Matlab a Simulink : úvod do používání /

Article · January 2005

Source: OAI

CITATIONS
7

READS
968

2 authors:

František Dušek
University of Pardubice
35 PUBLICATIONS **132** CITATIONS
SEE PROFILE

Daniel Honc
University of Pardubice
69 PUBLICATIONS **261** CITATIONS
SEE PROFILE

MATLAB a SIMULINK

úvod do používání

doc. Ing. František Dušek, CSc.
Ing. Daniel Honc, Ph.D.

Univerzita Pardubice

2005

Děkuji firmě HUMUSOFT Praha, díky jejíž pomoci bylo možné tato skripta realizovat. Jmenovitě panu Ing. Petru Byronovi za vstřícný a konstruktivní přístup.

ISBN 80-7194-776-8

Předmluva k prvnímu vydání

Tato skripta vychází z učebního textu "*Úvod do používání MATLAB*", který vznikl v roce 1997 a byl napsán jako pracovní materiál pro předmět základního kurzu MATLAB, a ze zkušeností se třemi roky výuky tohoto předmětu. Jsou určena především pro studenty technické vysoké školy a při jejich psaní jsem byl veden snahou studenty neodradit už při čtení skript a nezahltit je množstvím podrobností, při studiu kterých se ztrácí smysl proč se to vlastně dělá. Proto nechť laskavý čtenář promine poněkud lehčí tón některých pasáží[1]. Také je potřeba hned na začátku upozornit na to, že se předpokládá, že čtenář má možnost[2] MATLAB používat, příklady si zkusit[3] a využívat vestavěnou nápovědu.

Skripta by měla pomoci vyřešit dva problémy - jednak relativní nedostupnost (hlavně finanční[4]) literatury o MATLABu pro studenty a jednak nedostatek stručné dokumentace pro základní praktické používání.

První část skript je určena pro získání znalostí jak používat MATLAB při řešení úloh všeobecného zaměření. Pro plné porozumění příkladům se předpokládá znalost matematiky (a pro komplexní příklady i fyziky) na úrovni I. ročníku VŠ technického směru. Nezabývá se používáním toolboxů, které jsou zaměřeny na určitou oblast a jejich používání vyžaduje speciální znalosti z daného oboru.

Druhá část skript je věnována používání nadstavby MATLABu - SIMULINK. Tuto část lze studovat samostatně. Avšak vzhledem k tomu, že jde o nadstavbu MATLABu, je pro důkladnější pochopení a využití možností SIMULINKu vhodné mít aspoň základní znalosti MATLABu.

Jednou z předností MATLABu by mělo být, že není potřeba o něm příliš vědět a již je možné ho využívat i pro poměrně náročné výpočty. Tento text by měl představovat **stručný souhrn vybraných informací**[5] pro efektivní používání MATLABu. V žádném případě nelze text považovat za referenční příručku spíš za učebnicí. Naleznete zde proto jen některé vybrané základní příkazy a funkce, jejich použití v příkladech a upozornění na některé problémy či chyby často se vyskytující při běžném používání.

Cílem autora nebylo pouze popsat některé funkce a možnosti jednoho z nástrojů pro numerické řešení úloh, ale pokud možno i ukázat způsob řešení, respektive přístup k řešení problému. Chtěl bych zdůraznit, že mocný nástroj a jeho znalost sama o sobě žádný problém nevyřeší. Teprve když jsem daný problém zvládl (rozumím principu, vím co a proč to dělám) a jsem schopen navrhnout postup řešení, má význam sáhnout po nějakém nástroji umožňujícím konkrétní řešení. Samozřejmě znalost používaného nástroje a jeho možností může významně ovlivňovat návrh řešení. Výsledný postup jak návrhu řešení, tak i vlastního řešení je pak efektivní a rychlý. Na řešených příkladech kapitoly 9 by mělo být toto zpětné ovlivnění postupu řešení znalostí možností řešícího nástroje vidět (aspoň je to autorovo zbožné přání).

Jak již bylo řečeno skripta jsou rozdělena do dvou částí. První část - kapitola první až desátá jsou věnovány vlastnímu MATLABu. Část druhá - kapitoly deset až čtrnáct - se zabývají SIMULINKem. Uspořádání by mělo sledovat přirozený postup učení se něčemu novému. Nejprve je potřeba vyvolat zájem o to, co se budu učit.

K čemu je to dobré, proč zrovna toto a ne něco jiného, mám vůbec na to (myšleno hardware a software) - *kapitola první* **Úvod**.

Potom následuje první kontakt (co uvidím, na co mám máčknout, jak se to chová, jak se s tím zachází) - *kapitola druhá* **Začínáme.**

Když už vím co se mačká začne věda. Mělo by mě zajímat s čím se vlastně hlavně pracuje, s jakými základními prvky, na jakých principech je to postavené - *kapitola třetí* **Základní operace s maticemi**.

---

[1] , který má za účel zmírnit míru znechucení nešťastníka nuceného tento text studovat

[2] zcela dostačující je The Student Edition of MATLAB - součást publikací [152,153]

[3] Neustále potvrzovaná zkušenost ukazuje, že pokud je studentům něco vysvětlováno, tvrdí, že je to jasné. Teprve v okamžiku, kdy mají sami něco udělat, zjistí, že to je jasné jako tunel.

[4] tuto větu píši v okamžiku, kdy se ještě neví co budou skripta stát. Doufám, že bude platit i v době, kdy bude výsledná cena známa.

[5] znalci MATLABu nechť prominou, že jsem vynechal (ať už úmyslně či neúmyslně) velké množství funkcí a možností, které MATLAB poskytuje a těmi, na které se dostalo, se zabývám jen stručně a obvykle jen v té nejjednodušší podobě

Umí to ale také něco složitějšího než prostou posloupnost příkazů ? Jaké jsou možnosti řešení složitějších případů ? - *kapitola čtvrtá* **Řízení průběhu výpočtu**.

Výpočty jsou sice hezké ale obrázek je obrázek (jestli je to dobře nevím, ale obrázek vypadá hezky) - *kapitola pátá* **Vizualizace**.

Jsem líný[6]. Musím pokaždé znovu vypisovat všechny příkazy? A co když chci provést ten samý výpočet pouze s jinými daty, lze si ulehčit práci? - *kapitola šestá* **Tvorba skriptů a funkcí**.

Co užitečného to ještě standardně umí? - *kapitola sedmá* **Funkce funkcí**.

Proboha, je z toho zkouška. Co asi po nás bude chtít? - *kapitola osmá* **Jednoduché řešené příklady (MATLAB)**.

Zaplať Pán Bůh, že je už konec, dál už to stejně číst nebudu (něco konkrétního řešit nebo dokonce pracovat to je až ta poslední možnost jak se živit) - *kapitola devátá* **Komplexní řešené příklady (MATLAB)**.

Poslední kapitolou věnovanou vlastnímu MATLABu je *kapitola desátá* **Další možnosti MATLABu**, která obsahuje stručný přehled některých dalších vlastností a možností, které jsou v posledních verzích k dispozici a autor je považoval za natolik zajímavé či užitečné, že stojí za to se o nich zmínit.

Následující kapitoly - počínaje *kapitolou jedenáctou* **Co je to SIMULINK** - jsou věnovány nadstavbě MATLABu pro simulaci[7] (časové chování) dynamických soustav.

Dvanáctá kapitola **Jak se v SIMULINKu pracuje** se zabývá způsobem grafického zápisu simulovaného problému - modelu, základními bloky standardních knihoven a parametry vlastní simulace.

V *kapitole třinácté* **Subsystémy a knihovny** jsou ukázány možnosti usnadnění tvorby složitých či opakovaných modelů.

Poslední *kapitola čtrnáctá* **Řešené příklady (SIMULINK)** je věnována konkrétním příkladů na použití SIMULINKu. Tyto komentované příklady ukazují použití nejdůležitějších bloků standardních knihoven a postup konstrukce modelu ve složitějších případech.

Pro uspokojení autora a pro ty, kteří by chtěli využívat tento text, aby se něco naučili, je na konci ještě uveden **Rejstřík** názvů, funkcí a pojmů, které se v textu objevují a autor považoval za užitečné je samostatně uvést. A úplně nakonec poměrně rozsáhlý seznam **Literatury**, která se na MATLAB odkazuje, řazený podle technických oborů.

I když jsem byl veden snahou popisovat používání MATLABu v podobě pokud možno nezávislé na verzi MATLABu a použité platformě, byly všechny použité funkce a příkazy ověřovány na verzi 6.1 pod Windows 98, kterou jsem měl k dispozici. Též část věnovaná SIMULINKu[8] vychází z možností verze 4.0 pro Windows a o případných omezeních či rozšířeních na platformě UNIX není autor informován. Z tohoto pohledu je kapitola 10 výjimkou, neboť se zabývá možnostmi poskytovanými pouze posledními verzemi.

Pokud mi některý z čtenářů bude chtít sdělit jaké chyby v těchto skriptech objevil, co mu chybělo či co si myslí, že je naopak zcela zbytečné, může tak učinit nejlépe elektronickou poštou na adrese **frantisek.dusek@upce.cz**.

v Hradci Králové 29.února 2000 *František Dušek*

---

[6] dle názoru autora jedna z dominantních lidských vlastností

[7] dovoluji si zdůraznit, že jde o plnohodnotný simulační prostředek - ne pouze numerické řešení soustavy nelineárních diferenciálních rovnic. To znamená že má vlastnosti očekávané u prostředků pro simulaci (např. automatické přizpůsobení okamžiku výpočtu při nespojitých nelinearitách, volbu různých metod výpočtu, řešení počátečních vnitřních stavů atd.)

[8] SIMULINK jako nadstavba MATLABu se objevil až s verzí 4 pod Windows. Jeho použití je silně závislé na grafických možnostech prostředí, ve kterém je MATLAB spouštěn

Předmluva k druhému vydání

Základním podnětem pro druhé vydání byl příchod nové „velké“ verze MATLABu (Release 12). MATLAB je nyní ve verzi 6.1 a SIMULINK ve verzi 4. Zatímco změny v nové verzi MATLABu jsou zaměřeny spíše na vylepšení existujících vlastností a zlepšení komfortu uživatele, změny v SIMULINKu jsou podstatné. Otázkou je, do jaké míry tyto změny zahrnout do tohoto typu publikace, kde se popisuje to, co by mělo fungovat všude. Doufám, že se mi podařilo zachovat ráz publikace tj. zaměření na začínající uživatele MATLABu. Je potřeba udržet určitý kompromis mezi přílišnou jednoduchostí a přílišnou složitostí textu. Navíc v případě, že se doplňuje již napsaný text je obvykle snaha ho rozšiřovat a "nacpat" tam toho co nejvíc. I já jsem se neudržel, zářným příkladem budiž kapitola desátá, kde je nacpáno kde co. Omluvou snad je, že i to je jen zlomkem možností, které MATLAB nabízí. Vážený čtenář by ji měl brát jako "upoutávku" na další využití MATLABu. Obdobně i kapitoly 9 a 14 – Komplexní řešené příklady – je potřeba brát jako ukázku řešení (snad zajímavých) složitějších problémů.

Kromě aktualizace na nové vlastnosti a možnosti MATLABu jsem využil příležitosti a odstranil některé nedostatky předešlého vydání a obměnil a doplnil některé příklady. Kapitoly 1 až 8 jsou v podstatě pouze opraveny a aktualizovány. Kapitoly 9 a 14 – Komplexní řešené příklady – zůstaly zachovány ve stejné podobě (přes určité výhrady, které byly k jejich zařazení) protože stále zastávám názor, že MATLAB je pouze nástroj a pouze platonický popis jeho možností, bez použití na řešení reálných problémů, je sám o sobě k ničemu. Je zde zařazeno několik nových příkladů. Nezastupitelným úkolem kapitoly 14 je také na konkrétních příkladech ukázat funkci a použití jednotlivých bloků. Proto i pouze studium převodu matematického modelu na model v SIMULINKu může být užitečné.

Rozšířena je kapitola 10, která je zaměřena na pokročilejší uživatele a mimo jiné popisuje některé méně obvyklé možnosti MATLABu (výměna dat pomocí DDE, Active X a sériové linky). Také poslední dva příklady kapitoly 14 poněkud vybočují z původního záměru psát o základním použití. Slabší nátury nechť je při prvním čtení vynechají.

Značně přepracovány jsou všechny kapitoly věnované SIMULINKu a to nejen z důvodů změny vzhledu a uspořádání nové verze. SIMULINK zřejmě prochází výraznými principiálními změnami. Z relativně jednoduché grafické nadstavby MATLABu se postupně vyvíjí v plnohodnotný systém respektující filosofii MATLABu a poskytující služby, které se od něj očekávají. Přiznávám se, že jsem SIMULINKu nevěnoval tolik pozornosti jak bych chtěl a proto jsem vděčný za pomoc a informace, které mi poskytl kolega ing. Daniel Honc.

v Hradci Králové 26.července 2002 *František Dušek*

Rád bych poděkoval Ing. Danielu Honcovi, Ph.D. za pečlivé přečtení textu, cenné připomínky a v neposlední řádě i za pomoc při závěrečném zalomení skripta.

Můj dík za trpělivost a dodatečná omluva též patří všem, kteří museli snášet moje nepublikovatelné projevy během litého boje s textovým editorem MS Word 2000, ve kterém jsem připravoval druhé vydání.

Předmluva k třetímu vydání

Od druhého vydání již uplynuly tři roky a firma MathWorks nezahálela. MATLAB je nyní aktuálně (duben 2005) ve verzi 7.0.4 a SIMULINK ve verzi 6.2 (Release 14 SP2). Nové verze dále rozšiřují nejen možnosti použití, ale také zvyšují nároky na znalosti uživatele, který je chce využívat. Tento přístup sice dovoluje mít jeden prostředek, ve kterém lze udělat skoro vše, ale na druhou stranu odrazuje potencionálního uživatele začít MATLAB používat, protože se v nabízených možnostech neorientuje[9]. Rozhodnutí mezi jednoduchostí (neodradit začínající uživatele) a komplexní nabídkou (uspokojit zkušené uživatele) je i problémem těchto skript.

Základní členění zůstalo zachováno, nicméně v tomto vydání bylo provedeno poměrně dost změn. Skripta jsou nyní (snad) více zaměřena na začínající uživatele MATLABu. Jsou poněkud omezeny komplexní příklady kapitol 10 a 16 – několik příkladů[10] bylo vyřazeno. Naopak jsou posíleny jednoduché příklady na procvičování. Z tohoto důvodu jsou zařazeny i dvě nové kapitoly. V části věnované MATLABu je nová kapitola *Pro netrpělivé uživatele (aneb MATLAB v kostce)*. Zde je shrnuta základní filosofie MATLABu a na příkladech ukázány možnosti řešení různých úloh od jednoduchých po komplexní včetně příkladů využívajících Symbolic Math Toolbox. V části zabývající se SIMULINKem je doplněna kapitola *Jednoduché řešené příklady (SIMULINK)*.

Kapitoly 12 až 16 věnované SIMULINKu jsou aktualizované a doplněné Ing. Danielem Honcem, Ph.D., kolegou, kterému vděčím za pomoc při přípravě předchozích vydáních. SIMULINK je nejvíce se měnící částí MATLABu (od minulého vydání přibyly dvě „velké“ verze“) což se projevuje mimo jiné narůstajícím počtem bloků v knihovnách a stále se měnícím systém zařazování bloků do knihoven. Důsledkem je, že verzi od verze je na první pohled všechno jinak. Naštěstí tentokráte tu práci (všechno projít, vybrat to nejdůležitější a stručně a jasně to popsat) za mě udělal někdo jiný.

Přes velkou snahu se nepodařilo udržet rozsah minulé verze. Ačkoliv byla maximálně využívána[11] plocha stránky, byl zrušen seznam obrázků a silně redukován seznam literatury (nyní obsahuje pouze odkazy na literaturu týkající se vlastního MATLABu a SIMULINKu) došlo k zvětšení rozsahu o 14 stránek.

v Pardubicích 6. června 2005 *František Dušek, Daniel Honc*

[9] toho jsou si vědomi i jiní. Proto vznikají systémy kopírující jednoduchou filosofii MATLABu, ale nenabízející tolik možností a některé jsou zdarma. Typickým příkladem je **Octave** (www.octave.org), které je údajně na cca 80% kompatibilní s MATLABem. **Scilab** (http://scilabsoft.inria.fr/) je také podobný MATLABu, ale zahrnuje již opět mnoho možností.

[10] nicméně dvě „chuťovky“ (poslední dva příklady kapitoly 16) pouze pro silné jedince zůstaly

[11] což je proti trendu poslední doby. Nyní je módní využívat efektivní plochu stránky tak na 70% zejména u publikací zabývající se výpočetní technikou. Otázkou je zda to je móda nebo důsledek používání textového editoru MS Word. Vytvořit publikaci s větším počtem obrázků obtékaných textem ve Wordu vyžaduje hodně času a hodně pevné nervy.

Obsah

1. Úvod

V překotně se vyvíjejícím světě výpočetní techniky a programového vybavení mají od prvopočátku nezastupitelné místo programy pro numerické výpočty. Zpočátku byla podpora uživatele realizována matematickými knihovnami pro obecné programovací jazyky. Později se vyvinuly samostatné programy, které je možné rozdělit asi do tří oblastí - tabulkové procesory s rozšířenými matematickými funkcemi, balíky specializované na řešení určitého okruhu problémů (statistika, kvantová chemie, chemické inženýrství) a profesionální univerzální matematické balíky. Do posledně jmenované oblasti patří mj. MathCad, Mathematica a MATLAB[12] .

1.1 Co to je MATLAB ?

Název MATLAB vznikl z anglického MATrix LABoratory. MATLAB byl napsán, aby poskytoval jednoduchý přístup k matematickým knihovnám vyvinutým v projektech[13] LINPACK a EISPACK. Byl původně určen pro operační systém UNIX a tato okolnost se dodnes (i v prostředí Windows) projevuje ve velmi jednoduchém základním komunikačním rozhraní - příkazové řádce. V posledních verzích (od roku 2000) jsou nyní využívány matematické knihovny ARPACK (Arnoldi PACKage), LAPACK (Linear Algebra PACKage) a BLAS (Basic Linear Algebra Subroutines).

Co to tedy je MATLAB ? Slovy firemní literatury *"MATLAB je vysoce výkonný jazyk pro technické výpočty. Integruje výpočty, vizualizaci a programování do jednoduše použitelného prostředí, kde problémy i řešení jsou vyjádřeny v přirozeném tvaru"*. Jde o interaktivní systém jehož základním datovým typem je dvourozměrné[14] pole (bez nutnosti deklarovat rozměry). Tato vlastnost spolu s množstvím zabudovaných funkcí umožňuje relativně snadný zápis a řešení mnoha technických problémů, speciálně takových, které vedou na vektorovou či maticovou formulaci, v mnohem kratším čase než řešení v klasických jazycích jako je "C" nebo FORTRAN. Typické oblasti použití jsou:

- inženýrské výpočty
- vývoj algoritmů
- modelování, simulace a vývoj prototypů
- analýza dat a jejich vizualizace
- inženýrská grafika
- vývoj aplikací včetně tvorby grafického uživatelského rozhraní

V univerzitním prostředí jde o standardní nástroj využívaný ve výuce matematiky a inženýrských oborech. V průmyslu je využíván jako vysoce efektivní nástroj pro výzkum, vývoj i analýzu dat. O tom, že je skutečně používán svědčí přes 800 publikací[15] z různých oborů, které MATLAB při řešení problémů používají a řešení ve formě m-funkcí v knize nejen uvádějí, ale ve formě souborů buď na doprovodné disketě (CD) nebo na některém FTP serveru také nabízejí.

Aktuální informace o MATLABu lze získat na WWW serveru (http://www.mathworks.com) americké firmy The MathWorks, Inc. nebo fy HUMUSOFT Praha (http://www.humusoft.cz), výhradního zástupce The MathWorks pro Českou republiku, Slovensko, Rusko, Ukrajinu, Bělorusko, Moldávii a Kazachstán.

1.2 Kdy použít MATLAB ?

Ve srovnání s jinými obdobnými produkty je MATLAB blíže programovacímu jazyku. Nemá tolik předpřipravených komplexních matematických funkcí jako např. Mathematica, nemá integrované vlastnosti

---

[12] MathCad® je chráněná značka fy MathSoft, Inc. (www.mathsoft.com), Mathematica® je chráněná značka fy Wolfram Software Inc. (www.wolfram.com), MATLAB® je chráněná značka fy The MathWorks, Inc. (www.mathworks.com)

[13] jeden z autorů první verze MATLABu, Cleve Moler profesor matematiky a computer science, byl také jedním z autorů těchto knihoven

[14] od verze 5 existuje možnost obecnějšího datového typu - vícerozměrného pole

[15] podle seznamu literatury využívající MATLAB evidovaného na www firmy MathWorks k 14.4.2005

textového editoru jako např. MathCad. Zrovna tak není jeho standardní součástí podpora symbolických výpočtů jako u obou výše jmenovaných produktů.

Ale to neznamená, že je o tyto možnosti ochuzen. Obsahuje množství (více než 500) jednoduchých[16] i složitějších[17] matematických funkcí implementovaných ve formě vysoce efektivních a robustních algoritmů, které jsou součástí jádra (built-in function). Z těchto funkcí lze složit podle konkrétní aplikace další libovolně složité funkce. Skupiny takovýchto funkcí respektive funkcí hodících se k řešení určitého okruhu problémů se v MATLABu nazývají toolboxy[18]. Obvykle byly vytvořeny (nebo aspoň jejich základ) na univerzitách týmem seskupeným okolo významného odborníka v dané oblasti. Velkou výhodou, aspoň pro teoretické využití je to, že dokumentace k toolboxům obsahuje stručný princip a odkazy na literaturu, kde jsou použité algoritmy podrobně rozebrány.

Symbolickou matematiku a výpočty s definovanou přesností[19] lze zahrnout pomocí Symbolic Math Tbx. Tento toolbox zahrnuje výpočetní jádro MAPLE[20] 8.

Pro případ, že potřebujeme výpočty v MATLABu jako součást např. zprávy, je možné exportovat vytvořené obrázky či grafy do různých grafických formátů. Od verze 4 je součástí instalace pro operační systém (OS) Windows na PC a počítače MacIntosh přímo šablona pro textový procesor MS Word, která umožňuje volat příkazy MATLABu a vkládat výsledky (včetně obrázků) přímo v prostředí editoru. Od verze 6 je tato šablona v adresáři notebook.

Dalším rozšířením (se kterým nemá autor konkrétní zkušenost) je propojení MS Excelu s MATLABem pro snazší zadávání dat a prezentaci výsledků.

Mezi další možnosti MATLABu, se kterými autor nemá zkušenosti patří například

- vývoj aplikací (filtrace signálů, řízení) pro signálové procesory
- možnost překladu odladěného algoritmu do funkce v jazyce "C" použitelné ve vlastním programu
- převod hotové aplikace v MATLABu (včetně grafického uživatelského rozhraní) do samostatného spustitelného tvaru (bez nutnosti přítomnosti jádra MATLABu)
- použití MATLABu jako WEB serveru pro zajištění výpočtů v rámci WWW stránky

Jako samostatná nadstavba MATLABu existuje SIMULINK – časové řešení soustavy nelineárních diferenciálních rovnic s grafickým zadáváním řešené soustavy připomínajícím zapojení na analogovém počítači. Umožňuje graficky sledovat průběhy veličin v libovolném místě zapojení. Používá se např. pro simulaci dynamického chování sledovaného systému.

Další samostatnou nadstavbou MATLABu byl původně FEMLAB[21]. Tento, nyní samostatný produkt, je vyvíjen švédskou firmou COMSOL. Jde o systém na modelování problémů popsaných kombinací soustav parciálních diferenciálních rovnic s obyčejnými diferenciálními a algebraickými rovnicemi. V současné době (jaro 2005) je nabízen s nadstavbovými moduly Chemical Engineering Module (transportní jevy a chemická kinetika), Earth Science Module (geofyzikální problémy a životní prostředí), Electromagnetismus Module, Heat Transfer Module, MEMS Module (mikro elektromechanika) a Structural Mechanics Module (nelineární materiály, velké deformace).

---

[16] všechny běžné matematické funkce (pracující i s komplexními hodnotami) a základní operace s maticemi

[17] např. Besselovy funkce, gama funkce, generování prvočísel, transformace souřadnic, Fourierova transformace, maticové funkce, funkce pro práci s řídkými maticemi, funkce pro práci s polynomy, numerická integrace funkce, nulová hodnota fce, minimum funkce více proměnných, řešení soustavy diferenciálních rovnic I. řádu atd.

[18] problémově orientované (specializované) soubory funkcí pro řešení problematiky určitých oborů - Control System, Communications, Financial, Frequency Domain System Identification, Fuzzy Logic, Higher-Order Spectral Analysis, Image Processing, LMI Control, Model Predictive Control, μ-Analysis and Synthesis, NAG Foundation, Neural Network, Optimizatiom, Partial Differentuial Equation, QFT Control Design, Robust Control, Signal Processing, Spline, Statistics, Symbolic Math, System Identification

[19] např. hodnotu s p přesností 500 cifer

[20] MAPLE® je chráněná značka Waterloo Maple Software, Inc. (www.maplesoft.com)

[21] FEMLAB® je chráněná značka COMSOL AB, (www.femlab.com)

Kdy tedy MATLAB použít? Odpověď na tuto otázku je velice důležitá a měl by si ji položit každý. Odpověď není jednoznačná. Záleží na konkrétních požadavcích a podmínkách konkrétního uživatele. Obecné zdůvodnění snad lze shrnout v následujících větách.

V případě, že potřebuji robustní výpočty, nestandardně jednorázově zpracovávat rozsáhlé datové soubory, pracovat s velkými maticemi a vůbec v případech, kdy řešení problému se dá převést na vektorové či maticové operace. Vzhledem k možnostem programování je MATLAB s výhodou použitelný i v případech rozvětvených případně iteračních algoritmů řešení. Nezanedbatelnou výhodou je i to, že díky jednoduchému ovládání lze po cca 6 hodinách[22] s MATLABem samostatně pracovat a řešit jednodušší příklady. Dále je běžným prostředkem používaným na většině univerzit technického zaměření na světě. Také to, že MATLAB je k dispozici na všech běžných platformách může při výběru hrát roli.

Kdy použití MATLABu není vhodné? Použití MATLABu není vhodné v situaci, kdy potřebujeme provádět opakovaně stále stejné (byť složité) výpočty nebo opakovaně zpracovávat velké objemy dat. Zkrátka v situacích, kdy je rozhodující vlastní rychlost výpočtu a ne vývoj algoritmu výpočtu. V tomto případě je vhodné hotový algoritmus zapsat v nějakém programovacím jazyce, který umožní vytvořit spustitelný kód a nepoužívat MATLAB, který je stále jen interpret.

1.3 Jak se vyvíjel MATLAB

V roce 1997 jsem při přípravě prvního učebního textu napsal: „*Původně byl MATLAB vyvinut pro počítače pracující pod OS Unix. Především pro Unix (na různých platformách[23]) je určen MATLAB dodnes. Donedávna nebyl výkon počítačů PC dostatečný pro profesionální práci. Dnes už to sice není tak úplně pravda, ale firma MathWorks nepovažuje PC za nástroj pro profesionální práci. Svědčí o tom krom jiného i ta okolnost, že verze MATLABu pro PC nejsou nikterak chráněny proti nedovolenému kopírování na rozdíl od verzí pro Unix, kde je instalace vázána na sériové číslo procesoru*“. Dnes (rok 2005) je situace zcela jiná. Platforma na bázi procesorů INTEL (AMD) je výkonově srovnatelná s RISCovými procesory a hlavně podstatně levnější. Operační systém Windows ve verzích odvozených od Windows NT (2000, XP) je již dostatečně stabilní a zejména rozšířenější než UNIX (nebo LINUX).

Na platformě Windows se využívá pro ochranu nedovoleného kopírování MATLABu toho, že je nutné programy instalovat. Instalační program dovolí z instalačních CD (které obsahují všechny soubory k příslušné verzi MATLABu tzv. Release v zašifrované podobě) po zadání příslušného PLP[24] instalovat pouze ty části, na které si uživatel zakoupil licenci. V rámci instalace se vygeneruje soubor (LICENSE.DAT) s jedinečnou identifikací instalovaných součástí a počítače, na kterém byla instalace provedena. Na tento soubor se MATLAB při startu obrací. Pokud není správný, MATLAB se nespustí. Podobný koncept je použit při správě tzv. plovoucích licencí, kdy v síti na jednom počítači běží správce licencí, na kterého se MATLAB při spuštění obrátí. Tento správce licencí pak omezuje počet současně spuštěných instancí MATLABu na počet zakoupených licencí.

První verze pro PC XT se objevila kolem roku 1985. Základním problém byl nedostatek paměti - a z toho plynoucí omezení na maximální velikost matic. Potom, co se objevilo PC AT, rychle následovala verze

---

[22] nutno brát s rezervou - záleží na zkušenostech a inteligenci potencionálního uživatele. Ale názvy knih **Learn MATLAB 5 in 6 hours!** (Finn Haugen 1997, ISBN 82-91748-02-0) a **Learn SIMULINK 2 in 3 hours!** (Finn Haugen 1997, ISBN 82-91748-03-9) to tvrdí.

[23] v BENCHi dodávaném s verzí 5.1 jsou uvedeny pro porovnání doby provádění testovacího programu na těchto počítačích: SUN Ultra 2/200, SGI Impact 1000/195, PentiumPro 200/LINUX, PentiumPro 200/Win95, PentiumPro 200/WinNT, IBM RS6000, DEC Alpha 400/233, MAC 8500/180, HP 735/125, SGI RS4400/150, SUN Sparc 10/41, Mac 5300c Powerbook, SUN Sparc 2.

Ve verzi 5.2: DEC Alpha 500, Sparc Ultra 2 (Dual 300), SGI Octane 195, Power Mac G3 /500, HP 780 /180, Mac Powerbook G3 /250, SGI O2 /180, Pentium II NT /300

Ve verzi 6.1: DEC Alpha 21264A 667, PII NT 867, HP PA-85000, Athlon 600, Sun UltraSPARC-II, SGI R12000/300, IBM Power3 AIX 200

Ve verzi 7.0: Pentium 4 3.0 GHz (WinXP SP1), AMD Opteron Dual 2.1 GHz (SuSE Linux 9.0), Pentium 4 2.0 GHz (Win2000 SP4), Macintosh G5 Dual 2 GHz (OS X 10.3.2), AMD Athlon Dual 1.667 MHz (Red Hat Linux 8.0), Macintosh G4 Dual 1.25 GHz (OS X 10.3.2), Sun UltraSPARC-III Dual 1.2 GHz (Solaris 5.8), HP-UX Dual 875 MHz (HP-UX 11.0)

[24] Personal Licence Password – série 5-ti místných čísel uvedená dvojicí čísel odpovídajících Release (např. 14-12345-12345-12345-12345-12345)

MATLABu pro tento počítač. Zde byla maximální velikost matice omezena fyzickou velikostí paměti (pro PC AT max. 16 MB). Osazení větší pamětí než 1 MB však, vzhledem k tehdejším cenám pamětí, stejně nebylo (zvláště v našich podmínkách) běžné. Dalším omezením (opět speciálně v našich podmínkách) byla nezbytná přítomnost[25] matematického koprocesoru.

Velké obliby doznala verze MATLAB 386, která byla, jak už název naznačuje, určena pro počítače PC s procesorem 80386. Využívala jednu z možností tohoto procesoru – podporu virtuální paměti. Což znamená, že program pracuje s virtuální pamětí, která může být větší než je skutečná fyzická paměť. Dochází k odkládání dat na pevný disk počítače. Tato skutečnost sice vede (pakliže k odkládání dojde) k zpomalení výpočtů , ale je možné výpočty na velkých maticích[26] vůbec provést. Poslední verze MATLAB386 g z října 1995 byla používána velmi dlouho, protože byla vzhledem k tomu, že pracuje pod OS DOS, nepřekonatelná v rychlosti výpočtů. Všechny tyto verze MATLABu pracovaly pod OS DOS.

Přibližně v roce 1994 byla uvedena na trh verze MATLAB for Windows, která nabízela podstatně bohatší možnosti grafického rozhraní než předchozí verze. V této verzi se objevil poprvé SIMULINK. Po výpočetní stránce mnoho nového nenabízela (naopak výpočty na tom samém počítači byly, díky většímu vytížení procesoru režií operačního systému, pomalejší). MATLAB for Windows skončil verzí 4.2c, která se přestala dodávat koncem roku 1996.

Nástupcem verze 4 byla verze 5 určená v rámci platformy Intel do 32ti bitového prostředí (pod OS Windows 95 nebo Windows NT). Verze 6 byla plně 32 bitová a přinesla nové možnosti plynoucí z využití objektového přístupu – zejména nové datové struktury, rychlejší grafiku, nové vizualizační funkce a další rozšíření nabídky základních funkcí. Novinkou bylo, že obsahovala čtyři různá výpočetní jádra optimalizovaná pro různé typy procesorů.

Momentálně poslední verzí je MATLAB 7.0.4 z ledna 2005. Tato verze vyžaduje Windows NT/2000/XP. Obsahuje 4 optimalizovaná výpočetní jádra pro procesory Pentium (PII, PPro, PIII, P4) a jedno pro Athlon. Tato verze je dostupná kromě pro Windows také pro platformy HP-UX 11, Linux, Mac OS X, Solaris a částečně Linux x86-64 (ne všechny toolboxy).

1.4 Požadavky na instalaci

Všechny verze od prvopočátku bezpodmínečně vyžadují numerický koprocesor. Dnes v době procesorů Pentium je to sice samozřejmost, ale v případě, že chceme využít starší počítač či notebook je potřeba na to myslet. Další požadavky se liší podle toho, jakou verzi MATLABu máme k dispozici. V případě verze 7 jsou základní požadavky dané tím, že lze instalovat pouze na počítač s Windows NT/2000/XP. Velikost paměť je minimálně 256 MB (doporučeno 512 MB) a vzhledem k grafickým možnostem je doporučená rychlá grafická karta. s 3D akcelerátorem a podporou OpenGL. Minimální požadavek na velikost diskového prostoru je od 400 MB (pouze pro MATLAB + help) po více než 1 GB (podle instalovaných toolboxů). Instalace se provádí po zadání příslušného PLP z CD.

On-line nápověda ve verzi MATLABu 5 vyžadovala funkční WWW prohlížeč např. standardně s Windows 95 dodávaný Microsoft Internet Explorer (tedy i na počítači nepřipojeném k Internetu). Od verze 6 je použit vlastní prohlížeč HTML souborů tvořících nápovědu. Úplná verze elektronické dokumentace (kopie tištěné dokumentace) ve formátu Adobe PDF byla k dispozici do verze 6. K jejímu prohlížení a tisku je nutný program fy Adobe Systems Inc. - Acrobat Reader. Verze 3.0 byla součástí instalačního CD verze 5. Od verze 7 není součástí instalace ani instalačních CD dokumentace ve formátu PDF. V případě připojení k internetu lze využívat systém nápovědy (http://www.mathworks.com/access/helpdesk/help/helpdesk.html) na firemním serveru.

---

[25] i tento požadavek vyplýval z profesionálního zaměření produktu. Matematický koprocesor významně urychluje výpočty což je hlavní činnost MATLABu. Matematický koprocesor je integrální součástí procesorů INTEL (AMD) od verze 80486.

[26] např. na počítači PC386/40 s 4 MB fyzické paměti bylo možné provést výpočet inverze matice 1000×1000. Samotná matice potřebuje pro uložení 8 milionů bytů tj. skoro 8 MB. Výpočet sice trval několik minut, ale byl proveditelný.

Od 1.1.2000 došlo k určité změně licenční politiky fy MathWorks v oblasti poskytování licencí pro školství. Nadále je poskytována ***Školní licence pro všeobecné využití***[27]. Změna je u multilicence pro výuku – ***Classroom Kit***[28] – (nyní pouze v podobě síťové multilicence) a nově je zavedena ***Studentská licence***[29]. Ta nahradila dříve samostatně nabízené ***The Student Edition of MATLAB***.

Dříve bylo možné zakoupit *The Student Edition of MATLAB 5* a *The Student Edition of SIMULINK 2*. V obou případech šlo o knihu se stručným popisem příslušného produktu a součástí bylo instalační medium s "demo" verzí, která se od "ostré" verze lišila pouze omezením na maximální velikost použitých matic.

1.5 Co za nás MATLAB nevyřeší

Jako poslední bod tohoto obecného povídání bych si dovolil upozornit na podstatnou věc, která neplatí jenom pro MATLAB. MATLAB je pouze nástroj, sice výkonný a efektivní ale je to pouze prostředek k dosažení cíle. Musím tedy nezávisle na použitém prostředku vědět:

- **co chci** - mít jasnou a jednoznačnou formulaci problému
- **jak toho dosáhnout** - vědět (aspoň principiálně) jak se daný problém řeší
- **jak ověřit výsledek** - být schopen ověřit či odhadnout zda je získaný výsledek správný

Tyto tři body za nás žádný sebedokonalejší výpočetní systém nevyřeší. Může ale výrazným způsobem ovlivnit efektivitu a rychlost řešení. Výkonnost a efektivita těchto nástrojů se ocení teprve až při řešení složitějších a rozsáhlejších problémů.

---

[27] **Školní licence pro všeobecné použití** - plné verze MATLABu, SIMULINKu a ostatních produktů za školní ceny. Licence jsou určeny pro vzdělávací, výzkumnou a publikační činnost studentů, pedagogů a ostatních zaměstnanců ve vzdělávacích institucích.

[28] **Classroom Kit (CK)** - síťová konfigurace MATLABu, SIMULINKu a ostatních souvisejících produktu,která je určena pouze pro pravidelnou výuku studentu zapsaných do kurzu v institucích poskytujících akademické vzdělání. CK nesmí být používán k jiným účelům než je pravidelná výuka. Zejména je zakázáno použití CK k vědecké a publikační činnosti pedagogů, doktorandů i studentů (mimo pravidelnou výuku)! Cena CK je podstatně nižší než cena konfigurace pro všeobecné školní použití a závisí na počtu jednotlivých uživatelských licencí v CK konfiguraci. CK je k dispozici pro všechny podporované platformy! CK je možné zakoupit pro všechny produkty systému MATLAB s výjimkou C/C++ Math Library, C/C++ Graphics Library a ML Web Serveru.

[29] **Studentská licence** - individuální konfigurace pro platformu PC. Licence jsou určeny pro provoz na soukromých počítačích studentů pro studijní účely. Programové vybavení musí být zakoupeno univerzitou v minimálním množství 10 kusů a dále distribuováno jednotlivým studentům do osobního vlastnictví. Minimální konfigurace (ML Suite) obsahuje MATLAB, SIMULINK a Symbolic Math Toolbox. Studentskou licenci je možné rozšířit i o toolboxy, ale pouze v okamžiku nákupu celé konfigurace a pro všechny uživatele stejně. Studentské licence není možné aktualizovat za update ceny. Ke studentským licencím není dodávána tištěná dokumentace.

2. Pro netrpělivé (aneb MATLAB v kostce)

Tato kapitola je určena pro netrpělivé zájemce o používání MATLABu, kteří chtějí hned začít a nechtějí nic moc studovat. Po krátkém úvodu do filosofie MATLABu a způsobu ovládání jsou na sérii konkrétních příkladů ukázány některé možnosti MATLABu. Předpokládá se znalost matematického aparátu potřebného pro řešení úloh.

2.1 Historie

MATLAB vznikl na platformě UNIX a důsledek je patrný dodnes – veškeré ovládání je možné z příkazové řádky a rozlišují se velká a malá písmena v názvech proměnných. Zaměření na platformu PC+Windows je relativně nové a využívání grafického prostředí Windows se objevuje postupně u nových verzí (od verze 4).

2.2 Filosofie MATLABu

Při prvním pohledu je MATLAB podobný BASICu – napíše se příkaz, který se ihned provede. Pokud chci sestavit více příkazů, které se pak provedou najednou – napíšu je do textového souboru s příponou .m. Napsání jména souboru na příkazový řádek je pokyn pro MATLAB, aby příkazy ze souboru provedl. Tento typ zápisu se nazývá **skript** – sdílí se proměnné pracovního prostoru (Workspace) příkazové řádky. Je-li na prvním řádku uvedeno **function** jde o funkci tj. proměnné funkce jsou lokální a přenos dat je přes parametry funkce.

Jména proměnných se tvoří standardním způsobem (nesmí začínat číslicí) a jsou jednoho datového typu – komplexní matice (reálné číslo – skalár je matice 1x1 bez komplexní části). Proměnné není potřeba předem deklarovat a to ani z hlediska rozměrů.

Součástí MATLABu je velké množství různých funkcí a příkazů. Mnoho z nich je realizováno formou kombinace jednodušších příkazů a funkcí, které jsou umístěny v textových souborech s příponou .m – nazývají se **m-funkce**. Tento způsob dovoluje vytvářet vlastní (uživatelské) m-funkce, které rozšiřují množinu příkazů.

Tento princip je využit u tzv.TOOLBOXů. Obvykle jde o soubor m-funkcí, které jsou zaměřeny na potřeby řešení úloh v určité specifické oblasti (např. Statictics Tbx rozšiřuje statistické funkce, Neural Network Tbx obsahuje funkce potřebné pro práci s neuronovými sítěmi atd.).

Kromě těchto toolboxů existuje ještě funkční nadstavby jako je např. SIMULINK (simulace časového chování systémů popsaných složitými soustavami nelineárními diferenciálních rovnic). Tyto nadstavby mají obvykle specifický způsob ovládání a formulaci problému.

Vzhledem k obrovskému množství funkcí a ovládání z příkazové řádky je velmi důležitá úloha nápovědy. Nápověda je ve dvou úrovních – jednoduchá na úrovni příkazové řádky (příkaz `HELP` případně `HELP` *`název`*) a komfortní hypertextová ve formátu HTML (položka hlavního menu HELP) zpřístupňující dokumentaci v elektronické podobě.

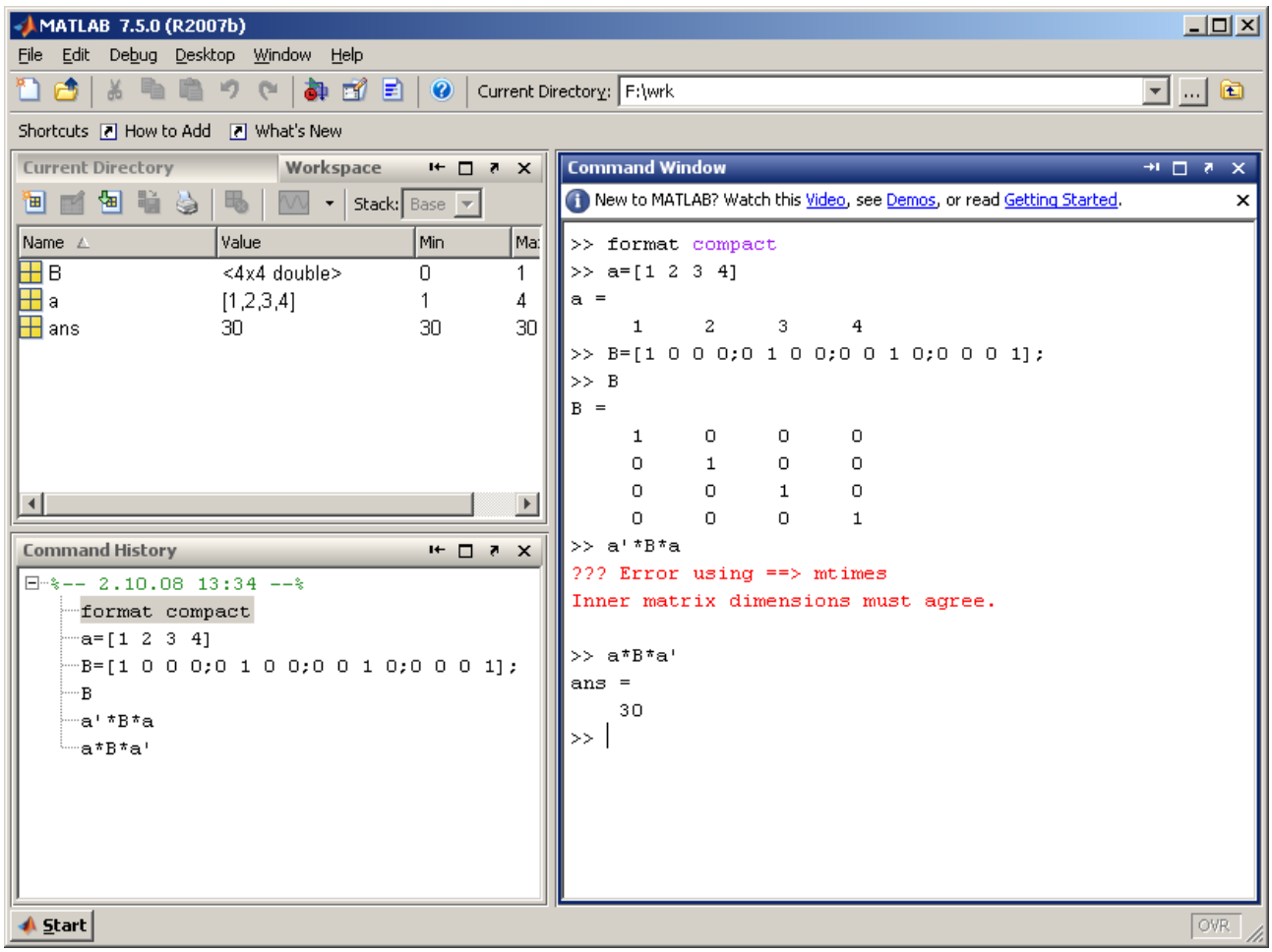
2.3 Pracovní plocha (desktop)

Pracovní plocha je od verze 6 organizována do několika oken – implicitní nastavení obsahuje 3 okna. Okno *Command History* obsahuje seznam (historii) příkazů napsaných na příkazový řádek. Okno v levé horní části má dvě záložky dovolující zobrazit buď seznam souborů v aktuálním adresáři (*Current Directory*) nebo seznam vytvořených proměnných (*Workspace*). Nejdůležitější je okno *Command*, do kterého se píší příkazy. Ukázka pracovní plochy MATLABu, kde jsou zachyceny i některé základní činnosti, je na obrázku 2-1.

2.4 Vytvoření a zápis proměnných

Jestliže příkazový řádek začíná znakem >> je dokončen předchozí příkaz a je možné zadávat nový příkaz. První příkaz příkazového okna na obrázku 2-1 ukazuje zadání řádkového vektoru s vypsáním výsledku (hranaté závorky, oddělení mezerami, bez středníku za příkazem). Druhý příkaz je příkladem zadání matice bez výpisu výsledku (hranaté závorky, oddělovač prvků na řádku - mezery, oddělovač řádků - středník,

středník za příkazem). Třetí příkaz – zapsání názvu proměnné – představuje požadavek na vypsání obsahu proměnné. Čtvrtý příkaz je požadavek na vyhodnocení maticového výrazu (transpozice a násobení), který nelze vyhodnotit. Je vidět reakce MATLABu na chybu. Pátý příkaz představuje totéž zapsané správně – není předepsána proměnná pro uložení výsledku (uloží se do proměnné s názvem `ans`) a není potlačen požadavek výpisu (chybí středník za příkazem).



obrázek 2-1 Standardní desktop MATLABu

2.5 Příklady MATLAB

Plnění a manipulace s vektory a maticemi

```
>> x=0:0.1:10;          % vytvoření řádkového vektoru x o 101 prvcích
>> A=zeros(4,5);        % vytvoření matice 4x5 naplněné nulami
>> A=ones(9,6);         % vytvoření matice 9x6 naplněné jednotkami
>> A=rand(7,8);         % vytvoření matice 7x8 naplněné náhodným číslem 0-1
>> A=eye(12,8);         % vytvoření matice 12x8 s jedničkami na hl. diagonále
>> a=A(2,8);            % výběr jednoho prvku matice A
>> a=A(:,8);            % výběr osmého sloupce matice A
>> B(1,2)=7;            % zápis do vybraného prvku matice
>> B=A(2:10,3:end);     % výběr submatice v matici A
>> b=rand(1,100);       % vytvoření řádkového vektoru o 100 náhodných prvcích
>> b=[1, b(2:end)];     % posun vektoru s naplněním prvního prvku hodnotou 1
```

Základní operace s vektory a maticemi

```
>> v=x*x';                          % skalární součin vektorů
>> V=x'*x;                          % vektorový součin vektorů
>> X=2*rand(1000,1000)+1;           % vytvoření matice naplněné náhodnými čísly –1 až 1
>> v=det(X);                        % determinant matice
>> V=inv(X);                        % inverzní matice
```

Používání funkcí

```
>> x=2:-0.5:-2;                     % vytvoření řádkového vektoru od 2 do –2 s krokem –0.5
>> y=sin(x);                        % aplikace funkce na jednotlivé prvky vstupní proměnné
>> sqrt(x)                          % plynulý přechod do komplexní oblasti
ans = 1.4142  1.2247  1.0000  0.7071 0 0+0.7071i 0+1.0000i 0+1.2247i 0+1.4142i
```

Maticové dělení

Řešme soustavu lineárních rovnic $A*x=b$, matice A je naplněna náhodnými čísly z intervalu <1,3> a vektor b náhodnými čísly z intervalu <0,5> (v obou případech rovnoměrné rozložení)

```
>> A=2*rand(3,3)+1                  >> b=5*rand(3,1)
A =  2.9003   1.9720   1.9129       b =  2.2235
     1.4623   2.7826   1.0370            3.0772
     2.2137   2.5242   2.6428            3.9597
```

a) Pro případ že matice soustavy A je čtvercová - počet řádků je stejný jako počet sloupců tj.

```
>> [nr,ns]=size(A)
nr =     3       ns =     3
```

je řešení pomocí inverze nebo pomocí maticového dělení zleva ekvivalentní.

```
>> x=inv(A)*b      nebo             >> x=A\b
x =  -0.6126                       x = -0.6126
      1.0530                            1.0530
      1.0057                            1.0057
```

b) Pro případ přeurčeného systému (větší počet rovnic než neznámých) jsou maticovým dělením zleva určeny takové hodnoty, které zajistí aby odchylka od pravé strany byla ve smyslu nejmenších čtverců minimální.

```
>> Ap=A(:,1:2)                      >> x=Ap\b              >> b=Ap*x
Ap =  2.9003   1.9720               x =  0.0516            ans = -0.3522
      1.4623   2.7826                    1.2303                  -0.4217
      2.2137   2.5242                                             0.7400
```

c) V případě nedourčeného systému (více neznámých než je rovnic) maticové dělení zleva určí takové hodnoty neznámých, aby co nejvíce hodnot bylo nenulových

```
>> An=A(1:2,:)                      >> bn=b(1:2)           >> x=An\bn
An = 2.9003   1.9720   1.9129       bn = 2.2235            x =  0.0230
     1.4623   2.7826   1.0370            3.0772                 1.0938
                                                               0
```

Operace prvek po prvku

Kromě maticových operací je potřeba rozlišovat tzv. operace prvek po prvku, kdy požadovaná operace (násobení, dělení a umocňování) je aplikována mezi odpovídajícím prvky obou operandů. To je označeno tečkou před požadovaným operátorem. Např. operace maticové mocnění na přirozené číslo je definována pouze pro čtvercové matice a jde o opakované maticová násobení. Zatímco mocnění ve smyslu operace prvek po prvku je možné pro libovolný rozměr matice.

```
>> A=magic(4)          >> A.^2                  >> A^2
A = 16   2  3  13      ans =256   4   9  169     ans =345 257 281 273
     5  11 10   8            25 121 100   64          257 313 305 281
     9   7  6  12            81  49  36  144          281 305 313 257
```

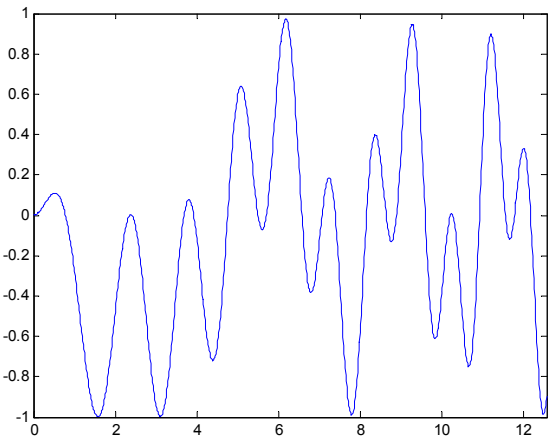
```
     4   14  15    1              16 196 225     1              273 281 257   34
```

Vizualizace – 2D a 3D grafy

a) Nakreslete průběh funkce

$$y(x)=\sin\left(\frac{x^2}{\sqrt{x+1}}\right)\cos(2x) \quad x\in\langle 0,4\pi\rangle$$

v 1000 bodech rovnoměrně rozložených po zadaném intervalu.



Řešení:

```
>> x=0:4*pi/999:4*pi;
>> y=sin(x.^2./sqrt(x+1)).*cos(2*x);
>> plot(x,y)
>> axis([0,4*pi,-1,1])
```

b) Nakreslete plochu odpovídající funkci $z(x,y)=\left(x^2+y^2\right)\cos(xy) \quad x\in\langle -\pi,\pi\rangle \quad y\in\langle -\pi,\pi\rangle$ při rovnoměrném dělení 50 bodů v každém směru

Řešení: vytvoření vektorů x a y

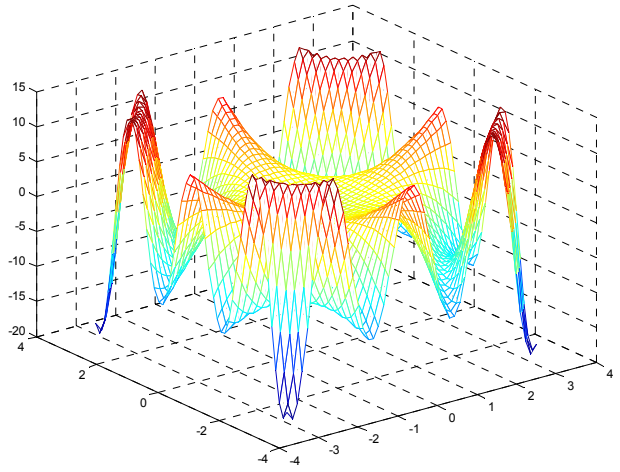
```
>> x=-pi:2*pi/49:pi;
>> y=x;
```

vytvoření pomocných matic X a Y dovolujících použít původní tvar funkce

```
>> [X,Y]=meshgrid(x,y);
>> Z=(X.^2+Y.^2).*cos(X.*Y);
```

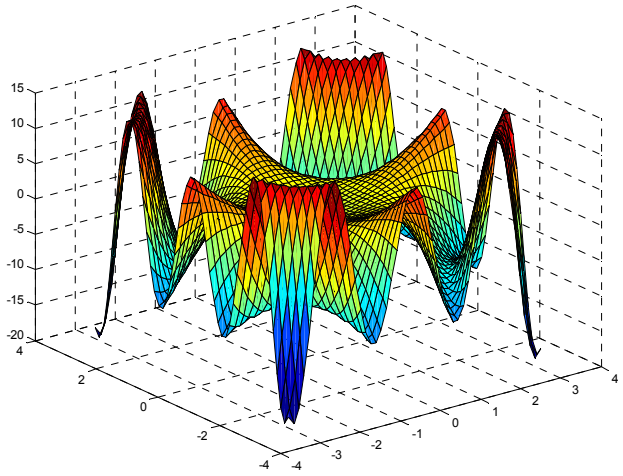
vytvoření drátového modelu

```
>> mesh(x,y,Z)
```



vytvoření modelu s plochami

```
>> surf(x,y,Z)
```



Funkce funkcí

a) Hledání kořene funkce – **`x=FZERO('fce',x0)`**

Nalezněte kořen(y) funkce $f(x)=xe^{-x}-0.1$.

Řešení: pro použití funkce `fzero` je nutné vytvořit uživatelskou m-funkci obsahující definici požadované funkce tj. vytvořit textový soubor např. f1.m obsahující text uvedený v rámečku vpravo. Můžeme využít editor, který je

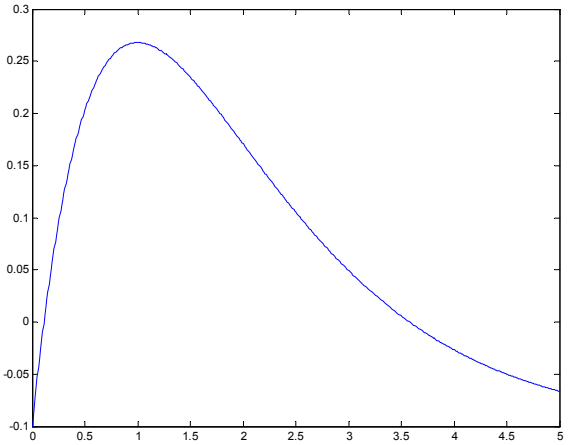
```
function y=f1(x)
y=x.*exp(-x)-0.1;
```

součástí MATLABu

```
>> edit f1.m
```

Pro určení počtu kořenů využijeme vytvořenou funkci jejíž graf na intervalu 0-5 je na obrázku.

```
>> x=0:0.01:5;
>> plot(x,f1(x))
```



Z obrázku odhadneme počet a přibližnou polohu kořenů – dva kořeny cca 0.2 a 3.5

```
>> x1=fzero('f1',0.2)
x1 =    0.1118
>> x2=fzero('f1',3.5)
x2 =    3.5772
```

b) Výpočet určitého integrálu – **S=QUADL('fce',a,b)**

Určete plochu nad osou x ohraničenou průběhem funkce $f(x) = xe^{-x} - 0.1$

Řešení: $S = \int_a^b \left(xe^{-x} - 0.1\right)dx$ Pro použití funkce `quadl` je nutné vytvořit uživatelskou m-funkci obsahující definici integrandu (využijeme m.funkci z předchozího příkladu). Dalším problémem je určení dolní a horní meze integrálu. Opět využijeme řešení z předchozího příkladu. Zápis řešení může být pak na jednom řádku.

```
>> S=quadl('f1',fzero('f1',0.5),fzero('f1',3.5))
S =    0.5197
```

c) Výpočet soustavy diferenciálních rovnic – **[T,Y]=ODE45('fce',t,Y0)**

Nalezněte numerické řešení soustavy diferenciálních rovnic (Lotka-Voltera predator/prey population model) s danými počátečními podmínkami na časovém intervalu od 0 do 15

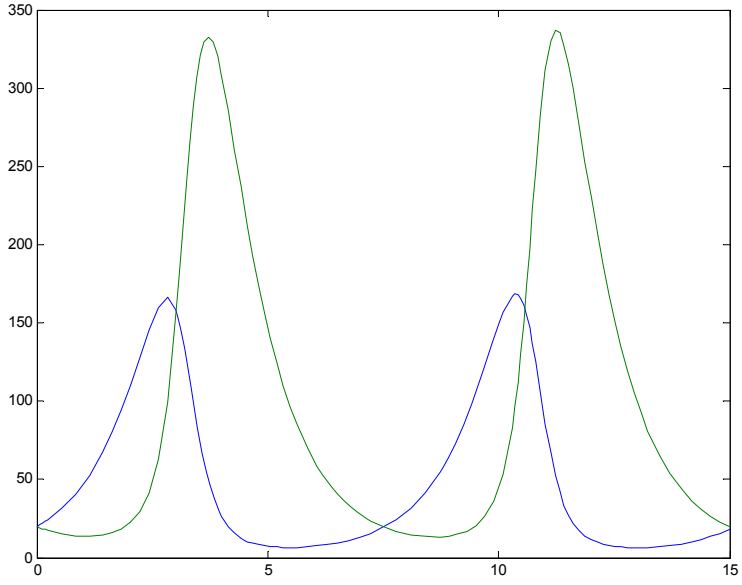
```
function dY=f3(t,Y)
% Lotka-Voltera predator/prey model
y1=Y(1); y2=Y(2);
dY=[(1-0.01*v2)*v1;(-1+0.02*v1)*v2];
```

$$\frac{dy_1}{dt} = \left(+1 - 0.01y_2\right)y_1 \quad y_1(t=0) = 20$$

$$\frac{dy_2}{dt} = \left(-1 + 0.02y_1\right)y_2 \quad y_2(t=0) = 20$$

Řešení: Pro použití funkce ode45 je nutné vytvořit uživatelskou m-funkci obsahující definici soustavy rovnic I. řádu ve vektorové podobě – s formálním tvarem d**Y**/dt=fce(t,**Y**) (viz rámeček)

```
>> t=[0,15];
>> Y0=[20;20];
>> [T,Y]=ode45('f3',t,Y0);
>> plot(T,Y)
>> size(Y)
ans =    97     2
```



d) Hledání minima funkce více proměnných – **X =**

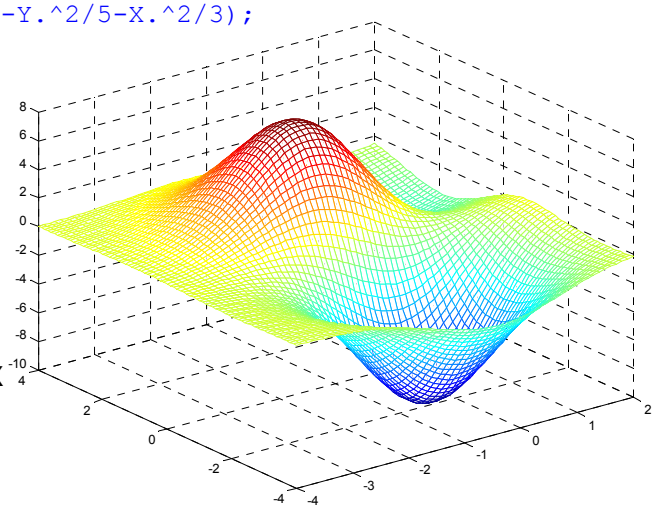
FMINSEARCH('fce',X0)

Nalezněte maximum funkce $z(x,y)=\left(x^2-5xy-2x+6y-2y^2\right)e^{-\frac{x^2}{3}-\frac{y^2}{5}}$

Řešení: Pro použití funkce `fminsearch` je nutné vytvořit uživatelskou m-funkci obsahující definici funkce jejíž minimum hledáme. Protože hledáme maximum otočíme znaménko. Je vhodné ověřit zda existuje pouze jedno maximum.

```
function z=f3(P)
x=P(1); y=P(2);
z=(x^2-5*x*y-2*x+6*y-2*y^2)*exp(-y^2/5-x^2/3);
z=-z;
```

```
>> x=-4:0.1:2; y=-4:0.1:4;
>> [X,Y]=meshgrid(x,y);
>> Z=(X.^2-5*X.*Y-2*X+6*Y-2*Y.^2).*exp(-Y.^2/5-X.^2/3);
>> mesh(x,y,Z)
>> p=fminsearch('f4',[1,1])
p =   -1.0790    1.1859
>> f4(p)
ans =   -7.1803
```



2.6 Příklady - Symbolic Math Toolbox

Vytvoření symbolických proměnných

```
>> x=sym('x'); y=sym('y'); z=sym('z');
>> syms a b c
```

Manipulace s výrazy

Zobrazení a zjednodušení výrazu $\frac{(a+b)^2}{a^2-b^2}\left(\frac{a+b}{ab-b^2}\right)^{-1}$

```
>> v=sym('(a+b)^2/(a^2-b^2)*((a+b)/(a*b-b^2))^(-1)')
v = (a+b)^2/(a^2-b^2)*((a+b)/(a*b-b^2))^(-1)
>> pretty(v)
                                    2
                          (a + b)  (b a - b )
                          ------------------
                                2     2
                               a  - b
>> simplify(v)
ans = b
```

Lineární algebra

Determinant, inverze, vlastní čísla a charakteristický polynom

```
>> M=sym('[a b;x y]');
>> det(M)
ans = a*y-b*x
 >> inv(M)
ans = [ -y/(-a*y+b*x),  b/(-a*y+b*x)]
      [  x/(-a*y+b*x), -a/(-a*y+b*x)]
>> eig(M)
ans = [ 1/2*a+1/2*y+1/2*(a^2-2*a*y+y^2+4*b*x)^(1/2)]
      [ 1/2*a+1/2*y-1/2*(a^2-2*a*y+y^2+4*b*x)^(1/2)]
>> poly(M)
 ans = t^2-t*y-a*t+a*y-b*x
```

Řešení algebraických rovnic

Polynom $x^3+7x^2-x/5+11=0$

```
>> v=solve('x^3+7*x^2-x/5+11=0');
>> pretty(v)
          [        1/3          1         ]
          [-1/15 %1   - 248/3 ----- - 7/3]
          [                     1/3      ]
          [                   %1         ]

          [       1/3           1                   1/2 /         1/3           1  \]
          [1/30 %1    + 124/3 ----- - 7/3 + 1/2 i 3    |- 1/15 %1    + 248/3 -----|]
          [                     1/3                    |                       1/3|]
          [                   %1                       \                     %1   /]

          [       1/3           1                   1/2 /         1/3           1  \]
          [1/30 %1    + 124/3 ----- - 7/3 - 1/2 i 3    |- 1/15 %1    + 248/3 -----|]
          [                     1/3                    |                       1/3|]
          [                   %1                       \                     %1   /]

                                                    1/2
                              %1 := 62225 + 15 8734785
>> double(v)
ans =  -7.2376
        0.1188 - 1.2271i
        0.1188 + 1.2271i
```

Soustava algebraických rovnic $\begin{aligned}(1+ax)y&=c\\(1+by)x&=c\end{aligned}$

```
>> v=solve('(1+a*x)*y=c','(1+b*y)*x=c')
v =  x: [2x1 sym]
     y: [2x1 sym]
>> v.x
>> simplify(v.x)
 ans =
[ -(-1-b*c-c*a+(1+2*b*c+2*c*a+b^2*c^2-2*b*c^2*a+c^2*a^2)^(1/2))/(-1+b*c-
    - c*a+(1+2*b*c+2*c*a+b^2*c^2-2*b*c^2*a+c^2*a^2)^(1/2))/a]
[ -(1+b*c+c*a+(1+2*b*c+2*c*a+b^2*c^2-2*b*c^2*a+c^2*a^2)^(1/2))/(1-
    - b*c+c*a+(1+2*b*c+2*c*a+b^2*c^2-2*b*c^2*a+c^2*a^2)^(1/2))/a]
>> simplify(v.y)
 ans =[ 1/2/b*(-1+b*c-c*a+(1+2*b*c+2*c*a+b^2*c^2-2*b*c^2*a+c^2*a^2)^(1/2))]
      [ -1/2*(1-b*c+c*a+(1+2*b*c+2*c*a+b^2*c^2-2*b*c^2*a+c^2*a^2)^(1/2))/b]
```

Symbolická integrace a derivace

Neurčitý integrál $\int \frac{x^2}{\sqrt{x^2+a^2}}dx$ a derivace výsledku

```
>> vp=sym('x*x/sqrt(x^2+a^2)');
>> v=int(vp)
v = 1/2*x*(x^2+a^2)^(1/2)-1/2*a^2*log(x+(x^2+a^2)^(1/2))
>> pretty(v)
                      2     2  1/2          2           2     2  1/2
               1/2 x (x  + a )    - 1/2 a log(x + (x  + a )    )
>> vd=diff(v)
vd =  1/2*(x^2+a^2)^(1/2)+1/2*x^2/(x^2+a^2)^(1/2)-
    - 1/2*a^2*(1+1/(x^2+a^2)^(1/2)*x)/(x+(x^2+a^2)^(1/2))
>> pretty(simplify(vd))
                                  2
                                 x
                            ------------
                               2     2 1/2
                              (x  + a )
```

Řešení diferenciálních rovnic

Soustava diferenciálních rovnic

$$\frac{dy}{dt} = a\,x\,y \qquad y(t=0) = c$$

$$\frac{dx}{dt} = a(x+y) \qquad x(t=0) = c$$

```
>> v=dsolve('Dy=a*(y*x)','Dx=b*(x+y)','y(0)=c','x(0)=c')
v = x: [1x1 sym]
    y: [1x1 sym]
>> pretty(v.x)
                        1
                  -----------
                  /     1 \
                  |t + ---| a
                  \     c /
>> pretty(v.y)
                 2    /     1 \
                c  a  |t + ---|
                      \     c /
```

2.7 Komplexní příklad

Mějme průnik dvou kružnic o poloměru *r* (čárkovaná) a *R* (plná) dle obrázku 2-2. Úkolem je určit *r* takové, průnik byl přesně polovinou obsahu kružnice poloměru *R*. Tedy

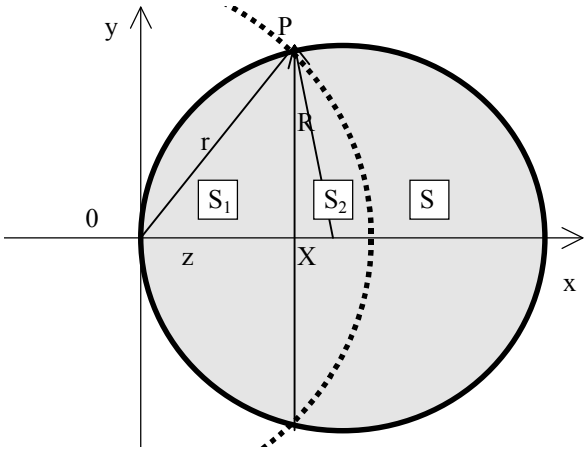
$$S_1 + S_2 = S$$

kde $S = \pi R^2 / 4$

$$S_1 = \int_0^z \sqrt{R^2 - (x-R)^2}\,dx$$

$$S_2 = \int_z^r \sqrt{r^2 - x^2}\,dx$$

$$z = \frac{r^2}{2R}$$



obrázek 2-2 Průnik dvou kružnic

Substitucemi $t = \frac{x}{R}$ $u = \frac{x}{r}$ $\mathbf{p} = \frac{r}{R}$ převedeme výpočetní vztahy na výslednou rovnici s jednou proměnnou *p*

$$\int_0^{\frac{\mathbf{p}^2}{2}} \sqrt{1-(1-t)^2}\,dt + \mathbf{p}^2 \int_{\frac{\mathbf{p}}{2}}^1 \sqrt{1-u^2}\,du - \pi/4 = fce(p) = 0$$

Vytvořme tři uživatelské funkce (viz rámeček). První dvě představují integrandy obou integrálů a jsou nutné pro vyhodnocení určitých integrálů funkcí `quadl`. Řešením je pak kořen výše uvedené rovnice (funkce *fce_p*). Hledaná hodnota vystupuje jako horní a dolní mez integrálů.

```
>> p=fzero('fce_p',1)
p =    1.1587
```

```
function y=fce_int1(x)
y=sqrt(1-(1-x).^2);

-----------------------
function y=fce_int2(x)
y=sqrt(1-x.^2);

-----------------------
function y=fce_p(p)
S1=quadl('fce_int1',0,p^2/2);
S2=quadl('fce_int2',p/2,1);
y=S1+p^2*S2-pi/4;
```

3. Začínáme

Příkazy a funkce: `cd, clear, delete, dir, format, help, lookfor, type, who, whos`

Problém: zápis příkazů, editace příkazového řádku, předdefinovaná jména, proměnné a informace o nich, výpis proměnných, pojem funkce, práce se soubory, nápověda

Tato kapitola by měla dát základní informace o používání MATLABu, o principech práce, o možnostech prostředí a další informace nesouvisející s vlastními výpočty. Budeme se zabývat pouze zjednodušeným přístupem, vynecháme mnoho možností, které MATLAB poskytuje. Proto informace zde uvedené (i když jsou předváděny na verzi 7.0.4) by měly být použitelné pro všechny verze.

Na začátku bych rád zdůraznil aspoň dvě skutečnosti, které je vhodné si uvědomit (a hlavně zapamatovat). MATLAB používá poměrně hutný způsob zápisu příkazů. To je na jednu stranu příjemné, neboť toho nemusíme moc napsat, ale na druhou stranu zápis není příliš přehledný. A hlavně, každý znak má své místo a význam – nemůžu libovolně zaměňovat tečku, čárku či středník nebo jiné znaky. Je potřeba dodržovat tzv. syntaxi jazyka - tedy věci, které by měly být jasné uživatelům zvyklým používat nějaký programovací jazyk, které ale nemusí být samozřejmé pro ostatní.

Druhým, zdánlivě samozřejmým faktem, je ***přečíst si pořádně informace***, které se vypíší při výskytu chyby. Souhlasím plně s názorem váženého čtenáře, který si nyní určitě řekl "*on si snad o mě myslí, že jsem úplně blbej*". Vzpomeňte si na toto upozornění až po dlouhém hledání chyby vlastními silami si konečně ze zoufalství přečtete, co Vám to vlastně oznamuje. Na druhou stranu, když si jednoduše přečtu kde je chyba, nezažiji ten slastný pocit vítězství jako v případě, že na chybu přijdu sám.

3.1 Desktop – první pohled

Od verze 6 došlo na první pohled ke změně. Místo jednoho – příkazového okna (*Command Window*) – se po spuštění MATLABu v základním nastavení objeví okna hned tři, jak je ukázáno na obr. 3-1. Aby tento úvod byl univerzální (platný i pro starší verze) budeme se budeme dále zabývat pouze příkazovým oknem neboť vše lze jeho pomocí udělat. Ostatní okna (ve skutečnosti jsou čtyři) pouze elegantněji zpřístupňují některé funkce a činnosti. K dalším možnostem desktopu se vrátíme na konci této kapitoly. Pokud uživatelé starších verzí preferují původní vzhled, nemusí zoufat. Příkazem *View->Desktop Layout->Command Window Only* se zobrazí pouze příkazové okno.

3.1.1 Jak to funguje

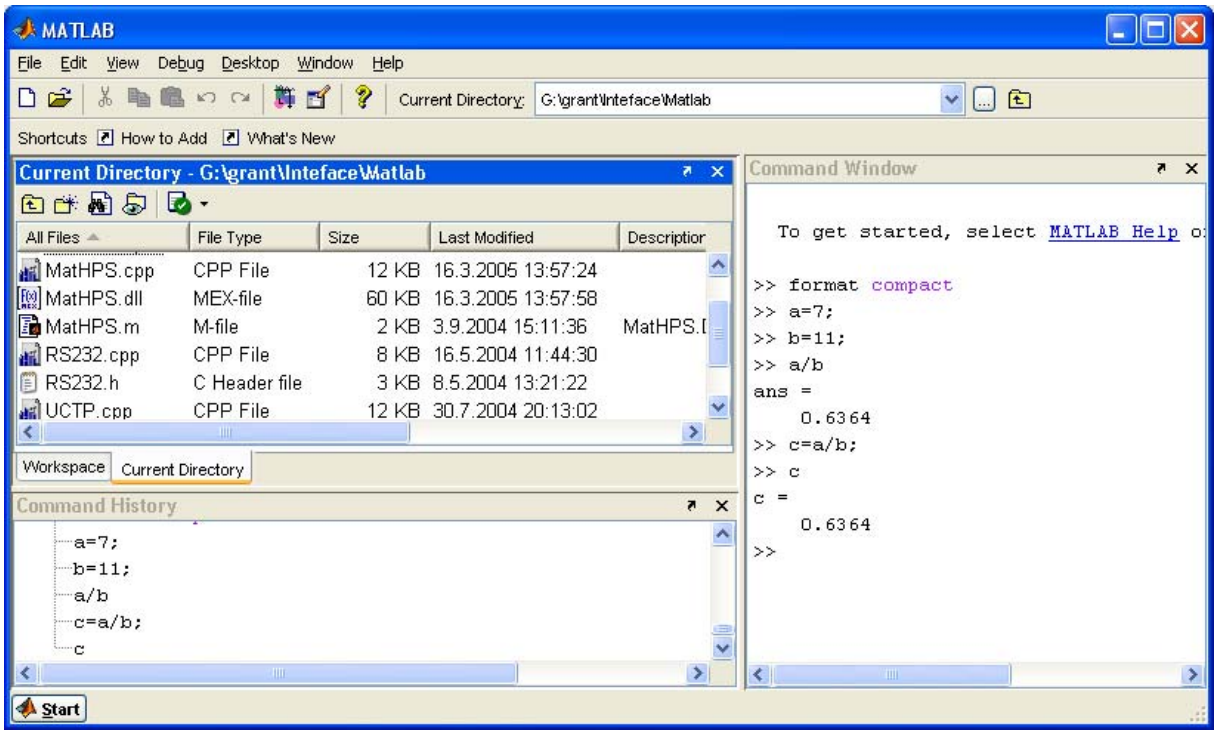
V nejjednodušším pohledu se můžeme na MATLAB dívat jako na velkou kalkulačku nebo na něco podobného interpretu jazyka BASIC. Příkazy se píší do příkazové řádky. Každý zapsaný příkaz se po stisku klávesy Enter okamžitě provede. V případě, že v příkazu neuvedeme název proměnné, do které se má uložit výsledek, uloží se do proměnné s názvem `ans`. Pokud příkaz ukončíme znakem středník (;) pak se příkaz pouze provede, ale výsledek se nevypíše[30]. Neuvedeme-li na konci příkazu znak středník, pouze příkaz ukončíme klávesou Enter, výsledek se i vypíše. Příkazové okno na obrázku 3-1 ukazuje na ilustračním příkladě naplnění proměnných *a* a *b* hodnotami a naplnění proměnné *c* podílem *a/b*. Napíšeme-li pouze název proměnné (bez středníku) vypíše se obsah proměnné. Implicitně je nastaven výpis na 4 desetinná místa. Vnitřně jsou čísla uložena s přesností 15 platných cifer[31].

---

[30] Tato zdánlivá maličkost může značně znepříjemnit práci. Vzhledem k tomu, že výsledek příkazu může být třeba matice 100x100, pak příkaz bez středníku představuje výpis 10 000 čísel na obrazovku. To znamená, kromě toho že tato informace nemá pro normálního člověka naprosto žádný smysl, také to že to dost dlouho trvá.

[31] 8 byte double precision floating point (standard IEEE) v rozsahu $x \in \{-\text{Inf}, <-1.7977\times10^{308}, -2.2251\times10^{-308}>, 0, <+2.2251\times10^{-308}, +1.7977\times10^{308}>, +\text{Inf}\}$ (základ z=2, mantisa k=52 bit, exponent p=11 bit), $z^{-p-1} \le |x| \le \left(1-z^{-k}\right)z^{p}$

Očekává-li MATLAB příkaz (dokončil-li předchozí příkaz), indikuje tento stav na začátku řádky úvodními znaky >> (prompt).



obrázek 3-1 Základní uspořádání okna MATLABu – desktop

Další užitečné všeobecně platné informace:

a) chceme-li přerušit provádění příkazu stiskneme kombinaci kláves Ctrl+C.
b) klávesy šipka nahoru/dolu umožňují pohyb v historii dříve napsaných příkazů. Vyhledání příkazu v historii zapsaných příkazů usnadní zapsání několika prvních znaků (masky) hledaného příkazu a poté vyhledání pomocí šipka nahoru. Vypisují se pouze příkazy splňující zapsanou masku
c) příkaz na řádku lze editovat tj. pohybovat se klávesami šipka vpravo/vlevo po napsaném textu a klávesami Delete či BackSpace mazat napsaný text a standardním způsobem zapisovat opravy
d) klávesa Esc vymaže celý řádek

3.1.2 Proměnné a funkce

Proměnné v MATLABu musí mít název začínající písmenem a název může být až 63[32] znaků dlouhý. Pozor ***rozlišují se velká a malá písmena***. Můžeme použít libovolný název, ale určitá jména mají dopředu přiřazenou hodnotu. Použijeme-li toto jméno, změníme původní hodnotu. Některá důležitá předdefinovaná jména jsou v následujícím seznamu.

ans	proměnná používaná systémem pro uložení výsledku, když neuvedeme vlastní název proměnné
pi	Ludolfovo číslo (3.1415926535897...)
i, j	imaginární jednotka
Inf	nekonečno (např. výsledek výrazu 1/0)
NaN	neplatná numerická hodnota (např. výsledek výrazu 0/0)

Proměnné jsou zjednodušeně pouze dvojího typu - komplexní matice[33] nebo řetězec znaků. Proměnné není třeba deklarovat - vytváří se automaticky v potřebné velikosti při prvním použití. Po vytvoření proměnná

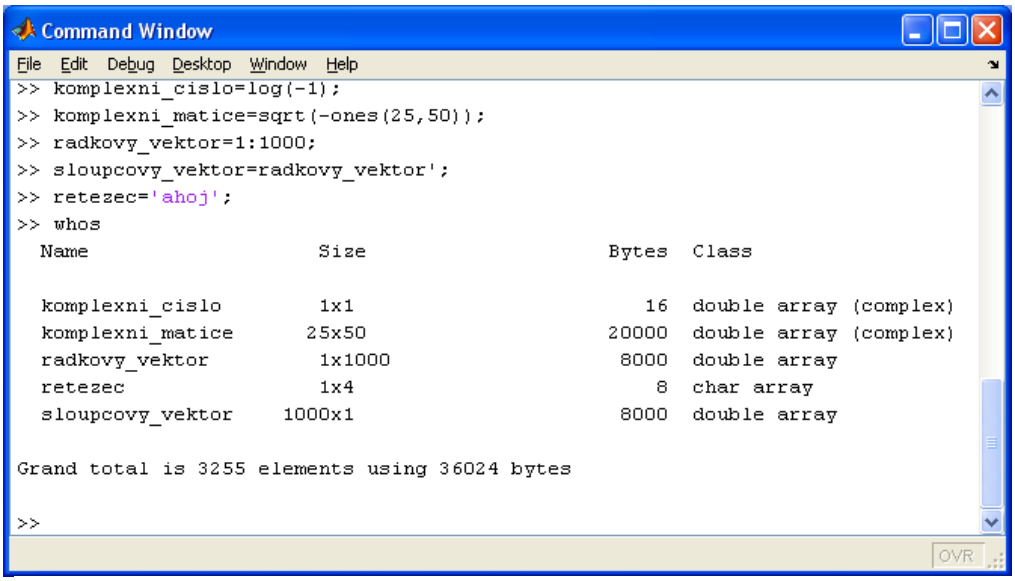
[32] aktuální maximální délka se zjistí funkcí `namelengthmax` (od verze 7)

[33] ostatní běžné datové typy jako reálné číslo, vektor jsou vlastně speciálním případem komplexní matice

existuje do ukončení programu nebo do zrušení příkazem **`clear`** *`jméno_proměnné_1 jméno_proměnné_2 ...`*. Příkaz `clear` použitý bez názvů proměnných vymaže všechny proměnné.

Během práce můžeme vytvořit velké množství proměnných navíc s různou velikostí. Příkaz **`who`** vypíše seznam existujících jmen a příkaz **`whos`** opět seznam jmen, ale navíc s velikostí proměnných. Použití příkazu `whos` je na obrázku 3-2.

Hodnoty, které proměnné zastupují, jsou v paměti počítače uloženy v binárním formátu pohyblivé řádové čárky (každé číslo 8 byte). V případě, že je vypisujeme na obrazovku musí být konvertovány z tohoto formátu do řetězce znaků (číslic). Parametry této konverze můžeme nastavit příkazem `format`.



obrázek 3-2 Výpis použitých proměnných

Hlavní síla MATLABu je ve funkcích. Funkce je obvykle složitější činnost, která jeden nebo více vstupních parametrů (proměnných nebo konstant) zpracuje do jednoho nebo více výstupních parametrů (proměnných) podle určitého předpisu (algoritmu). Funkce mohou být jednoduché (např. funkce sinus) nebo složité (např. vyhledání minima funkce více proměnných). Funkce mohou být přímo součástí jádra MATLABu (**built-in function**) nebo ve formě předpisu (kombinujícího matematické operace, built-in funkce či jiné m-funkce), který je uložen v samostatném souboru na disku s příponou .m. Takovéto funkce budeme nazývat **m-funkce**. Součástí standardní instalace MATLABu je velké množství m-funkcí organizovaných v adresářích sdružujících m-funkce podle jejich zaměření. Seznam adresářů, kde jsou m-funkce uloženy, se vytváří při instalaci.

M-funkce jsou standardní ASCII znakové soubory a proto si uživatel může nejen vytvořit vlastní m-funkce ale i prohlédnout m-funkce jiných autorů. Je dobrým zvykem (a vřele doporučuji ho dodržovat) svoje uživatelské m-funkce ukládat do samostatného adresáře. Nejjednodušší je nastavit tento adresář jako aktuální (nebo též pracovní) adresář. V případě že aktuální adresář je jiný, je nutné adresář s uživatelskými funkcemi zařadit do systému MATLABovských adresářů. Blíže viz kapitolu 7.3.

Pro manipulaci se soubory a adresáři jsou do MATLABu zahrnuty příkazy podobné jako jsou v OS DOS. Příkaz **`cd`** použitý bez parametrů vypíše cestu do aktuálního adresáře, s parametry umožňuje změnit aktuální adresář (včetně disku), příkaz **`dir`** vypíše obsah adresáře, příkaz **`type`** vypíše obsah určeného souboru, příkaz **`delete`** vymaže určený soubor.

Z hlediska používání není mezi built-in funkcemi a m-funkcemi rozdíl. Vždy je potřeba vědět jak se jmenuje funkce, která dělá to, co potřebuji, kolik má vstupních parametrů a jaký mají jednotlivé parametry význam a potom samozřejmě v jaké formě funkce vrací výsledek (kolik výstupních parametrů a jaký mají jednotlivé parametry význam). Obecně vypadá volání funkce (syntaxe příkazu) následovně:

```
[prom1,prom2,...] = nazev_funkce(par1,par2,...);
```

Nyní zbývá jen maličkost - jak tyto informace zjistit.

3.1.3 Nápověda - HELP

Začínáme-li s MATLABem (a nejenom tehdy) je patrně nejpoužívanějším příkazem[34] příkaz **`help`**. Tento příkaz nám slouží ke zjištění, jaké funkce máme k dispozici (včetně instalovaných toolboxů) a pokud je známo jméno funkce, pak i k určení syntaxe[35] dané funkce.

Tedy postupně - příkaz **`help`** sám o sobě vypíše seznam skupin funkcí, které jsou v daném okamžiku k dispozici např.

```
>> help
HELP topics

matlab\general       -  General purpose commands.
matlab\ops           -  Operators and special characters.
matlab\lang          -  Programming language constructs.
matlab\elmat         -  Elementary matrices and matrix manipulation.
matlab\elfun         -  Elementary math functions.
matlab\specfun       -  Specialized math functions.
matlab\matfun        -  Matrix functions - numerical linear algebra.
matlab\datafun       -  Data analysis and Fourier transforms.
matlab\polyfun       -  Interpolation and polynomials.
matlab\funfun        -  Function functions and ODE solvers.
matlab\sparfun       -  Sparse matrices.
matlab\scribe        -  Annotation and Plot Editing.
matlab\graph2d       -  Two dimensional graphs.
matlab\graph3d       -  Three dimensional graphs.
matlab\specgraph     -  Specialized graphs.
matlab\graphics      -  Handle Graphics.
matlab\uitools       -  Graphical user interface tools.
matlab\strfun        -  Character strings.
matlab\imagesci      -  Image and scientific data input/output.
matlab\iofun         -  File input and output.
matlab\audiovideo    -  Audio and Video support.
matlab\timefun       -  Time and dates.
matlab\datatypes     -  Data types and structures.
matlab\codetools     -  Commands for creating and debugging code.
matlab\helptools     -  Help commands.
matlab\winfun        -  Windows Operating System Interface Files (COM/DDE)
simulink\simulink    -  Simulink
simulink\blocks      -  Simulink block library.
simulink\components  -  Simulink components.
```

Zajímají-li nás například speciální matematické funkce, použijeme příkaz

```
>> help specfun
  Specialized math functions.

  Specialized math functions.
    airy        - Airy functions.
    besselj     - Bessel function of the first kind.
    bessely     - Bessel function of the second kind.
    besselh     - Bessel functions of the third kind (Hankel function).
    besseli     - Modified Bessel function of the first kind.
    besselk     - Modified Bessel function of the second kind.
    beta        - Beta function.
    betainc     - Incomplete beta function.
    betaln      - Logarithm of beta function.
```

[34] v tomto textu je pojem příkaz používán v obdobném významu jako funkce - v obou případech je to pokyn pro MATLAB, aby něco udělal. Pojem funkce budeme používat pro pokyn jehož základním smyslem je vypočítat nějakou hodnotu (např. sin), pojem příkaz bude používán pro pokyn, která primárně nevrací číslo, ale vykonává nějakou činnost (např. příkaz cd, help, plot)

[35] formální způsob zápisu, který musí volání dané funkce splňovat

```
    ellipj      - Jacobi elliptic functions.
    ellipke     - Complete elliptic integral.
    erf         - Error function.
    erfc        - Complementary error function.
    erfcx       - Scaled complementary error function.
    erfinv      - Inverse error function.
    expint      - Exponential integral function.
    gamma       - Gamma function.
    gammainc    - Incomplete gamma function.
    gammaln     - Logarithm of gamma function.
    psi         - Psi (polygamma) function.
    legendre    - Associated Legendre function.
    cross       - Vector cross product.
    dot         - Vector dot product.

  Number theoretic functions.
    factor      - Prime factors.
    isprime     - True for prime numbers.
    primes      - Generate list of prime numbers.
    gcd         - Greatest common divisor.
    lcm         - Least common multiple.
    rat         - Rational approximation.
    rats        - Rational output.
    perms       - All possible permutations.
    nchoosek    - All combinations of N elements taken K at a time.
    factorial   - Factorial function.

  Coordinate transforms.
    cart2sph    - Transform Cartesian to spherical coordinates.
    cart2pol    - Transform Cartesian to polar coordinates.
    pol2cart    - Transform polar to Cartesian coordinates.
    sph2cart    - Transform spherical to Cartesian coordinates.
    hsv2rgb     - Convert hue-saturation-value colors to red-green-blue.
    rgb2hsv     - Convert red-green-blue colors to hue-saturation-value.
```

Máme-li např. provést transformaci mezi kartézskými a sférickými souřadnicemi, potřebujeme znát syntaxi funkce `cart2sph`. Napíšeme tedy

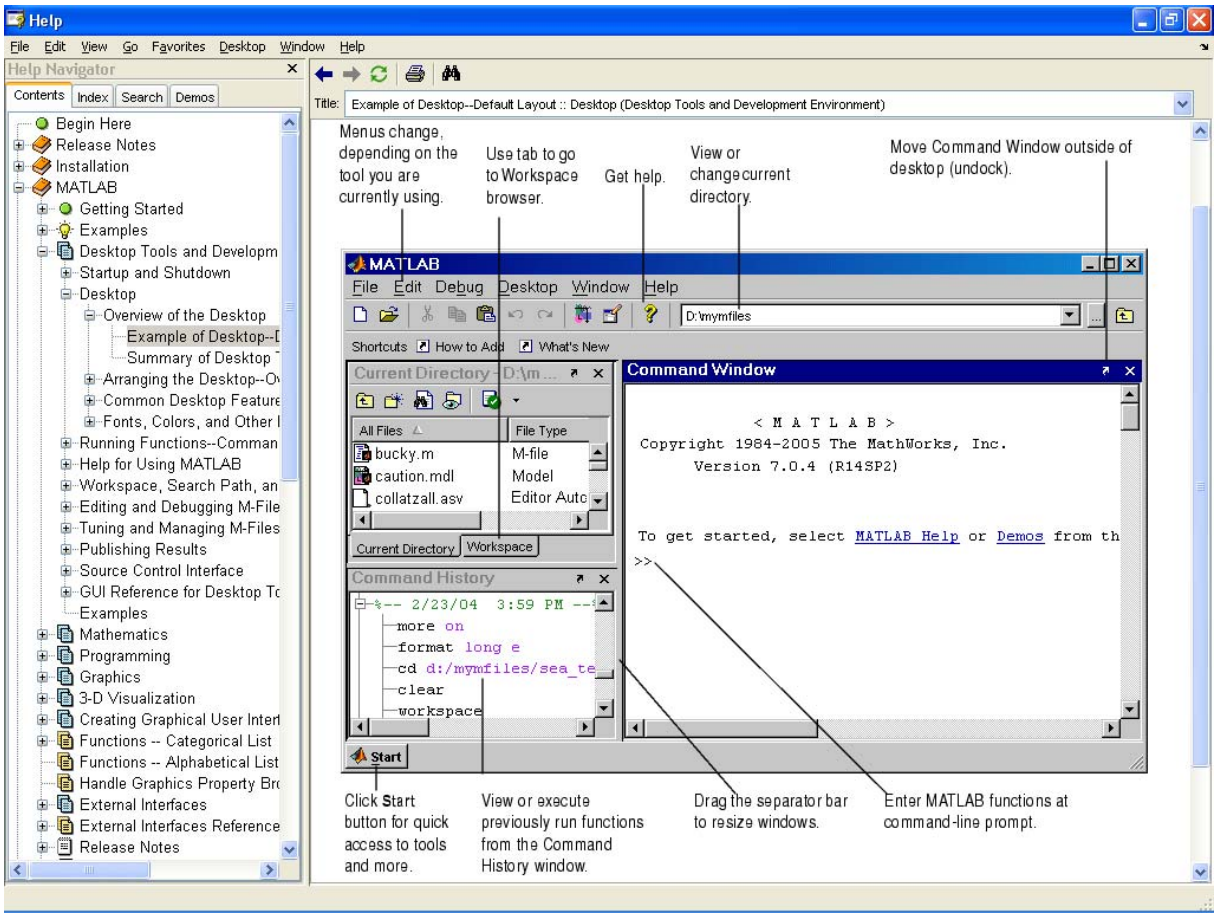
```
>> help cart2sph
 CART2SPH Transform Cartesian to spherical coordinates.
    [TH,PHI,R] = CART2SPH(X,Y,Z) transforms corresponding elements of
    data stored in Cartesian coordinates X,Y,Z to spherical
    coordinates (azimuth TH, elevation PHI, and radius R).  The arrays
    X,Y, and Z must be the same size (or any of them can be scalar).
    TH and PHI are returned in radians.

    TH is the counterclockwise angle in the xy plane measured from the
    positive x axis.  PHI is the elevation angle from the xy plane.

    Class support for inputs X,Y,Z:
       float: double, single

    See also cart2pol, sph2cart, pol2cart.

    Reference page in Help browser
       doc cart2sph
```

Užitečný je také příkaz **`lookfor`** za který když uvedeme slovo, pak jsou vyhledány všechny výskyty tohoto slova v nápovědách všech funkcí. Např. když chceme vědět, kde se vyskytuje slovo gamma, napíšeme

```
>> lookfor gamma
GAMMA  Gamma function.
GAMMAINC Incomplete gamma function.
```

```
GAMMALN Logarithm of gamma function.
PSI  Psi (polygamma) function.
CMGAMDEF Default gamma correction table.
CMGAMMA Gamma correct colormap.
DHINFOPT Discrete H-Infinity control synthesis via Gamma iteration.
HINFOPT H-Infinity control synthesis via Gamma iteration.
GAMCDF Gamma cumulative distribution function.
GAMFIT Parameter estimates for gamma distributed data.
GAMINV Inverse of the gamma cumulative distribution function (cdf).
GAMLIKE Negative log-likelihood for the gamma distribution.
GAMPDF Gamma probability density function.
GAMRND Random arrays from gamma distribution.
GAMSTAT Mean and variance for the gamma distribution.
RANDG Gamma random numbers (unit scale).
HINFOPT H-Infinity control synthesis via Gamma iteration.
```

Od verze 5 pod Windows je součástí systém souborů s nápovědou v HTML formátu využívající hypertextové odkazy (s pomocí Internet Exploreru). Od verze 6 je součástí MATLABu je součástí vlastní prohlížeč systému nápovědy. K dispozici je kompletní elektronická dokumentace. Od verze 7 není instalace nápovědy volitelná – instaluje se vždy ke všem instalovaným součástem. Pro představu je na obrázku 3-3 ukázka nápovědy k základnímu oknu – k desktopu.

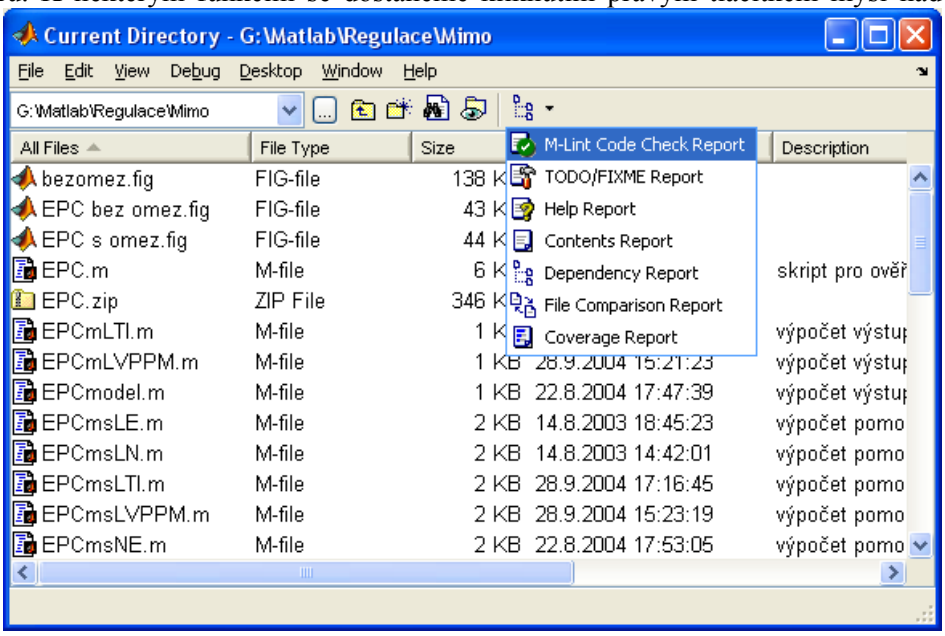


obrázek 3-3 Nápověda k používání desktopu

3.2 Desktop – druhý pohled

Desktop neboli základní pracovní plocha MATLABu ve verzi 7 může, kromě již zmiňovaného příkazového okna (***Command Window***), obsahovat ještě další tři okna. Základní představu o využití jednotlivých oken lze získat z obrázku 3-3. Okno ***Command History*** ukazuje historii zapsaných příkazů a dovoluje dvojklikem na příkazu v seznamu tento příkaz znovu vykonat. Okno ***Workspace*** (pracovní prostor proměnných) ukazuje

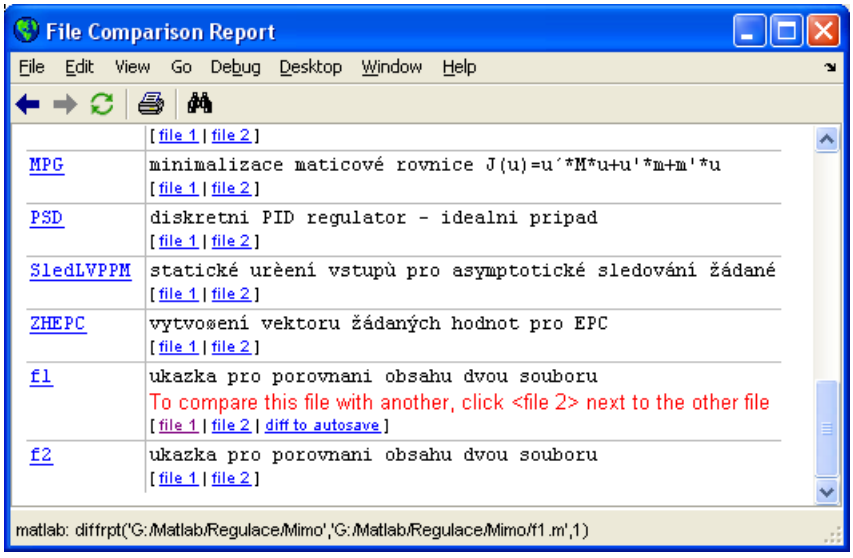
existující proměnné (podobně jako příkaz `whos`) a navíc dovoluje i jejich editaci[36] (dvojklik na jménu proměnné) v editoru připomínajícím tabulkový procesor. Okno ***Current Directory*** zobrazuje obsah aktuálního adresáře a kromě základních operací se složkami (přesun do nadřízené složky či podřízené složky, vytvoření nové složky, výmaz existující složky) a se soubory (nový soubor, výmaz souboru, přejmenování souboru) dovoluje i prohlížení (*Open As Text*) a načtení dat (*Import Data*) do proměnné v pracovním prostoru (*Workspace*) MATLABu. Posledně jmenované činnosti jsou samozřejmě dostupné pouze pro podporované formáty souborů. K některým funkcím se dostaneme kliknutím pravým tlačítkem myši nad vybraným objektem, některé funkce jsou přístupné z tlačítkové lišty nad příslušným oknem.



obrázek 3-4 Uvolněné okno Current Directory

Jednotlivá okna lze z celkového rámce uvolnit a osamostatnit, případně vrátit do původního rámce. Tyto akce se provedou pomocí tlačítka *Undock* (zahnutá šipka v pravém horním rohu).

Tlačítková lišta skrývá i nové funkce. Na obrázku 3-4 je uvolněné okno *Current Directory* s rozvinutým seznamem funkcí tlačítka, které je nejvíc napravo *Directory Reports*. Příkazy z tohoto seznamu se vykonávají nad soubory MATLABu v aktuálním adresáři. Např. příkaz *M-lint Code Check Report* provede rozšířenou kontrolu skriptů a m-funkcí MATLABu a vytvoří výslednou zprávu. Mimo běžnou syntaktickou kontrolu se zjišťuje např. přítomnost vytvořených, ale nepoužívaných proměnných atd.

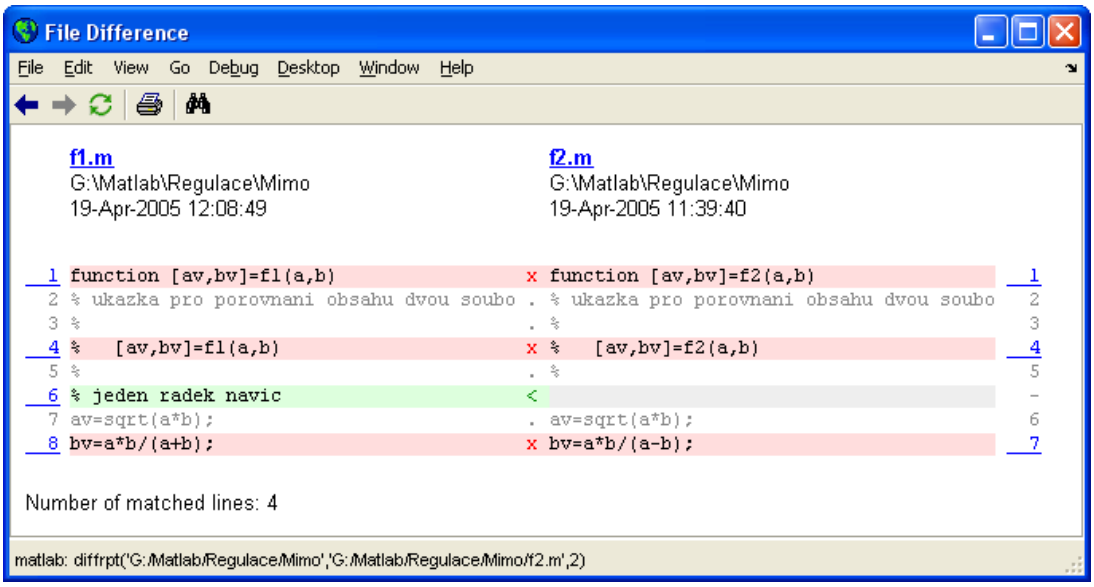


obrázek 3-5 Funkce File Comparission Report - výběr souborů

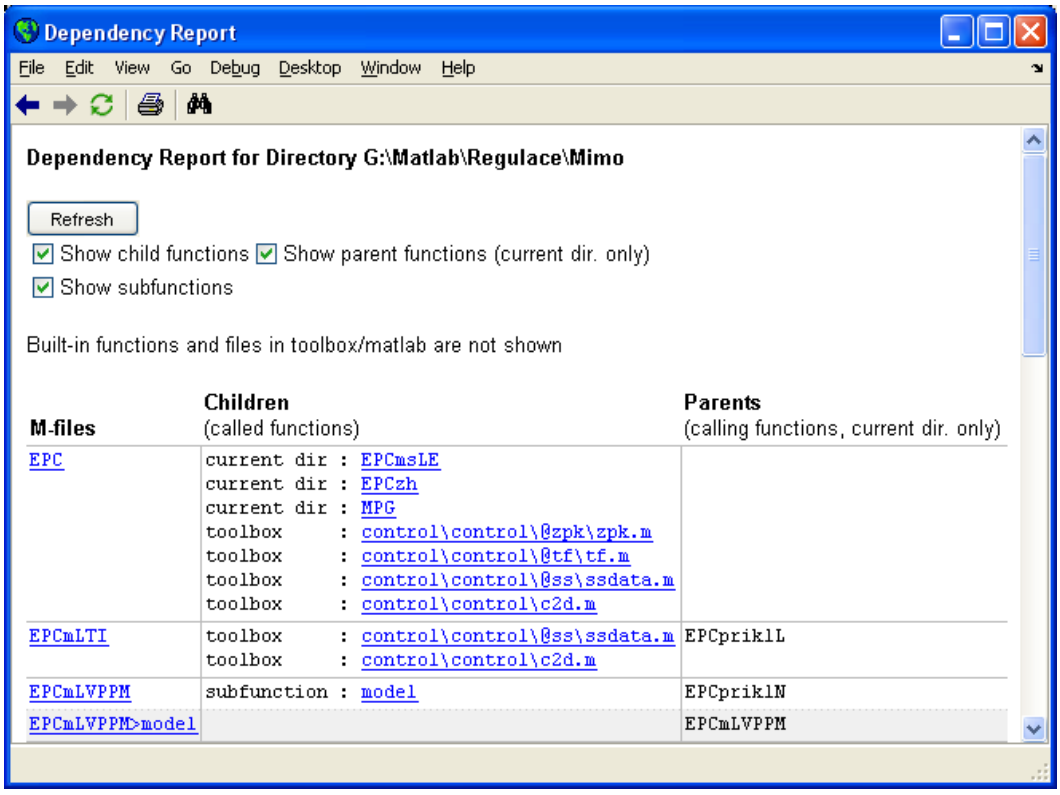
Užitečný může být příkaz *File Comparission Report*. Tento příkaz dovolí vybrat dva soubory (obrázek 3-5) ze seznamu v aktuálním adresáři a porovnat jejich obsah (zpráva na obr. 3-6).

Další užitečnou funkcí je funkce *Dependenci Report* tj. kontrola vzájemné závislosti volání funkcí. Na obrázku 3-7 je část výsledné zprávy, ze které lze vyčíst, nejen které další funkce definované v souborech aktuálního adresáře (případně z toolboxů) jsou v souborech volány, ale i ze kterého soubor jsou funkce volány.

[36] velmi užitečná možnost i pro prosté prohlížení – zvláště u strukturovaných proměnných. V případě objektů je spuštěn *Property Inspektor*.



obrázek 3-6 Funkce File Difference Report - výsledek porovnání obsahu dvou souborů



obrázek 3-7 Zpráva funkce Dependency Report

4. Základní operace s maticemi

Příkazy a funkce: `conv, det, diag, exp, eye, fopen, fprintf, fscanf, interp1, inv, length, load, log, magic, ones, polyfit, polyval, rand, randn, roots, save, sin, size, sqrt, strcat, strfind, zeros`

Problém: plnění vektorů a matic, aritmetická řada čísel, komplexní čísla, transpozice, inverze, maticové násobení a dělení, operace prvek po prvku, speciální znaky, načtení a uložení dat do souboru, operace s polynomy, práce se soubory a znakovými řetězci

Kapitola čtvrtá se zabývá základními operacemi s vektory a maticemi - jejich vytvořením a naplněním hodnotami. Ukážeme si jak vytvořit některé speciální tvary matic, jak pracovat se sloupcem či řádkem matice. Dále si ukážeme rozdíl mezi maticovými operacemi a operacemi aplikovanými mezi jednotlivými prvky matic. Také si ukážeme využití vektorového zápisu pro práci s polynomy.

Na základě zkušeností s výukou doporučuji této kapitole (práci s maticemi) věnovat dostatečnou pozornost, neboť nedovolené operace (např. násobení matic s nesouhlasnými rozměry) s maticemi jsou nejčastější chybou vyskytující se při práci s MATLABem.

Kompletní seznam základních příkazů, matematických a jiných funkcí nabízených MATLABem nejen pro práci a manipulaci s maticemi je uveden v kapitole 4.5.

4.1 Vytváření vektorů a matic

Skalár je matice s oběma rozměry rovnými jedné. Vektor je matice, kdy jeden rozměr má hodnotu jedna. Rozlišujeme vektory sloupcové a řádkové. Je zvykem, že první rozměr označuje počet řádků a druhý počet sloupců. Řekneme-li, že matice je rozměru 7×12, znamená to, že má 7 řádků a 12 sloupců. Pokud se hovoří o vektoru bez upřesnění (sloupcový či řádkový) předpokládá se sloupcový vektor. Toto platí jak v matematických zápisech, tak i v MATLABu. Pro vyjádření opačného vektoru se používá symbol transpozice[37]. V matematice používané označení transpozice (velké T jako pravý horní index tj. ^{T}) je v zápisu MATLABu nahrazeno znakem ' (apostrof). Běžný matematický zápis

$$a^T=[1,2,3,4,5]\quad b=[1,2,3,4,5]^T\quad c=[1,2,3,4,5]$$

pak vyjadřuje, že a ,b jsou sloupcové vektory a c je řádkový vektor. Všechny vektory o pěti prvcích s hodnotami 1,2,3,4,5.

Na matici můžeme pohlížet i jako na vektor vektorů. Potom nepřekvapí zápis matice též pomocí již definovaných vektorů či matic

$$A=\begin{bmatrix}1,2,3,4,5\\1,2,3,4,5\end{bmatrix}=\begin{bmatrix}c\\1,2,3,4,5\end{bmatrix}=\begin{bmatrix}a^T\\b^T\end{bmatrix}=\begin{bmatrix}c\\c\end{bmatrix}$$

4.1.1 Plnění vektorů a matic

Zápis vektorů a, b, c a matice A v MATLABu pak vypadá následovně

```
>> a=[1;2;3]                % sloupcový vektor (oddělovač středník) s výpisem (na konci příkazu není středník)
a =  1
     2
     3
>> c=[1,2,3]                % řádkový vektor (oddělovač čárka nebo mezera) s výpisem
c =  1     2     3
>> b=[1,2,3]';              % sloupcový vektor (oddělovač čárka, transpozice) bez výpisu
```

[37] transpozici si nejsnáze představíme tak, že matici (vektor) doplníme na čtvercovou matici, tato se otočí kolem hlavní diagonály a odebere se přidaná část. To znamená, že z matice 2x5 se stane matice 5x2 a z řádkového vektoru se stane sloupcový vektor

Zápis matice s uvedením všech prvků je možný např.

```
>> A=[1,2,3;1,2,3]        % dva řádkové vektory mezi sebou oddělené středníkem s výpisem
A =  1     2     3
     1     2     3
```

Využijeme-li již existujícího vektoru *c* můžeme vytvořit matici *A* jednodušeji

```
>> A=[c;c]
      A =  1     2     3
           1     2     3
```

Následující příklad ukazuje nepřípustnou kombinaci (kombinuje do matice sloupcový a řádkový vektor - MATLAB ohlásí chybu)

```
>> C=[a,c]
??? Error using ==> horzcat
All matrices on a row in the bracketed expression must have the same number of
rows
```

Potřebujeme-li pracovat s komplexními čísly, použijeme zápis ekvivalentní algebraickému zápisu komplexních čísel. Využijeme předdefinovanou konstantu *i* nebo *j* - imaginární jednotku. Např. vektor s kombinací reálných a komplexních čísel získáme

```
>> x=[1.23,7-3.6*i,1/7+3/11*i,pi*i]
x = 1.2300      7.0000 - 3.6000i      0.1429 + 0.2727i        0 + 3.1416i
```

Všimněte si, že transpozice komplexní proměnné znamená současně převod na komplexně sdružená čísla

```
>> y=x'
y = 1.2300
    7.0000 + 3.6000i
    0.1429 - 0.2727i
         0 - 3.1416i
```

Často potřebujeme vytvořit rozsáhlý vektor tvořený aritmetickou řadou např. od 0.75 s krokem 0.05 do 10. Takovýto vektor vytvoříme následujícím příkazem

```
>> v=0.75:0.05:10;        % počáteční hodnota : krok : konečná hodnota
```

Nyní nás zajímá jaký se vlastně vektor vytvořil tj. kolik má prvků a jakého rozměru. Na zjištění těchto vlastností použijeme funkci **`size`**. Tato funkce vrací dvě hodnoty - počet řádků a počet sloupců. V následujícím příkladu použijeme tuto funkci, necháme uložit hodnoty do proměnných a zároveň vypsat na obrazovku

```
>> [poc_radku,poc_sloupcu]=size(v)
poc_radku =   1
poc_sloupcu = 186
```

neuvedeme-li výstupní parametry funkce `size` a samozřejmě ani ukončení středníkem, vypíšou se pouze hodnoty na obrazovku respektive uloží se do speciální proměnné `ans` a vypíšou

```
>> size(v)
ans =   1   186
```

Vektor v má 186 prvků a je řádkový. Pro zjištění počtu prvků (délky) vektoru slouží funkce **`length`**, která vrátí větší z čísel získaných funkcí `size`.

Vektory vzniklé plněním tímto způsobem (aritmetická řada) jsou řádkové. Chceme-li vytvořit sloupcový vektor, použijeme transpozici. Chceme-li vytvořit tímto způsobem matici použijeme např. tento příkaz. Všimněte si, že v prvním řádku je vynechán krok. V takovém případě MATLAB použije hodnotu 1

```
>> B=[1:5;5:-1:1]
B =  1     2     3     4     5
     5     4     3     2     1
```

Shrňme nyní nejpoužívanější speciální znaky a jejich význam. Zdůrazňuji, že není jedno zda napíšeme čárku či středník, kulatou či hranatou závorku - každý znak má svůj přesný význam.

;	*středník*	ukončení příkazu bez výpisu, oddělovač sloupců
,	*čárka*	obecný oddělovač, oddělovač v řádku
	mezera	obecný oddělovač, oddělovač v řádku
:	*dvojtečka*	oddělovač ve výčtu (určení rozsahu)
'	*apostrof*	označení transpozice (umístěn za proměnnou)
()	*kulaté závorky*	přednost v mat. výrazu, seznam parametrů funkce, index matice
[]	*hranaté závorky*	určují, že jde o vektor či matici
{}	složené závorky	určují, že jde o strukturu

4.1.2 Vektory a matice se speciálními hodnotami

Často jsou potřeba matice naplněné speciálními hodnotami. Jedním z nejčastěji se vyskytujících požadavků je vytvoření matice (vektoru) zadaných rozměrů naplněných konstantou. MATLAB poskytuje dvě funkce pro tyto účely. Funkce **`zeros`** vytvoří a naplní matici hodnotou nula a funkce **`ones`** hodnotou jedna. Následující seznam uvádí příklady použití funkcí vytvářejících některé často se vyskytující speciální matice

x=zeros(11,50)	matice o rozměrech 11×50 naplněná nulami
x=zeros(100)	nulová čtvercová matice 100×100
x=ones(1,1000)	řádkový vektor o 1000 prvcích plný hodnot 1
x=eye(100)	jednotková matice o rozměru 100×100
x=eye(25,50)	matice 25×50 s jednotkami na hlavní diagonále
x=rand(5,7)	matice 5×7 náhodná čísla v intervalu (0,1) - rovnoměrné rozložení
x=randn(1,9)	matice 1×9 náhodná čísla s normálním rozložením N(0,1)

Informace o dalších speciálních typech matic viz `help elmat`. Jako příklad uveďme tzv. magický čtverec[38]

```
>> x=magic(6)              % čtvercová matice 6×6 - magický čtverec
x = 35     1     6    26    19    24
     3    32     7    21    23    25
    31     9     2    22    27    20
     8    28    33    17    10    15
    30     5    34    12    14    16
     4    36    29    13    18    11
```

4.1.3 Práce s částí matice či vektoru

V příkladech využijeme vektory a matice vytvořené v kap.4.1.1. Práce s pouze jednou hodnotou vektoru či matice je standardní.

```
>> a(2)                    % zjištění hodnoty druhého prvku vektoru a
ans =   2
>> c(5)=x(2,3)             % hodnotu z druhého řádku a třetího sloupce matice x dej do pátého prvku vektoru c
c =     1     2     3     0     7
```

Pokud pozorně čtete[39], měl by vám být předchozí příklad podezřelý. Ano, vektor *c* byl definován pouze o třech prvcích. Přesto není problém uložit hodnotu do jeho pátého prvku – chybějící prvky se naplní nulovou hodnotou.

V MATLABu lze také pracovat s částí vektoru či submaticí matice. V tomto případě se používá pro určení rozsahu indexů znak dvojtečka. Např. část matice *x* (submatici) lze získat příkazem

```
>> x(2:5,1:2)
```

[38] čtvercová matice kde součet každého řádku, sloupce i obou diagonál dává stejné číslo

[39] upřímně řečeno, pokud si toho někdo všiml, má můj obdiv

```
ans = 3    32
     31     9
      8    28
     30     5
```

Často je potřebné vyjmout celý sloupec či řádek. To je přirozeně možné i předchozím způsobem - pomocí funkce `size` zjistíme rozměry a tyto použijeme pro určení rozsahu. Existuje ale elegantnější způsob. Použití samotné dvojtečky na místě příslušného indexu získáme celý rozsah.

```
>> y=x(3,:)                % vyjmi třetí řádek, ulož do proměnné y a vypiš
y = 31     9     2    22    27    20
```

Dalším užitečným trikem je použití klíčového slova **`end`**[40] - v případě, že potřebujeme rozsah indexu od určité hodnoty do konce - např.

```
>> r=3; s=4; y=x(r:end,s:end)
y = 22    27    20
    17    10    15
    12    14    16
    13    18    11
```

V tomto příkladu jsme navíc použili tři příkazy na jednom příkazovém řádku což je samozřejmě možné. Pozor jen, že musí být odděleny oddělovačem (obvykle středník nebo čárka).

V případě, že potřebujeme získat hodnoty na hlavní diagonále matice je užitečná funkce **`diag`**

```
>> diag(y)
ans = 22
      10
      16
```

4.1.4 Zápis dat do souboru a čtení ze souboru

V MATLABu existují v podstatě dvě varianty jednoduchého uložení dat do souboru a odpovídajícího čtení ze souboru. První způsob je nejčastěji používaný v případě, že máme něco v MATLABu rozpracováno tj. máme naplněná data v proměnných a musíme ukončit práci. Při prostém ukončení programu nás příště čeká pracné plnění dat znova. Pro tento případ je určený příkaz **`save.`** Použitý bez dalších parametrů uloží všechny použité proměnné (tzv. pracovní prostor) do souboru *matlab.mat* v aktuálním adresáři. Obnovit proměnné z tohoto souboru lze příkazem **`load`**. Použití těchto příkazů s jedním parametrem *jmeno* (jméno souboru bez přípony) uloží/načte proměnné do/z souboru *jmeno.mat*.

```
>> save stav050502          % uložení všech proměnných (včetně názvů) do souboru stav050502.mat
```

Pozor na to, že formát souboru je binární, speciální pro MATLAB a ukládají se i názvy proměnných. Uvedeme-li za jméno souboru ještě názvy proměnných, uloží se pouze vyjmenované proměnné.

Druhá varianta umožňuje výměnu dat s jinými programy. Ukládá data v textovém (čistém ASCII) formátu. Požadavek na uložení v tomto formátu je rozpoznán na základě uvedení klíčového slova `-ASCII` v příkazu `save`. Následující příklad uloží matici *x* v textovém formátu do souboru *pokus.txt* v aktuálním adresáři:

```
>> save pokus.txt x -ascii  % uložení hodnot z proměnné x do souboru pokus.txt
```

Příkaz `load` načte tento soubor a uloží do proměnné stejného jména[41] jako je jméno souboru (bez přípony)

```
>> load pokus.txt           % načtení dat ze souboru pokus.txt do proměnné pokus
```

Data v textovém souboru musí být ve stejně dlouhých sloupcích. Čísla v řádcích by měla být oddělena nejlépe některým ze standardních oddělovačů (mezera, čárka, středník, tabelátor), oddělovač desetinné části je tečka. Viz např. obsah souboru *pokus.txt* (ukázka v rámečku na následující straně), kde je jako oddělovač čísel použita mezera a čísla jsou v tzv. semilogaritmickém tvaru.

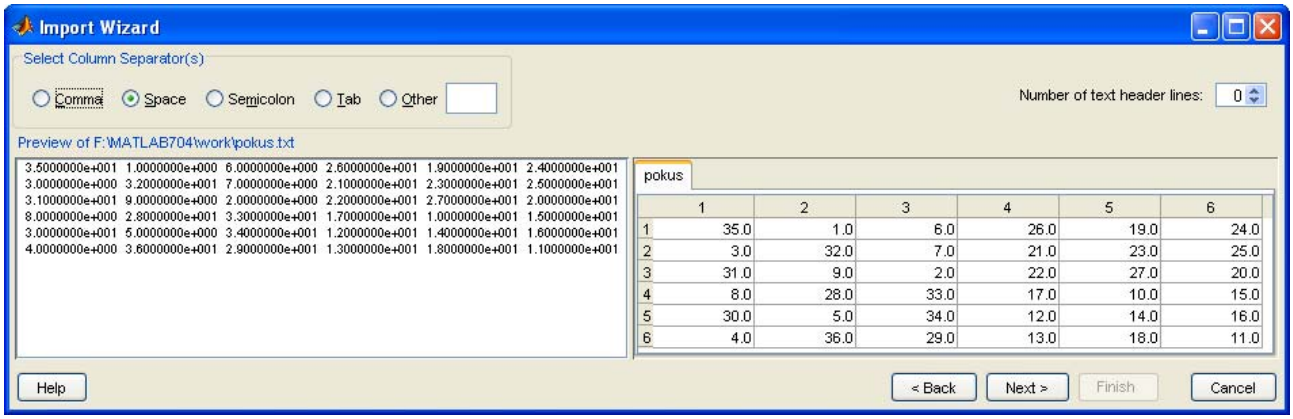
---

[40] lze používat až od verze 5

[41] nelze pomocí příkazu load načíst ASCII data do proměnné jiného jména než je jméno souboru, ze kterého se data čtou

Pro načtení dat v různých formátech souborů můžeme z okna *Current Directory* desktopu také použít ***Import Wizard*** – průvodce načtením dat ze souboru do proměnné (klik pravým tlačítkem myši nad zvoleným datovým souborem a výběr *Import Data*). Viz také obrázek 4-1, kde je zobrazen první krok průvodce pro načtení souboru *pokus.txt*. *Import Wizard* je možné spustit také přímo příkazem **`uiimport`**.

3.5000000e+001	1.0000000e+000	6.0000000e+000	2.6000000e+001	1.9000000e+001	2.4000000e+001
3.0000000e+000	3.2000000e+001	7.0000000e+000	2.1000000e+001	2.3000000e+001	2.5000000e+001
3.1000000e+001	9.0000000e+000	2.0000000e+000	2.2000000e+001	2.7000000e+001	2.0000000e+001
8.0000000e+000	2.8000000e+001	3.3000000e+001	1.7000000e+001	1.0000000e+001	1.5000000e+001
3.0000000e+001	5.0000000e+000	3.4000000e+001	1.2000000e+001	1.4000000e+001	1.6000000e+001
4.0000000e+000	3.6000000e+001	2.9000000e+001	1.3000000e+001	1.8000000e+001	1.1000000e+001



obrázek 4-1 Import Wizard - textový soubor

Kromě textových formátů je možná práce ještě se zvukovými soubory formátu .WAV (MS Windows) a .AU (NexT/SUN) pomocí příkazů **`wavread`** a **`wavwrite`** (**`auread`**, **`auwrite`**). Je podporováno více kanálů a osmi i šestnácti bitové vzorkování (od verze 5.2). Dále je možné pracovat s některými obrazovými formáty (.BMP, .GIF, .HDF, .JPG, .PCX, .PNG, .TIF, .XWD), video soubory (.AVI) a soubory tabulkových procesorů (.WK1,.XLS). Jsou také možné formátované i binární operace s datovými soubory. Od verze 6 jsou také přímo podporovány I/O operace nad sériovou linkou (kapitola 11.5). Seznam příkazů pro import a export dat je v kapitole 4.5.1 (File I/O)

4.2 Základní operace a funkce

V dalším výkladu se předpokládá, že čtenář je seznámen se základními pravidly práce s maticemi a maticovými operacemi.

4.2.1 Sčítání, maticové násobení, inverze, operace prvek po prvku

Nyní ukážeme používání matematických operací pro práci s maticemi v MATLABu. Vytvoříme si znovu několik proměnných, na kterých předvedeme základní operace. Pokud dále v této kapitole nejsou v příkladech definovány použité proměnné, jsou myšleny tyto vektory a matice. Vektory budeme v této kapitole označovat malými písmeny a matice velkými.

```
>> a=[1:3];                % řádkový vektor 1×3
>> b=[3:-1:1]';            % sloupcový vektor 3×1
>> A=[a,b';b',a];          % matice 2×6
>> B=A';                   % matice 6×2
```

chybné zadání matice

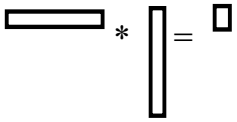
```
>> C=[1 2 3 4              % první řádek matice (4 sloupce)
5 6 7 8                    % druhý řádek matice (4 sloupce)
9 10 11]                   % třetí řádek matice (3 sloupce) chyba
??? Error using ==> vertcat
All rows in the bracketed expression must have the same number of columns.
```

Následující příklad ukazuje, že násobení matic není asociativní a že sčítat se v MATLABu dají pouze odpovídající matice. Prostudujte si pozorně všechny varianty násobení a sčítání dvou vektorů.

chyba

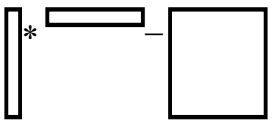
```
>> a*a
??? Error using
==> *
Inner matrix
dimensions must
agree.
```

```
>> c=a+a
c =  2     4     6
```



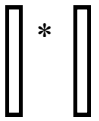
```
>> c=a*a'
c =    14
```

```
>> a+a'
??? Error using
==> +
Matrix
dimensions must
agree.
```



```
>> C=a'*a
C =  1     2     3
     2     4     6
     3     6     9
```

```
>> a'+a
??? Error using ==> +
Matrix dimensions
must agree.
```



chyba

```
>> a'*a'
??? Error using
==> *
Inner    matrix
dimensions must
agree.
```

```
>> c=a'+a'
c =  2
     4
     6
```

Obdobně jako při práci se skaláry odpovídá dělení v maticovém počtu operaci násobení převrácenou hodnotou. Tato operace - získání převrácené hodnoty - se nazývá inverzí a lze ji provést pouze se čtvercovou maticí. Stejně jako není definováno dělení nulou, neexistuje inverze k tzv. singulárním maticím. Singulární matice mají determinant rovný nule. Příkladem takové matice je matice *C* (třetí varianta násobení dvou vektorů z předchozího příkladu). Výpočet determinantu se v MATLABu provede funkcí **`det`** a výpočet inverzní matice funkcí **`inv`**. Inverze matice a determinant jsou v MATLABu definovány pro čtvercové matice.

```
>> 1/0                      % dělení skalární nulou
Warning: Divide by zero.
ans = Inf
>> det(C)                   % výpočet determinantu matice C
ans = 0
>> inv(C)                   % výpočet inverzní matice k matici C
Warning: Matrix is singular to working precision.
ans = Inf   Inf   Inf
      Inf   Inf   Inf
      Inf   Inf   Inf
```

Vytvořme nyní matici $D = A * A^T$

```
>> D=A*A'                   % násobení matice a transponované matice (D je symetrická matice)
D = 28    20
    20    28
```

a pro zajímavost spočítejme inverzní matici k matici *D*

```
>> inv(D)                   % výpočet inverzní matice
ans = 0.0729   -0.0521
     -0.0521    0.0729
```

Pro skaláry platí pro $x \neq 0$ rovnost $x*x^{-1}=1$. Ukažme si obdobný vztah pro matici D

```
>> D*inv(D)                 % násobení matice a matice k ní inverzní dá jednotkovou
ans = 1     0
      0     1
```

Operace násobení, jak byla do tohoto místa používána, je klasické maticové násobení. Kromě maticového násobení je v MATLABu definováno násobení člen po členu. Rozdíl mezi oběma případy je ukázán na definici obou násobení

$$A[r,m] \quad B[m,s] \quad C[r,s] \qquad C = A * B \quad \Leftrightarrow \quad C(i,j) = \sum_{k=1}^{m} A(i,k) * B(k,j)$$

$$D[r,s] \quad E[r,s] \quad F[r,s] \qquad F = D.* E \quad \Leftrightarrow \quad F(i,j) = D(i,j) * E(i,j)$$

```
>> C=D*D                          % maticové násobení
C =     1184        1120
        1120        1184
>> F=D.*D                         % násobení prvek po prvku
F =     784         400
        400         784
```

Některé operace na maticích mají smysl pouze prvek po prvku např.

```
>> A.^0.75
ans = 1.0000    1.6818    2.2795    2.2795    1.6818    1.0000
      2.2795    1.6818    1.0000    1.0000    1.6818    2.2795
```

Zastavme se u operací prvek po prvku. Výhodou maticového zápisu MATLABu není jen možnost přirozené práce s maticemi, ale i možnost jednou operací zpracovat velké množství dat. Následující příkaz spočítá všechny hodnoty výrazu odpovídající příslušným hodnotám *x* a uloží do proměnné *y*

```
>> x=1:5;
>> y=1./(1+x)
y = 0.5000    0.3333    0.2500    0.2000    0.1667
```

Nyní podrobněji k operátorům / (maticové dělení zprava) a \ (maticové dělení zleva). Maticové dělení zprava je v podstatě ekvivalentní násobení inverzní maticí - *A/B=A*inv(B)* - s tím rozdílem, že je prováděno numericky jiným způsobem (efektivněji[42] než při použití inverze). Maticové dělení zleva je poněkud složitější. V případě dělení čtvercových matic stejné velikosti jde o obdobný případ jako u maticového dělené zprava tj. *A\B=inv(A)*B*. V případě, že *A* je čtvercová matice rozměru *N*×*N* a *b* je sloupcový vektor rozměru *N*, pak výsledkem je sloupcový vektor *x*, který je řešením rovnice *A*x=b* získaným Gaussovou eliminací. Nejsložitější případ nastane, když matice není čtvercová, ale rozměru *N*×*M* a vektor *b* je sloupcový o rozměru *N* (případ *N* rovnic o *M* neznámých). Pokud je *N*>*M* (soustava je přeurčená) je výsledek počítán tak, aby odchylka od pravé strany byla ve smyslu nejmenších čtverců minimální. V případě, že je *N*<*M* (soustava nedourčená) je výsledný vektor počítán tak, aby bylo co nejvíce elementů výsledku nenulových.

```
>> A=[1 2 3;2 3 1;3 2 1];  b=[1; 2; 4];
>> x=A\b; x'                      % systém tří rovnic pro tři neznámé
ans = 1.5833   -0.4167    0.0833
>> A=[1 2 3;2 3 1;3 2 1;1 1 1];  b=[1; 2; 4; 2];
>> x=A\b; x'                      % přeurčený systém (čtyři rovnice pro tři neznámé)
ans = 1.6667   -0.5000    0.1667
>> A=[1 2 3;2 3 1];  b=[1; 2];
>> x=A\b; x'                      % nedourčený systém (dvě rovnice pro tři neznámé)
ans = 0    0.7143   -0.1429
```

4.2.2 Některé základní funkce

MATLAB disponuje většinou základních matematických funkcí. Specialitou je, že tyto funkce jsou aplikovány na všechny prvky proměnné (vektoru či matice). Např. zajímají-li mne hodnoty funkce sinus na intervalu <0,2π> v 1000 bodech, lze toho dosáhnout dvěma příkazy

[42] např. operace *A/A* trvá cca 78% času operace *A*inv(A)*

```
>> x=0:2*pi/999:2*pi; y=sin(x);        % v proměnné y je tisíc hodnot funkce sin(x)
```

Jako argumenty funkce lze použít všechny hodnoty, pro které je daná funkce definovaná

```
>> x=-2:2; y=sqrt(x)                   % odmocnina (i ze záporných čísel)
y =   0 + 1.4142i      0 + 1.0000i      0           1.0000           1.4142
>> log(x)                              % přirozený logaritmus
Warning: Log of zero.
ans = 0.6931 + 3.1416i 0 + 3.1416i      -Inf        0                0.6931
>> A=[1 i;-i 1]; B=exp(A)              % exponenciální funkce
B =   2.7183             0.5403 + 0.8415i
      0.5403 - 0.8415i   2.7183
>> acsch(x)                            % inverzní hyperbolický kosekant
Warning: Divide by zero.
> In G:\MATLAB6P1\toolbox\matlab\elfun\acsch.m at line 8
ans = -0.4812   -0.8814       Inf    0.8814    0.4812
```

Seznam všech elementárních matematických funkcí použitelných v MATLABu získáme příkazem `help elfun` (kap. 4.5.2), seznam speciálních matematických funkcí příkazem `help specfun` (kap. 4.5.2) a seznam maticově orientovaných funkcí příkazem `help matfun` (kap. 4.5.2).

4.3 Práce s polynomy a interpolace

Práce s polynomy je běžná v inženýrských výpočtech. MATLAB tuto oblast též podporuje. Využívá se vektorové reprezentace polynomů. V MATLABu je zavedena určitá konvence uložení polynomu. Koeficient u nejvyšší mocniny polynomu je uložen v prvním prvku vektoru a ostatní koeficienty u klesajících mocnin polynomu jsou umístěny postupně do dalších prvků. Nesmíme zapomenout zapsat i nulové koeficienty. Přepišme dva následující polynomy *p(x)* a *q(x)* do vektorů *p* a *q*

$$p(x) = 4x^5 + 3.1x^3 - 7x^2 + 11 \quad q(x) = -x^4 + x^3 - x$$

```
>> p=[4 0 3.1 -7 0 11]; q=[-1 1 0 -1 0];
```

Pro práci s polynomy je k dispozici několik[43] funkcí, z nichž uvedeme pouze některé:

```
>> x=[-0.1,0,0.1,0.2];
>> y=polyval(p,x)                      % vyčíslení polynomu p pro všechny hodnoty x
y =   10.9269   11.0000   10.9331   10.7461
>> r=conv(p,q)                         % násobení polynomů p a q
r =  -4.0000  4.0000 -3.1000  6.1000 -7.0000 -14.1000 18.0000  0  -11.0000     0
>> koreny=roots(p)                     % kořeny[44] polynomu p
koreny = 0.9727 + 0.5747i
         0.9727 - 0.5747i
        -0.5124 + 1.4413i
        -0.5124 - 1.4413i
        -0.9207
```

Nalezení koeficientů polynomu zvoleného stupně prokládajícího data podle kritéria nejmenších čtverců

```
>> x=[1 2 3 4 5]; y=[5.5 43.1 100.2 190.7 218.4];
>> r=polyfit(x,y,3)                    % nalezení koeficientů polynomu 3. stupně
r =   -6.8583   62.6964 -110.3452   61.5800
```

S proložením bodů pomocí polynomu úzce souvisí interpolace[45]. Respektive proložení polynomem je jeden typ interpolace. MATLAB poskytuje několik standardních funkcí pro různé typy interpolačních funkcí. Popis všech funkcí a jejich použití je přesahuje rozsah tohoto textu.

---

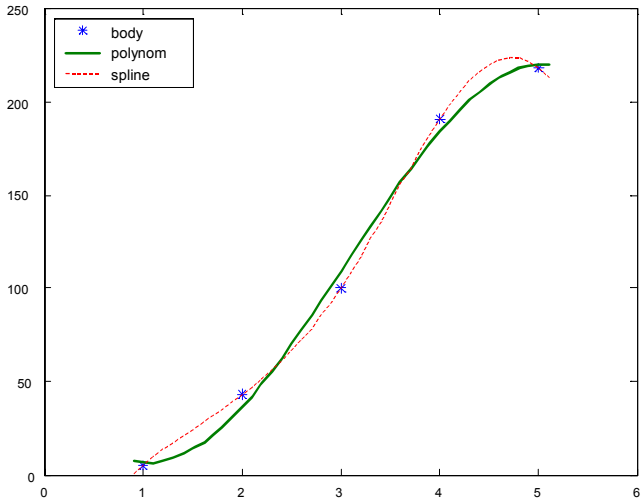
[43] seznam funkcí získáme příkazem `help polyfun` (viz též kap. 4.5.3)

[44] funkce roots není všemocná. Přesnost určení hodnot násobných kořenů nemusí být v určitých případech nejlepší.

[45] nalezení hodnot závislosti v předepsaném místě mezi zadanými body

Pro názornou ukázku použití funkce **polyval** předběhneme další výklad a použijeme příkaz `plot` pro grafické zobrazení původních bodů a bodů získaných proložením polynomem 3. stupně a interpolační funkcí - kubickými spline (viz obrázek 4-2).

```
% body x, ve kterých budou počítány aproximace
>> xx=0.9:0.1:5.1;
% body y získané polynomiální interpolací (MNČ)
>> yp=polyval(r,xx);
% body y získané splinovou interpolací
>> yi=interp1(x,y,xx,'spline');
>> plot(x,y,'*',xx,yp,xx,yi,'--')
>> legend('body','polynom','spline')
```



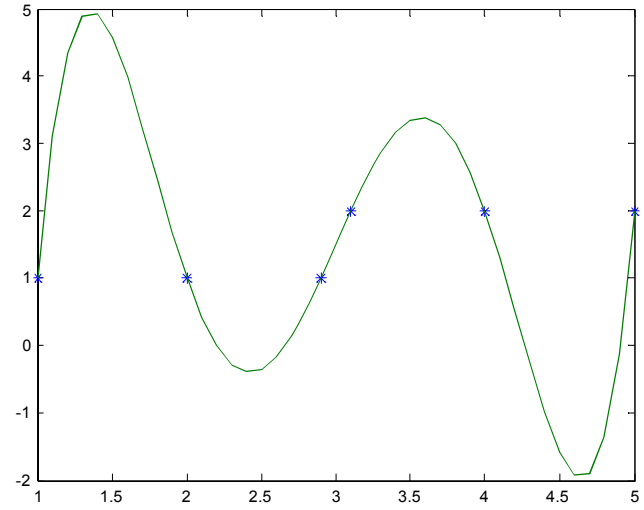
obrázek 4-2 Polynomiální a splinová interpolace

Ukažme si příklad, který ukazuje nebezpečí bezmyšlenkového použití polynomiální aproximace. Mějme následující data (body označené křížky na obrázku 4-3).

```
>> x=[1 2 2.9 3.1 4 5];
>> y=[1 1 1 2 2 2];
```

Pokud je aproximujeme polynomem 5. stupně a dopočítáme hodnoty pro dané x dostaneme ideální shodu. Pokud se však podíváme i na hodnoty mimo sledované body už spokojeni nebudeme. Aproximace se sice přesně shoduje v zadaných bodech, ovšem průběh mezi zadanými body se značně odchyluje od očekávaného průběhu. Polynomiální aproximace není vhodná pro aproximaci průběhů s ostrými zlomy.

```
>> p=polyfit(x,y,5);
>> xp=1:0.1:5; yp=polyval(p,xp);
>> plot(x,y,'*',xp,yp)
```



obrázek 4-3 Nevhodně použitá polynomiální interpolace

4.4 Práce se soubory a znakovými řetězci

Ačkoliv je MATLAB primárně určen na výpočty tj. práci s čísly, při praktickém používání se explicitní[46] práci s textem tj. řetězci znaků nevyhneme. MATLAB práci se znakovými řetězci podporuje v dostatečném rozsahu. Tyto funkce jsou potřeba často v souvislosti se zpracováním různých souborů s daty. Je tedy vhodné se aspoň stručně zmínit i o možnostech rozšířené práce se soubory.

Nejprve tedy k práci se soubory. Pro běžnou práci můžeme vystačit s již dříve zmíněnými příkazy `load` a `save`. Ale chceme-li pracovat se složitěji strukturovanými soubory potřebujeme obecnější funkce. MATLAB používá velice podobné funkce jako jazyk "C". Soubor se kterým budeme pracovat (ať už z něj číst či do něj zapisovat) musíme "otevřít" pomocí funkce **fopen**. Tato funkce vrací pro každý úspěšně otevřený soubor jedinečný identifikátor, který se používá v ostatních funkcích, které s tímto souborem

[46] implicitně s řetězci znaků stejně pracujeme. Napíšeme-li příkaz MATLABu jde o textový řetězec (posloupnost znaků zadaných z klávesnice či přečtených ze souboru), který MATLAB musí "rozluštit" aby poznal co po něm chceme. Číslo – různě dlouhá posloupnost znaků (číslic) – se musí nejprve převést do vnitřního (binárního) zobrazení a teprve potom je možné s nimi provádět matematické operace. Obdobně chceme-li se dozvědět (vidět zobrazenou) hodnotu výsledku musí se nejprve provést konverze z vnitřního zobrazení do textu – řetězce znaků. Podobu této konverze můžeme ovlivnit příkazem `format`

pracují. Jako musíme každý soubor "otevřít" měli[47] bychom po skončení práce se souborem také soubor "zavřít" pomocí funkce **`fclose`**. S daty můžeme pracovat buď textově – při čtení (zápisu) dochází ke konverzi z textového do vnitřního (z vnitřního do textového) zobrazení – nebo binárně tj. ke konverzím nedochází. Který způsob se použije, záleží na použité funkci. Pozor - při otevření souboru specifikujeme zda budeme pracovat v textovém nebo binárním módu, ale tato volba má vliv pouze na interpretaci znaku konce řádku[48] při textově orientovaných funkcích. Prací v binárním módu se zde nebudeme zabývat.

Pro práci v textovém (znakovém) režimu jsou nejčastěji používány funkce[49]

`fscanf`[50]	čtení jednoho či více čísel v textové podobě s konverzí do vnitřního (binárního) zobrazení
`fprintf`	zápis jednoho či více čísel s konverzí z vnitřního zobrazení do textové podoby
`fgets`	načtení řádku textu bez konverze

Více o funkcích pro práci se soubory viz `help iofun` (také výpis v kapitole 4.5.1)

Ukažme použití některých příkazů na příkladě, kdy jsou čísla (s desetinou částí) přečtena ze souboru *zdroj.txt* a uložena do souboru *cil.txt*. V příkladě je použito klíčové slovo **`while`** (které je vysvětleno v kapitole 5).

```
>> z=fopen('zdroj.txt','rt'); % otevři soubor pouze pro čtení v textovém módu
>> c=fopen('cil.txt','wt');   % otevři soubor pro zápis v textovém režimu
>> [x,p]=fscanf(z,'%f',1);    % přečti jedno desetinné číslo – počet v proměnné p
>> while p==1                 % dokud rovno 1, prováděj příkazy do end
fprintf(c,'%7.5f\n',x);       % zapiš číslo (7 znaků celkem, 5 deset. míst) z x do souboru + konec řádku
[x,p]=fscanf(z,'%f',1);       % přečti jedno desetinné číslo – počet v proměnné p (konec p=0)
end
>> fclose(z);                 % uzavři soubor
>> fclose(c);                 % uzavři soubor
```

Pro práci s textovými (znakovými) řetězci je k dispozici značné množství funkcí. Jak je v MATLABu znakový řetězec reprezentován ? Je to řádkový vektor o velikosti prvku 1 znak[51]. Ukažme jaký je rozdíl mezi číslem a řetězcem znaků. Napišme dva příkazy

```
>> cislo=123.456789           % zadání čísla
cislo = 123.45678             % implicitní formát výpisu je na 5 desetinných míst
>> text='123.456789'          % zadání znakového řetězce
text = 123.456789
```

Příkazem `whos` si můžeme vypsat informace o proměnných

```
>> whos
  Name        Size            Bytes  Class
  cislo       1x1                 8  double array
  text        1x10               20  char array
Grand total is 11 elements using 28 bytes
```

Se znakovou proměnnou nelze provádět matematické operace. Se znakovými řetězci lze ale provádět jiná kouzla. Řetězce lze spojovat (**`strcat`**), převádět text na velká (**`upper`**) či malá (**`lower`**) písmena, řetězec číslic lze převádět na číslo (**`str2num`**) a číslo na řetězec (**`num2str`**), lze vyhledat pozici(e) výskytu(ů)

---

[47] úmyslně píši měli a ne musíme. Na rozdíl od funkce `fopen`, kterou musíme použít (neměli bychom identifikátor souboru), když neuzavřeme otevřené soubory, uzavře je operační systém při ukončení celého programu. Problémy mohou nastat při neuzavření souborů otevřených pro zápis – poslední zápisy do souboru se nemusí zrealizovat (dat zůstanou ve vyrovnávacích pamětech)

[48] na platformě Windows se používají pro ukončení řádku dva znaky (**L**ine **F**eed, **C**arriage **R**eturn) na rozdíl od Unixu, kde používá pouze znak LF.

[49] jde o funkce známé z jazyka "C"

[50] pozor – formátové specifikace jsou aplikovány ne na celý vektor, ale postupně na jednotlivá čísla vektoru. To znamená, že při změně délky vektoru je potřeba změnit formátovací řetězec

[51] původně byl jeden znak uložen v 1 byte. Nyní je v MATLABu 1 znak uložen na 2 byte – pravděpodobně buď přímo v kódování UNICODE nebo je pro toto kódování vše připraveno.

jednoho řetězce uvnitř druhého (**strfind**), porovnávat dva řetězce (**strcmp**) atd. Viz také `help strfun` (kapitola 4.5.3). Ukažme si několik příkladů na práci s řetězci.

```
t1='Ahoj'; t2='Nazdar';          % vytvoření dvou řetězců
t=strcat(t1,'-a-',t2)            % spojení tří řetězců
t = Ahoj-a-Nazdar
>> lower(t)                      % převod na malá písmena
ans = ahoj-a-nazdar
>> strfind(t,'a')                % najdi pozice výskytu řetězce 'a' v řetězci t
ans =     6     9    12
```

4.5 Seznam funkcí a příkazů, operátorů a speciálních znaků

Na závěr třetí kapitoly uvedeme seznam všeobecných příkazů, příkazů pro práci s daty, funkcí pro práci s maticemi, vytváření speciálních matic, standardních i speciálních matematických funkcí, funkcí pro práci s polynomy a znakovými řetězci.

4.5.1 Všeobecné příkazy, operátory a práce s daty

Seznam příkazů a funkcí umístěných v adresářích matlab\general, matlab\ops a matlab\iofun

```
>> help general

  General purpose commands.
 MATLAB Version 7.0.4 (R14SP2)
     21-Jan-2005

  General information.
syntax      - Help on command syntax.
demo   - Run demonstrations.
ver    - MATLAB, Simulink and toolbox
version information.
version - MATLAB version information.

  Managing the workspace.
who    - List current variables.
whos   - List current variables, long
form.
clear  - Clear variables and functions
from memory.
pack   - Consolidate workspace memory.
load   - Load workspace variables from
disk.
save   - Save workspace variables to
disk.
saveas - Save Figure or model to
desired output format.
memory - Help for memory limitations.
recycle - Set option to move deleted
files to recycle folder.
quit   - Quit MATLAB session.
exit   - Exit from MATLAB.

  Managing commands and functions.
what   - List MATLAB-specific files in
directory.
type   - List M-file.
open   - Open files by extension.

which - Locate functions and files.
pcode - Create pre-parsed pseudo-code
file (P-file).
mex    - Compile MEX-function.
inmem - List functions in memory.
namelengthmax - Maximum length of
MATLAB function or variable name.

  Managing the search path.
path   - Get/set search path.
addpath - Add directory to search
path.
rmpath      - Remove directory from
search path.
rehash      - Refresh function and
file system caches.
import      - Import Java packages
into the current scope.
finfo - Identify file type against
standard file handlers on path.
genpath - Generate recursive toolbox
path.
savepath - Save the current MATLAB
path in the pathdef.m file.

  Managing the java search path.
javaaddpath - Add directories to the
dynamic java path.
javaclasspath - Get and set java path.
javarmpath - Remove directory from
dynamic java path.

  Controlling the command window.
echo   - Echo commands in M-files.
more   - Control paged output in
command window.
diary - Save text of MATLAB session.
format - Set output format.
beep   - Produce beep sound.
desktop - Start and query the MATLAB
Desktop.
preferences - Bring up MATLAB user
settable preferences dialog.
```

```
  Operating system commands.
cd    - Change current working
directory.
copyfile - Copy file or directory.
movefile - Move file or directory.
delete - Delete file or graphics
object.
pwd   - Show (print) current working
directory.
dir   - List directory.
ls    - List directory.
fileattrib - Set or get attributes of
files and directories.
isdir - True if argument is a
directory.
mkdir - Make new directory.
rmdir - Remove directory.
getenv - Get environment variable.
!     - Execute operating system
command (see PUNCT).
dos   - Execute DOS command and return
result.
unix  - Execute UNIX command and
return result.
system - Execute system command and
return result.
perl  - Execute Perl command and
return the result.
computer - Computer type.
isunix - True for the UNIX version of
MATLAB.
ispc  - True for the PC (Windows)
version of MATLAB.

  Debugging.
debug - List debugging commands.
mexdebug - Debug MEX-files.

  Tools to locate dependent functions
of an M-file.
depfun - Locate dependent functions of
an M-file or P-file.
depdir - Locate dependent directories
of an M-file or P-file.

  Loading and calling shared
libraries.
calllib - Call a function in an
external library.
libpointer - Creates a pointer object
for use with external libraries.
libstruct - Creates a structure
pointer for use with external
libraries.
libisloaded - True if the specified
shared library is loaded.
loadlibrary - Load a shared library
into MATLAB.
libfunctions - Return information on
functions in an external library.
libfunctionsview - View the functions
in an external library.
unloadlibrary - Unload a shared
library loaded with LOADLIBRARY.
java  - Using Java from within MATLAB.
usejava - True if the specified Java
feature is supported in MATLAB.

  See also lang, datatypes, iofun,
graphics, ops, strfun, timefun,
matfun, demos, graphics, datafun,
uitools, doc, punct, arith.

>> help ops
  Operators and special characters.

  Arithmetic operators.
plus  - Plus                        +
uplus - Unary plus                  +
minus - Minus                       -
uminus - Unary minus                -
mtimes - Matrix multiply            *
times - Array multiply              *
mpower - Matrix power               ^
power - Array power                 ^
mldivide - Backslash or left matrix
divide                              \
mrdivide - Slash or right matrix
divide                              /
ldivide - Left array divide         \
rdivide - Right array divide        /
kron  - Kronecker tensor product kron

  Relational operators.
eq    - Equal                       ==
ne    - Not equal                   ~=
lt    - Less than                   <
gt    - Greater than                >
le    - Less than or equal          <=
ge    - Greater than or equal       >=

  Logical operators.
      Short-circuit logical AND     &&
      Short-circuit logical OR      ||
and   - Element-wise logical AND    &
or    - Element-wise logical OR     |
not   - Logical NOT                 ~
xor   - Logical EXCLUSIVE OR
any   - True if any element of vector
is nonzero
all   - True if all elements of vector
are nonzero

  Special characters.
colon - Colon                       :
paren - Parentheses and subscripting
                                    ( )
paren - Brackets                    [ ]
paren - Braces and subscripting     { }
punct - Function handle creation    @
punct - Decimal point
```

punct - Structure field access .
punct - Parent directory ..
punct - Continuation ...
punct - Separator ,
punct - Semicolon ;
punct - Comment %
punct - Invoke operating system command !
punct - Assignment =
punct - Quote '
transpose - Transpose '
ctranspose - Complex conjugate transpose '
horzcat - Horizontal concatenation [,]
vertcat - Vertical concatenation [;]
subsasgn - Subscripted assignment ( ),{ },
subsref - Subscripted reference ( ),{ },
subsindex - Subscript index

Bitwise operators.
bitand - Bit-wise AND.
bitcmp - Complement bits.
bitor - Bit-wise OR.
bitmax - Maximum floating point integer.
bitxor - Bit-wise XOR.
bitset - Set bit.
bitget - Get bit.
bitshift - Bit-wise shift.

Set operators.
union - Set union.
unique - Set unique.
intersect - Set intersection.
setdiff - Set difference.
setxor - Set exclusive-or.
ismember - True for set member.

See also arith, relop, slash, function_handle.

```
>> help iofun
```
File input and output.

File import/export functions.
dlmread - Read ASCII delimited file.
dlmwrite - Write ASCII delimited file.
importdata - Load data from a file into MATLAB.
daqread - Read Data Acquisition Toolbox (.daq) data file.
matfinfo - Text description of MAT-file contents.

Spreadsheet support.
xlsread - Get data and text from a spreadsheet in an Excel workbook.
xlswrite - Stores numeric array or cell array in Excel workbook.
xlsfinfo - Determine if file contains Microsoft Excel spreadsheet.
wk1read - Read spreadsheet (WK1) file.
wk1write - Write spreadsheet (WK1) file.
wk1finfo - Determine if file contains Lotus WK1 worksheet.
str2rng - Convert spreadsheet range string to numeric array.
wk1wrec - Write a WK1 record header.

Internet resource.
urlread - Returns the contents of a URL as a string.
urlwrite - Save the contents of a URL to a file.
ftp - Create an FTP object.
sendmail - Send e-mail.

Zip file access.
zip - Compress files into a zip file.
unzip - Extract the contents of a zip file.

Formatted file I/O.
fgetl - Read line from file, discard newline character.
fgets - Read line from file, keep newline character.
fprintf - Write formatted data to file.
fscanf - Read formatted data from file.
textscan - Read formatted data from text file.
textread - Read formatted data from text file.

File opening and closing.
fopen - Open file.
fclose - Close file.

Binary file I/O.
fread - Read binary data from file.
fwrite - Write binary data to file.

File positioning.
feof - Test for end-of-file.
ferror - Inquire file error status.
frewind - Rewind file.
fseek - Set file position indicator.
ftell - Get file position indicator.

Memory-mapped file support.
memmapfile - Construct memory-mapped file object.

```
  File name handling.
fileparts - Filename parts.
filesep - Directory separator for this
platform.
fullfile - Build full filename from
parts.
matlabroot - Root directory of MATLAB
installation.
mexext - MEX filename extension for
this platform.
partialpath - Partial pathnames.
pathsep - Path separator for this
platform.
prefdir - Preference directory name.
tempdir - Get temporary directory.
tempname - Get temporary file.

  XML file handling.
xmlread - Parse an XML document and
return a Document Object Model node.
xmlwrite - Serialize an XML Document
Object Model node.
xslt  - Transform an XML document
using an XSLT engine.

  Serial port support.
seriál - Construct serial port object.
instrfindall - Find all serial port
objects with specified property
values.
freeserial - Release MATLAB's hold on
serial port.
instrfind - Find serial port objects
with specified property values.

  Timer support.
timer - Construct timer object.
timerfindall - Find all timer objects
with specified property values.
timerfind - Find visible timer objects
with specified property values.

  Command window I/O.
clc   - Clear command window.
home  - Send the cursor home.

   SOAP support.
createClassFromWsdl - Create a MATLAB
object based on a WSDL-file.
callSoapService - Send a SOAP message
off to an endpoint.
createSoapMessage - Create the SOAP
message, ready to send to the server.
parseSoapResponse - Convert the
response from a SOAP server into
MATLAB types.

  See also general, lang, audiovideo,
imagesci, graphics, uitools.
```

4.5.2 Základní příkazy a funkce pro práci s maticemi

Seznam příkazů a funkcí umístěných v adresářích matlab\elmat a matlab\elfun

```
>> help elmat
    Elementary matrices and matrix
              manipulation.

  Elementary matrices.
zeros - Zeros array.
ones  - Ones array.
eye   - Identity matrix.
repmat - Replicate and tile array.
rand  - Uniformly distributed random
numbers.
randn - Normally distributed random
numbers.
linspace - Linearly spaced vector.
logspace - Logarithmically spaced
vector.
freqspace - Frequency spacing for
frequency response.
meshgrid - X and Y arrays for 3-D
plots.
accumarray - Construct an array with
accumulation.
:     - Regularly spaced vector and
index into matrix.

  Basic array information.
size  - Size of array.
length - Length of vector.
ndims - Number of dimensions.
numel - Number of elements.
disp  - Display matrix or text.
isempty - True for empty array.
isequal - True if arrays are
numerically equal.
isequalwithequalnans - True if arrays
are numerically equal.

  Matrix manipulation.
cat   - Concatenate arrays.
reshape - Change size.
diag  - Diagonal matrices and
diagonals of matrix.
blkdiag - Block diagonal
concatenation.
tril  - Extract lower triangular part.
triu  - Extract upper triangular part.
fliplr - Flip matrix in left/right
direction.
flipud - Flip matrix in up/down
direction.
```

```
flipdim - Flip matrix along specified
dimension.
rot90 - Rotate matrix 90 degrees.
:     - Regularly spaced vector and
index into matrix.
find  - Find indices of nonzero
elements.
end   - Last index.
sub2ind - Linear index from multiple
subscripts.
ind2sub - Multiple subscripts from
linear index.

  Multi-dimensional array functions.
ndgrid - Generate arrays for N-D
functions and interpolation.
permute - Permute array dimensions.
ipermute - Inverse permute array
dimensions.
shiftdim - Shift dimensions.
circshift - Shift array circularly.
squeeze - Remove singleton dimensions.

  Array utility functions.
isscalar - True for scalar.
isvector - True for vector.

  Special variables and constants.
ans   - Most recent answer.
eps   - Floating point relative
accuracy.
realmax - Largest positive floating
point number.
realmin - Smallest positive floating
point number.
pi    - 3.1415926535897....
i, j  - Imaginary unit.
inf   - Infinity.
NaN   - Not-a-Number.
isnan - True for Not-a-Number.
isinf - True for infinite elements.
isfinite - True for finite elements.
why   - Succinct answer.

  Specialized matrices.
compan - Companion matrix.
gallery - Higham test matrices.
hadamard - Hadamard matrix.
hankel - Hankel matrix.
hilb  - Hilbert matrix.
invhilb - Inverse Hilbert matrix.
magic - Magic square.
pascal - Pascal matrix.
rosser - Classic symmetric eigenvalue
test problem.
toeplitz - Toeplitz matrix.
vander - Vandermonde matrix.
wilkinson - Wilkinson's eigenvalue
test matrix.
```

```
>> help elfun
  Elementary math functions.

  Trigonometric.
sin   - Sine.
sind  - Sine of argument in degrees.
sinh  - Hyperbolic sine.
asin  - Inverse sine.
asind - Inverse sine, result in
degrees.
asinh - Inverse hyperbolic sine.
cos   - Cosine.
cosd  - Cosine of argument in degrees.
cosh  - Hyperbolic cosine.
acos  - Inverse cosine.
acosd - Inverse cosine, result in
degrees.
acosh - Inverse hyperbolic cosine.
tan   - Tangent.
tand  - Tangent of argument in
degrees.
tanh  - Hyperbolic tangent.
atan  - Inverse tangent.
atand - Inverse tangent, result in
degrees.
atan2 - Four quadrant inverse tangent.
atanh - Inverse hyperbolic tangent.
sec   - Secant.
secd  - Secant of argument in degrees.
sech  - Hyperbolic secant.
asec  - Inverse secant.
asecd - Inverse secant, result in
degrees.
asech - Inverse hyperbolic secant.
csc   - Cosecant.
cscd  - Cosecant of argument in
degrees.
csch  - Hyperbolic cosecant.
acsc  - Inverse cosecant.
acscd - Inverse cosecant, result in
degrees.
acsch - Inverse hyperbolic cosecant.
cot   - Cotangent.
cotd  - Cotangent of argument in
degrees.
coth  - Hyperbolic cotangent.
acot  - Inverse cotangent.
acotd - Inverse cotangent, result in
degrees.
acoth - Inverse hyperbolic cotangent.

  Exponential.
exp   - Exponential.
expm1 - Compute exp(x)-1 accurately.
log   - Natural logarithm.
log1p - Compute log(1+x) accurately.
log10 - Common (base 10) logarithm.
log2  - Base 2 logarithm and dissect
floating point number.
pow2  - Base 2 power and scale
floating point number.
```

```
realpow - Power that will error out on
complex result.
reallog - Natural logarithm of real
number.
realsqrt - Square root of number
greater than or equal to zero.
sqrt  - Square root.
nthroot - Real n-th root of real
numbers.
nextpow2 - Next higher power of 2.

  Complex.
abs   - Absolute value.
angle - Phase angle.
complex - Construct complex data from
real and imaginary parts.
conj  - Complex conjugate.
imag  - Complex imaginary part.
real  - Complex real part.
unwrap - Unwrap phase angle.
isreal - True for real array.
cplxpair - Sort numbers into complex
conjugate pairs.

  Rounding and remainder.
fix   - Round towards zero.
floor - Round towards minus infinity.
ceil  - Round towards plus infinity.
round - Round towards nearest integer.
mod   - Modulus (signed remainder
after division).
rem   - Remainder after division.
sign  - Signum.
```

4.5.3 Další příkazy a funkce pro práci s maticemi, polynomy a řetězci

Seznam příkazů a funkcí umístěných v adresářích matlab\matfun, matlab\datafun, matlab\sparfun, matlab\polyfun a matlab\strfun

```
>> help matfun
  Matrix functions - numerical linear
                algebra.

  Matrix analysis.
norm  - Matrix or vector norm.
normest - Estimate the matrix 2-norm.
rank  - Matrix rank.
det   - Determinant.
trace - Sum of diagonal elements.
null  - Null space.
orth  - Orthogonalization.
rref  - Reduced row echelon form.
subspace - Angle between two
subspaces.

  Linear equations.
\ and / - Linear equation solution;
use "help slash".
linsolve - Linear equation solution
with extra control.
inv   - Matrix inverse.
rcond - LAPACK reciprocal condition
estimator
cond  - Condition number with respect
to inversion.
condest - 1-norm condition number
estimate
normest1 - 1-norm estimate.
chol  - Cholesky factorization.
cholinc - Incomplete Cholesky
factoriz.
lu    - LU factorization.
luinc - Incomplete LU factorization.
qr    - Orthogonal-triangular
decomposition.
lsqnonneg - Linear least squares with
nonnegativity constraints.
pinv  - Pseudoinverse.
lscov - Least squares with known
covariance.

  Eigenvalues and singular values.
eig   - Eigenvalues and eigenvectors.
svd   - Singular value decomposition.
gsvd  - Generalized singular value
decomposition.
eigs  - A few eigenvalues.
svds  - A few singular values.
poly  - Characteristic polynomial.
polyeig - Polynomial eigenvalue
problem.
condeig - Condition number with
respect to eigenvalues.
hess  - Hessenberg form.
schur - Schur decomposition.
qz    - QZ factorization for
generalized eigenvalues.
ordschur - Reordering of eigenvalues
in Schur decomposition.
ordqz - Reordering of eigenvalues in
QZ factorization.
ordeig - Eigenvalues of
quasitriangular matrices.

  Matrix functions.
expm  - Matrix exponential.
logm  - Matrix logarithm.
sqrtm - Matrix square root.
funm  - Evaluate general matrix
function.

  Factorization utilities
qrdelete - Delete a column or row from
QR factorization.
qrinsert - Insert a column or row into
QR factorization.
rsf2csf - Real block diagonal form to
complex diagonal form.
```

cdf2rdf - Complex diagonal form to real block diagonal form.
balance - Diagonal scaling to improve eigenvalue accuracy.
planerot - Givens plane rotation.
cholupdate - rank 1 update to Cholesky factorization.
qrupdate - rank 1 update to QR factorization.

```
>> help datafun
```
Data analysis and Fourier transforms.

Basic operations.
max - Largest component.
min - Smallest component.
mean - Average or mean value.
median - Median value.
std - Standard deviation.
var - Variance.
sort - Sort in ascending order.
sortrows - Sort rows in ascending order.
sum - Sum of elements.
prod - Product of elements.
hist - Histogram.
histc - Histogram count.
trapz - Trapezoidal numerical integration.
cumsum - Cumulative sum of elements.
cumprod - Cumulative product of elements.
cumtrapz - Cumulative trapezoidal numerical integration.

Finite differences.
diff - Difference and approximate derivative.
gradient - Approximate gradient.
del2 - Discrete Laplacian.

Correlation.
corrcoef - Correlation coefficients.
cov - Covariance matrix.
subspace - Angle between subspaces.

Filtering and convolution.
filter - One-dimensional digital filter.
filter2 - Two-dimensional digital filter.
conv - Convolution and polynomial multiplication.
conv2 - Two-dimensional convolution.
convn - N-dimensional convolution.
deconv - Deconvolution and polynomial division.
detrend - Linear trend removal.
Fourier transforms.
fft - Discrete Fourier transform.
fft2 - Two-dimensional discrete Fourier transform.
fftn - N-dimensional discrete Fourier Transform.
ifft - Inverse discrete Fourier transform.
ifft2 - Two-dimensional inverse discrete Fourier transform.
ifftn - N-dimensional inverse discrete Fourier Transform.
fftshift - Shift zero-frequency component to center of spectrum.
ifftshift - Inverse FFTSHIFT.

```
>> help sparfun
```
Sparse matrices.

Elementary sparse matrices.
speye - Sparse identity matrix.
sprand - Sparse uniformly distributed random matrix.
sprandn - Sparse normally distributed random matrix.
sprandsym - Sparse random symmetric matrix.
spdiags - Sparse matrix formed from diagonals.

Full to sparse conversion.
sparse - Create sparse matrix.
full - Convert sparse matrix to full matrix.
find - Find indices of nonzero elements.
spconvert - Import from sparse matrix external format.

Working with sparse matrices.
nnz - Number of nonzero matrix elements.
nonzeros - Nonzero matrix elements.
nzmax - Amount of storage allocated for nonzero matrix elements.
spones - Replace nonzero sparse matrix elements with ones.
spalloc - Allocate space for sparse matrix.
issparse - True for sparse matrix.
spfun - Apply function to nonzero matrix elements.
spy - Visualize sparsity pattern.

Reordering algorithms.
colamd - Column approximate minimum degree permutation.
symamd - Symmetric approximate minimum degree permutation.
colmmd - Column minimum degree permutation.
symmmd - Symmetric minimum degree permutation.

symrcm - Symmetric reverse Cuthill-McKee permutation.
colperm - Column permutation.
randperm - Random permutation.
dmperm - Dulmage-Mendelsohn permutation.

Linear algebra.
eigs - A few eigenvalues, using ARPACK.
svds - A few singular values, using eigs.
luinc - Incomplete LU factorization.
cholinc - Incomplete Cholesky factorization.
normest - Estimate the matrix 2-norm.
condest - 1-norm condition number estimate
sprank - Structural rank.

Linear Equations (iterative methods).
pcg - Preconditioned Conjugate Gradients Method.
bicg - BiConjugate Gradients Method.
bicgstab - BiConjugate Gradients Stabilized Method.
cgs - Conjugate Gradients Squared Method.
gmres - Generalized Minimum Residual Method.
lsqr - LSQR Method.
minres - Minimum Residual Method.
qmr - Quasi-Minimal Residual Method.
symmlq - Symmetric LQ Method.

Operations on graphs (trees).
treelayout - Lay out tree or forest.
treeplot - Plot picture of tree.
etree - Elimination tree.
etreeplot - Plot elimination tree.
gplot - Plot graph, as in "graph theory".

Miscellaneous.
symbfact - Symbolic factorization analysis.
spparms - Set parameters for sparse matrix routines.
spaugment - Form least squares augmented system.

>> help polyfun
Interpolation and polynomials.

Data interpolation.
pchip - Piecewise cubic Hermite interpolating polynomial.
interp1 - 1-D interpolation (table lookup)
interp1q - Quick 1-D linear interpolation.
interpft - 1-D interpolation using FFT method.
interp2 - 2-D interpolation (table lookup)
interp3 - 3-D interpolation (table lookup)
interpn - N-D interpolation (table lookup)
griddata - Data gridding and surface fitting.
griddata3 - Data gridding and hyper-surface fitting for 3-dimensional data.
griddatan - Data gridding and hyper-surface fitting (dimension >= 2).

Spline interpolation.
spline - Cubic spline interpolation.
ppval - Evaluate piecewise polynomial.

Geometric analysis.
delaunay - Delaunay triangulation.
delaunay3 - 3-D Delaunay tessellation.
delaunayn - N-D Delaunay tessellation.
dsearch - Search Delaunay triangulation for nearest point.
dsearchn - Search N-D Delaunay tessellation for nearest point.
tsearch - Closest triangle search.
tsearchn - N-D closest triangle search.
convhull - Convex hull.
convhulln - N-D convex hull.
voronoi - Voronoi diagram.
voronoin - N-D Voronoi diagram.
inpolygon - True for points inside polygonal region.
rectint - Rectangle intersection area.
polyarea - Area of polygon.

Polynomials.
roots - Find polynomial roots.
poly - Convert roots to polynomial.
polyval - Evaluate polynomial.
polyvalm - Evaluate polynomial with matrix argument.
residue - Partial-fraction expansion (residues).
polyfit - Fit polynomial to data.
polyder - Differentiate polynomial.
polyint - Integrate polynomial analytically.
conv - Multiply polynomials.
deconv - Divide polynomials.

```
>> help strfun
  Character strings.

  General.
char  - Create character array
(string).
strings - Help for strings.
cellstr - Create cell array of strings
from character array.
blanks - String of blanks.
deblank - Remove trailing blanks.

  String tests.
iscellstr - True for cell array of
strings
ischar - True for character array
(string).
isspace - True for white space
characters.
isstrprop - Check if string elements
are of a specified category.

  String operations.
regexp - Match regular expression.
regexpi - Match regular expression,
ignoring case.
regexprep - Replace string using
regular expression.
strcat - Concatenate strings.
strvcat - Vertically concatenate
strings.
strcmp - Compare strings.
strncmp - Compare first N characters
of strings.
strcmpi - Compare strings ignoring
case.
strncmpi - Compare first N characters
of strings ignoring case.
strread - Read formatted data from
string.
findstr - Find one string within
another.
strfind - Find one string within
another.
strjust - Justify character array.
strmatch - Find possible matches for
string.
strrep - Replace string with another.
strtok - Find token in string.
strtrim - Remove insignificant
whitespace.
upper - Convert string to uppercase.
lower - Convert string to lowercase.

  Character set conversion.
native2unicode - Convert bytes to
Unicode characters.
unicode2native - Convert Unicode
characters to bytes.

  String to number conversion.
num2str - Convert numbers to a string.
int2str - Convert integer to string.
mat2str - Convert a 2-D matrix to a
string in MATLAB syntax.
str2double - Convert string to double
precision value.
str2num - Convert string matrix to
numeric array.
sprintf - Write formatted data to
string.
sscanf - Read string under format
control.

  Base number conversion.
hex2num - Convert hexadecimal string
to double precision number.
hex2dec - Convert hexadecimal string
to decimal integer.
dec2hex - Convert decimal integer to
hexadecimal string.
bin2dec - Convert binary string to
decimal integer.
dec2bin - Convert decimal integer to a
binary string.
base2dec - Convert base B string to
decimal integer.
dec2base - Convert decimal integer to
base B string.
num2hex - Convert singles and doubles
to IEEE hexadecimal strings.

    See also general, lang, iofun,
ops, datatypes.
```

5. Řízení průběhu výpočtu

Příkazy a funkce: `break, case, catch, else, end, for, if, otherwise, switch, tic, toc, try, while`

Problém: programové konstrukce - podmínky a cykly, měření časového intervalu, obsluha výjimek

V první kapitole jsme přirovnávali MATLAB k programovacímu interaktivnímu jazyku BASIC. Tento příměr byl použit nejen pro podobnost interaktivního režimu, kdy každý příkaz je hned proveden, ale i pro možnost použití programových konstrukcí. Je možné používat 4 základní programové konstrukce:

I. neúplný a úplný podmíněný příkaz
příklad: výpočet absolutní hodnoty a
výpočet výrazu 1/a s kontrolou na dělení nulou, výsledek je v tomto případě 1
omezení hodnoty v a na interval <1,2> s nastavením příznaku o zda k omezení došlo

if *logický_výraz*
  příkaz
end

```
if a<0
   a=-a;
end
```

if *logický_výraz*
  příkaz1
else
  příkaz2
end

```
if a~=0
   x=1/a;
else
   x=1;
end
```

if *log_výraz1*
  příkaz1
elseif *log_výr2*
  příkaz2
else
  příkaz3
end

```
if a<1
  a=1;o=1;
elseif a>2
  a=2;o=1;
else
  o=0
end
```

II. přepínač (výsledkem výrazu musí být hodnota nebo řetězec, která se porovnává s hodnotami za klíčovými slovy `case`)
příklad: určit, zda první písmeno řetězce v proměnné text je písmeno a nebo b (bez ohledu na velikost)

switch *výraz*
  case *hodnota1*
    příkaz1
  case *hodnota2*
    příkaz2
  otherwise
    příkaz
end

```
switch lower(text(1))
  case 'a'
    disp('První znak je a/A')
  case 'b'
     disp('První znak je b/B')
  otherwise
     disp('První znak není ani a/A ani b/B')
end
```

III. cykl s podmínkou na začátku
příklad: počkat 10 sekund

while *výraz*
  příkaz
end

```
x=0;                % pomocná proměnná, naplň hodnotou 0
tic                 % nastav časovou značku
while x<10          % pokud je podmínka splněna proveď příkaz(y)
   x=toc;           % ulož čas, který uplynul od časové značky
end
```

IV. cykl s pevným počtem opakování (není-li krok explicitně uveden, použije se hodnota 1)
příklad: sečíst druhé mocniny čísel ve vektoru a

for *index=start:krok:konec,*
  příkaz
end

```
s=0;                % nulování součtové proměnné
for k=1:length(a)   % pro k od 1 s krokem 1 do délky vektoru a
   x=x+a(k)^2;      % k min. hod. přičti kvadrát k-tého prvku vekt. a
end
```

Úplný seznam příkazů týkajících se této oblasti je uveden v kapitole 7.5 (`help lang`). Užitečnou vlastností MATLABu od verze 5 může být obsluha výjimek podporovaná příkazy **`try`** a **`catch`**. Tyto příkazy se hodí k řešení situace, kdy může při vykonávání nějaké posloupnosti příkazů dojít k chybě. Standardně v této situaci MATLAB ukončí vykonávání příkazů a ohlásí chybu. Pokud bychom chtěli provést vlastní obsluhu chybové situace, pak se bez těchto příkazů neobejdeme. Před první z příkazů, které chceme "hlídat" umístíme příkaz `try` a za poslední ze sledovaných příkazů `catch`. Za příkazem `catch` musí následovat příkazy, které se mají provést při výskytu chyby, ukončené příkazem `end`. Když k chybě nedojde, provedou se standardně příkazy mezi `try` a `catch` a příkazy mezi `catch` a `end` se neprovedou. Dojde-li při vykonávání některého z příkazů mezi `try` a `catch` k chybě, zbývající příkazy do `catch` se neprovedou a místo nich se provedou příkazy mezi `catch` a `end`. V obou případech se pak pokračuje dalšími příkazy následujícími za příslušným příkazem `end`.

Hlavní využití programových konstrukcí je asi až při psaní uživatelských m-funkcí, ale i v interaktivním režimu jsou občas užitečné. Zvláště příkaz `for` je často využíván. Především lidé, kteří mají zkušenosti s programováním, tento příkaz často využívají i když se dá mnoho případů vyřešit daleko efektivněji důsledným používáním maticových operací.

Nejjednodušší příklad na použití příkazu cyklu `for` může být např. naplnění vektoru o 100 prvcích i-tou odmocninou čísla 100, kde i je pořadové číslo prvku vektoru. Vyjádřeno v matematickém zápisu,

$$x(i) = \sqrt[i]{100} \quad i = 1 \cdots 100$$

a zápis v MATLABu

```
>> for i=1:100
 x(i)=100^(1/i);
 end
```

Všimněte si, že jednotlivé příkazy mohou být zapsány na samostatné řádky a vykonávání se započne až po uvedení posledního klíčového slova `end`. Též je možný praktičtější zápis do jednoho řádku[52]. V případě zápisu do jednoho řádku musí být příkazy programových konstrukcí odděleny čárkou.

Řešení pomocí maticových a vektorových operací je výrazně rychlejší než použití příkazu `for`. Ukažme si vytvoření čtvercové matice 800×800 takové, že první řádek tvoří čísla od 1 do 800 , druhý řádek je dvojnásobkem prvního, třetí trojnásobkem prvního atd. Vytvořme tuto matici pomocí dvou vnořených cyklů a pomocí maticových operací. Pro zajímavost změříme i čas potřebný pro provedení pomocí funkcí **`tic`** - nastavení začátku časového intervalu a **`toc`** - dotaz na dobu, která uplynula od začátku časového intervalu

```
>> tic; for b=1:800, for a=1:800, X(b,a)=a*b; end; end; toc
elapsed_time = 11.138828 (při opakování 1.248099)
>> tic; a=[1:800]; X=a'*a; toc
elapsed_time = 0.104737 (při opakování 0.014495)
```

Vidíme, že maticové řešení je v tomto případě asi o dva řády rychlejší.

Užitečný je v určitých případech příkaz **`break`**, který ukončuje smyčku. Potřebujeme-li např. nalézt ve vektoru *x* první číslo, které je větší než 99 můžeme použít následující konstrukci

```
>> for i=1:length(x), if x(i)>99, break; end, end
```

Cyklus se ukončí při splnění podmínky a v proměnné i bude index prvního čísla splňujícího podmínku. Všimněte si také použití funkce `length` pro určení horní hranice cyklu. V případě, že tuto konstrukci použijeme z příkazového řádku asi budeme velikost vektoru *x* vědět, ale v případě použití ve skriptu či vlastní m-funkci je potom tato konstrukce obecná (platná pro každou velikost vektoru *x*).

[52] praktičtější v tomto případě znamená, že je možné klávesou šipka nahoru se k tomuto (jednomu) řádku vrátit a příkaz najednou zopakovat nebo případně opravit

6. Vizualizace

Příkazy a funkce: `fplot, grid, legend, mesh, meshgrid, plot, plot3, subplot, surf, title, xlabel, ylabel`

Problém: tvorba dvou a třírozměrných grafů, rozdělení oblasti kreslení, popis a editace grafu

Důležitou vlastností programů typu MATLABu jsou jejich možnosti vizualizace. MATLAB v tomto směru poskytuje značné možnosti jak pro dvourozměrnou tak i třírozměrnou vizualizaci a tvorbu speciálních druhů grafů. Umožňuje též vytváření uživatelských rozhraní. Představu co lze vytvořit, získáme nejlépe spuštěním standardní připravené ukázky (příkazem **demo**). V této kapitole se budeme zabývat pouze základními příkazy pro vizualizaci. Bližší informace o grafických příkazech je možné získat pomocí `help graph2d`, `help graph3d` a `help specgraph` (viz kapitola 6.4). Další možnosti vizualizace jsou též zmíněny v kapitole 11.2.

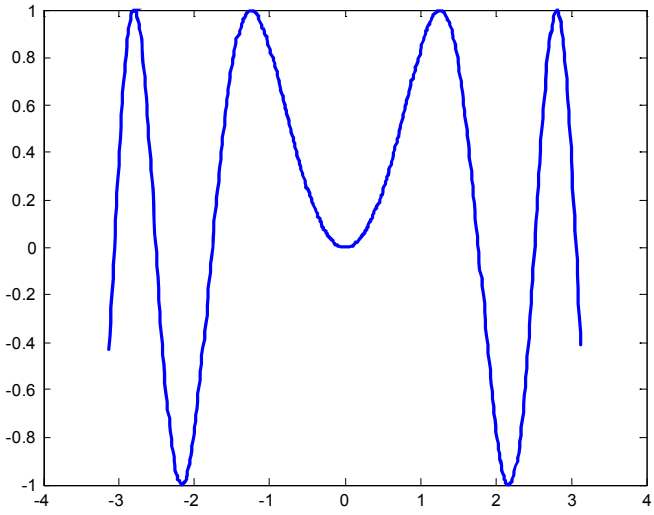
6.1 Dvourozměrné grafy

Základním příkazem pro vykreslení dvourozměrného grafu je příkaz **plot**. Pro úplný popis možností příkazu plot použijte `help plot`. Zde si ukážeme několik příkladů použití.

Máme nakreslit průběh funkce *y=sin(x^2)* na intervalu $x \in \langle -\pi, \pi \rangle$

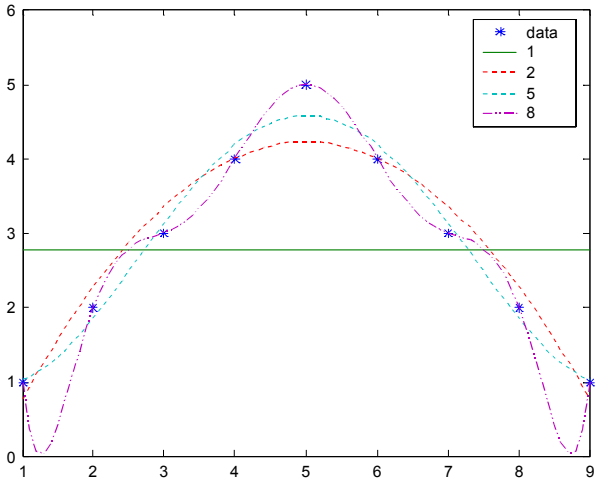
```
>> x=-pi:0.01:pi;
>> y=sin(x.^2);
>> plot(x,y)
```

Výsledek je na obrázku 6-1.



obrázek 6-1 Použití příkazu plot

Chceme-li nakreslit několik průběhů do jednoho grafu (obrázek 6-2), musíme si připravit datové vektory odpovídající jednotlivým grafům. Zkusme nadefinovat několik bodů a proložit jimi polynomy různých stupňů pomocí nám již známé funkce `polyfit`. Prokládané body jsou dány vektory *x* a *y*. V grafu budou vyznačeny hvězdičkami.



obrázek 6-2 Více průběhů v jednom grafu

```
>> x=[1 2 3 4 5 6 7 8 9];
>> y=[1 2 3 4 5 4 3 2 1];
>> x1=1:0.1:9;
>> y1=polyval(polyfit(x,y,1),x1);
>> y2=polyval(polyfit(x,y,2),x1);
>> y5=polyval(polyfit(x,y,5),x1);
>> y8=polyval(polyfit(x,y,8),x1);
>>plot(x,y,'*',x1,y1,x1,y2,x1,y5,x1,y8)
```

Užitečný může být, pokud chceme vykreslit průběhy funkcí, také příkaz **fplot**. Tato funkce je určená přímo na vykreslení průběhu funkce v zadaném rozsahu osy x a případně i y. Nakresleme průběh funkce sin(x^2) a sin(x^0.5) v intervalu $x \in \langle 0, 2\pi \rangle$. Měřítko osy y budiž od -1.5 do 1.5. Výsledek příkazu je na obrázku 6-3.

```
>> fplot('[sin(x.^2),sin(sqrt(x))]',[0 2*pi -1.5 1.5])
```

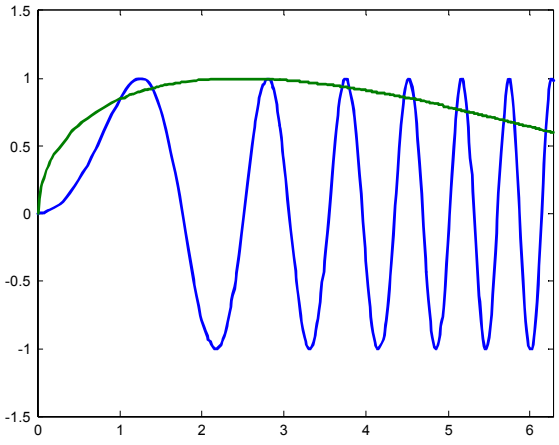
V případě, že potřebujeme několik celých grafů umístit do jednoho obrázku, je užitečný příkaz **`subplot`**. Tento příkaz rozdělí obrázek na několik části a umístí výstup následujícího příkazu `plot` do vybrané části. Ukažme tento případ na příkladě, kdy chceme umístit do obrázku rozděleného na čtyři části čtyři samostatné grafy opatřené titulkem (obrázek 6-4)

$$f_1(x) = \sin(x) \qquad x \in \langle 0, 2\pi \rangle$$

$$f_2(x) = \frac{\sin(x) + \sin(2x)}{2}$$

$$f_3(x) = \frac{\sin(x) + \sin(2x) + \sin(3x)}{3}$$

$$f_4(x) = \frac{\sin(x) + \sin(2x) + \sin(3x) + \sin(4x)}{4}$$

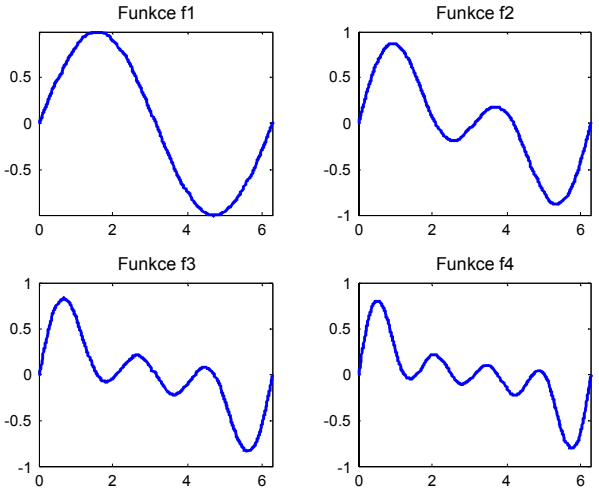


obrázek 6-3 Použití příkazu fplot

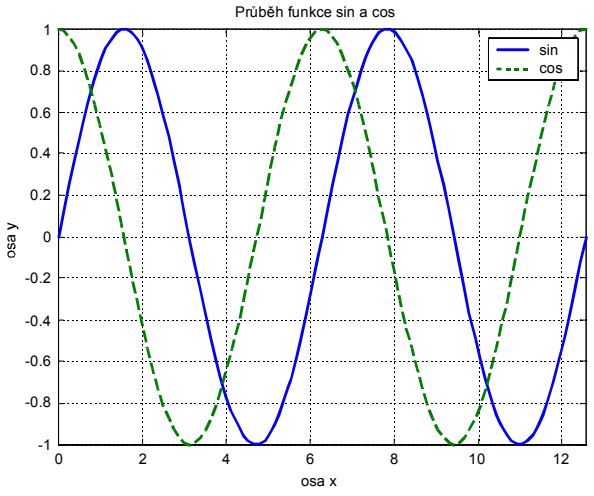
```
>> x=0:0.01:2*pi;
>> f1=sin(x);       f2=f1+sin(2*x);
>> f3=f2+sin(3*x); f4=f3+sin(4*x);
>> subplot(2,2,1); plot(x,f1); title('Funkce f1');
>> subplot(2,2,2); plot(x,f2/2); title('Funkce f2');
>> subplot(2,2,3); plot(x,f3/3); title('Funkce f3');
>> subplot(2,2,4); plot(x,f4/4); title('Funkce f4');
```

Popis grafu lze doplnit o popis osy x (**`xlabel`**), osy y (**`ylabel`**), případně o popis křivek (**`legend`**) a mřížku (**`grid`**) např. jak je uvedeno v následujícím příkladě (obrázek 6-5)

```
>> t=0:4*pi/100:4*pi;
>> y1=sin(t); y2=cos(t); plot(t,y1,'-',t,y2,':')
>> grid, title('Průběh funkce sin a cos')
>> xlabel('osa x'), ylabel('osa y'), legend('sin','cos')
```



obrázek 6-4 Použití příkazu subplot



obrázek 6-5 Popis grafu

Další příkazy pro popis grafu použitelné z příkazové řádky viz kapitola 6.4.

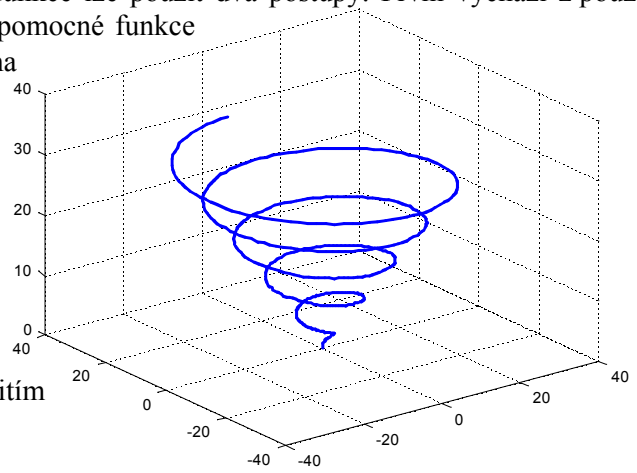
6.2 Třírozměrné grafy

Příkazy pro kreslení třírozměrných grafů jsou v základní podobě sice také jednoduché, ale pozornost je potřeba věnovat přípravě dat pro vykreslení grafu. Nejen co se týká vlastních hodnot, ale mít také rozumné požadavky na jejich počet[53].

Grafy v prostoru mohou být buď čárové nebo plošné. Nakreslit čárový graf v prostoru je velmi jednoduché (příkaz **`plot3`**). Stačí mít k dispozici tři vektory stejné délky, které obsahují souřadnice x, y a z (průběh funkce). Např. spirálu na obr. 6-6 nakreslíme příkazy

```
>> t=0:pi/50:10*pi; plot3(t.*sin(t),t.*cos(t),t); grid
```

Vykreslení funkce dvou proměnných (plochy) v prostoru vyžaduje minimálně matici s hodnotami funkce (Z) případně ještě vektory hodnot X a Y definujícími síť, ve které jsou funkční hodnoty matice vypočteny. Pro vytvoření matice funkčních hodnot dvourozměrné funkce lze použít dva postupy. První vychází z použití příkazu `for`, druhý využívá maticových operací a pomocné funkce **`meshgrid`**. Oba postupy budou ukázány na následujícím příkladu.



obrázek 6-6 Použití příkazu plot3

Nakresleme jak vypadá následující funkce (s dělením 21 bodů v každé ose)

$$z = \cos(x^2 + y^2) \quad x \in \langle -\pi, \pi \rangle \quad y \in \langle -\pi, \pi \rangle$$

definování bodů sítě na osách x a y

```
>> x=[-pi/2:pi/20:pi/2]; y=x;
```

Varianta 1: naplnění matice funkčních hodnot s využitím příkazu for

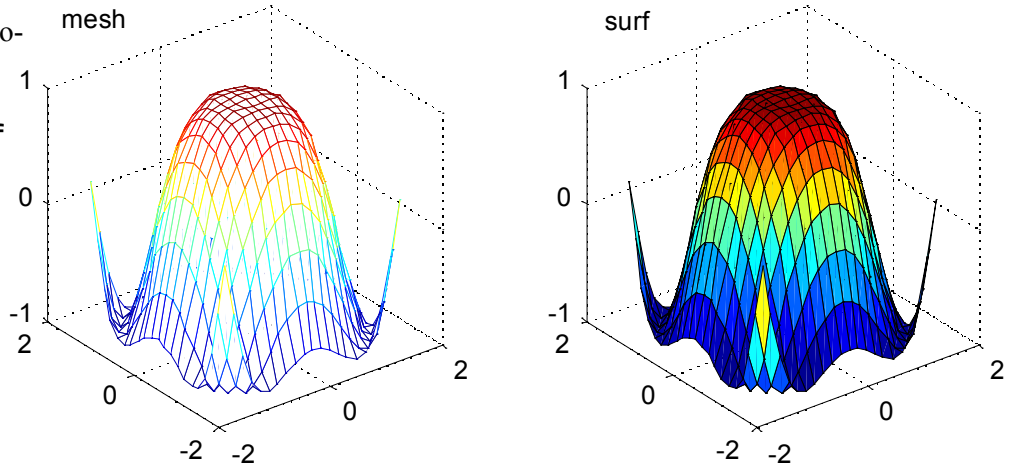
```
>> for a=1:length(x), for b=1:length(y),
Z(a,b)=cos(x(a)^2+y(b)^2);end, end
```

Varianta 2: naplnění matice funkčních hodnot s využitím maticových operací

```
>> [X,Y]=meshgrid(x,y);          % pomocné matice
>> Z=cos(X.^2+Y.^2);             % výpočet funkčních hodnot maticovými operacemi
```

Vlastní vykreslení prostorového pohledu se provede buď příkazem **`mesh`** (drátový model) nebo **`surf`** (stínovaná plocha). Použijeme oba příkazy a obě varianty vykreslíme do jednoho obrázku (obr. 6-7)

```
>> subplot(1,2,1);
>> mesh(x,y,Z);
>> subplot(1,2,2);
>> surf(x,y,Z)
```



obrázek 6-7 Použití příkazů mesh a surf

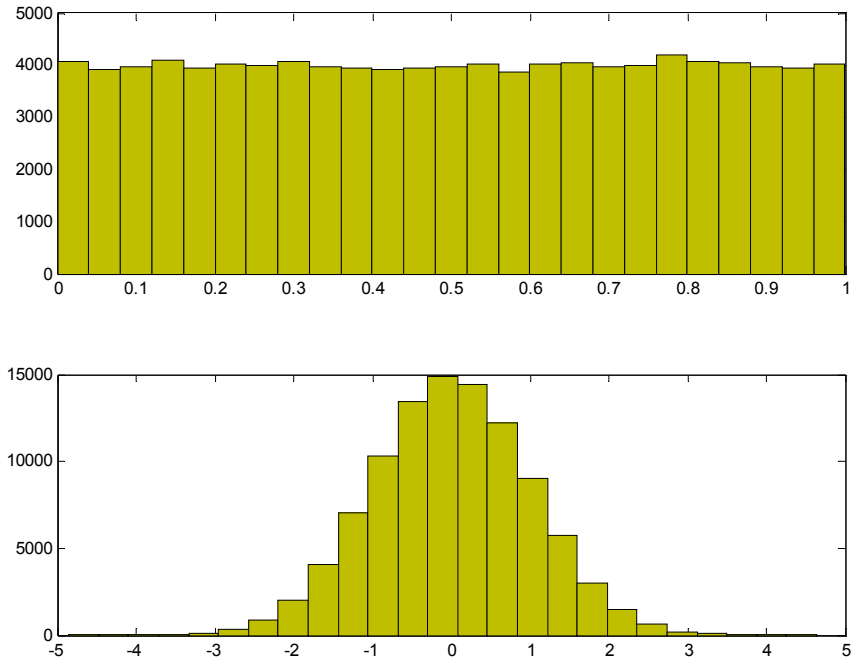
[53] při tvorbě matice definující plochu v prostoru je třeba mít na paměti, že počet bodů narůstá s kvadrátem rozměru matice a že vykreslení plochy složené z např. 10000 bodů (matice 100×100) už může trvat (v závislosti na rychlosti procesoru a grafické karty) historicky významnou dobu.

6.3 Další typy grafů

Kromě standardních typů grafů uvedených v předchozích kapitolách nabízí MATLAB ještě další běžné typy grafů. Uveďme několik příkladů.

Poměrně častou úlohou je vykreslení histogramu. Ověřme pomocí histogramu rozložení dat generované pomocí funkcí `rand` (rovnoměrné rozložení) a `randn` (normální rozložení). Oběmi funkcemi vygenerujeme 100000 čísel a histogram vykreslíme v 25 skupinách. Výsledek je na obrázku 6-8.

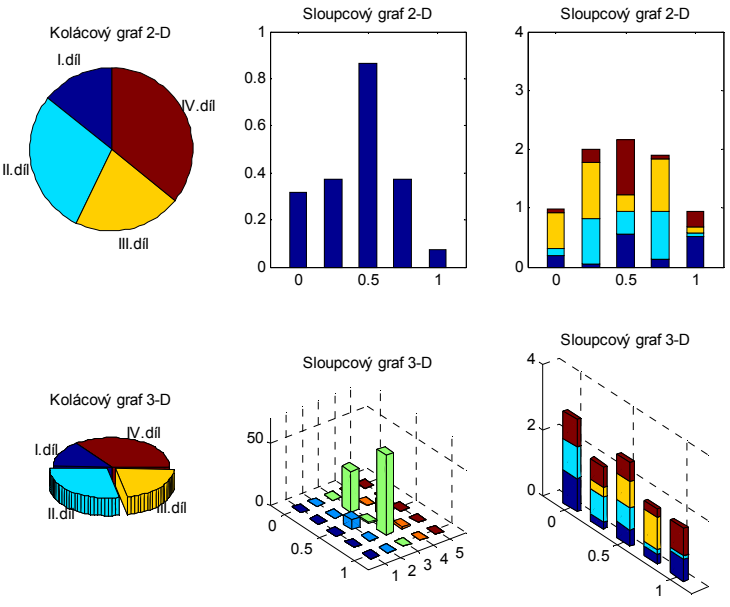
```
>> x1=rand(100000,1);
>> x2=randn(100000,1);
>> subplot(211)
>> hist(x1,25)
>> subplot(212)
>> hist(x2,25)
```



obrázek 6-8 Histogram

Další typy grafů jsou uvedeny na obrázku 6-9. Zde jsou ukázány 2-D a 3-D varianty koláčového a sloupcového grafu.

```
>> % koláčové grafy
>> subplot(231), pie([2 4 3 5],{'I.díl','II.díl','III.díl','IV.díl'})
>> title('Kolácový graf 2-D')
>> subplot(234), pie3([2 4 3 5],[0 1 1 0],{'I.díl','II.díl','III.díl','IV.díl'})
>> title('Kolácový graf 3-D')
>> % sloupcové (bar) grafy
>> subplot(232)
>> bar(0:.25:1,rand(5,1),0.5)
>> title('Sloupcový graf 2-D')
>> axis([-0.2,1.2,0,1])
>> subplot(233)
>> bar(0:.25:1,rand(5,4),0.5,…
   'stacked')
>> title('Sloupcový graf 2-D')
>> axis([-0.2,1.2,0,4])
>> subplot(235)
>> bar3(0:.25:1,…
   peaks(5).^2,0.5)
>> title('Sloupcový graf 3-D')
>> axis([0,6,-0.2,1.2,0,70])
>> subplot(236)
>> bar3(0:.25:1,rand(5,4),0.5,…
  'stacked')
>> title('Sloupcový graf 3-D')
>> axis([0,2,-0.2,1.2,0,4])
```



obrázek 6-9 Koláčové a sloupcové grafy

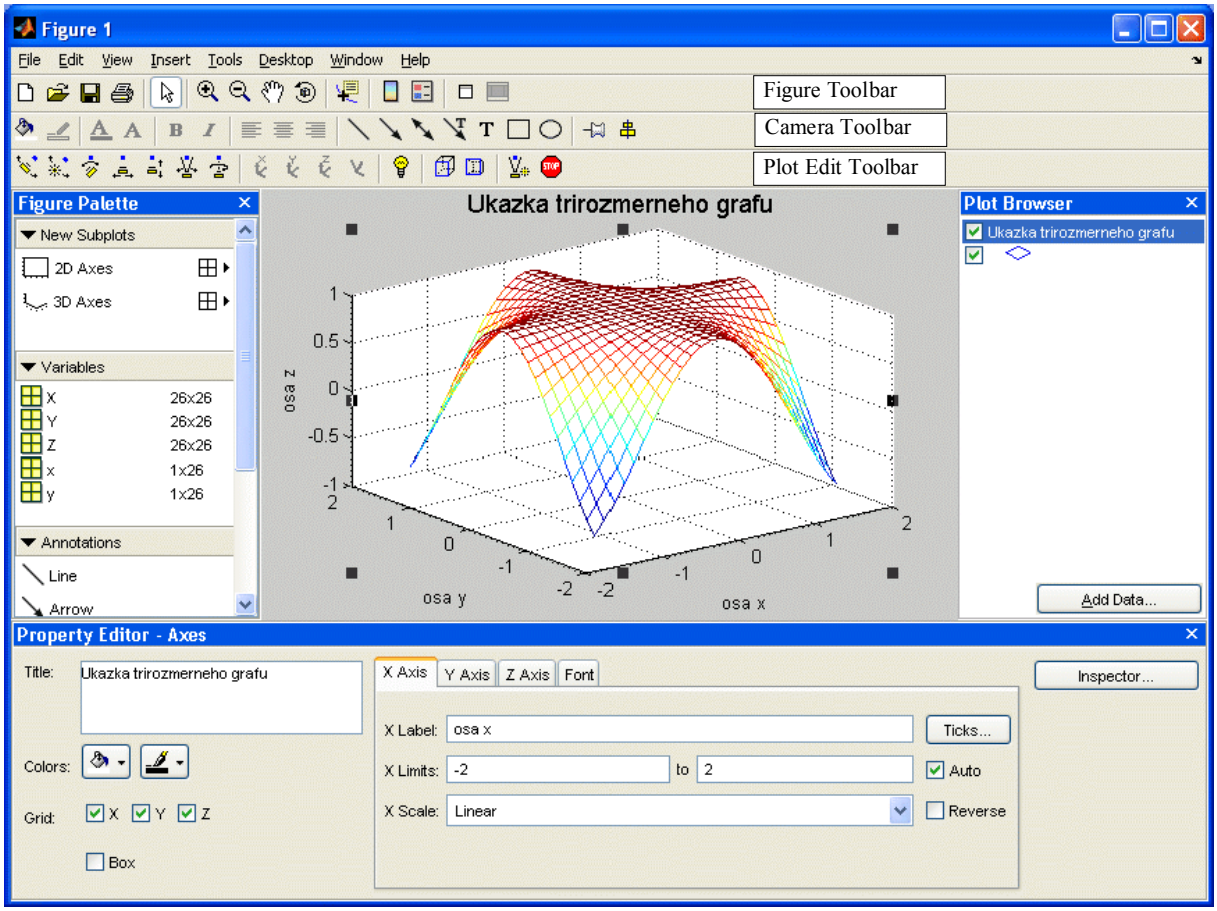
6.4 Editace grafů

Od verze 5 je možné editovat grafy – interaktivně doplňovat či měnit titulek, popisy os, vlastnosti jednotlivých čar v grafu atd. Vytvoříme-li graf příkazy

```
>> x=-pi/2:pi/25:pi/2;
>> y=x;
>> [X,Y]=meshgrid(x,y);
>> Z=cos(X.*Y);
>> mesh(x,y,Z)
```

dostaneme graf plochy v prostoru. Na tomto grafu ukážeme možnosti úprav. V případě třírozměrného grafu může být velmi užitečná možnost rotace grafu v prostoru – funkce umožněná tlačítkem ***Rotate*** (deváté odleva) tlačítkové lišty *Figure Toolbar* na obrázku 6-10. V nabídce ***View*** můžeme zvolit i zobrazení dalších tlačítkových lišt s nástroji pro nasvícení a pohled na graf (*Camera Toolbar*) a přidání standardních prvků do grafu (*Plot Edit Toolbar*). Graf můžeme dále upravovat pomocí příkazů z příkazové řádky MATLABu nebo jednodušeji pomocí předpřipravených nástrojů na editaci. Nejjednodušší editace grafu je pomocí tlačítka ***Edit Plot*** (páté odleva na *Figure Toolbar*). Po aktivaci tohoto tlačítka je možné vybrat myší objekt grafu a dvojklikem či pomocí pravého tlačítka myši se dostat k jeho vlastnostem. Toto je postup při vhodný při požadavku modifikovat základní vlastnost určitého objektu v grafu (tloušťka, barva, typ čáry grafu atd.). Pomocí položky základního menu ***Insert*** lze další objekty přidat (např. šipky, čáry, legendu atd.).

Potřebujeme-li měnit vlastnosti, které nejsou jednoduše přístupné výběrem nebo vlastnosti všech objektů určitého typu najednou je asi nejlepší použít rovnou jednu z voleb *Edit->... Properties*. Tato volba spustí *Property Editor* dovolující nastavit vše možné i nemožné pro všechny objekty grafu. Na obrázku 6-10 jsou otevřena i okna *Figure Palette* a *Plot Browser* – opět v nabídce hlavního menu ***View***.



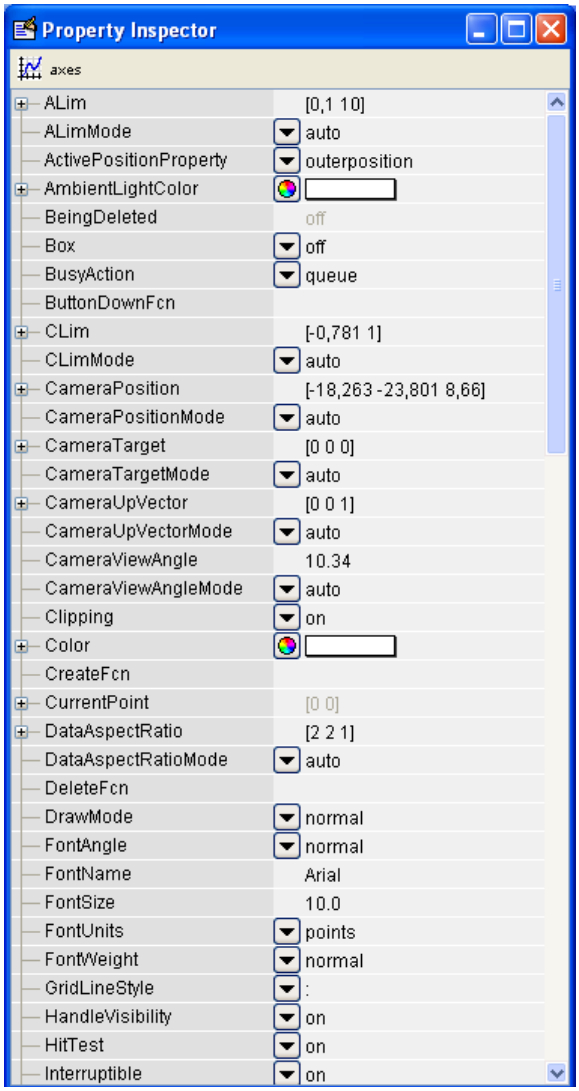
obrázek 6-10 Graf se všemi nástroji pro jeho vizuální editaci

Uspořádání a ovládání editace grafu se verzi od verze liší. Ve verzi 7 obsahuje okno *Property Editor* základní vlastnosti vybraného objektu. Na kompletní seznam vlastností se lze dostat v okně *Property Inspector*. Toto okno vyvoláme tlačítkem *Inspector ...* v pravém horním rohu okna *Property Editor*. Na obrázku 6-11 je okno *Property Inspector* s částí seznamu vlastností objektu grafu *Axes*.

6.5 Seznam funkcí a příkazů pro vytvoření a úpravu grafů

Na závěr uvedeme seznam příkazů a funkcí vztahujících se k tvorbě grafů a vizualizaci. Jde o funkce získané příkazy `help graph2d`, `help graph3d` a `help specgraph`.

```
>> help graph2d
  Two dimensional graphs.

  Elementary X-Y graphs.
plot  - Linear plot.
loglog - Log-log scale plot.
semilogx - Semi-log scale plot.
semilogy - Semi-log scale plot.
polar - Polar coordinate plot.
plotyy - Graphs with y tick labels on
the left and right.

  Axis control.
axis  - Control axis scaling and
appearance.
zoom  - Zoom in and out on a 2-D
plot.
grid  - Grid lines.
box   - Axis box.
hold  - Hold current graph.
axes  - Create axes in arbitrary
positions.
subplot - Create axes in tiled
positions.

  Graph annotation.
plotedit - Tools for editing and
annotating plots.
title - Graph title.
xlabel - X-axis label.
ylabel - Y-axis label.
texlabel - Produces TeX format from a
character string
text  - Text annotation.
gtext - Place text with mouse.

  Hardcopy and printing.
print - Print graph or Simulink
system; or save graph to M-file.
printopt - Printer defaults.
orient - Set paper orientation.
```



obrázek 6-11 Část seznamu vlastností objektu grafu Axes

```
>> help graph3d
  Three dimensional graphs.

  Elementary 3-D plots.
plot3 - Plot lines and points in 3-D
space.
mesh  - 3-D mesh surface.
surf  - 3-D colored surface.
fill3 - Filled 3-D polygons.

  Color control.
colormap - Color look-up table.
caxis - Pseudocolor axis scaling.

shading - Color shading mode.
hidden - Mesh hidden line removal
mode.
brighten - Brighten or darken color
map.
colordef - Set color defaults.
graymon - Set graphics defaults for
gray-scale monitors.
```

Lighting.
surfl - 3-D shaded surface with lighting.
lighting - Lighting mode.
material - Material reflectance mode.
specular - Specular reflectance.
diffuse - Diffuse reflectance.
surfnorm - Surface normals.

Color maps.
hsv - Hue-saturation-value color map
hot - Black-red-yellow-white color map.
gray - Linear gray-scale color map.
bone - Gray-scale with tinge of blue color map.
copper - Linear copper-tone color map.
pink - Pastel shades of pink color map.
white - All white color map.
flag - Alternating red, white, blue, and black color map.
lines - Color map with the line colors.
colorcube - Enhanced color-cube color map.
vga - Windows colormap for 16 colors.
jet - Variant of HSV.
prism - Prism color map.
cool - Shades of cyan and magenta color map.
autumn - Shades of red and yellow color map.
spring - Shades of magenta and yellow color map.
winter - Shades of blue and green color map.
summer - Shades of green and yellow color map.

Transparency.
alpha - Transparency (alpha) mode.
alphamap - Transparency (alpha) look-up table.
alim - Transparency (alpha) scaling

Axis control.
axis - Control axis scaling and appearance.
zoom - Zoom in and out on a 2-D plot.
grid - Grid lines.
box - Axis box.
hold - Hold current graph.
axes - Create axes in arbitrary positions.
subplot - Create axes in tiled positions.
daspect - Data aspect ratio.
pbaspect - Plot box aspect ratio.
xlim - X limits.
ylim - Y limits.
zlim - Z limits.

Viewpoint control.
view - 3-D graph viewpoint specification.
viewmtx - View transformation matrix.
rotate3d - Interactively rotate view of 3-D plot.

Camera control.
campos - Camera position.
camtarget - Camera target.
camva - Camera view angle.
camup - Camera up vector.
camproj - Camera projection.

High level camera control.
camorbit - Orbit camera.
campan - Pan camera.
camdolly - Dolly camera.
camzoom - Zoom camera.
camroll - Roll camera.
camlookat - Move camera and target to view specified objects.
cameratoolbar - Interactively manipulate camera.

High level light control.
camlight - Creates or sets position of a light.
lightangle - Spherical position of a light.

Graph annotation.
title - Graph title.
xlabel - X-axis label.
ylabel - Y-axis label.
zlabel - Z-axis label.
text - Text annotation.
gtext - Mouse placement of text.
plotedit - Experimental graph editing and annotation tools.

Hardcopy and printing.
print - Print graph or Simulink system; or save graph to M-file.
printopt - Printer defaults.
orient - Set paper orientation.
vrml - Save graphics to VRML 2.0 file.

```
>> help specgraph
  Specialized graphs.

  Specialized 2-D graphs.
area  - Filled area plot.
bar   - Bar graph.
barh  - Horizontal bar graph.
comet - Comet-like trajectory.
compass - Compass plot.
errorbar - Error bar plot.
ezplot - Easy to use function
plotter.
ezpolar - Easy to use polar
coordinate plotter.
feather - Feather plot.
fill  - Filled 2-D polygons.
fplot - Plot function.
hist  - Histogram.
pareto - Pareto chart.
pie   - Pie chart.
plotmatrix - Scatter plot matrix.
rose  - Angle histogram plot.
scatter - Scatter plot.
stem  - Discrete sequence or "stem"
plot.
stairs - Stairstep plot.

  Contour and 2-1/2 D graphs.
contour - Contour plot.
contourf - Filled contour plot.
contour3 - 3-D Contour plot.
clabel - Contour plot elevation
labels
ezcontour - Easy to use contour
plotter.
ezcontourf - Easy to use filled
contour plotter.
pcolor - Pseudocolor (checkerboard)
plot.
voronoi - Voronoi diagram.

  Specialized 3-D graphs.
bar3  - 3-D bar graph.
bar3h - Horizontal 3-D bar graph.
comet3 - 3-D comet-like trajectories.
ezgraph3 - General purpose surface
plotter.
ezmesh - Easy to use 3-D mesh
plotter.
ezmeshc - Easy to use combination
mesh/contour plotter.
ezplot3 - Easy to use 3-D parametric
curve plotter.
ezsurf - Easy to use 3-D colored
surface plotter.
ezsurfc - Easy to use combination
surf/contour plotter.
meshc - Combination mesh/contour
plot.
meshz - 3-D mesh with curtain.
pie3  - 3-D pie chart.
ribbon - Draw 2-D lines as ribbons in
3-D.
scatter3 - 3-D scatter plot.
stem3 - 3-D stem plot.
surfc - Combination surf/contour
plot.
trisurf - Triangular surface plot.
trimesh - Triangular mesh plot.
waterfall - Waterfall plot.

  Volume and vector visualization.
vissuite - Visualization suite.
isosurface - Isosurface extractor.
isonormals - Isosurface normals.
isocaps - Isosurface end caps.
isocolors - Isosurface and patch
colors.
contourslice - Contours in slice
planes.
slice - Volumetric slice plot.
streamline - Streamlines from 2D or
3D vector data.
stream3 - 3D streamlines.
stream2 - 2D streamlines.
quiver3 - 3D quiver plot.
quiver - 2D quiver plot.
divergence - Divergence of a vector
field.
curl  - Curl and angular velocity of
a vector field.
coneplot - 3D cone plot.
streamtube - 3D stream tube.
streamribbon - 3D stream ribbon.
streamslice - Streamlines in slice
planes.
streamparticles - Display stream
particles.
interpstreamspeed - Interpolate
streamline vertices from speed.
subvolume - Extract subset of volume
dataset.
reducevolume - Reduce volume dataset.
volumebounds - Returns x,y,z and
color limits for volume data.
smooth3 - Smooth 3D data.
reducepatch - Reduce number of patch
faces.
shrinkfaces - Reduce size of patch
faces.

  Images display and file I/O.
image - Display image.
imagesc - Scale data and display as
image.
colormap - Color look-up table.
gray  - Linear gray-scale color map.
contrast - Gray scale color map to
enhance image contrast.
brighten - Brighten or darken color
map.
```

colorbar - Display color bar (color scale).
imread - Read image from graphics file.
imwrite - Write image to graphics file.
imfinfo - Information about graphics file.
im2java - Convert image to Java image.

Movies and animation.
getframe - Get movie frame.
movie - Play recorded movie frames.
rotate - Rotate object about specified orgin and direction.
frame2im - Convert movie frame to indexed image.
im2frame - Convert index image into movie format.

Color related functions.
spinmap - Spin color map.
rgbplot - Plot color map.
colstyle - Parse color and style from string.
ind2rgb - Convert indexed image to RGB image.

Solid modeling.
cylinder - Generate cylinder.
sphere - Generate sphere.
ellipsoid - Generate ellipsoid.
patch - Create patch.
surf2patch - Convert surface data to patch data.

7. Tvorba skriptů a funkcí

Příkazy a funkce: `addpatch, edit, function, nargin, nargout, path, rmpath`

Problém: zápis skriptu a funkce, vstupní a výstupní parametry, komentář, editor, debbuger, profiler

Do tohoto okamžiku jsme s MATLABem pracovali v interaktivním režimu tj. každý příkaz byl po jeho dopsání ihned proveden. Často však potřebujeme určitou posloupnost příkazů opakovat - ať už v rámci jednoho spuštění MATLABu či při příštím spuštění. Pokud tato opakovaná činnost pracuje stále se stejnými daty nebo nad proměnnými určitých jmen je to úkol pro tzv. **skript**. Pokud potřebuji určitou činnost vykonávat nad různými daty a výsledek uložit do proměnné či proměnných předem neznámých jmen je vhodné napsat vlastní funkci.

Několik velice užitečných funkcí MATLABu vyžaduje definovat zadání ve formě funkce - např. řešení soustavy diferenciálních rovnic, integrál funkce, hledání minima funkce jedné nebo více proměnných atd.

Skripty i funkce se vytvářejí v textové podobě (ASCII) libovolným editorem, který tento výstupní formát umožňuje. Ve Windows může být používán standardní Notepad (Poznámkový blok). Od verze 5 je součástí MATLABu vlastní editor/debugger (viz kapitola 7.4).

7.1 Skript

Skript je posloupnost příkazů zapsaná do souboru s příponou .m. Napíšeme-li jméno tohoto souboru (bez přípony) do příkazové řádky MATLABu, jsou jednotlivé příkazy postupně vykonány jako bychom je napsali z klávesnice. Proměnné, které byly před spuštěním skriptu definovány, lze ve skriptu použít. Proměnné vytvořené v rámci provádění skriptu zůstanou po jeho skončení zachovány.

Využití může být např. pro zpracování souboru naměřených dat. Uveďme si příklad skriptu, který proloží naměřená data uložená v souboru na disketě polynomem 3. stupně a do dalšího souboru uloží hodnoty získané vyhodnocením polynomu v 101 bodech rovnoměrně rozdělených mezi minimální a maximální hodnotou *x*. Soubor s původními daty se jmenuje *data.dat* a obsahuje tři sloupce dat - pořadové číslo měření, hodnoty *x* a hodnoty *y*. Soubor s výsledky se bude jmenovat *datarx.dat* a bude obsahovat dva sloupce dat – nové hodnoty *x* a *y*.

```
load a:\data.dat          % načtení dat do matice jménem data
x=data(:,2);              % vyjmutí druhého sloupce dat do proměnné x
y=data(:,3);              % vyjmutí třetího sloupce dat do proměnné y
r=polyfit(x,y,3);         % určení koeficientu prokládajícího polynomu 3.stupně
xmin=min(x);              % minimální hodnota x
xmax=max(x);              % maximální hodnota x
xn=xmin:(xmax-xmin)/100:xmax; % 101 rovnoměrně rozl. hodnot mezi xmin a xmax
yn=polyval(r,xn);         % výpočet hodnot y podle polynomu
xn=xn(:);yn=yn(:);        % výsledky budou sloupc. vektory
A=[xn;yn];                % vytvoření matice 100x2
save a:\datarx.dat A -ascii   %uloženi matice A do výstupního souboru
```

Pomocí editoru MATLABu (spuštění příkazem **`edit`**) nebo textovým editorem je nutné vytvořit soubor, který bude obsahovat příslušné řádky a uložit jej pod jménem např. *zprdata.m* do aktuálního adresáře.

Nyní napíšeme-li

```
>> zprdata
```

příkazy se provedou a soubor s daty a:\data.dat se zpracuje. Pokud by nás nyní zajímalo jak vypadají naměřená data a proložení můžeme rovnou napsat příkazy

```
>> ya=polyval(r,x);          % výpočet proložených hodnot v původních bodech x
>> plot(x,y,'*',x,ya)        % nakreslení jednotlivých bodů a křivky
```

Vykreslí se aktuální data, protože použité proměnné zůstaly definovány.

7.2 Funkce

Funkce jsou podstatně efektivnější a obecnější nástroj než skripty. Např. toolboxy - rozšíření MATLABu pro řešení určitého okruhu úloh - jsou většinou napsané ve formě uživatelských m-funkcí.

Čím se tedy liší funkce od skriptu, když obojí je tvořeno posloupností třeba stejných příkazů? Zdánlivě nepodstatnou, ale ve svých důsledcích nesmírně důležitou maličkostí. První řádek souboru tvořícího funkci obsahuje tzv. hlavičku funkce. Tato hlavička má formální tvar

function [out1,out2,...]=*jmeno_funkce*(inp1,inp2,...)

a zajišťuje přenos dat do a z funkce. Parametry na pravé straně v kulaté závorce jsou vstupní, parametry na levé straně v hranaté závorce jsou výstupní. Vše, co se děje ve funkci, je samostatné a oddělené od volajícího programu. Proměnné vytvořené uvnitř funkce jsou lokální tj. po skončení posledního příkazu funkce zase zaniknou a vůbec nesouvisí s proměnnými v pracovním prostoru MATLABu. Má to tu výhodu, že nemusíme přemýšlet nad názvy pracovních proměnných ve funkci, abychom náhodou nesmazali obsah nějaké důležité proměnné se stejným jménem mimo funkci. Jediné spojení směrem dovnitř funkce je zajištěno vstupními parametry a směrem z funkce výstupními parametry. Názvy parametrů definovaných v hlavičce (formální parametry) používáme ve výrazech uvnitř funkce a při použití funkce (volání) jsou nahrazeny skutečnými parametry, které při každém volání mohou být jiné.

První řádek - hlavička - musí začínat klíčovým slovem **`function`**, za kterým následuje seznam[54] výstupních argumentů (parametrů). Je-li jich více než jeden, je nutné je dát do hranatých závorek a oddělit čárkami. Poté následuje rovnítko a jméno funkce, které by mělo být shodné se jménem souboru[55], ve kterém je funkce uložena. Za jménem funkce je seznam vstupních argumentů (parametrů) uzavřený do kulatých závorek. Připomeňme to, že každý argument může být libovolného typu (skalár, vektor, matice, struktura, objekt atd.)

Na dalších řádcích se obvykle píše stručný popis (komentář) - co funkce dělá a jaké má parametry. Tyto informační řádky musí začínat znakem % (procento). Text na těchto řádcích se také vypíše, napíšeme-li **`help`** ***`jméno_funkce`***. Znak % je možné použít kdekoli a označuje komentář. Od znaku % do konce řádku se text při provádění ignoruje.

Na následujících řádcích se píší vlastní příkazy funkce - tzv. tělo funkce. V příkazech mohu používat jména vstupních parametrů - při konkrétním výpočtu se použijí hodnoty, které byly na místě příslušného vstupního parametru[56] při volání funkce zapsány. Mohu používat pomocné proměnné libovolného jména. Uložím-li něco do výstupního parametru, přenese se tato hodnota do proměnné, která byla při volání funkce na místě příslušného výstupního parametru použita.

```
function [f,vyk] = vsh(data,frek)
%
%         [f,vyk] = vsh(data,frek)
%
% Vykonova spektralni hustota signalu vzorkovaneho
% s frekvenci frek. Zmerene vzorky jsou ve
% vektoru data
% f...frekvence, na kterych lze zjistit vykon
% vyk  ...vykon na odpovidajich frekvencich f
% Prevod na nulovou stredni hodnotu =
% odstraneni stejnosmerne slozky

n = length(data);       % pocet bodu mereneho signalu
y = data - sum(data)/n; % odecteni stredni hodnoty
Y = fft(y);             % Fourierova transformace
Pyy = Y.*conj(Y)/n; % Vykonova spektralni hustota
f = frek/n*(0:n/2); % Hodnoty realnych frekvenci
vyk = Pyy(1:n/2+1); % Hodnoty vykonu
```

Jako příklad uveďme funkci, která určí výkonovou spektrální hustotu signálu vzorkovaného s frekvencí *frek* a změřená data jsou uložena ve vektoru *data*.

---

[54] na rozdíl oproti programovacím jazykům, které dovolují pouze jeden výstupní argument

[55] rozhodující je jméno souboru. Používat různá jména v názvu funkce a souboru je sice možné, ale matoucí. Windows 95 a vyšší umožňují používání dlouhých názvů, ale pro použití v MATLABu není vhodné je používat. Název funkce v MATLABu nesmí obsahovat např. mezeru.

[56] takzvané formální parametry jsou nahrazeny skutečnými parametry funkce

Napíšeme-li nyní na příkazovou řádku help vsh vypíší komentářové řádky následující za prvním řádkem funkce

```
>> help vsh

                  [f,vyk] = vsh(data,frek)

  Vykonova spektralni hustota signalu vzorkovaneho
  s frekvenci frek. Zmerene vzorky jsou ve
  vektoru data
  f    ...frekvence, na kterych lze zjistit vykon
  vyk  ...vykon na odpovidajich frekvencich f
  Prevod na nulovou stredni hodnotu =
  odstraneni stejnosmerne slozky
```

Funkce v MATLABu jsou značně flexibilní. Je možné vyvolat funkci nejen s různými typy vstupních parametrů (skalár, vektor, matice) ale i s různým počtem vstupních a výstupních parametrů. Uvnitř funkce je možné tuto skutečnost zjistit (funkce **`nargin`**, **`nargout`**) a např. podle přítomnosti výstupních parametrů měnit činnost funkce. Příkladem může být funkce **`dstep`** z Control System Tbx (výpočet přechodové charakteristiky soustavy), která podle počtu vstupních parametrů určuje zda soustava je popsána přenosovou funkcí nebo stavovým modelem a podle přítomnosti výstupních parametrů rozhoduje zda vypočtené hodnoty má uložit do těchto parametrů nebo je pouze nakreslit do grafu.

V novějších verzích MATLABu (od verze 6) navíc použití objektů dovoluje měnit význam funkce podle typu objektu. Např. již není nutné rozlišovat funkce `step` (pro určení odezvy na skok pro spojitý systém) a `dstep` (pro diskrétní systém). Použijeme-li popis soustavy ve formě objektu LTI pak funkce `step` provede výpočet a vykreslení korektně pro oba typy popisu soustavy sama.

Kromě předávání informace z/do funkce pomocí výstupních/vstupních argumentů MATLAB podporuje i používání globálních[57] proměnných. Klíčové slovo **`global`**, použité před názvem proměnné např. ve funkci a na příkazovém řádku, nastaví že příslušná proměnná je společná pro datový pracovní prostor (workspace) funkce i příkazového řádku.

7.3 Poznámka ke konceptu funkcí v MATLABu

Z určitého pohledu geniální[58] koncepce m-funkcí a skriptů v MATLABu začala s postupným rozvojem MATLABu narážet na problém s množstvím. Každá m-funkce či skript je reprezentován malým souborem na disku. Instalace MATLABu tak obsahuje velké množství souborů a každý toolbox přidává další. První problém (souvisí se souborovým systémem operačního systému) je s efektivitou využití místa na disku. Zvláště ve Windows (s jednoduššími souborovými systémy FAT či FAT32) je to markantní[59].

Druhým problémem je zajistit nalezení souboru (m-funkce) daného jména. MATLAB toto zajišťuje tím, že při instalaci vytváří seznam adresářů, které se při hledání prohlížejí. Tento seznam můžeme samozřejmě upravovat – buď příkazy z příkazové řádky (**`path`**, **`addpath`**, **`rmpath`** atd.) nebo editorem adresářových (položka menu *File->Set Path*). Napíšeme-li jméno m-funkce, hledá se nejprve v aktuálním adresáři a poté v jednotlivých adresářích prohledávaných v pořadí v jakém jsou zapsány v seznamu adresářových cest. Použije se soubor, který se nalezne jako první.

S hledáním funkce daného jména souvisí třetí, z pohledu uživatele, největší problém. Jak zajistit abych při vytváření své m-funkce nepoužil již existující jméno. Pokud použiji jméno built-in funkce, pak bude zavolána původní funkce MATLABu a moje funkce bude ignorována. Pokud použiji jméno existující jiné m-funkce pak bude záležet, ve kterém adresáři se bude nacházet původní a moje funkce. Bude-li moje m-

---

[57] stejně jako v programovacích jazycích způsob výměny informace, který se obecně nedoporučuje používat protože dovoluje vznik obtížně identifikovatelných chyb. Na rozdíl od programovacích jazyků je proměnná společná pouze pro ty datové prostory, v rámci kterých bylo klíčové slovo `global` použito.

[58] jednoduchá, univerzální, na uživatelské úrovni přenositelná mezi různými výpočetními platformami

[59] např. instalace MATLABu 6.1 (18 instalovaných toolboxů) pod Win98 s FAT32 na disku 16 GB (velikost alokační jednotky 16 kB) obsahuje 40 910 souborů v 989 adresářích. Tyto soubory obsahují cca 482 MB byte a zabírají na disku cca 1000 MB. Efektivita využití diskového prostoru je tedy cca 50%.

funkce v aktuálním adresáři, použije se (aktuální adresář je prohledáván první). Původně existující funkce bude nedostupná (zakryta mojí funkcí). Umístím-li moji funkci do adresáře, který je dále od začátku seznamu adresářových cest než adresář s existující funkcí, použije se existující funkce a moje bude nedostupná.

Pro běžnou práci je rozumné pracovat v samostatném adresáři, který nastavím jako pracovní. Problémy nastávají zejména pokud chci své funkce předat dál (např. ve formě toolboxu). Zde nabízí MATLAB možnost používat privátní funkce. Jsou to funkce (soubory) umístěné v adresáři ***private***. Funkce v tomto adresáři umístěné jsou "viditelné" pouze z funkcí umístěných v adresáři nadřazeném.

Další možností je využití subfunkcí. Jde o soubory, kde je několik klíčových slov **`function`**. Příkazy umístěné za prvním klíčovým slovem `function` do výskytu dalšího klíčového slova `function` tvoří standardní funkci jak ji již známe. Další funkce (i více) následující v tomto souboru jsou tzv. subfunctions tj. funkce použitelné ("viditelné") pouze ze základní funkce – viz příklad v rámečku. V MATLABu jsou funkce **`mean`** a **`median`** standardně k dispozici. V uvedeném příkladu se ale použijí vlastní definice těchto funkcí. První funkce v souboru je tzv. primární a pouze tato funkce je přístupná zvenčí. Tato možnost je užitečná např. v situaci, kdy moje m-funkce opakovaně používá na několika místech stejný výpočet s různými hodnotami (typická situace pro použití funkce), ale nikdo jiný ji nebude nepotřebovat. Pokud použiji standardní postup, musím vytvořit nový soubor s pomocnou funkcí a zvolit jeho jméno takové, aby nekolidovalo s již existujícími názvy nebo využít privátní adresář. Použití lokální funkce řeší problém kolize názvů a navíc je pomocná funkce pohromadě s hlavní v jednom souboru.

```
function [avg,med] = newstats(u)
% Primary function
% NEWSTATS Find mean and median with internal
% functions.
n = length(u);
avg = mean(u,n);
med = median(u,n);

function a = mean(v,n)          % Subfunction
% Calculate average.
a = sum(v)/n;

function m = median(v,n)        % Subfunction
% Calculate median.
w = sort(v);
if rem(n,2) == 1,   m = w((n+1)/2);
else                m = (w(n/2)+w(n/2+1))/2;
end
```

Objektová orientace MATLABu dovoluje ještě další způsob určení, která funkce má být použita – rozlišení podle typu objektu použitého jako vstupní parametr. V systému adresářů jsou adresáře se jménem začínajícím znakem @ a pokračující názvem typu objektu (class). V tomto adresáři jsou pak funkce, které se mají použít v případě argumentu tohoto typu. Například funkci s názvem fopen (otevření souboru) lze použít pro otevření souboru na disku i pro otevření komunikačního kanálu přes sériovou linku. V prvním případě je argumentem textový řetězec (použije se funkce *fopen.m* z adresáře *matlab\toolbox\matlab\iofun*) a v druhém případě objekt typu `serial` (použije se funkce *fopen.m* z adresáře *matlab\toolbox\matlab\iofun\@serial*).

7.4 Editor/debugger/profiler

Do verze 5.0 se k zápisu a opravě uživatelem vytvářených skriptů a funkcí používal jakýkoli[60] textový editor umožňující uložit čistý ASCII text. Ladění sice bylo systémem MATLAB podporováno, ale pouze na úrovni řádkových příkazů. Uživatel se musel starat[61] především sám. Od verze 5.0 je součástí MATLABu i vlastní editor, který zároveň slouží jako debugger[62]. Jde na první pohled o poměrně jednoduchý editor, který ale zahrnuje některé užitečné vlastnosti (zvýrazňuje klíčová slova MATLABu, při déle trvajícím umístění kurzoru myši nad názvem proměnné se otevře okno s hodnotou, označuje čísla řádků atd.).

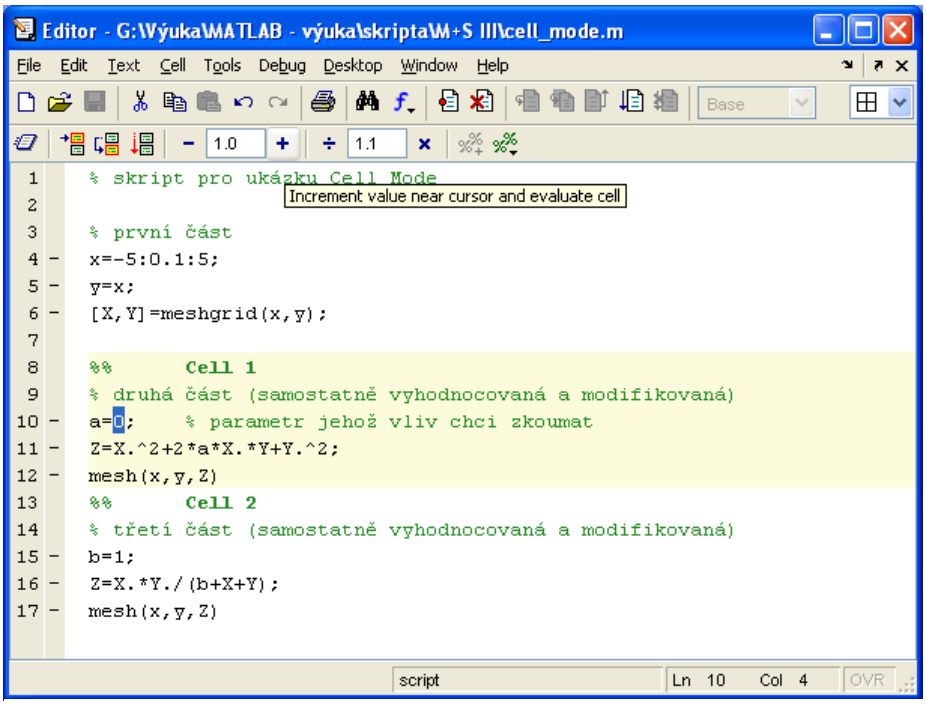
Praktickou možností je vykonání označených příkazů bez opuštění editoru. Označíme-li jeden či více příkazů v okně editoru a stiskneme-li klávesu F9 (či po stisku pravého tlačítka vybereme z kontextové nabídky volbu ***Evaluate Selection***), jsou vybrané příkazy provedeny v příkazovém okně MATLABu. Od verze 7 je tato

[60] obvykle editor, který je součásti sytému (EDIT pod MS DOS, NOTEPAD pod MS Windows)

[61] pomocný výpis proměnných, přepis laděné funkce do několika kratších skriptů, využití globálních proměnných atd.

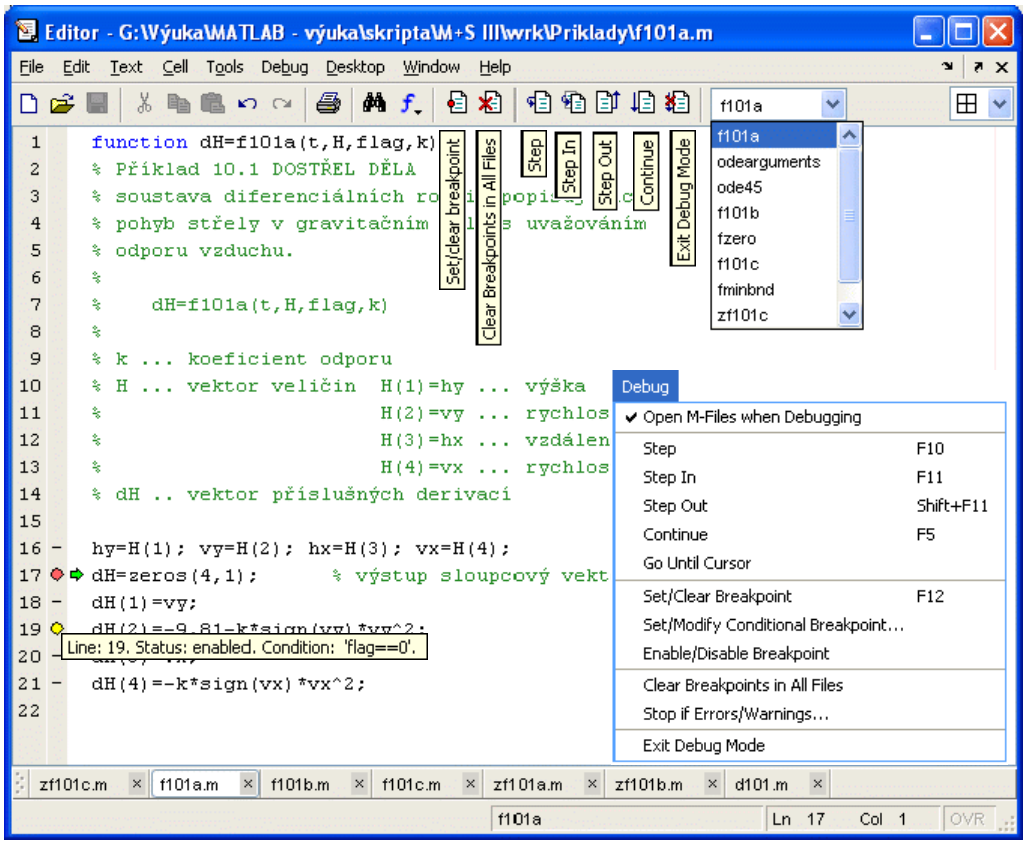
[62] debugger je speciální prostředek usnadňující ladění vytvořených skriptů a funkcí

možnost rozšířena a je možné pracovat v tzv. ***Cell Mode***. Do tohoto režimu je nutno editor přepnout (nabídka hlavního menu *Cell->Enable Cell Mode*). Nyní je možné nastavit začátky oblastí (*Cell*, začátek označen %%) např. pomocí druhého tlačítka zprava nástrojové lišty ***Cell Toolbar***. Potom lze nechat provést všechny příkazy oblasti, ve které je kurzor, po stisku klávesové zkratky Ctrl+ Enter. Pokud je kurzor nad příkazem nastavení hodnoty proměnné je možné stiskem příslušného tlačítka na tlačítkové liště *Cell Toolbar* hodnotu měnit (přičítání, odečítání, násobení či dělení nastavenou konstantou) a celou oblast vyhodnotit. Ukázková situace je na obrázku 7.1.



obrázek 7-2 Editor - Cell Mode

Užitečné je současné využití editoru pro účely ladění - **debugger**. Na obrázku 7-2 je okno editoru/ debuggeru s m-funkcí popisující soustavu diferenciálních rovnic z příkladu 10.1. Vykonávání funkce je zastaveno po vykonání příkazu na řádku 17. Do obrázku je doplněn rozvoj položky hlavního menu ***Debug***, rozvoj seznamu ***Stack*** (ukazuje posloupnost volání funkcí ze základního skriptu), výpis informace o podmíněném bodu zastavení a význam ikon na nástrojové liště týkajících ovládání debuggeru.



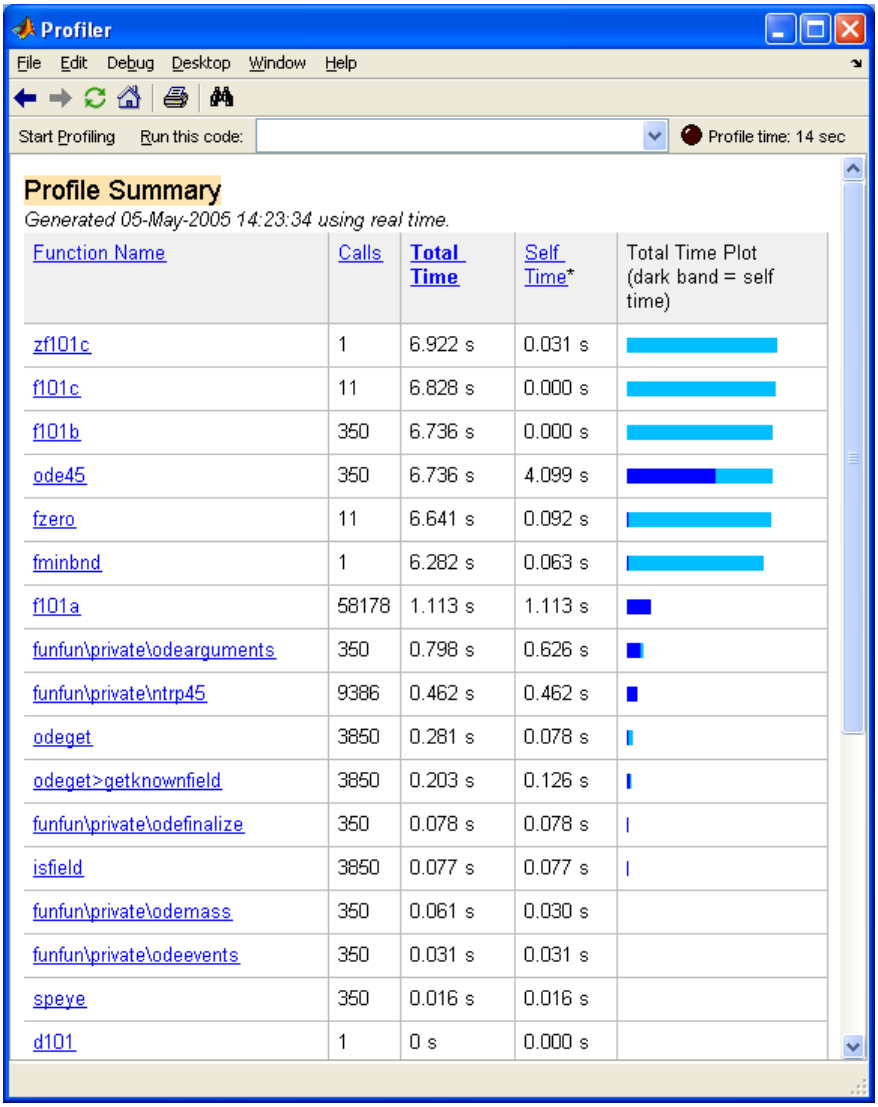
obrázek 7-1 Editor a ladění programu

Pro ladění jsou poskytovány základní funkce debuggeru:

- indikaci příkazu, který má být vykonán (zelená šipka)
- umístění a odebrání bodu zastavení (klávesa F12) - breakpoint (velká červená tečka)
- vytvoření a odebrání podmíněného bodu zastavení – conditional breakpoint (velká žlutá tečka)
- řízení běhu při ladění
  - vykonání jednoho příkazu (klávesa F10)
  - vnoření se do příkazu obsahujícího volání skriptu či funkce (klávesa F11)
  - vykonání příkazů do místa kde je kurzor
  - pokračování ve vykonávání příkazů
- indikaci úrovně vnoření[63] (řádek v záhlaví *Stack*)
- možnost zastavení provádění příkazů při výskytu chyby, varování či výsledku *NaN* a *Inf*

Ladění lze provádět též přímo z prostředí MATLABu pomocí příslušných příkazů (**`dbcont`**, **`dbstep`**, **`dbclear`**, **`dbtype`**, **`dbstack`**, **`dbup`**, **`dbdown`**, **`dbstatus`**, **`dbquit`**). Při ladění souborů (m-funkcí) umístěných ve složkách, které obsahují dlouhé názvy či znaky s diakritikou (čeština) byly ve starších verzích MATLABu pod českými Windows problémy. Od verze 6.1 již problémy s používáním českých znaků v názvech souborů nejsou.

Kromě možnosti ladění poskytuje MATLAB i prostředky pro vyhledání příkazů a funkcí, kde se stráví při průběhu výpočtu nejvíce času. Jde o tzv. **profiler**. Tato možnost je užitečná zejména při optimalizaci řešení z časového hlediska. Požadavek na zaznamenávání se provede příkazem **`profile on`**, pak se spustí sledovaná funkce či skript a ukončení zaznamenávání se provede příkazem **`profile off`**. Příkaz **`profile report`** vygeneruje protokol v HTML formátu a pomocí prohlížeče ho zobrazí. Zobrazí se celkový přehled a detailní informace o



obrázek 7-3 Profiler

[63] na obrázku 7-2 je tato záložka rozvinuta a je vidět postup vnoření (volání) jednotlivých funkcí. Ze základní úrovně (Base, je ve skryté části seznamu) je volán skript ***zf101c***, který volá funkci MATLABu **`fminbnd`** hledající minimum funkce ***f101c***. Tato používá funkci MATLABu **`fzero`**, která hledá kořen funkce ***f101b***. Funkce f101b využívá řešitele soustavy diferenciálních rovnic – funkci MATLABu **`ode45`**, která využívá funkci **`odearguments`**. Vlastní soustava diferenciálních rovnic je popsána funkcí ***f101a***, ve které je umístěn bod zastavení.

jednotlivých funkcích. Na obrázku 7-3 je ukázka části okna profileru s informací o provádění skriptu *zf101c.m*, který řeší příklad 10.1. Při interpretaci výsledků je potřeba rozlišovat čas strávený ve funkci či skriptu (Total Time) a čas strávený vykonáváním příkazů funkce (Self Time) Je vidět že např. funkce *f101a.m* (popis soustavy diferenciálních rovnic) je volána 58178 krát a celkově se při jejím vykonávání spotřebovalo 38.4 % (tj. 0.75/6.888*100) celkového času. V detailní informaci o jednotlivých funkcích lze zjistit i kolikrát se vykonávaly a kolik času se spotřebovalo na jednotlivé řádky. Je také vidět kolik pomocných funkcí je voláno standardními funkcemi – zejména funkcí ode45. Je to daň za univerzálnost příslušných funkcí

7.5 Seznam příkazů a funkcí pro tvorbu skriptů a funkcí

Příkazy a funkce využívané při tvorbě skriptů a funkcí jsou v adresáři matlab\lang a jejich seznam lze získat příkazem `help lang`

```
>> help lang
Programming language constructs.

  Control flow.
if      - Conditionally execute
statements.
else    - Execute statement if previous
IF condition failed.
elseif  - Execute if previous IF
failed and condition is true.
end     - Terminate scope of control
statements.
for     - Repeat statements a specific
number of times.
while   - Repeat statements an
indefinite number of times.
break   - Terminate execution of WHILE
or FOR loop.
continue - Pass control to the next
iteration of a loop.
switch  - Switch among several cases
based on expression.
case    - SWITCH statement case.
otherwise - Default SWITCH statement
case.
try     - Begin TRY block.
catch   - Begin CATCH block.
return  - Return to invoking function.
error   - Display message and abort
function.
rethrow - Reissue error.

  Evaluation and execution.
eval    - Execute string with MATLAB
expression.
evalc   - Evaluate MATLAB expression
with capture.
feval   - Execute the specified
function.
evalin  - Evaluate expression in
workspace.
builtin - Execute built-in function
from overloaded method.
assignin - Assign variable in
workspace.
run     - Run script.

  Scripts, functions, and variables.
script  - About MATLAB scripts and M-
files.
function - Add new function.
global  - Define global variable.
persistent - Define persistent
variable.
mfilename - Name of currently
executing M-file.
lists   - Comma separated lists.
exist   - Check if variables or
functions are defined.
mlock   - Prevent M-file from being
cleared.
munlock - Allow M-file to be cleared.
mislocked - True if M-file cannot be
cleared.
precedence - Operator Precedence in
MATLAB.
isvarname - True for valid variable
name.
iskeyword - Check if input is a
keyword.
javachk - Validate level of Java
support.
genvarname - Construct a valid MATLAB
variable name from a string.

  Argument handling.
nargchk - Validate number of input
arguments.
nargoutchk - Validate number of
output arguments.
nargin  - Number of function input
arguments.
nargout - Number of function output
arguments.
varargin - Variable length input
argument list.
varargout - Variable length output
argument list.
inputname - Input argument name.
```

Message display.
warning - Display warning message.
lasterr - Last error message.
lasterror - Last error message and related information.
lastwarn - Last warning message.
disp - Display array.
display - Display array.
intwarning - Controls the state of the 4 integer warnings.

Interactive input.
input - Prompt for user input.
keyboard - Invoke keyboard from M-file.

See also general, iofun, ops, datatypes, matfun, funfun, elfun, polyfun.

8. Funkce funkcí

Příkazy a funkce: `fminsearch, fzero, ode45, quad, quadl`

Problém: kořen funkce, numerická integrace, minimum funkce více proměnných, řešení soustavy diferenciálních rovnic

Při řešení inženýrských problémů velmi často narazíme na úlohy typu vyhledat minimum funkce či zjistit určitý integrál funkce nebo nalézt numerické řešení diferenciální rovnice.

MATLAB podporuje řešení těchto problémů:

a) nalezení nulové hodnoty funkce jedné proměnné
b) nalezení minima funkce jedné či více proměnných
c) nalezení hodnoty určitého integrálu jedné proměnné, dvou a tří proměnných
d) řešení soustavy diferenciálních rovnic

Všechny tyto matematické úlohy mají společné to, že příslušné operace je potřeba vykonávat s analyticky zadanými vztahy[64] a tyto vztahy lze zapsat jediným formálně stejným (třeba maticovým) zápisem.

Obecný postup přístup MATLABu je v těchto případech ten, že příslušné analytické vztahy se vyjádří uživatelskou m-funkcí a použije se příslušná standardní funkce (ať už built-in či m-funkce) MATLABu na řešení daného problému. Tyto standardní funkce mají jako první parametr jméno nově vytvořené m-funkce. Další parametry jsou dány syntaxí příslušné funkce. Podmínkou pro používání těchto standardních funkcí je tedy znalost tvorby m-funkcí.

8.1 Vyhledání nulové hodnoty funkce

Potřebujeme-li nalézt takovou hodnotu nezávisle proměnné nějaké funkce[65], pro kterou je funkční hodnota nula, pak je to úkol pro standardní funkci **`fzero`** (hledání kořene funkce). Jde tedy o úlohu $f(x) = 0$

Upozorňuji, že tato funkce nevyhledá všechny kořeny funkce. Jde o numerické vyhledání nějakého kořene v okolí výchozí hodnoty nezávisle proměnné[66]. Geometrický význam je průsečík funkce s osu x. Základní syntaxe funkce `fzero` je následující

hodnota_korene = **fzero**(´nazev_funkce´, pocatecni_hodnota_x),

kde *nazev_funkce* je jméno m-funkce, která definuje funkční závislost jejíž kořen hledáme. Zadává se buď jméno m-funkce (text) v apostrofech nebo jméno proměnné typu řetězec obsahující název m-funkce (název souboru na disku v některém z adresářů, které jsou nastaveny v cestách MATLABu). Tato m-funkce musí být napsána tak, aby pro zadanou hodnotu nezávisle proměnné vracela funkční hodnotu. Konstanta či proměnná *pocatecni_hodnota_x* představuje hodnotu nezávisle proměnné, v okolí které se vyhledává kořen. Návratová hodnota funkce `fzero` představuje hodnotu kořene určenou s určitou přesností. Bližší informace viz příkaz `help fzero`.

Ukažme si použití této funkce na řešení tří příkladů. Zároveň ukážeme, že pouze výkonný nástroj nestačí. I u takto jednoduchých příkladů je potřeba vědět co děláme a přemýšlet nad výsledkem. Mějme tyto tři rovnice

$$a)\ xe^x = 1 \quad b)\ x^2 + x + 1 = 0 \quad c)\ \sin(x) = x/10$$

Všechny rovnice nejprve přepíšeme do tvaru f(x)=0 a tento tvar zapíšeme jako m-funkce MATLABu pod názvy *fce_a*, *fce_b* a *fce_c*. Tyto m-funkce mohou být zapsány třeba tak, jak je uvedeno v rámečku. Nyní můžeme již úlohy řešit.

---

[64] řešení je samozřejmě numerické. Pro analytické řešení (existuje-li) je nutný Symbolic Math Toolbox

[65] speciálně jde-li o polynom, pak se všechny kořeny získají pomocí funkce **`roots`**

[66] pouze v oboru reálných čísel

Příkaz

```
>> k_a=fzero('fce_a',0)
k_a = 0.9128
```

nalezne kořen *fce_a*. Zkusme nyní stejným způsobem nalézt kořen *fce_ b*

```
>> k_b=fzero ('fce_b',0)
Exiting fzero: aborting search for an interval containing a sign change because NaN or Inf function value encountered during search (Function value at -1.716199e+154 is Inf) Check function or try again with a different starting value.
k_b = NaN
```

V tomto případě dostaneme poněkud neočekávaný výsledek. Co se stalo ? Chtěli jsme po MATLABu, aby řešil příklad, který v rámci předpokladů nemá řešení. V oboru reálných čísel kořen *fce_b* neexistuje - jde o kvadratickou rovnici s dvojicí komplexně sdružených kořenů. O tomto se lehce přesvědčíme např. pomocí nám již známé funkce **roots.**

```
function y=fce_a(x)
% uzivatel.fce pro 'Uvod do pouzivani MATLAB'
% y=x*exp(x/10)-1
%                        y=fce_a(x)
%

y=x.*exp(x/10)-1;    %nasobeni prvek po prvku
```

```
function y=fce_b(x)
% uzivatel.fce pro 'Uvod do pouzivani MATLAB'
% y=x^2+x+1
%                        y=fce_b(x)
%

y=x.^2+x+1;          %nasobeni prvek po prvku
```

```
function y=fce_c(x)
% uzivatel.fce pro 'Uvod do pouzivani MATLAB'
% y=sin(x)-x/10
%                        y=fce_c(x)
%

y=sin(x)-x/10;
```

```
>> roots([1 1 1])                %vyhledání kořenů polynomu x^2+x+1
ans = -0.5000 + 0.8660i
      -0.5000 - 0.8660i
```

Třetí příklad, *fce_c*, je, pokud nad ním nepřemýšlíme, nejzrádnější. Použijme opět známý příkaz

```
>> k_c=fzero('fce_c',0)
k_c = 0
```

a dostaneme správný výsledek. Avšak pokud použijeme např. poněkud modifikovaný příkaz

```
>> k_c=fzero('fce_c',10)
k_c = 8.4232
```

dostaneme výsledek také správný, ale zcela jiný. Zde je potřeba mít na mysli, že nelineární funkce mohou mít větší počet kořenů a ne vždy je jednoduché určit kolik jich je. Při určení odhadu počtu a polohy kořenů nám může pomoci MATLAB. Stačí určit interval, kde se kořeny mohou vyskytovat a na tomto intervalu si nechat průběh funkce vykreslit. Z grafu určíme počet a odhadneme polohu kořenů. Změnou měřítka můžeme touto metodou určit polohu vybraného kořene s požadovanou přesností. Pro případ funkce c je oblast, kde se mohou vyskytovat kořeny dána intervalem (-10,10) (rozmyslete si proč). Vykresleme průběh funkce c na tomto intervalu (horní graf v obrázku 8-1).

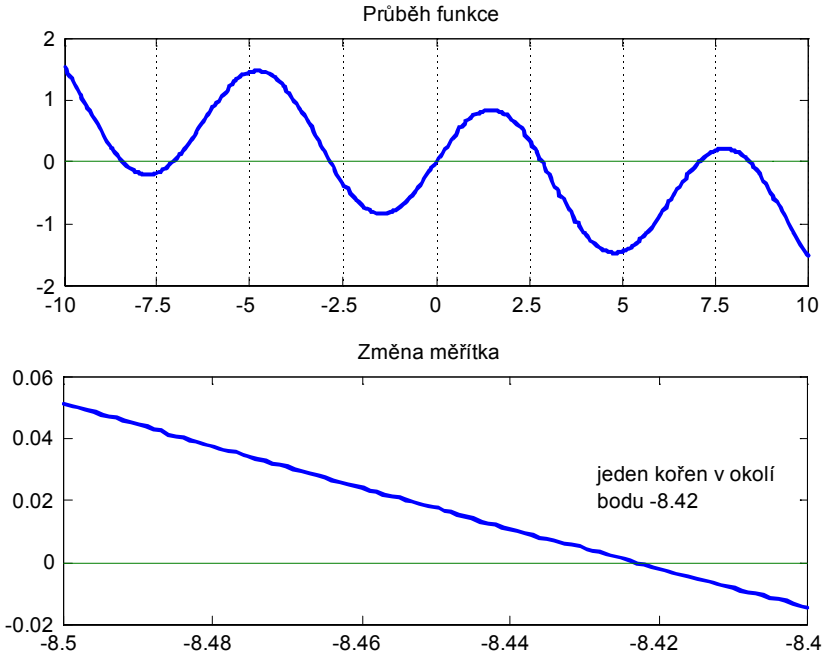
```
>> x=-10:0.05:10; subplot(211), plot(x,fce_c(x),x,zeros(size(x)))
>> x=-8.5:0.001:-8.4; subplot(212), plot(x,fce_c(x),x,zeros(size(x)))
```

Vidíme, že funkce má 7 kořenů. Chceme-li určit graficky odhad polohy kořene s nejmenší hodnotou tj. kořene okolo hodnoty -8.5 můžeme toho docílit např. příkazem pro vykreslení na vybraném intervalu.

Přesné hodnoty všech 7 kořenů získáme např. sadou příkazů

```
>> x1=fzero('fce_c',-10)
x1 = -8.4232
```

```
>> x2=fzero('fce_c',-6)
x2 = -7.0682
>> x3=fzero('fce_c',-4)
x3 = -2.8523
>> x4=fzero('fce_c',0)
x4 =  0
>> x5=fzero('fce_c',3)
x5 =  2.8523
>> x6=fzero('fce_c',7)
x6 =  7.0682
>> x7=fzero('fce_c',10)
x7 =  8.4232
```



obrázek 8-1 Grafický odhad kořene funkce

8.2 Numerická integrace

MATLAB podporuje numerickou integraci funkcí jedné proměnné tj. $I = \int_a^b f(t)dt$ dvěmi standardními funkcemi **quad** a **quadl**[67]. Funkce se liší použitým algoritmem numerické integrace. První z nich používá adaptivní Simpsonovo pravidlo a druhá adaptivní Lobattovo pravidlo. Základní syntaxe funkce `quad` (`quadl`) je následující

hodnota_integralu = **quad**(´nazev_funkce´,dolni_mez,horni_mez)

kde *nazev_funkce* je jméno m-funkce popisující závislost integrandu na integrační proměnné. Zadává se buď jméno m-funkce (text) v apostrofech nebo jméno proměnné typu řetězec obsahující název m-funkce. Tato m-funkce musí být napsána tak, aby pro zadaný vektor hodnot nezávisle proměnné vracela vektor funkčních hodnot. Následující dva parametry představují integrační meze.

Numerickou integraci funkce si ukážeme na dvou příkladech. Mějme křivku v prostoru (spirálu použitou v kapitole 6.2 pro dokumentaci třírozměrných grafů) a máme určit její délku. Přesné zadání následuje.

parametricky zadaná prostorová funkce

$x = t\sin(t) \quad y = t\cos(t) \quad z = t \quad t \in \langle 0, 10\pi \rangle$

délka obecné parametricky zadané funkce a konkrétní funkce

$$l = \int_{t_1}^{t_2} \sqrt{\left(dx/dt\right)^2 + \left(dy/dt\right)^2 + \left(dz/dt\right)^2}\, dx \qquad l = \int_0^{10\pi} \sqrt{\left(\sin(t) + t\cos(t)\right)^2 + \left(\cos(t) - t\sin(t)\right)^2 + 1}\, dt$$

Pro řešení je potřeba funkci, jejíž integrál hledáme, zapsat ve formě m-funkce např. s názvem *fce_i1* (viz rámeček). Vlastní výpočet integrálu realizujeme příkazem

```
>> l=quad('fce_i1',0,100*pi)
l =  4.9354618481803 90e+004
```

případně s využitím jiného algoritmu.

```
>> ll=quadl('fce_i1',0,100*pi)
ll =  4.9354618481641 93e+004
```

```
function y=fce_i1(x)
% uzivatelska funkce pro 'Uvod do ...'
% y=sqrt((sin(x)+x*cos(x))^2+(cos(x)-x*sin(x))^2+1)
%
%                    y=fce_i1(x)
%

y=sqrt((sin(x)+x.*cos(x)).^2+(cos(x)-
x.*sin(x)).^2+1);
```

[67] funkce `quadl` nahradila dříve používanou funkci `quad8`. Ta existuje dále ale je doporučeno kvůli výhodnějším numerickým vlastnostem funkci `quadl`.

Pro druhý příklad zvolíme trochu složitější případ. Mějme funkci y(x) a bude nás zajímat průběh funkce y(x) a integrálu funkce y(x) na daném intervalu.

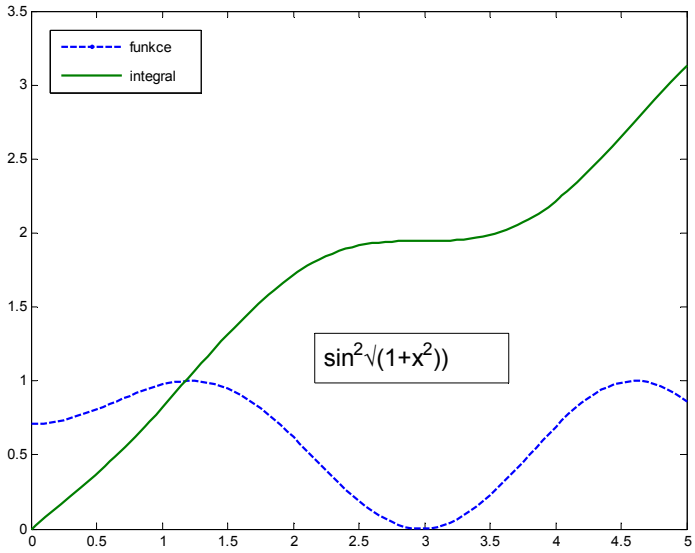
$$y(x) = \sin^2\sqrt{1+x^2}$$

$$iy(x) = \int_0^x y(t)dt \quad x \in \langle 0,5 \rangle$$

M-funkce *fce_i2* může vypadat např. jak je uvedeno v rámečku. Vlastní příkazy pro získání grafu obou funkcí jsou nyní trochu složitější. Průběh funkce y se spočítá jedním příkazem. V cyklu se pak počítá integrál pro různé hodnoty horní meze. Na závěr se oba průběhy vynesou do grafu (obrázek 8-2).

```
>> x=0:0.05:5;
>> y=fce_i2(x);
>> for a=1:length(x),
iy(a)=quadl('fce_i2',0,x(a));end
>> plot(x,y,x,iy)
>>legend('funkce','integrál')
```

```
function y=fce_i2(x)
% uzivatel.fce pro 'Uvod do pouzivani MATLAB'
% y=(sin(sqrt(1+x^2))^2
%
%                  y=fce_i2(x)
%

y=(sin(sqrt(1+x.^2))).^2;
```



obrázek 8-2 Integrál jako funkce horní meze

Použití dvojného integrálu tj. $I = \iint_O f(x,y)dxdy$ ukažme na příkladu určení objemu koule o jednotkovém poloměru. MATLAB poskytuje pro řešení dvojného integrálu funkci **dblquad**, která ale předpokládá pravoúhlou oblast integrace. Ukažme tedy zároveň jak využít možností MATLABu pro řešení tohoto problému. Horní polokoule (se středem v počátku souřadnic) je ohraničena plochou $z = \sqrt{1-x^2-y^2}$ a objem celé koule je tedy dvojnásobkem a určen hodnotou integrálu

$$V = \iiint_{V(x,y,z)} dxdydz = \iint_{V(x,x)} z(x,y)dxdy = 2\int_{-1}^{1}\left(\int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \sqrt{1-x^2-y^2}\,dy\right)dx$$

V MATLABu ale budeme řešit modifikovaný vztah $V = 2\int_{-1}^{1}\left(\int_{-1}^{1}\sqrt{f\left(1-x^2-y^2\right)}dy\right)dx$, kde funkce f(x,y) „vybírá“ pouze přípustné hodnoty (jinak vrací hodnotu 0). Zapišme m-funkci (viz rámeček), kde pomocí standardní funkce `max` vybereme přípustnou hodnotu. Vlastní řešení je pak triviální.

```
>> V1=dblquad('fce_i3',-1,1,-1,1)*2
V1 =    4.1888
>> V2=4/3*pi
V2 =    4.1888
```

```
function Integrand=fce_i3(x,y)
% pro výpočet objemu koule dvojným
integrálem
%
%   Integrand=fce_i3(x,y)
%

z=R^2-x.^2-y.^2;     % výpočet z
% pouze kladné hodnoty, jinak 0
Integrand=sqrt(max(z,0));
```

8.3 Minimum funkce více proměnných

Jako velice mocná funkce se ukazuje funkce vyhledání minima[68] funkce více proměnných **`fminsearch`**. Jde o hledání minima funkce více proměnných na neohraničeném intervalu $\min_{\mathbf{x}} f(\mathbf{x})$ metodou Nelder-Mead. MATLAB ještě obsahuje funkci **`fminbnd`**, která hledá minimum funkce pouze jedné proměnné, ale na zadaném intervalu. Další funkce vyhledávají extrémy některých speciálních funkcí a používající různé metody jsou v samostatném toolboxu (Optimization Toolbox).

Žádná z funkcí samozřejmě není schopná řešit problém globálního minima ani formulovat řešení ze slovního zadání. Avšak jsou velmi užitečné pro častou úlohu prokládání naměřených dat funkcí nelineární v parametrech – nelineární regresi.

Základní syntaxe funkce `fminsearch`[69] je následující

vekt_x = **fminsearch**(´nazev_funkce´,vekt_x0)

kde *nazev_funkce* je jméno m-funkce popisující funkční závislost jejíž minimum hledáme. Zadává se buď jméno m-funkce (text) v apostrofech nebo jméno proměnné typu řetězec obsahující název m-funkce. Tato m-funkce musí být napsána tak, aby pro zadaný vektor hodnot nezávisle proměnných vracela funkční hodnotu. Parametr *vekt_x0* je startovací bod v okolí kterého se vyhledává lokální minimum. Vracený vektor *vekt_x* obsahuje souřadnice vyhledaného lokálního minima. Bližší informace viz příkaz `help fminsearch`.

Ukažme si opět dva příklady na použití této funkce. Nejprve řešme standardní úlohu - nalezení minima funkce dvou proměnných. Dvě proměnné jsou zvoleny proto, aby bylo možné příslušnou plochu vykreslit a získat představu o poloze minim. Mějme tedy funkci definovanou jako

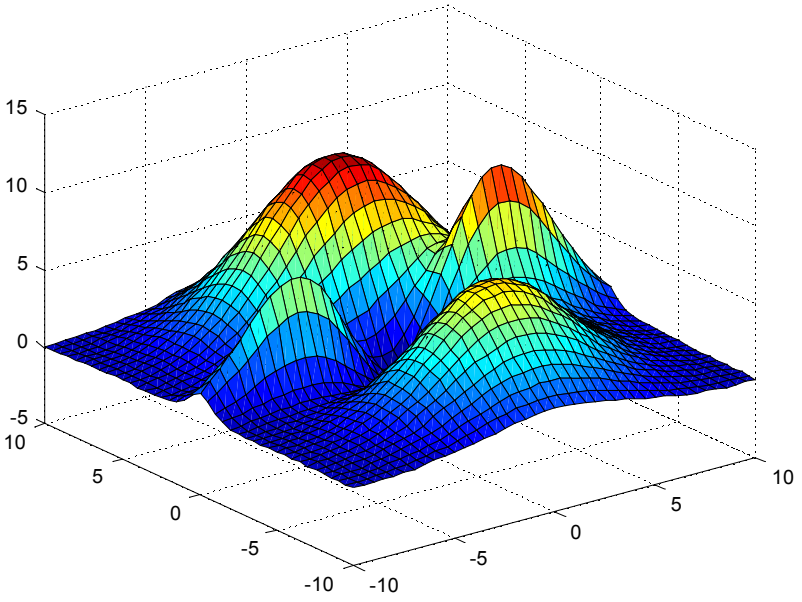
$$z(x,y)=\left(x^2+x+y+\frac{y^2}{1+x^2}\right)e^{-\frac{x^2+y^2}{25}}$$

$$x\in\langle -10,10\rangle \quad y\in\langle -10,10\rangle$$

Tuto funkci přepíšeme do tvaru m-funkce s názvem *fce_m1* např. jak je uvedeno v rámečku. Jak tato funkce vypadá zjistíme po vykreslení např. následujícími příkazy (viz obr. 8-4)

```
>> x=-10:0.5:10; y=x;
>> n=length(x);
>> for a=1:n,
for b=1:n,
Z(a,b)=fce_m1([x(a),y(b)]);
end
end
>> surf(x,y,Z)
```

```
function z=fce_m1(par)
%uzivatel.fce pro 'Uvod do pouzivani MATLAB'
%z=(x^2+x+y+y^2/(1+x^2))*exp(-(x^2+y^2)/25)
%
%              z=fce_m1(par)
%

x=par(1); y=par(2);
z=(x.^2+x+y+y.^2./(1+x.^2)).*exp(-(x.^2+y.^2)/25);
```



obrázek 8-3 Funkce dvou proměnných

Zdá se, že minimum plochy je někde v okolí bodu [0,0]. Zkusme vyhledat přesnou polohu a hodnotu minima příkazy

[68] maximum funkce se jednoduše hledá jako minimum záporně vzaté původní funkce

[69] do verze 5.3 včetně byla používána funkce s názvem `fmins`

```
>> sm=fminsearch('fce_m1',[0,0])
sm = -0.6339   -0.6787
>> hm=fce_m1(sm)
hm = -0.5624
```

Pokud by nás zajímala maxima, pak bychom museli otočit znaménko u výpočtu proměnné z v m-funkci *fce_m1* (vznikne nová m-funkce - např. *fce_m2*) a zkusit různé startovací hodnoty, abychom vyhledali všechny extrémy funkce

```
>> souradnice_maxima_1=fminsearch('fce_m2',[0,1])
souradnice_maxima_1 = 0.0218    4.7767
>> hodnota_maxima_1=fce_m2(souradnice_maxima_1)
hodnota_maxima_1 =  -11.0818                    % lokální extrém
>> souradnice_maxima_2=fminsearch('fce_m2',[0,-1])
souradnice_maxima_2 = 0.0180   -5.2819
>> hodnota_maxima_2=fce_m2(souradnice_maxima_2)
hodnota_maxima_2 =   -7.4123                    % lokální extrém
>> souradnice_maxima_3=fminsearch('fce_m2',[1,0])
souradnice_maxima_3 = 4.7345    0.4706
>> hodnota_maxima_3=fce_m2(souradnice_maxima_3)
hodnota_maxima_3 =  -11.1722                    % globální extrém
>> souradnice_maxima_4=fminsearch('fce_m2',[-1,0])
souradnice_maxima_4 =-5.2177    0.5759
>> hodnota_maxima_4=fce_m2(souradnice_maxima_4)
hodnota_maxima_4 =   -7.5042                    % lokální extrém
```

Jako druhý příklad na hledání minima funkce si ukážeme prokládání daných bodů libovolnou funkcí. V případě, že je funkce lineární v parametrech jde o relativně jednoduchý problém lineární regrese vedoucí na soustavu lineárních rovnic. V případě, že funkce lineární v parametrech není, jde o problém nelineární regrese vedoucí na optimalizační úlohy.

Pro jednoduchost vygenerujeme "naměřené" body analytickou funkcí a jako kritérium shody původních bodů a bodů získaných aproximační funkcí použijeme kritérium součtu absolutních hodnot odchylek. Zadání "generující funkce" a tvaru aproximační funkce následuje

$$\textit{původní funkce}: \quad y = \sin(x) \quad x \in \langle 0, \pi \rangle$$

$$\textit{aproximační funkce}: y = a(x-b)e^{c(x-d)^2}$$

Nejprve je nutné vytvořit m-funkci, která bude mít minimum pro takovou sadu parametrů *a-d*, která splňuje naše kriterium. Navíc tato m-funkce musí splňovat formální požadavky na tvar funkce použitelný pro standardní funkci `fminsearch`. Tj. první návratový parametr musí být skalár a jeho hodnota je minimalizována měněním hodnot vektoru parametrů, který musí být uveden jako první vstupní parametr m-funkce. Naším požadavkem je zjištění součtu odchylek od "naměřených" hodnot. Proto musí v m-funkci být k dispozici vektor naměřených hodnot. Uvedeme ho jako třetí vstupní parametr. Pro vykreslení průběhu aproximující funkce budeme potřebovat vektor aproximujících hodnot. Tento vektor je také potřeba pro určení odchylky – musí se počítat uvnitř funkce. Stačí ho pouze "zveřejnit" mimo funkci prostřednictvím druhého výstupního parametru. Potom m-funkce (s jménem *fce_m3*) splňující tyto požadavky může vypadat např. jak je uvedeno v rámečku

```
function [krit,ya]=fce_m3(par,x,y)
% uzivatel.fce pro 'Uvod do pouzivani MATLAB'
% aproximace bodu [x,y] funkci ya(x)
% ya=a*(x+b)*exp(c*(x-d)^2)
%
%            [krit,ya]=fce_m3(par,x,y)
%

a=par(1); b=par(2); c=par(3); d=par(4);

% prubeh aprox.fce pro aktualni sadu parametru
ya=a*(x+b).*exp(c*(x-d).^2);
% vypocet kriteria shody (minimalizuje se)
krit=sum(abs(ya-y));
```

Nyní můžeme vygenerovat zkušební data

```
>> x=0:0.01:pi; y=sin(x);
```

dále je vhodné připravit si počáteční vektor parametrů s "rozumnými" hodnotami a odzkoušet zda uživatelská funkce *fce_m3* dává smysluplný výsledek

```
>> p0=[0,0,0,0]; [kr,ya]=fce_m3(p0,x,y); kr
kr =  199.9990
```

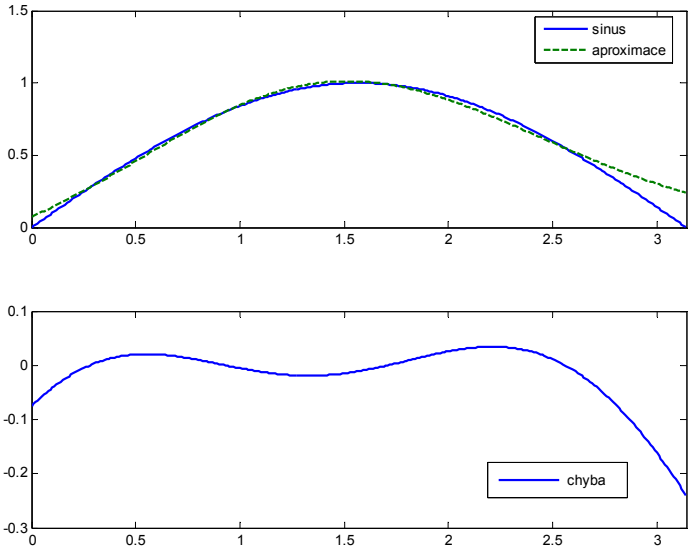
Protože naše uživatelská funkce obsahuje více než jeden vstupní parametr, musíme informovat funkci `fminsearch`, že je potřeba dodat další parametry pro výpočet *fce_m3*. Po prostudování helpu zjistíme, že to lze zajistit uvedením dalších parametrů ve funkci `fminsearch`**.** Třetí parametr představuje vektor parametrů algoritmu - implicitní hodnoty jsou použity při zadání prázdného vektoru. Následují parametry, které je potřeba přenést do uživatelské funkce.

```
>> p=fminsearch('fce_m3',p0,[],x,y)            % start z hodnot p0
p =    1.8021   -0.0763   -0.1414   -1.1603
>> p=fminsearch('fce_m3',p,[],x,y)             % start z minulého řešení
p =    0.7582    0.1331   -0.4321    0.8180
>> [kr,ya]=fce_m3(p,x,y); kr
kr =  10.6205
```

Výsledek aproximace jedné půlperiody funkce sinus danou funkcí je možné ukázat také graficky. Je zřejmé, že v tomto případě tvar aproximační funkce není vhodný - není schopný sledovat základní průběh aproximované funkce. Přesto vidíme, že není problém nalézt optimální (tj. minimalizující kritérium) hodnoty parametrů aproximující funkce.

```
>> subplot(211)
>> plot(x,y,x,ya)
>> subplot(212)
>> plot(x,y-ya)
```

Legenda byla doplněna editací výsledného grafu na obrázku 8-4.



obrázek 8-4 Určení parametrů funkce - nelineární regrese

8.4 Řešení soustavy diferenciálních rovnic

Poslední standardní funkcí, kterou se budeme v této kapitole zabývat, je funkce pro řešení soustavy obyčejných diferenciálních rovnic prvního řádu. (Ordinary Differential Equations). Pokud si kladete otázku, jak se budou řešit diferenciální rovnice vyšších řádů, pak je potřeba si uvědomit, že diferenciální rovnice vyššího řádu než prvního lze převést na ekvivalentní soustavu diferenciálních rovnic řádu prvního. $\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y})$

Vrátíme-li se k MATLABu, pak tento poskytuje několik standardních funkcí (řešitelů) pro řešení soustavy obyčejných diferenciálních rovnic. Základní standardní funkcí je funkce `ode45` pro nonstiff[70] rovnice nebo `ode15s` pro stiff rovnice. Základní syntaxe funkce **ode45** je následující

[T,Y] = **ode45**(´nazev_funkce´,casovy_interval,pocatecni_podminky)

kde *nazev_funkce* je jméno m-funkce popisující soustavu diferenciálních rovnic. Zadává se buď jméno m-funkce v apostrofech nebo jméno proměnné typu řetězec obsahující název m-funkce. Tato m-funkce musí mít syntaxi

[70] V souvislosti s numerickým řešením diferenciálních rovnic je potřeba upozornit na problém stiff systémů. Jde velmi jednoduše řečeno o diferenciální rovnice, u kterých řešení obsahuje zároveň rychle i pomalu se v čase měnící složky řešení. V případě lineárních diferenciálních rovnic to je např. v případě, že charakteristická rovnice má kořeny, jejichž absolutní hodnoty se velmi liší ve velikosti. Praktický důsledek tohoto jevu se při řešení pomocí standardního algoritmu, který s tímto nepočítá, projeví tak, že rychlé jevy nejsou v řešení zachyceny nebo naopak řešení trvá neúměrně dlouho. Proto na stiff problémy je potřeba používat speciální algoritmy.

dY=nazev_funkce(t,Y)

Výstupní sloupcový vektor *dY* představuje hodnoty derivací pro vektor hodnot *Y* v čase *t*. Parametr *casovy_interval* představuje vektor o dvou prvcích - počáteční čas řešení t_0 a konečný čas řešení. Parametr *pocatecni_podminky* představuje vektor počátečních podmínek Y_0 takový, že platí $Y(t_0)=Y_0$.

Výstupní parametry funkce `ode45` jsou dva. *T* - sloupcový vektor- obsahuje časové okamžiky, ve kterých jsou určeny hodnoty řešení *Y* a vlastní řešení uložené v matici *Y*. Počet řádků matice *Y* odpovídá počtu řádků *T*, počet sloupců odpovídá počtu rovnic (veličin *Y*) řešené soustavy.

Ukažme si příklad - řešení Van der Polovy rovnice popsané nelineární diferenciální rovnicí druhého řádu

$$\frac{d^2y_1}{dt^2} - \mu(1-y_1^2)\frac{dy_1}{dt} + y_1 = 0$$

Tuto rovnici druhého řádu lze přepsat na tvar soustavy dvou rovnic řádu prvního

$$\frac{dy_1}{dt} = y_2 \qquad \frac{dy_2}{dt} = \mu(1-y_1^2)y_2 - y_1$$

```
function dy=fce_vdp(t,y)
%uzivatel.fce pro 'Uvod do pouzivani MATLAB'
% Van der Polova diferencialni rovnice
%
%                    dy=fce_vdp(t,y)
%

mi=1;
dy=[y(2);mi*(1-y(1)^2)*y(2)-y(1)];
```

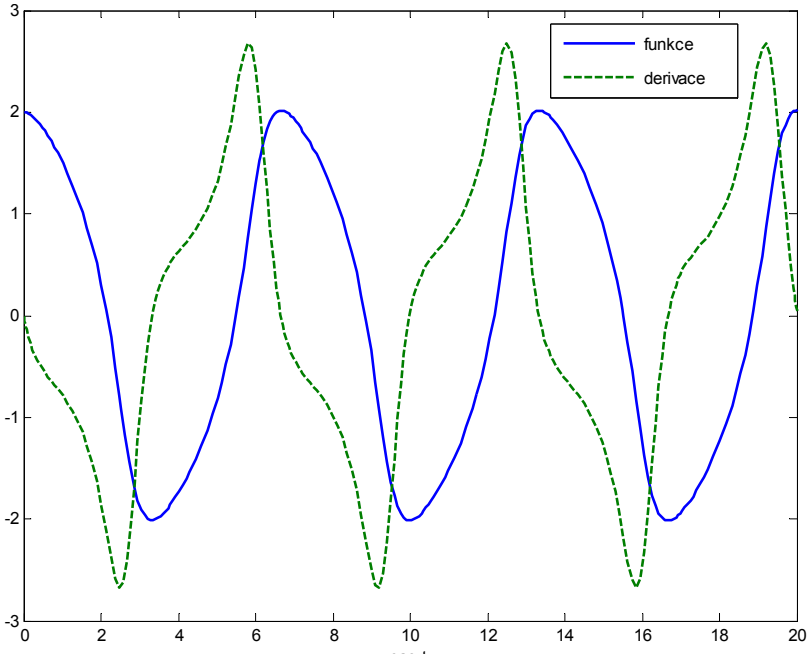
Takovouto soustavu můžeme vyjádřit formálně obecným vektorovým zápisem a v této podobě přepsat do uživatelské funkce *fce_vdp* např. tak jak je uvedeno v rámečku.

$$\frac{d\mathbf{y}}{dt} = \mathbf{F}(t,\mathbf{y}) \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \quad \frac{d\mathbf{y}}{dt} = \begin{bmatrix} dy_1/dt \\ dy_2/dt \end{bmatrix}$$

Určíme-li nyní počáteční podmínky (např. hodnota y_1 v čase 0 bude mít hodnotu 2 a derivace y_2 v čase 0 bude mít hodnotu 0), pak můžeme pro časový interval 0-20 rovnici řešit a vykreslit příkazy

```
>> [t,y]=ode45('fce_vdp',[0,20],[2;0]);
>> plot(t,y(:,1),'-',t,y(:,2),'--')
>> title('Reseni Van der Polovy rovnice, mi=1');
>> xlabel('cas t'), legend('funkce', 'derivace')
```

Poznámka: ve verzi 6 došlo oproti verzi 5 k některým změnám v systému řešitelů diferenciálních rovnic. Základní použití ukázané výše zůstalo beze změny. Změnil se ale význam některých parametrů definovaných příkazem ODESET. Také se změnil význam parametru flag předávaného do uživatelské m-funkce. Tato změna se projeví až v okamžiku, kdy budeme potřebovat do uživatelské m-funkce předávat dodatečné parametry. Pokud budete toto potřebovat, podívejte se do HELPu jak je předání parametrů řešeno ve vaší verzi MATLABu. Způsob používaný ve verzi 7 je ukázán v příkladu v kapitole 11.2.



obrázek 8-5 Průběh řešení Van der Polovy rovnice

8.5 Seznam "funkcí funkcí" a příkazů pro řešení difer. rovnic

```
>> help funfun
Function functions and ODE solvers

  Optimization and root finding.
fminbnd - Scalar bounded nonlinear
function minimization.
fminsearch - Multidimensional
unconstrained nonlinear minimization,
by Nelder-Mead direct search method.
fzero - Scalar nonlinear zero
finding.

  Optimization Option handling
optimset - Create or alter
optimization OPTIONS structure.
optimget - Get optimization
parameters from OPTIONS structure.

  Numerical integration (quadrature).
quad  - Numerically evaluate
integral, low order method.
quadl - Numerically evaluate
integral, higher order method.
quadv - Vectorized QUAD.
dblquad - Numerically evaluate double
integral.
triplequad - Numerically evaluate
triple integral.

  Plotting.
ezplot - Easy to use function
plotter.
ezplot3 - Easy to use 3-D parametric
curve plotter.
ezpolar - Easy to use polar
coordinate plotter.
ezcontour - Easy to use contour
plotter.
ezcontourf - Easy to use filled
contour plotter.
ezmesh - Easy to use 3-D mesh
plotter.
ezmeshc - Easy to use combination
mesh/contour plotter.
ezsurf - Easy to use 3-D colored
surface plotter.
ezsurfc - Easy to use combination
surf/contour plotter.
fplot - Plot function.

  Inline function object.
inline - Construct INLINE function
object.
argnames - Argument names.
formula - Function formula.
char  - Convert INLINE object to
character array.

  Differential equation solvers.
Initial value problem solvers for
ODEs. (If unsure about stiffness, try
ODE45 first, then ODE15S.)
ode45 - Solve non-stiff differential
equations, medium order method.
ode23 - Solve non-stiff differential
equations, low order method.
ode113 - Solve non-stiff differential
equations, variable order method.
ode23t - Solve moderately stiff ODEs
and DAEs Index 1, trapezoidal rule.
ode15s - Solve stiff ODEs and DAEs
Index 1, variable order method.
ode23s - Solve stiff differential
equations, low order method.
ode23tb - Solve stiff differential
equations, low order method.

  Initial value problem solver for
fully implicit ODEs/DAEs F(t,y,y')=0.
decic - Compute consistent intial
conditions.
ode15i - Solve implicit ODEs or DAEs
Index 1

  Initial value problem solver for
delay differential equations (DDEs).
dde23 - Solve delay differential
equations (DDEs) with constant
delays.

  Boundary value problem solver for
ODEs.
bvp4c - Solve two-point boundary
value problems for ODEs by
collocation.

  1D Partial differential equation
solver.
pdepe - Solve initial-boundary value
problems for parabolic-elliptic PDEs.

  Option handling.
odeset - Create/alter ODE OPTIONS
structure.
odeget - Get ODE OPTIONS parameters.
ddeset - Create/alter DDE OPTIONS
structure.
ddeget - Get DDE OPTIONS parameters.
bvpset - Create/alter BVP OPTIONS
structure.
bvpget - Get BVP OPTIONS parameters.

  Input and Output functions.
deval - Evaluates the solution of a
differential equation problem.
```

odextend - Extends the solutions of a differential equation problem.
odeplot - Time series ODE output function.
odephas2 - 2-D phase plane ODE output function.
odephas3 - 3-D phase plane ODE output function.
odeprint - Command window printing ODE output function.
bvpinit - Forms the initial guess for BVP4C.
pdeval - Evaluates by interpolation the solution computed by PDEPE.
odefile - MATLAB v5 ODE file syntax (obsolete).

9. Jednoduché řešené příklady (MATLAB)

V této kapitole jsou uvedeny řešené příklady od nejjednodušších po složitější na procvičení práce s MATLABem. U většiny příkladů existuje více možných postupů řešení, některá více některá méně elegantní či výpočetně efektivní. Rozhodující je to, aby výsledek byl správný. Obecně je preferováno vektorové řešení (je výrazně rychlejší) než programové řešení např. s využitím cyklu `for ... end`. Tento kurz je však úvodem do používání MATLABu, proto není rozhodující elegance, ale správný výsledek.

Kapitola je rozdělena na část se zadáními příkladů a část s řešeními. Toto rozdělení by mělo pomoci těm čtenářům, kteří si chtějí příklady sami spočítat. Přesněji pomoci neusnadnit si práci tím, že se „jen trochu kouknu jaký je výsledek“. Výsledek sice je uveden ale není aspoň hned u zadání.

9.1 Zadání příkladů

9.1.1 Plnění vektorů a matic - zadání

Vytvořte vektory či matice

a) sloupcový vektor *a* obsahující celá čísla 1..10
b) řádkový vektor *b* obsahující celá čísla 10..1
c) matici *A* obsahující dva řádky celých čísel 1..10 a 10..1
d) matici *B* obsahující dva sloupce celých čísel 1..10 a 10..1
e) jednotkovou matici *E* o rozměru 10×10
f) nulovou matici *N* rozměru 5×10
g) matici jednotek *J* o rozměru 10×5
h) matici náhodných čísel *NC* o rozměru 5×7
i) čtvercovou matici DD o rozměru 5×5 s hodnotami 5 na vedlejší diagonále
j) dolní trojúhelníkovou matici DT o rozměru 5×5 naplněnou jednotkami

9.1.2 Základní operace s maticemi, vektory a polynomy - zadání

Vypočtěte (s proměnnými z kapitoly 9.1.1.1)

a) součin vektorů *a*, *b* a *b*, *a*
b) druhou mocninu prvků vektoru *b*
c) součet kvadrátů prvků vektoru *b*
d) součin matic *A*, *B* a *B*, *A*
e) tabulku funkce (vektor) *ys* =sin(*x*) pro $x \in \langle 0,2\pi \rangle$ v 1000 bodech
f) tabulku funkce (vektor) *ys1*=sin(*x*^2)+sin(*x*)^2 pro $x \in \langle 0,2\pi \rangle$ v 1000 bodech
g) tabulku funkce (matici) *YS*=cos(*x***y*) pro *x*, $y \in \langle -\pi,\pi \rangle$ v 100 bodech
h) matici $NCN = NC*NC^T$, její determinant a matici inverzní
i) ukažte, že součin matice *NCN* a její inverze je jednotková matice
j) součet čísel od 1 do 1000
k) součin polynomů $p_1(x) = x^3 - x^2 + x + 1 \quad p_2(x) = x^3 + x^2 + x - 2 \quad p_3(x) = x^3 + x$
l) integrál a derivaci polynomu $p(x) = x^3 + x^2 + x + 1$
m) kořeny polynomu $p(x) = x^3 + x^2 + x + 1$
n) polynom p1 jehož kořeny jsou celá čísla od 1 do 5 a polynom p2 s kořeny 1+i,1+2i a 1+3i

9.1.3 Kreslení funkcí - zadání

Nakreslete průběh funkce

a) *y* = sin(*x*) pro $x \in \langle 0,2\pi \rangle$ v 1000 bodech
b) *y* = sin(*x*^2)+sin(*x*)^2 pro $x \in \langle 0,2\pi \rangle$ v 1000 bodech

c) $y_{\sin}(x)=\sum_{k=1}^{100}\frac{1}{k}\sin(k\times x)$ $y_{\cos}(x)=\sum_{k=1}^{100}\frac{1}{k}\cos(k\times x)$ $x\in\langle 0,4\pi\rangle$ v 1000 bodech. Oba průběhy umístěte do jednoho grafu rozděleného svisle a opatřete titulky. Měřítko nastavte tak, aby odpovídalo přesně velikosti grafů.

d) kružnice, epicykloidy (horní znaménko) a hypocykloidy (dolní znaménko) do jednoho grafu. Poloměr kružnice je R=12 a poloměr kotálející se kružnice je r=4. Křivky vykreslete z 1000 bodů. Graf opatřete legendou a mřížkou. Měřítko obou os nastavte na rozsah ±21.

$$x=(R\pm r)\cos(t)\mp r\cos\left(\frac{R\pm r}{r}t\right)\quad y=(R\pm r)\sin(t)-r\sin\left(\frac{R\pm r}{r}t\right)\quad t\in\langle 0,2\pi\rangle$$

e) $Z = \cos(x*y)$ pro $x,y\in<-\pi,\pi>$ v 50 bodech v každém směru. Graf (drátový model) označte titulkem a popisem os. Editací grafu změňte velikost písma titulku na 16 bodů.

f) dvou hyperbolických paraboloidů $z_1=\frac{x^2}{4}-\frac{y^2}{6}$ $z_2=-\frac{x^2}{6}+\frac{y^2}{4}$ do jednoho grafu. Rozsah v obou osách volte ±21. Graf vykreslete jako souvisle vybarvenou plochu s postupným přechodem šedé (interpolované barvy) a přidejte stupnici přiřazení intenzity barvy k hodnotám na ose z (colorbar)

g) vytvořte vektor deseti náhodných čísel z rozsahu 0-10 a čísla znázorněte ve formě koláčového grafu 2D a 3D. Zobrazte histogram a bar graf. Všechny grafy umístěte do jednoho okna.

9.1.4 Tvorba uživatelských funkcí - zadání

Vytvořte m-funkci, která:

a) vypočte hodnotu funkce $y(x)=\frac{e^x+e^{-x}}{2}$ (parametr x může být i vektor). Uveďte příklad použití (volání vytvořené funkce)

b) vypočte hodnotu funkce $y(x)=x\cos(x)$ a uveďte příklad použití pro vykreslení průběhu této funkce na intervalu $x\in\langle -2\pi,2\pi\rangle$ z 1000 bodů

c) naplní matici hodnot funkce $y(x,y)=\sin(x^2+y^2)$ v bodech daných hodnotami vektoru x a y a uveďte příklad použití pro vykreslení průběhu této funkce (plochy v prostoru) na intervalu $x\in\langle -2\pi,2\pi\rangle$ $y\in\langle -2\pi,2\pi\rangle$ z 100 bodů na každém intervalu

d) určí parametry lineární regrese $y(x)=kx+q$ (směrnici k a posun q přímky prokládající měřená data optimálně z hlediska minima součtu kvadrátu odchylek) z dat zadaných body [x,y] (x a y jsou vektory stejné délky)

9.1.5 Použití uživatelských funkcí - zadání

S využitím vlastních m-funkcí a standardních funkcí vypočtěte

a) nalezněte řešení rovnice $xe^x=10$ (funkce **`fzero`**)

b) nalezněte řešení rovnice $\sin(x)+\frac{x}{10}=0$ (funkce **`fzero`**)

c) vypočtěte integrál od 0 do 10 funkce dle 9.1.5b (funkce **`quadl`**)

d) vypočtěte a nakreslete průběh integrálu od 0 do t funkce dle 9.1.5b pro $t\in<0,10>$ v 1000 bodech (funkce **`quadl`**)

e) řešení diferenciální rovnice $2\frac{dh}{dt}+0.5\sqrt{h}=5\quad h(t=0)=0\quad t\in\langle 0,100\rangle$ (funkce **`ode45`**)

f) řešení diferenciální rovnice $\frac{d^2y}{dx^2} + 0.75\frac{dy}{dx} + 6y = \sin(\sqrt{x})$, $\left.\frac{dy}{dx}\right|_{x=0} = 0, \quad y(0) = 1 \quad x \in <0,10>$

9.1.6 Kombinace více funkcí - zadání

Vyřešte následující příklady

a) nalezněte hodnotu parametru *a*, pro který platí rovnice $\int_{-0.5}^{+0.5} e^{-x^2/a}\,dx = 0.5$ (funkce **quadl** a **fzero**).

b) určete, v kterém čase *t* dosáhne řešení y(t) příkladu 9.1.5e hodnoty y(t)=50 (funkce **ode45** a **fzero**)

c) nalezněte aproximaci horní polokružnice (se středem v počátku souřadnic a poloměrem 10) polynomem 4. řádu s kritériem (funkce **fminsearch**)
- minima sumy kvadrátů odchylek
- minima sumy absolutních hodnot
- minimaxu[71]. Nakreslete průběh chyby (odchylku) pro všechny aproximace

d) nalezněte parametry *a, b, c, d, e* funkce $y = \left(a + bx + cx^2\right)e^{dx+ex^2}$ takové, aby aproximace horní polokružnice (o středu v počátku souřadnic a poloměru 10) touto funkcí byla z hlediska minima maximální odchylky optimální (funkce **fminsearch**). Nakreslete průběh chyby.

9.1.7 Práce se soubory a řetězci - zadání

Vyřešte následující příklady

a) Vytvořte v aktuálním adresáři textový soubor *zdroj.txt* naplněný 100 náhodnými čísly s rovnoměrným rozložením v rozsahu 0 až 100. Každé číslo bude na samostatném řádku. Řešení proveďte jednak s využitím funkce **`save`** a jednak s využitím funkcí pro práci se soubory **`fopen`**, **`fclose`** a **`fprintf`**.

b) Čísla na lichých řádcích ze souboru *zdroj.txt* vynásobte dvěma a uložte do textového souboru *cil01.txt*. Čísla na sudých řádcích vydělte dvěma a uložte do souboru *cil02.txt*. Řešení proveďte jednak s využitím funkcí **`load`** a **`save`** a jednak s využitím funkcí pro práci se soubory **`fopen`**, **`fclose`** , **`fscanf`** a **`fprintf`**.

c) Uložte do souboru s názvem *imagic5.txt* hodnoty inverzní matice k magickému čtverci o rozměrech 5 x 5. Data budou dále zpracovávána v programu, který jako oddělovač desetinné části předpokládá čárku.

d) Mějme soubor s naměřenými experimentálními daty (např. taková jak je uvedeno ve vedlejším rámečku). Soubor obsahuje měřené hodnoty oddělené středníkem. Jako oddělovač desetinné části je použita čárka. Všechna data jsou na jednom řádku. Vždy dvě po sobě jdoucí čísla tvoří jeden bod experimentu. Naším úkolem je vytvořit funkci, která soubor s daty v tomto formátu zpracuje tak, že vytvoří dvousloupcovou matici s příslušnými daty – dvojicemi hodnot tvořících bod experimentu. Funkce, kterou máme vytvořit, bude mít jeden vstupní parametr – název souboru – a jeden výstupní parametr – matici s čísly.

```
-5,7512;928,2;-5,01;913,56;-4,7111;
901,697;-4,6;850,6;-4,123;823,08;-3,89;
752,875;-3,712;732,1;-2,3;729,03;-5,51;
903,6;-4,681;895,781;-4,4457;845,216;
-4,23;813,8;-3,802;772,75;-3,2;712,1;
-2,3;720,03;
```

[71] maximální odchylka aproximace na uvažovaném intervalu je ze všech přípustných aproximací nejmenší

9.2 Řešení příkladů

9.2.1 Plnění vektorů a matic - řešení

a)
```
>> a=1:10; a=a';
```

---

b)
```
>> b=10:-1:1;
```

---

c)
```
>> A=[1:10;10:-1:1];
```
nebo
```
>> A=[a';b];
```

---

d)
```
>> B=[[1:10]',[10:-1:1]'];
```
nebo
```
>> B=[a,b'];
```

---

e)
```
>> E=eye(10,10);
```
nebo
```
>> E=eye(10);
```

---

f)
```
>> N=zeros(5,10);
```

---

g)
```
>> J=ones(10,5);
```

---

h)
```
>> NC=rand(5,7);
```
nebo
```
>> NC= randn(5,7);
```

---

i)
```
>> DD=eye(5,5)*5;
>> DD=rot90(DD);
```

---

j)
```
>> DT=ones(5,5);
>> DT=tril(DT,-1);
```

9.2.2 Základní operace s maticemi, vektory a polynomy - řešení

a)
```
>> a*b
ans = 1     2     3     4     5     6     7     8     9    10
      2     4     6     8    10    12    14    16    18    20
      3     6     9    12    15    18    21    24    27    30
      4     8    12    16    20    24    28    32    36    40
      5    10    15    20    25    30    35    40    45    50
      6    12    18    24    30    36    42    48    54    60
      7    14    21    28    35    42    49    56    63    70
      8    16    24    32    40    48    56    64    72    80
      9    18    27    36    45    54    63    72    81    90
     10    20    30    40    50    60    70    80    90   100
>> b*a
ans =   385
```

---

b)
```
>> b.^2
ans =   1     4     9    16    25    36    49    64    81   100
```

---

c)
```
>> b*b'
ans   =   385
```

---

d)
```
>> A*B
ans   = 385   385
        220   220
      >> B*A
ans   = 11    11    11    11    11    11    11    11    11    11
        22    22    22    22    22    22    22    22    22    22
        33    33    33    33    33    33    33    33    33    33
        44    44    44    44    44    44    44    44    44    44
        55    55    55    55    55    55    55    55    55    55
        66    66    66    66    66    66    66    66    66    66
        77    77    77    77    77    77    77    77    77    77
        88    88    88    88    88    88    88    88    88    88
        99    99    99    99    99    99    99    99    99    99
       110   110   110   110   110   110   110   110   110   110
```

---

e)
```
>> x=0:2*pi/999:2*pi;
>> ys=sin(x);
```

---

f)
```
>> x=0:2*pi/999:2*pi;
>> ys1=sin(x.^2)+sin(x).^2;
```

---

g)
```
>> x=-pi:2*pi/99:pi;
>> y=x;
```

<u>řešení první</u> (přímočaré)
```
>> for a=1:length(x),
for b=1:length(y),
XY(a,b)=x(a)*y(b);
end
end
>> YS=cos(XY);
```

<u>řešení druhé</u> (s využitím funkce MATLABu **meshgrid**[72])
```
>> [X,Y]=meshgrid(x,y);
>> YS=cos(X.*Y);
```

---

h)
```
>> NCN=NC*NC'                    % součin matice a matice transponované
NCN   = 2.0716    1.5065    1.7063    1.7785    1.7123
        1.5065    2.4858    2.4758    2.2091    1.8211
        1.7063    2.4758    3.2830    1.9539    1.9714
        1.7785    2.2091    1.9539    2.5666    1.7849
        1.7123    1.8211    1.9714    1.7849    2.0978
      >> D=det(NCN)                 % determinant matice
D     =    0.2946
      >> iNCN=inv(NCN)              % inverzní matice
iNCN  = 2.3591    1.9015   -0.8484   -1.6971   -1.3350
  1.9015    5.8267   -2.5514   -3.5780   -1.1682
 -0.8484   -2.5514    1.8540    1.3771   -0.0067
 -1.6971   -3.5780    1.3771    3.3639    0.3350
 -1.3350   -1.1682   -0.0067    0.3350    2.3017
```

---

[72] tato funkce vytvoří dvě matice - jednu se stejnými hodnotami v každém řádku a druhou se stejnými hodnotami v každém sloupci. Tyto matice je pak možné použít (ve spojení s operacemi člen po členu) jako přímou náhradu proměnných x a y ve skalárních matematických výrazech

i)
```
>> NCN*inv(NCN)
ans   = 1.0000    0.0000    0.0000    0.0000    0.0000
        0.0000    1.0000    0.0000         0         0
        0.0000    0.0000    1.0000    0.0000         0
        0.0000    0.0000    0.0000    1.0000         0
        0.0000         0    0.0000    0.0000    1.0000
```

---

j) <u>řešení první</u> (přímočaré)
```
>> S=0;
>> for a=1:1000,
S=S+a;
end;
>> S
S     =   500500
```

<u>řešení druhé</u> (práce s vektory - součin řádkového a sloupcového vektoru stejné délky)
```
>> S=[1:1000]*ones(1000,1)
S     =   500500
```

---

k)
```
>> p1=[1,-1,1,1];                    % x^3-x^2+x+1
>> p2=[1,1,1,-2];                    % x^3+x^2+x-2
>> p3=[1,0,1,0];                     % x^3+x
>> v=conv(p1,conv(p2,p3));
v     = 1   0   2  -1   5  -2   2  -1  -2   0     % x^9+2x^7- x^6+5x^5-2x^4+2x^3-x^2-2x
```

---

l)
```
>> p=[1,1,1,1];                      % x^3+x^2+x+1
>> Int=polyint(p)
Int   = 0.2500  0.3333  0.5000  1.0000  0     % x^4 /4+x^3/3+x^2/2+x+0
>> Der=polyder(p)
Der   = 3       2       1                     % 3x^2+2x+1
```

---

m)
```
>> p=[1,1,1,1];                      % x^3+x^2+x+1
>> roots(p)
ans   = -1.0000
        -0.0000 + 1.0000i
        -0.0000 - 1.0000i
```

---

n)
```
>> k=1:5;
>> p1=poly(k)
p1    = 1  -15   85  -225   274  -120          % x^5-15 x^4 +85x^3-225x^2+274x-120
>> k=[1+i,1+2*i,1+3*i];
>> p=poly(k)
p2    = 1.0  -3.0-6.0i  -8.0+12.0i   10.0     % x^3-(3+6i)x^2-(8-12i)x+10
```

9.2.3 Kreslení funkcí - řešení

Nakreslete průběh funkce

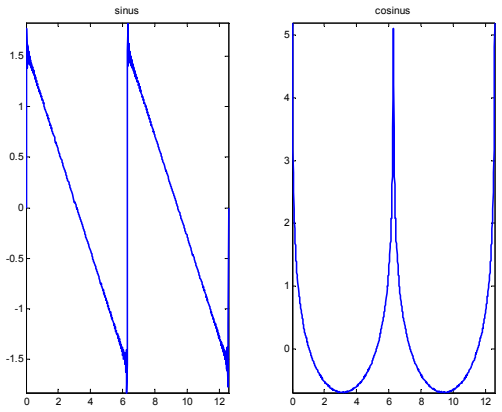
a)
```
>> x=0:2*pi/999:2*pi;
>> y=sin(x);
>> plot(x,y);
```

---

b)
```
>> x=0:2*pi/999:2*pi;
>> y=sin(x.^2)+sin(x).^2;
>> plot(x,y);
```

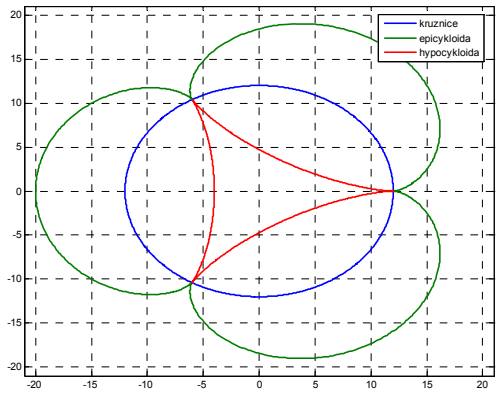
---

c)
```
>> x=0:4*pi/999:4*pi;
>> ys=sin(x); yc=cos(x);
>> for i=2:101,
ys=ys+sin(i*x)/i;
yc=yc+cos(i*x)/i;
end
>> subplot(121)
>> plot(x,ys)
>> title('sinus')
>> axis([0,4*pi,min(ys),max(ys)])
>> subplot(122)
>> plot(x,yc)
>> title('cosinus')
>> axis([0,4*pi,min(yc),max(yc)])
```



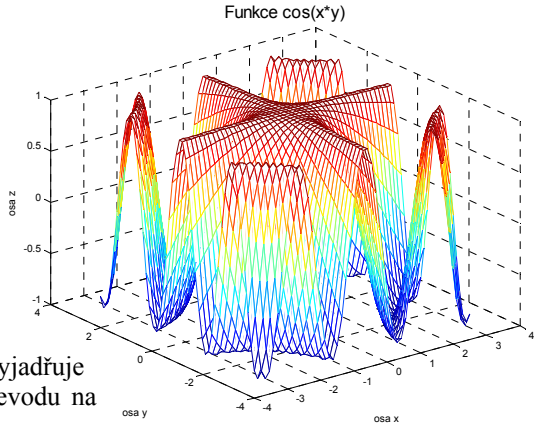
---

d)
```
>> t=0:2*pi/999:2*pi;
>> R=12;
>> r=4;
>> x=R*sin(t); y=R*cos(t);
>> xE=(R+r)*cos(t)-r*cos((R+r)/r*t);
>> yE=(R+r)*sin(t)-r*sin((R+r)/r*t);
>> xH=(R-r)*cos(t)+r*cos((R-r)/r*t);
>> yH=(R-r)*sin(t)-r*sin((R-r)/r*t);
>> plot(x,y,xE,yE,xH,yH,':')
>> grid
>> legend('kruznice',...
   'epicykloida','hypocykloida')
>> axis([-21,21,-21,21])
```



---

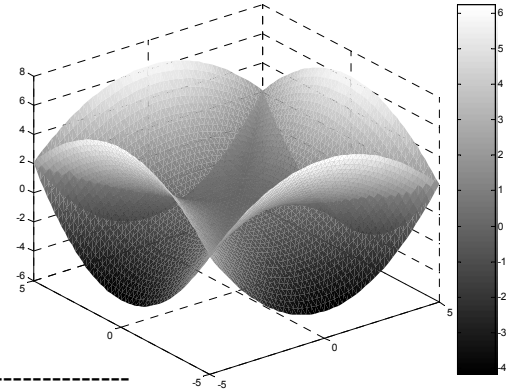
e)
```
>> x=-pi:2*pi/99:pi;
>> y=x;
>> [X,Y]=meshgrid(x,y);
>> Z=cos(X.*Y);
>> mesh (x,y,Z)
>> title('Funkce cos(x*y)')
>> xlabel('osa x')
>> ylabel('osa y')
>> zlabel('osa z')
```



Poznámka k obrázku. Originál je barevný, kde barevný přechod vyjadřuje výšku bodu nad základnou. Žlutozelená oblast (střed) se při převodu na černobílý tisk nezobrazí

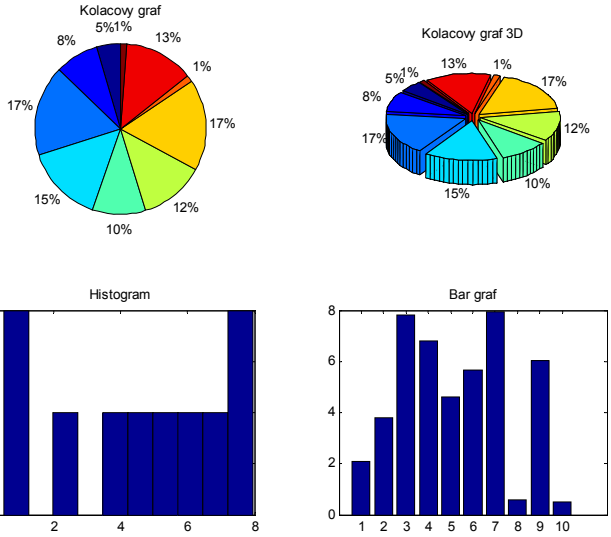
---

f)
```
>> x=-5:0.2:5;  y=x;
>> [X,Y]=meshgrid(x,y);
>> Z1=X.^2/4-Y.^2/6;
>> Z2=-X.^2/6+Y.^2/4;
>> surf(x,y,Z1)
>> hold
Current plot held
>> surf(x,y,Z2)
>> shading interp
>> colormap('gray')
>> colorbar
```



---

g)
```
>> x=rand(1,10)*10;
>> subplot(221)
>> pie(x)
>> title('Kolacovy graf')
>> subplot(222)
>> pie3(x,ones(1,10))
>> title(' Kolacovy graf 3D')
>> subplot(223)
>> hist(x)
>> title('Histogram')
>> subplot(224)
>> bar(x)
>> title('Bar graf')
```



9.2.4 Tvorba uživatelských funkcí - řešení

a) Příklad použití

```
>> t=-2*pi:4*pi/999:2*pi;
>> plot(t,f924a(t));
```

```
function y=f924a(x)
%
%              y=fce924a(x)

y=(exp(x)+exp(-x))/2;
```

---

b) Poznámka k řešení: musí se použít operace násobení člen po členu, aby nedošlo k vektorovému násobení)

Příklad použití

```
>> x=-2*pi:4*pi/999:2*pi;
>> plot(x,f924b(x))
```

```
function y=f924b(x)
%
%              y=fce924b(x)

v=x.*cos(x);
```

---

c) Příklad použití

```
>> x=-2*pi:4*pi/99:2*pi;
>> y=x;
>> [X,Y]=meshgrid(y,x);
>> mesh(x,y,f924c(X,Y))
```

```
function z=f924c(x,y)
%
%              z=f924c(x,y)

z=sin(x.^2+y.^2);
```

---

d) Příklad použití

```
>> x=-2:0.1:5;
>> y=0.1*x+2;
>> y=y+randn(1,length(x));
>> [ko,qo]=f924d(x,y)
ko    = 0.0827
qo    = 1.9169
```

```
function [k,q]=f924d(x,y)
%
%              [k,q]=f924d(x,y)

r=polyfit(x,y,1);
k=r(1); q=r(2);
```

9.2.5 Použití uživatelských funkcí

a) Řešení: rovnice má pouze jedno řešení

```
>> koren=fzero('f925a',0)
koren =  1.7455
```

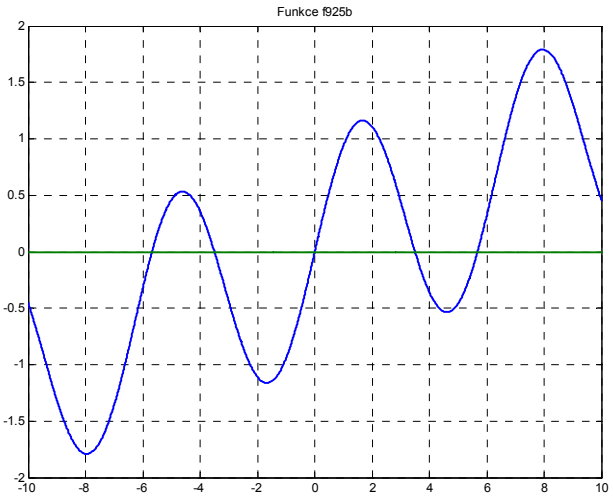
```
function y=f925a(x)
%
%              y=f925a(x)

y=x.*exp(x)-10;
```

---

b) Řešení: rovnice má více řešení. Pomocí průběhu funkce určíme počet a přibližnou polohu řešení.

```
>> x=-10:0.01:10;
>> z=zeros(size(x));
>> plot(x,f925b(x),x,z)
>> title('Funkce f925b)')
>> grid
```

```
function y=f925b(x)
%
%              y=f925b(x)

y=sin(x)+x/10;
```



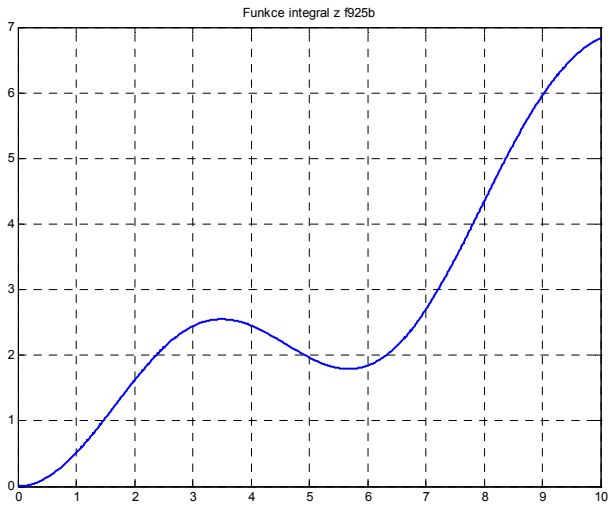
Z obrázku je zřejmé, že příklad má 5 řešení - je potřeba volit výchozí body poblíž řešení, ke kterému chceme dospět. Vlastní řešení získání použitím funkce `fzero` s startem vyhledávání v okolí odhadnutých řešení

```
>> k1=fzero('f925b',-6)
k1    = -5.6792
>> k2=fzero('f925b',-3)
k2    = -3.4991
>> k3=fzero('f925b',0)
k3    =  0
>> k4=fzero('f925b',3)
k4    =  3.4991
>> k5=fzero('f95b',6)
k5    =  5.6792
```

---

c) Řešení: využijeme m-funkci z příkladu 9.1.5b

```
>> quadl('f925b',0,10)
ans = 6.8391
```



---

d) Řešení: využijeme m-funkci z příkladu 9.1.5b

```
>> t=0:10/999:10;
>> for a=1:1000,
y(a)=quadl('f925b',0,t(a));
end
>> plot(t,y)
>> title('Funkce integral z f925b')
>> grid
```

---

e) Řešení:

```
>> [t,y]=ode45('f925e',[0;100],0);
```

```
function dy=f925e(t,x)
%
%          dy=f925e(t,x)

dy=(5-0.5*sqrt(x(1)))/2;
```

---

f) Řešení: Diferenciální rovnici řádu druhého je nutné převést na soustavu dvou diferenciálních rovnic řádu prvého

$$\frac{dy}{dx}=z \qquad \frac{dz}{dx}+0.75z+6y=\sin(\sqrt{x}) \qquad Y=\begin{bmatrix} z \\ y \end{bmatrix} \qquad \frac{dY}{dx}=\begin{bmatrix} \frac{dz}{dx} \\ \frac{dy}{dx} \end{bmatrix}=\begin{bmatrix} \sin(\sqrt{x})-0.75z-6y \\ z \end{bmatrix}$$

```
>> [x,y]=ode45('f925f',[0;10],[0;1]);
>> plot(x,y(:,2))
>> title('Funkce 925f')
```

```
function dy=f925f(x,y)
%
%          dy=f925f(x,y)
dy=zeros(2,1);
dy(1)=sin(sqrt(x))-0.75*y(1)-6*y(2);
dy(2)=y(1);
```

9.2.6 Kombinace více funkcí - řešení

a) Řešení: nejprve je potřeba vytvořit m-funkci definující zadaný integrand (*f926a1*) a potom další m-funkci (*f926a*) definující funkci jejichž nulovou hodnotu hledáme. Vlastní řešení nalezneme pomocí funkce `fzero`

```
>> a=fzero('f926a',0.1)
a     =   0.0818
```

```
function y=f926a1(x,a)
%
%          y=f926a1(x,a)
y=exp(-x.^2/a);
```

```
function y=f926a(a)
%
%          y=f926a(a)
y=quad8('fce_86a1',-0.5,0.5,[],[],a)-0.5;
```

---

b) Řešení: využijeme již hotové m-funkce *f925e*, která byla použita pro řešení diferenciální rovnice. Dále vytvoříme další m-funkci, která definuje funkci jejichž nulovou hodnotu hledáme (*f926b*). Vlastní řešení nalezneme pomocí funkce `fzero`

```
>> t=fzero('fce_86b',50)
t     =   41.6672
```

```
function y=f926b(t)
%
%          y=f926b(t)

[x,yp]=ode45('f925e',[0;t],0);
y=yp(end)-50; % rozdíl koncové hodnoty a 50
```

---

c) Řešení: je nutné vytvořit m-funkci, která určí průběh aproximační křivky pro danou sadu parametrů a vyhodnotí hodnotu zvoleného kritéria. Vytvořme funkci, která podle hodnoty parametru *Kriterium* vrací hodnotu kriteria vypočtenou podle různých požadavků. Je vhodné, aby tato m-funkce měla jako vstupní parametr vektor původních hodnot a jako druhý výstupní parametr vracela přímo vektor odchylek. Protože tato m-funkce bude argumentem funkce `fminsearch` musí mít jako první výstupní parametr hodnotu, která se minimalizuje (kritérium) a jako první vstupní parametr vektor parametrů (*f926c*)

data polokružnice a počáteční hodnoty parametrů

```
>> x=-10:0.25:10;
>> y=sqrt(10^2-x.^2);
>> p0=[0 0 0 0 1];
```

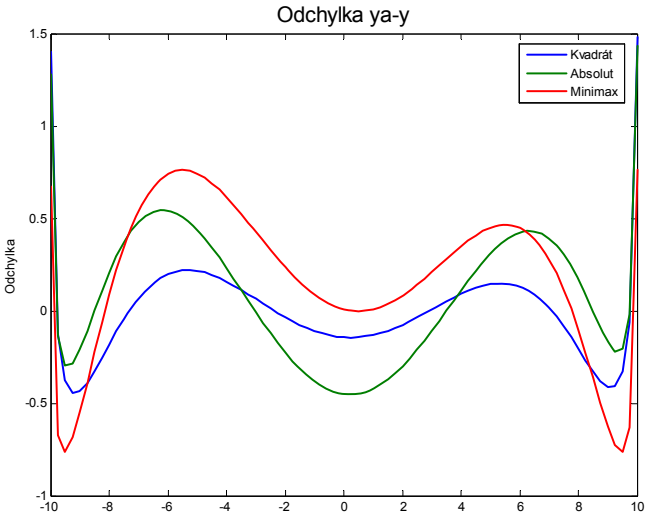
vyhledání minima

```
>> p1=fminsearch('f926c',p0,…
        [],x,y,'kvadrat');
>> p2=fminsearch('f926c',p0,…
        [],x,y,'absolut');
>> p3=fminsearch('f926c',p0,…
        [],x,y,'minimax');
```

```
function [Kr,e]=f926c(par,x,y,Kriterium)
%
%      [Kr,e]=f926c(par,x,y,Kriterium)

N=length(x);    % pocet prvku vektoru
x=x(:);y=y(:); % preved na sloup. vektory
ya=polyval(par,x); % vyp. aprox. v bodech x
e=ya-y;         % odchylka v bodech x
% suma kvadratu odchylek
Kr1=e'*e;       % skalarni soucin vektoru
% suma absolutnich hodnot
Kr2=sum(abs(e));
% maximalni odchylka (v absolutni hodnote)
Kr3=max(abs(e));
switch lower(Kriterium)
  case 'kvadrat', Kr=Kr1;
  case 'absolut', Kr=Kr2;
  case 'minimax', Kr=Kr3;
  otherwise, disp('Nezname kriterium');
end
```

výpočet odchylky aproximace

```
>> [kr,e1]=f926c(p1,x,y,…
        'kvadrat');
>> [kr,e2]=f926c(p2,x,y,…
        'absolut');
>> [kr,e3]=f926c(p3,x,y,…
        'minimax');
>> plot(x,e1,x,e2,…
        '--',x,e3,':');
>> title('Odchylka ya-y');
>> ylabel('Odchylka');
>> legend('Kvadrát','Absolut',…
        'Minimax');
```



---

d) Řešení: vytvořme m-funkci, která vypočte hodnotu kritéria pro danou sadu parametrů aproximační křivky. Pomocí funkce `fminsearch` nalezneme sadu parametrů minimalizující kritérium.

```
function [Kr,e]=f926d(p,x,y)
%
%          [Kr,e]=f926d(p,x,y)

x=x(:);y=y(:);
ya=(p(1)+p(2)*x+p(3)*x.^2).*exp(p(4)*x+p(5)*x.^2);
e=ya-y;
Kr=max(abs(e));
```

data polokružnice

```
>> x=-10:0.5:10;
>> y=sqrt(10^2-x.^2);
```

"nástřel" parametrů

```
>> p0=[1 1 0 0 0];
```

výpočet parametrů minim. kritérium

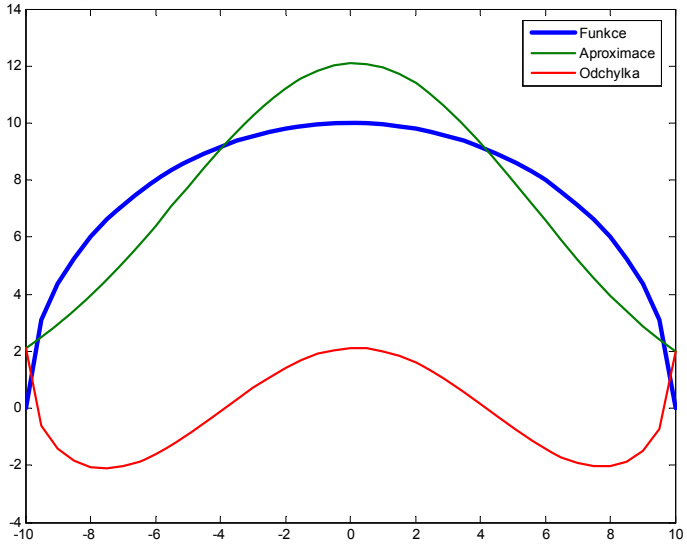
```
>> p=fminsearch('f926d',p0,[],x,y)
p     =   12.3264    0.4814    -0.0066    -0.0332    -0.0163
```

výpočet aproximace

```
>> ya=(p(1)+p(2)*x+p(3)*x.^2).*exp(p(4)*x+p(5)*x.^2);
```

výpočet odchylky a vykreslení původní funkce, aproximace a odchylky

```
>> plot(x,y,x,ya,':',x,…
   ya-y,'-.')
>> legend('Funkce',…
  'Aproximace','Odchylka')
```



9.2.7 Práce se soubory a řetězci - řešení

a) Řešení 1: pomocí standardní funkce MATLABu **save**

```
>> x=rand(100,1)*100;
      % sloupcový vektor
>> save zdroj.txt x -ascii          % uloz jako text (8 platných cifer)
```

Řešení 2a: pomocí obecných funkcí pro práci se soubory (**fopen**, **fclose** a **fprintf**) – standard "C"

```
>> z=fopen ('zdroj.txt','wt');      % otevri textovy soubor pro zapis
>> for k=1:100,                     % opakuj 100x
fprintf(z,'%e\n',rand(1,1)*100);    % 1 číslo na řádek (8 platných cifer)
end
>> fclose(z)                        % uzavři soubor
```

Řešení 2b: pomocí obecných funkcí pro práci se soubory (**fopen**, **fclose** a **fprintf**) – pouze MATLAB

```
>> x=rand(100,1)*100;                % sloupcový vektor
>> z=fopen ('zdroj.txt','wt');       % otevri textovy soubor pro zapis
>> fprintf(z,'%e\n',x);              % 1 číslo na řádek (8 platných cifer)
>> fclose(z)                         % uzavři soubor
```

---

b) Řešení 1: pomocí standardních funkcí MATLABu **load** a **save**

```
>> load zdroj.txt                    % načti soubor do matice
>> [nr,ns]=size(zdroj);              % počet řádků a sloupců
>> licha=1:2:nr;                     % indexy lichých řádků
>> suda=2:2:nr;                      % indexy sudých řádků
>> cil1=zdroj(licha,:)*2;            % vyjmi všechny liché řádky
>> cil2=zdroj(suda,:)/2;             % vyjmi všechny sudé řádky
>> save cil01.txt cil1 -ascii        % ulož liché do textového souboru
>> save cil02.txt cil2 -ascii        % ulož sudé do textového souboru
```

Řešení 2: pomocí obecných funkce pro práci s textovými soubory (**fopen**, **fclose**, **fscanf** a **fprintf**)

```
>> z=fopen ('zdroj.txt','rt');       % otevři textový soubor pro čtení
>> c1=fopen('cil01.txt','wt');       % otevři textový soubor pro zápis
>> c2=fopen('cil02.txt','wt');       % otevři textový soubor pro zápis
>> x=fscanf(z,'%e');                 % vyčti vše ze souboru do proměnné x
>> licha=1:2:nr;                     % indexy lichých řádků
>> suda=2:2:nr;                      % indexy sudých řádků
>> cil1=zdroj(licha,:)*2;            % vyjmi všechny liché řádky
>> cil2=zdroj(suda,:)/2;             % vyjmi všechny sudé řádky
>> fprintf(c1,'%e\n',cil1);          % 1 číslo na řádek (8 platných cifer)
>> fprintf(c2,'%e\n',cil2);          % 1 číslo na řádek (8 platných cifer)
>> fclose (z);                       % uzavři soubor
>> fclose(c1);                       % uzavři soubor
>> fclose(c2);                       % uzavři soubor
```

---

c) Řešení: prosté uložení v ASCII (textové podobě) pomocí příkazu **save** *imagic6.txt* data -ASCII je možné, ale nastanou problémy při čtení v příslušném programu, protože jako oddělovač desetinné části čísel bude použita tečka. Jedno z možných řešení je následující posloupnost příkazů

```
>> M=magic(5);                       % vytvoř magický čtverec
>> X=inv(M);                         % inverzní matice
>> XS=num2str(X,16);    % každý řádek převeď na text, každé číslo na 16 cifer
>> file=fopen('imagic5.txt','wt');   % otevři textový soubor pro zápis
>> for k=1:5                         % pro každý řádek matice
 XS(k,strfind(XS(k,:),'.'))=',';     % najdi tečky v řádku a nahraď čárkami
 fprintf(file,'%s\n',XS(k,:));       % zapiš jako řetězec + přechod na nový řádek
 end                                 % konec cyklu
>> fclose(file);                     % uzavři soubor
```

---

d) Řešení: hledaná funkce může vypadat např. tak, jak je uvedeno v rámečku na další straně. Nejprve je nutné data načíst do MATLABu. To zajistí příkazy **fopen** a **fgets**. Po ukončení práce se souborem nezapomeneme uzavřít soubor příkazem **fclose**. Výsledkem příkazu `fgets` je vektor (data) obsahující jednotlivé znaky souboru – řetězec znaků. MATLAB jako oddělovač desetinné části používá tečku. Nejprve tedy nahradíme všechny čárky tečkami. To zajistíme pomocí funkce **strfind**, která vrací vektor s pozicemi výskytu hledaného řetězce uvnitř jiného řetězce. Najdeme tedy pozice všech čárek a nahradíme tečkami. Pak zjistíme pozice všech středníků oddělujících čísla a tím i počet čísel. Připravíme si tři pomocná ukazovátka (*k*, *r*, *zac*) a v cyklu vyjmeme znaky prvního čísla dvojice, převedeme na číslo příkazem **str2num** a uložíme do matice výsledků. Posuneme ukazovátka na další číslo. Zopakujeme pro druhé číslo dvojice a uzavřeme cyklus.

```
function X=f287d(soubor)
%
%      X=f287d(soubor)
%
% soubor - textový řetězec s názvem souboru
% X - matice čísel

file=fopen(soubor,'rt');          % otevři text. soubor pro čtení
data=fgets(file);                 % načti vše jako znakový řetězec
fclose(file)                      % uzavři spojení se souborem
poz=strfind(data,',');            % najdi pozice všech čárek
data(poz)='.';                    % na pozicích nahraď tečkou
poz=strfind(data,';');            % pozice všech středníků
N=length(poz);                    % počet středníků = počet čísel
X=zeros(N/2,2);                   % matice pro výsledek
k=1;                              % ukazovátko na zpracovávané číslo
r=1;                              % řádek v matici výsledků
zac=1;                            % pozice prvního znaku zprac.čísla
while k<N
    s=data(zac:poz(k)-1);         % vyjmi znaky čísla (bez ;)
    X(r,1)=str2num(s);            % řetězec na číslo a ulož
    zac=poz(k)+1;                 % pozice za zpracovávané číslo
    k=k+1;                        % ukazovátko na další číslo
    s=data(zac:poz(k)-1);         % vyjmi znaky
    X(r,2)=str2num(s);            % řetězec na číslo a ulož
    zac=poz(k)+1;                 % pozice za zpracovávané číslo
    k=k+1;                        % ukazovátko na další číslo
    r=r+1;                        % další řádek v matici výsledků
end
```

10. Komplexní řešené příklady (MATLAB)

Následující příklady by měly dokumentovat využití MATLABu při řešení složitějších úloh. Princip řešení jednotlivých úloh nemá samozřejmě s MATLABem nic společného, ale znalost možností tohoto prostředku umožňuje jednodušší návrh řešení. Lze navrhnout celkové řešení bez nutnosti blíže upřesňovat dílčí řešení poměrně složitých částí, které už jsou kompletně řešitelné prostředky MATLABu.

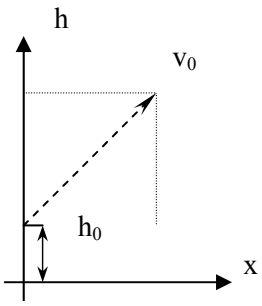
Možnosti MATLABu a relativní jednoduchost jeho použití umožňují provádění i složitých pomocných výpočtů. Např. nemám-li k dispozici tabulky s tabelovanými hodnotami některých fyzikálních konstant (třeba koeficienty přestupu tepla, koeficienty odporu prostředí atd.) můžu provést pomocné výpočty pro získání smysluplných odhadů jejich hodnot. Také možnost grafického zobrazení průběhu veličin může přispět k verifikaci (ověření) toho, zda spočítaný výsledek je správný či zda neexistují ještě další řešení.

Většina následujících příkladů vychází z jednoduchých a snadno pochopitelných zadání a předpokládá pouze všeobecné znalosti matematiky a fyziky a schopnost logického uvažování. Poslední příklady vyžadují určité znalosti z obecné chemie.

10.1 Dostřel děla (MATLAB)

První ze složitějších příkladů má dokumentovat vícenásobné použití MATLABu při řešení jednoho příkladu. Jde o výpočet optimálního náměru děla tj. takového úhlu osy hlavně, při kterém dělo s danými parametry dostřelí nejdále. Tento příklad bude využit i v kapitole 16 zabývající se řešením v SIMULINKu. Na tomto příkladě je ukázáno modulární řešení úlohy, kdy se řešení určité části úlohy používá jako hotový modul v řešení části navazující.

Mějme problém zadaný následovně: "Z děla umístěného na pahorku ve výšce h_0 nad okolním terénem je pod úhlem α vystřelena železná koule o poloměru r s úsťovou rychlostí v_0. Určete dráhu střely v rovině kolmé na terén a procházející osou hlavně. Při výpočtu uvažujte, že odpor prostředí je úměrný (konstanta c_x) ploše průřezu střely S a kvadrátu rychlosti v." V případě, že bychom neuvažovali odpor prostředí, lze celý problém řešit analyticky. Pokud odpor prostředí uvažujeme je možné řešení pouze numerické.



obrázek 10-1 Schéma pro příklad Dostřel děla

Jak je popsán pohyb koule v gravitačním poli země při uvažování odporu prostředí ? Pro sestavení rovnic popisujících dráhu střely vyjdeme z rovnováhy sil. Skutečné síly rozložíme na složky do osy x a y.

Rovnováha sil v ose y

$$\underbrace{F_y}_{\text{síla zrychlení}} + \underbrace{F_{yo}}_{\text{síla odporu}} + \underbrace{F_g}_{\text{síla gravitační}} = 0$$

$$m\frac{d^2h_y}{dt^2} + c_x S\underbrace{\left(\frac{d\,h_y}{dt}\right)^2}_{v_y^2} = -mg \quad h_y(t=0) = h_0 \quad \left.\frac{d\,h_y}{dt}\right|_{t=0} = v_0\sin(\alpha)$$

Rovnováha sil v ose x

$$\underbrace{F_x}_{\text{síla zrychlení}} + \underbrace{F_{xo}}_{\text{síla odporu}} = 0$$

$$m\frac{d^2h_x}{dt^2} + c_x S\underbrace{\left(\frac{d\,h_x}{dt}\right)^2}_{v_x^2} = 0 \quad h_x(t=0) = h_{x0} \quad \left.\frac{d\,h_x}{dt}\right|_{t=0} = v_0\cos(\alpha)$$

Hmotnost a čelní plocha střely

$$V = \tfrac{4}{3}\pi r^3 \quad m = \rho V \quad S = \pi r^2$$

Výsledné rovnice

$$\frac{d^2h_y}{dt^2} + c_x \frac{S}{m}\left(\frac{dh_y}{dt}\right)^2 = -g \qquad h_y(t=0) = y_0 \qquad \left.\frac{dh_y}{dt}\right|_{t=0} = v_0 \sin(\alpha)$$

$$\frac{d^2h_x}{dt^2} + c_x \frac{S}{m}\left(\frac{dh_x}{dt}\right)^2 = 0 \qquad h_x(t=0) = x_0 \qquad \left.\frac{dh_x}{dt}\right|_{t=0} = v_0 \cos(\alpha)$$

$$\frac{S}{m} = \frac{3}{4\rho r} = k$$

Pro další práci v MATLABu je vhodné výsledné rovnice přepsat do tvaru soustavy rovnic I. řádu

$$\frac{dh_y}{dt} = v_y \qquad h_y(t=0) = h_0$$

$$\frac{dv_y}{dt} = -c_x \frac{3}{4\rho r} sign(v_y) v_y^2 - g \qquad v_y(t=0) = v_0 \sin(\alpha)$$

$$\frac{dh_x}{dt} = v_x \qquad h_x(t=0) = 0$$

$$\frac{dv_x}{dt} = -c_x \frac{3}{4\rho r} sign(v_x) v_x^2 \qquad v_y(t=0) = v_0 \cos(\alpha)$$

kde h_y výška střely nad povrchem; v_y rychlost střely ve směru osy y
h_x vzdálenost střely od ústí hlavně v ose x; v_x rychlost střely ve směru osy x

Funkce **sign**[73] zajišťuje, že i při změně smyslu rychlosti, síla odporu působí proti působící síle.

Jeden z možných přepisů do m-funkce MATLABu (*f101a.m*) je ukázán ve vedlejším rámečku. Kromě dvou povinných parametrů (první je čas řešení, druhý jsou hodnoty proměnných), jejichž existenci předpokládá (a připravuje) řešitel soustavy diferenciálních rovnic (např. **ode45**), potřebujeme předat do výpočtu ještě další parametr (hodnotu parametru k tj. přepočtený koeficient odporu). Řešitel předání uživatelem vyžadovaných parametrů dovoluje, ale musíme dodržet jeho pravidla. Řešitel ve skutečnosti do naší funkce dodává vždy parametry tři (parametr označený *flag*). Tento třetí parametr můžeme ignorovat, pokud žádné další parametry nenásledují. Ovšem pokud další parametry předáváme, musíme třetí parametr uvést, aby se pořadí čtvrtého a případně dalších parametrů dalo rozpoznat.

```
function dH=f101a(t,H,flag,k)
% Příklad 10.1 DOSTŘEL DĚLA
% soustava diferenciálních rovnic popisujících
% pohyb střely v gravitačním poli s uvažováním
% odporu vzduchu. Předpokládá se, že funkce je
% volaná z řešitele (ODE45) a předáván parametr k
% k ... koeficient odporu
% H ... vektor veličin     H(1)=hy ... výška
%                          H(2)=vy ... rychlost
%                          H(3)=hx ... vzdálenost
%                          H(4)=vx ... rychlost
% dH .. vektor příslušných derivací

hy=H(1); vy=H(2); hx=H(3); vx=H(4);
dH=zeros(4,1);             % výstup sloupcový vektor
dH(1)=vy;
dH(2)=-9.81-k*sign(vy)*vy^2;
dH(3)=vx;
dH(4)=-k*sign(vx)*vx^2;
```

[73] funkce signum je definována tak, že pro kladné hodnoty argumentu nabývá hodnoty 1, pro záporné hodnoty argumentu hodnoty –1 a pro nulovou hodnotu argumentu nabývá hodnotu 0

Zvolme nějaké smysluplné hodnoty parametrů – např. ty, které jsou uvedeny v tabulce. Protože tyto hodnoty budeme používat vícekrát, zapíšeme si je v podobě skriptu, který uložíme pod názvem *d101.m*.

Označení	Rozměr	Hodnota	Význam
h_0	m	10.0	Výška ústí hlavně nad terénem
h_{x0}	m	0.0	Poloha ústí hlavně v ose x
v_0	m s^{-1}	1000	Úsťová rychlost střely
r	m	0.05	Poloměr střely
ρ	kg m^{-3}	7800	Hustota materiálu střely
c_x	kg m^{-1}	0.26	Součinitel aerodynamického odporu
g	m s^{-1}	9.81	Gravitační zrychlení

Obsah skriptu definujícího parametry úlohy je v rámečku vlevo. V rámečku vpravo je skript *zf101a.m* realizující výpočet dráhy střely pro uvedené hodnoty parametrů a náměr 45°. Výpočet je proveden pro odhadnutý interval výpočtu 55 s.

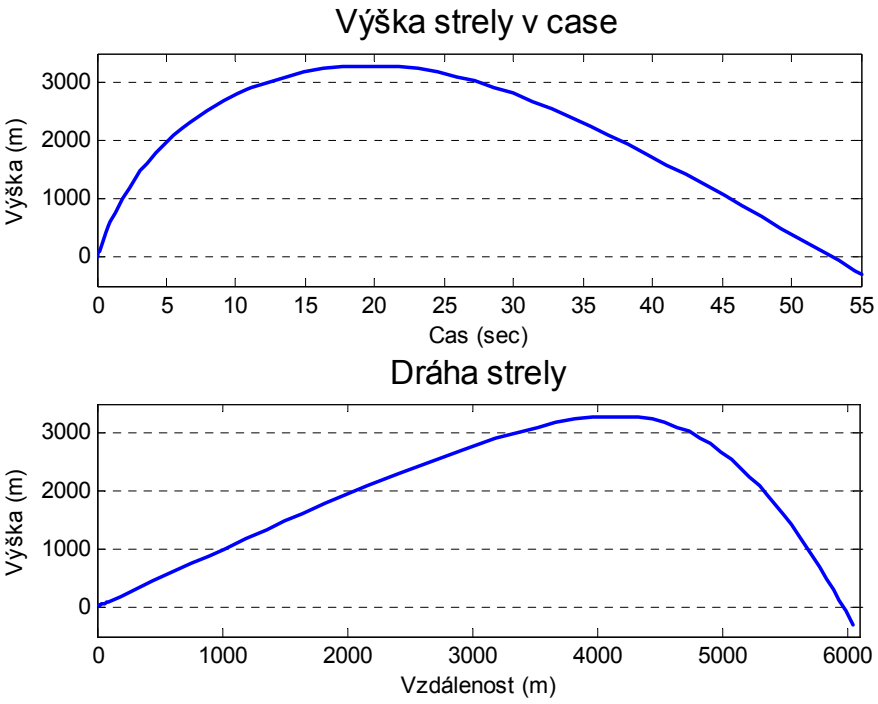
```
% skript d101.m
% Příklad 10.1
% definice parametrů

h0=10;    %(m) výška
al=45;    % (o) náměr
v0=1000;  %(m/s) rychlost
r=0.05;   %(m) poloměr střely
ro=7800;  %(kg/m3) hustota
%(kg/m) koef. aer.odporu
cx=0.26;
% pomocná konstanta k
k=3*cx/4/ro/r;
```

```
% skript zf101a.m
% Příklad 10.1 DOSTŘEL DĚLA
% výpočet dráhy střely

d101          % naplnění parametrů
x=2*pi/360;

al=45;
% vektor počátečních hodnot
H0=[h0;v0*sin(x*al);0;v0*cos(x*al)];
% vektor určující časový interval řešeni
cas=[0,55];
% vlastní řešení
[t,H]=ode45('f101a',cas,H0,[],k);
% nakreslení průběhů
% a) výška střely v čase
% b) výška střely v závislosti na vzdálenosti
subplot(211), plot(t,H(:,1))
title('Výška střely v čase')
xlabel('Čas (sec)'), ylabel('Výška (m)')
subplot(212), plot(H(:,3),H(:,1))
title('Dráha střely')
xlabel('Vzdálenost (m)'), ylabel('Výška (m)')
```

Na obrázku 10-2 vidíme časový průběh výšky střely a průběh výšky v závislosti na vzdálenosti získaný provedením skriptu *zf101a.m*. Na konci zvoleného časového intervalu řešení je výška již záporná.

```
>> zf101a
```

První problém je nalézt přesný čas dopadu střely na zem. Jinak formulováno, znamená to nalézt takový čas, pro který je jedna proměnná z řešení soustavy diferenciálních rovnic (výška střely) rovno nule. Jde tedy o hledání kořene funkce – v MATLABu funkce **fzero**. Abychom ji mohli použít, musíme vytvořit m-funkci realizující funkci (*f101b.m*), jejíž kořen hledáme. Tato funkce může vypadat např. tak, jak je uvedeno v rámečku na další straně. Funkce vrací dva parametry – výšku a vzdálenost střely pro zadaný čas. První parametr – výška střely – bude využívat funkce



obrázek 10-2 Dráha střely v čase

`fzero` a druhý parametr – vzdálenost střely – bude funkce `fzero` ignorovat. Tento druhý parametr si připravíme pro budoucí využití.

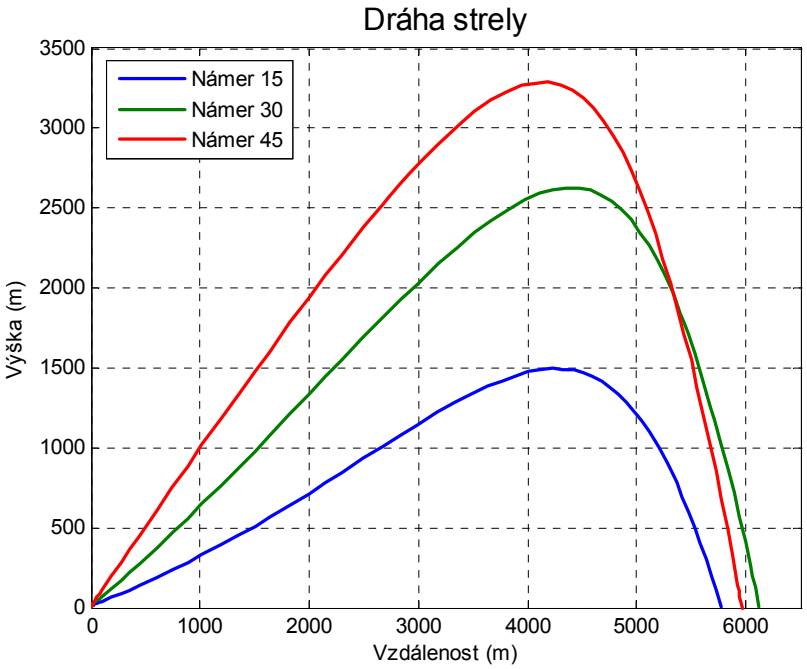
Tuto funkci odzkoušíme pomocí skriptu *zf101b.m* uvedeného v rámečku vpravo. Kromě ověření funkčnosti m-funkce *f101b* si zároveň ukážeme jak vypadá závislost průběhu výšky střely na vzdálenosti pro tři náměry $\alpha = 15°$, 30° a 45° (obr. 10-3). Tento obrázek může sloužit i pro odhad rozsahu náměrů ve kterém budeme hledat optimální hodnotu náměru maximalizující dostřel.

```
function [hy,hx]=f101b(t,H0,k)
% Příklad 10.1
% funkce pro určení výšky střely h
%
%      [hy,hx]=f101b(t,H0,k)

% řešení diferenciálních rovnic
[t,H]=ode45('f101a',[0,t],H0,[],k);
% výška v čase t je na konci sloupce
hy=H(end,1);     % výška v čase t
hx=H(end,3);     % vzdálenost v čase t
```

```
% skript zf101b.m
% Příklad 10.1 DOSTŘEL DĚLA
% výpočet dráhy střely pro různé náměry
d101      % načtení parametrů
x=2*pi/360;
al=15; % (o) úhel osy hlavně
% vektor počátečních hodnot
H0=[h0;v0*sin(x*al);0;v0*cos(x*al)];
% vlastní řešení
t1=fzero('f101b',100,[],H0,k)
[t1,H1]=ode45('f101a',[0,t1],H0,[],k);
al=30; % (o) úhel osy hlavně
H0=[h0;v0*sin(x*al);0;v0*cos(x*al)];
t2=fzero('f101b',100,[],H0,k)
[t2,H2]=ode45('f101a',[0,t2],H0,[],k);
al=45; % (o) úhel osy hlavně
H0=[h0;v0*sin(x*al);0;v0*cos(x*al)];
t3=fzero('f101b',100,[],H0,k)
[t3,H3]=ode45('f101a',[0,t3],H0,[],k);
% nakreslení průběhu
plot(H1(:,3),H1(:,1),H2(:,3),H2(:,1),…
H3(:,3),H3(:,1))
title('Dráha střely')
xlabel('Vzdálenost (m)'),ylabel('Výška (m)')
legend('Náměr 15','Náměr 30','Náměr 45')
```

```
>> zf101b
```

Funkce *f101b* nám pro zadaný čas vrátí kromě výšky i vzdálenost. A tuto hodnotu chceme volbou náměru maximalizovat. Jde tedy o úlohu vyhledání extrému funkce jedné proměnné – funkce **`fminbnd`** MATLABu. Opět potřebujeme vytvořit uživatelskou m-funkci (*f101c.m*), která bude mít minimum pro hledané řešení. Minimum proto, že funkce `fminbnd` vyhledává minimum na zadaném intervalu. Proto je poslední příkaz m-funkce *f101c* změna znaménka u hodnoty vzdálenosti (dostřelu), aby maximální dostřel byl minimální hodnotou m-funkce *f101c* a bylo možné využít funkci MATLABu `fminbnd`. Jeden z možných zápisů m-funkce *f101c* a vlastní řešení úlohy nalezení náměru maximalizujícího dostřel ve formě skriptu *zf101c.m* jsou v rámečcích na následující straně.



obrázek 10-3 Dráha střely pro různé náměry

Řešení celé úlohy – určení optimálního náměru, času dopadu a dostřelu - pak představuje spuštění skriptu *zf101c.m*

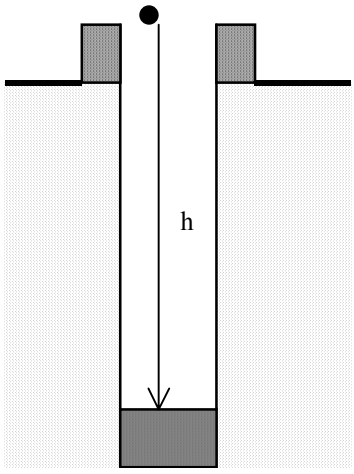
```
>> zf101c
alf    = 31.1597
t  = 47.5426
hx     = -6.1236e+003
```

```
function [hxm,t]=f101c(alfa,v0,k,hy0)
% Příklad 10.1 DOSTŘEL DĚLA
% funkce pro urceni dostrelu
%
%  [hxm,t]=f101c(alfa,v0,k,hy0)
%
% alfa . namer (ve stupnich)
% v0 ... ustova rychlost strely
% k .... parametry strely
% hy0 .. počátečni výška

vy=v0*sin(2*pi/360*alfa);
hx0=0;              % posun v ose x
vx=v0*cos(2*pi/360*alfa);
H0=[hy0;vy;hx0;vx]; %poč.podmínky

% vyhledání nulové hodnoty funkce
%(čas dopadu)
t=fzero('f101b',200,[],H0,k);
% souřadnice dopadu (y,x)
[hy,hx]=f101b(t,H0,k);
hxm=-hx;            % z maxima minimum
```

```
% skript zf101c.m
% Příklad 10.1 DOSTŘEL DĚLA
% výpočet max. dostřelu

d101          % načtení parametrů

% vyhled.min.hodnoty funkce
% (hodnota náměru)
alf=fminbnd('f101c',15,60,[],v0,k,h0);%
výpočet dostřelu a času dopadu
[hx,t]=f101c(alf,v0,k,h0);
alf
t
hx
```

10.2 Hloubka studně

Často se při návštěvě nějakého hradu dozvíme různé zcela zbytečné informace mimo jiné třeba i hloubku hradní studně. Jak ale tuto informaci zjistit, když není po ruce průvodce (který by si ji pohotově vymyslel) a vsadili jsme se třeba s přítelkyní? Jedna možnost je vytáhnout olovnici, dostatečně dlouhý provaz a hloubku k hladině změřit. Takto vybaveni na výlet s přítelkyní obvykle nevyrážíme. Zato hodinky se stopkami jsou poměrně běžné a tak můžeme provést jiné měření. Spustíme stopky a současně vrhneme do studně ocelovou kuličku[74] a posloucháme až to šplouchne. V tomto okamžiku zastavíme stopky. Pak už jen stačí provést referenční experiment na studni známé hloubky, sehnat počítač s instalovaným MATLABem a když nezapomeneme změřené časy, tak po chvíli počítání můžeme oznámit hloubku studně. Vzhledem k tomu, že to přítelkyně po nás pravděpodobně nedokáže přepočítat, určitě sázku vyhrajeme.



obrázek 10-4 Schéma pro příklad Hloubka studně

Jak tedy ze známého času t, který uplynul od okamžiku upuštění předmětu (s nulovou počáteční rychlostí, v gravitačním poli Země) do okamžiku, kdy jsme zaslechli šplouchnutí, spočítat hloubku studně h? Pokud zanedbáme odpor prostředí a rychlost šíření zvuku pak jde o triviální výpočet. Sice dospějeme k značně odlišnému výsledku od skutečnosti, ale výpočet je jednoduchý. Dokonce i v případě uvažování rychlosti šíření zvuku jde o jednoduchý výpočet. Komplikace nastávají v případě uvažování odporu prostředí a to ze dvou důvodů - jednak musíme řešit nelineární diferenciální rovnici a jednak potřebujeme znát fyzikální konstantu - součinitel odporu vzduchu c_x pro použitý předmět vržený do studny. S prvním problémem nám pomůže MATLAB a pro určení součinitele odporu můžeme provést referenční experiment[75], který vyhodnotíme opět v MATLABu.

Řešení zahájíme opět aplikací základního pravidla - nakreslíme obrázek s označením viz obr. 10-4. V tomto případě je obrázek jednoduchý. Zaveďme konvenci, že dráha padající kuličky s bude uvažována s kladnou hodnotou.

Jednoduchou rovnici popisující náš problém získáme na základě následující úvahy. Celkový čas t_{exp}, který změříme je dán součtem dvou složek. Času t_k, který potřebuje kulička k překonání vzdálenosti od místa puštění k hladině, a času t_z, který potřebuje zvuk k překonání té samé vzdálenosti. Je-li dráha kuličky

[74] výjimky, které s sebou ocelové kuličky nenosí, mohou do studny plivnout. Tuto variantu obzvláště nedoporučuji v případě, že se někde v okolí vyskytuje kombinace cedule s nápisem Ochranné pásmo vodního zdroje a hlídač. Nemuseli byste mít dostatek času na dokončení experimentu.

[75] aby byly experimenty srovnatelné, je vhodné vrhat tentýž předmět (proto doporučená ocelová kulička). Z tohoto důvodu nedoporučuji vrhat přítelkyni. Jednak byste měli problém s provedením referenčního experimentu (nutnost sehnat novou přítelkyni stejných parametrů) a jednak by neměl kdo ocenit vaší schopnost používání MATLABu.

v závislosti na čase popsána funkcí s_k(t) a dráha zvuku s_z(t), pak můžeme naší úlohu matematicky formulovat jako dvě rovnice

$$s_k(t_k) = s_z(t_z) \quad t_{\exp} = t_k + t_z$$

Rychlost šíření zvuku ve vzduchu budeme uvažovat jako konstantní s hodnotou v_z = 330 ms^{-1} tj. dráha s_z, kterou zvuk urazí za čas t je popsána rovnicí $s_z(t) = v_z t$.

Složitější je nalezení druhé funkce s_k(t). Tato funkce je řešením diferenciální rovnice $m\frac{d^2 s_k}{dt^2} = mg$ popisující pohyb tělesa v gravitačním poli bez odporu prostředí. Neuvažujeme-li odpor prostředí, jde o triviální řešení a pro případ nulových počátečních podmínek (v(t=0)=0 a s(t=0)=0) je řešením této rovnice známý vztah

$$s_k(t) = \frac{1}{2} g t^2$$

a celkové řešení úlohy spočívá ve vyřešení kvadratické rovnice (pouze jeden kořen má fyzikální význam) pro určení času t_k (nebo t_z) a pomocí něj i hloubky h

$$t_{\exp} = t_k + t_z \quad \wedge \quad v_z t_z = \frac{1}{2} g t_k^2 \quad \Leftrightarrow \quad v_z(t_{\exp} - t_k) = \frac{1}{2} g t_k^2 \quad \Leftrightarrow \quad \frac{1}{2} g t_k^2 + v_z t_k - v_z t_{\exp} = 0$$

$$t_k = \frac{-v_z \pm \sqrt{v_z^2 + 2 g v_z t_{\exp}}}{g} \qquad h = v_z(t_{\exp} - t_k)$$

Pro případ uvažování odporu vzduchu jde o nelineární diferenciální rovnici, pro kterou analytické řešení neexistuje. Musíme vyjít ze základní rovnice popisující chování reálné kuličky v gravitačním poli. V těchto úlohách klasické mechaniky se vychází z rovnováhy sil - gravitační síly F_g a proti působící síly setrvačné F_s a síly odporu prostředí F_o. Odpor prostředí je úměrný kvadrátu rychlosti. Konstanta úměrnosti tohoto vztahu je tvořena součinem čelní plochy S a součinitele aerodynamického odporu[76] c_x. Pro kuličku lze čelní plochu S i hmotnost m vyjádřit pomocí poloměru r a hustoty ρ.

$$F_s + F_o = F_g \quad m = \frac{4}{3}\pi r^3 \rho \quad S = \pi r^2$$

$$m\frac{d^2 s_k}{dt^2} + c_x S\left(\frac{ds_k}{dt}\right)^2 = mg \quad \Leftrightarrow \quad \frac{d^2 s_k}{dt^2} + c_x \frac{S}{m}\left(\frac{ds_k}{dt}\right)^2 = g \quad \Leftrightarrow \quad \frac{d^2 s_k}{dt^2} + \underbrace{c_x \frac{4}{3r\rho}}_{K}\left(\frac{ds_k}{dt}\right)^2 = g$$

Tato diferenciální rovnice určuje časový průběh dráhy padající kuličky tj. s_k(t). Řešení je zcela stejné jako v předchozí variantě. Opět hledáme takovou hodnotu t_k, pro kterou je splněna základní rovnice

$$s_k(t_k) = s_z(t_z) \quad t_{\exp} = t_k + t_z \quad \Leftrightarrow \quad s_k(t_k) - v_z(t_{\exp} - t_k) = 0$$

Tuto rovnice nelze řešit analyticky a je nutné použít numerické řešení. Zde nám pomůžou funkce MATLABu **`fzero`** a **`ode45`**. Než ji však začneme řešit je potřeba znát hodnotu součinitele aerodynamického odporu c_x. V případě kuličky by to neměl být až takový problém, ve slušných fyzikálních tabulkách by se měly dát hodnoty součinitele c_x pro základní geometrické tvary nalézt. Ovšem ti jedinci, kteří s sebou kuličky nenosí a vrhli něco jiného budou mít asi smůlu. Tito nešťastníci budou muset provést referenční experiment a koeficient z výsledku dopočítat. Tato úloha je ekvivalentní původní úloze.

Pro řešení v MATLABu potřebujeme pro obě úlohy přepsat původní diferenciální rovnici druhého řádu do soustavy dvou rovnic řádu prvního a zapsat ve formě m-funkce (*f102a*)

[76] veličina důvěrně známá především fandům automobilů

$$\frac{ds_k}{dt} = v_k \qquad K = c_x \frac{4}{3r\rho}$$

$$\frac{dv_k}{dt} = g - Kv_k^2$$

Protože pro řešení budeme využívat standardní funkci `ode45` s předáváním parametrů do volané funkce, jsou první tři parametry m-funkce pevně dány[77]. Od čtvrtého parametru už význam parametrů závisí na nás. Čtvrtým parametrem bude pro nás koeficient odporu prostředí.

V tomto bodě řešení je vhodné ověřit, zda rovnice i přepis do MATLABu je správný a smysluplný. Nakresleme si jak bude vypadat prvních 10 s dráhy a rychlosti volného pádu železné kuličky o poloměru r=0.5 cm ve vakuu (c_x=0) a pro dvě hodnoty součinitele aerodynamického odporu (c_x=0.3 a c_x=0.9) a logicky odhadněme jakého výsledku (v hrubých rysech) bychom měli dosáhnout. Ve vakuu se bude rychlost neustále zvyšovat (odpovídá výše uvedenému analytickému řešení), při existenci nějakého odporu prostředí se rychlost ustálí na určité maximální hodnotě, která bude záviset na hodnotě c_x a to nepřímo úměrně. Čím vyšší hodnota c_x tím nižší ustálená rychlost.

Na obrázku 10-5 je vidět, že očekávané chování se potvrdilo. Dále je zřetelné, že maximální rychlosti se pro zvolené hodnoty parametrů dosáhne rychle.

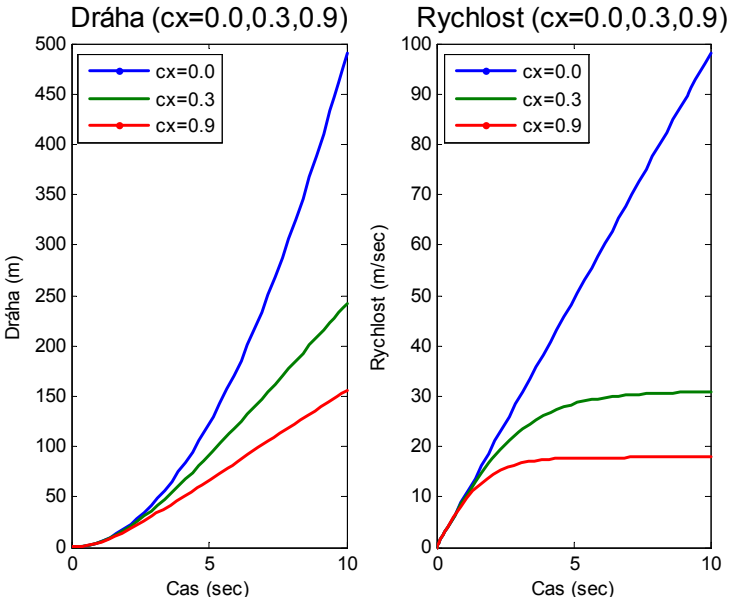
Po této kontrole můžeme pokračovat v řešení. Předpokládejme, že pomocí referenčního experimentu jsme naměřili čas t_{exp}=7.5 s při hloubce studny 20 m. Tuto vzdálenost urazí zvuk za čas t_z=20/330 s. Je tedy potřeba nalézt takovou hodnotu koeficientu odporu K, pro kterou za čas 7.5-t_z s bude mít řešení diferenciální rovnice hodnotu 20 - tedy řešit nulovou hodnotu funkce

$$s_k(K, t_{\exp} - t_z) - h = 0$$

$$t_{\exp} = 7.5 \quad h = 20 \quad t_z = \frac{h}{v_z} = \frac{20}{330}$$

```
function ds=f102a(t,s,flag,K)
% Příklad 10.2 HLOUBKA STUDNĚ
% diferenciální rov. volného pádu real. tělesa
%
%     ds=f102a(t,s,flag,K)
%
% s=[sk     ds=[dsk/dt
%    vk]        dvk/dt]

g=9.81;         % gravitační zrychleni
ds=zeros(2,1); % příprava sloup.vystup.vektoru
ds(1)=s(2);     % první derivace je rychlost
ds(2)=g-K*s(2)^2;
```

```
% skript zf102a.m
% Příklad 10.2 HLOUBKA STUDNĚ
ro=7800;                        % hustota železa
r=0.005;                       % poloměr kuličky
s0=[0;0];    % poč. podmínky (poloha,rychlost)
K1=0.0;           % koeficient odporu pro cx=0
K2=0.3*4/3/r/ro;                  % pro cx=0.3
K3=0.9*4/3/r/ro;                  % pro cx=0.9
[t1,s1]=ode45('f102a',[0;10],s0,[],K1);%cx=0.0
[t2,s2]=ode45('f102a',[0;10],s0,[],K2);%cx=0.3
[t3,s3]=ode45('f102a',[0;10],s0,[],K3);%cx=0.9
subplot(1,2,1); plot(t1,s1(:,1),…
        t2,s2(:,1),t3,s3(:,1));
title('Dráha (cx=0.0,0.3,0.9)');
xlabel('Čas (sec)'); ylabel('Dráha (m)');
legend('cx=0.0','cx=0.3','cx=0.9')
subplot(1,2,2);
plot(t1,s1(:,2),t2,s2(:,2),t3,s3(:,2));
title('Rychlost (cx=0.0,0.3,0.9)');
xlabel('Cas (sec)');
ylabel ('Rychlost (m/sec)');
legend('cx=0.0','cx=0.3','cx=0.9')
```



obrázek 10-5 Dráha a rychlost při různém odporu prostředí

[77] čas *t*, který se v našem případě nevyužívá, vektor hodnot *s* v čase *t* a příznak typu volání *flag*. Význam tohoto parametru se změnil. Od verze 6 není jeho hodnota nastavována.

Použijeme funkci `fzero`. Musíme vytvořit novou m-funkci (přepsat předchozí rovnici do m-funkce *f102b*), která bude pro funkcí `fzero` navržené hodnoty K dopočítávat rozdíl řešení diferenciální rovnice a hodnoty 20. Tato m-funkce je uvedena v rámečku. Vlastní řešení - vyhledání hodnoty koeficientu K z dat referenčního experimentu - získáme příkazem

```
>> K=fzero('f102b',1)
K      =     1.2871
```

```
function f=f102b(K)
% Příklad 10.2  HLOUBKA STUDNĚ
% rovnice pro urceni koef. odporu prostredi
%
%    f=f102b(K)
%

tk=7.5-20/330;                % cas padu
[t,h]=ode45('f102a',[0;tk],[0;0],[],K);
f=h(end,1)-20;
```

Nyní můžeme konečně přikročit k výpočtu vlastní úlohy. Zopakujme tvar základní rovnice, kterou je třeba řešit. Předpokládejme, že naměřený čas byl t_{exp}=31 sec.

$$s_k(K,t_k) - v_z(t_{\exp} - t_k) = 0$$

$$K = 1.2871 \quad v_z = 330 \quad t_{\exp} = 31$$

```
function f=f102c(tk)
% Příklad 10.2  HLOUBKA STUDNĚ
% rovnice pro urceni casu tk
%
%        f=f102c(tk)
%

tc=31;              %celkovy cas
vz=330;             %rychlost zvuku
K=1.2871;           %koeficient odporu prostredi
tint=[0;tk];        %casovy interval reseni
[t,h]=ode45('f102a',tint,[0;0],[],K);
f=h(end,1)-330*(tc-tk);
```

Musíme opět vytvořit m-funkci *f102c*, kterou použijeme ve standardní funkci `fzero`. Vlastní řešení - určení času pádu předmětu t_k z naměřeného času t_{exp} a koeficientu odporu prostředí K určeného z referenčního experimentu a určení hloubky h - získáme příkazy

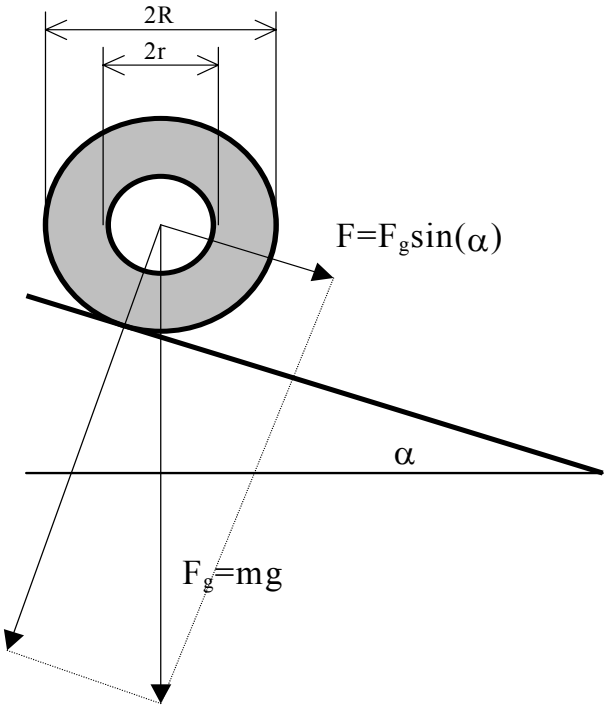
```
>> tk=fzero('f102c',31)
tk     =    30.7444
>> h=330*(31-tk)
h      =    84.3395
```

Pro zvolený čas 31 s a uvedené parametry vyšla hloubka studně 84.3 m. V případě kdyby se neuvažoval odpor vzduchu, by vyšla hloubka 2613 metrů.

10.3 Dutá koule

Představte si, že máte dvě stejně velké, těžké a stejně vypadající koule. Jedna z nich je dutá a druhá je plná. Vaším úkolem je určit, která z nich je dutá. Nemáte k dispozici žádné speciální vybavení (rentgen atd.) a nesmíte koule poškodit (provrtat atd.). Vymyslet jednoduchou použitelnou metodu pro jednoznačné odlišení není snadné.

Rozlišení koulí je možné podle jedné fyzikální vlastnosti, která je u obou koulí různá – moment setrvačnosti. Tato veličina závisí na rozložení měrné hmotnosti (hustoty) materiálu vzhledem k ose otáčení. Protože koule jsou jinak stejné (rozměr a váha) musí dutá koule mít větší hustotu materiálu při okraji a tudíž větší moment setrvačnosti.



obrázek 10-6 Schéma pro příklad Dutá koule

Jak se změna momentu setrvačnosti projeví ? Postavíme-li obě koule na nakloněnou rovinu a pustíme je, začnou se obě pohybovat. Koule s větším momentem setrvačnosti by se měla opožďovat za plnou koulí.

Mějme tedy dvě koule o poloměru R a hmotnosti m. Jedna z těchto koulí má uprostřed dutinu tvaru soustředné koule o poloměru r. Obě tyto koule umístíme na šikmou plochu o sklonu s úhlem α a pustíme

(viz obr. 10-6). Nyní nás bude zajímat časový průběh dráhy obou koulí. Budeme uvažovat pouze odpor prostředí a nikoliv odpor valivého tření[78]. Následující obrázek ukazuje dutou kouli umístěnou na šikmé ploše a rozložení sil vyvolané gravitačním působením. Obě koule se pokouší uvést do pohybu stejná gravitační síla F. Proti působí odpor prostředí F_{op} a síla setrvačná sestávající se ze dvou složek - F_p posuvný pohyb a F_o otáčivý pohyb. Rovnice popisující tyto dvě síly jsou

$$F_p = m\frac{d^2 s}{dt^2} \qquad M = RF_o = J\frac{d^2\varphi}{dt^2}$$

kde M je moment síly (síla působící na rameni R), J je moment setrvačnosti vzhledem k ose a φ je úhel otáčení.

Rozbor koeficientu odporu K pro kouli byl uveden v příkladě o měření hloubky studně. Základní rovnici popisující sledovaný systém - rovnováhu sil - lze popsat jako

$$F_{op} + F_p + F_o = F$$

$$K\left(\frac{ds}{dt}\right)^2 + m\frac{d^2 s}{dt^2} + \frac{1}{R}J\frac{d^2\varphi}{dt^2} = mg\sin(\alpha)$$

Souvislost mezi drahou s, kterou koule urazí po nakloněné rovině a úhlem otočení φ je dána vztahem $s=R\varphi$. Pro homogenní tělesa (konstantní hustota ρ) lze moment setrvačnosti vzhledem k ose vyjádřit pomocí objemového momentu setrvačnosti J_v vztahem $J=\rho J_v$. Pro kouli o poloměru R lze v tabulkách nalézt hodnotu J_{vp} a pro naší dutou kouli lze získat vztah J_{vd}. Hustota materiálu plné koule musí být m/V_R a duté koule m/V_{Rr}.

$$J_{vp} = \frac{8}{15}\pi R^5 \qquad J_{vd} = \frac{8}{15}\pi(R^5 - r^5)$$

$$\rho_p = \frac{3m}{4\pi R^3} \qquad \rho_d = \frac{3m}{4\pi(R^3 - r^3)}$$

$$J_p = \rho_p J_{vp} = \frac{2}{5}mR^2 \qquad J_d = \rho_d J_{vd} = \frac{2m(R^5 - r^5)}{5(R^3 - r^3)}$$

Po dosazení za koeficient odporu K a za moment setrvačnosti J pro plnou a dutou kouli a nahrazení úhlu otáčení φ uraženou drahou s, můžeme základní diferenciální rovnice zapsat jako

plná koule

$$K\left(\frac{ds_p}{dt}\right)^2 + m\frac{d^2 s_p}{dt^2} + \frac{1}{R^2}J_p\frac{d^2 s_p}{dt^2} = mg\sin(\alpha) \quad \Leftrightarrow \quad \frac{7}{5}\frac{d^2 s_p}{dt^2} + \frac{K}{m}\left(\frac{ds_p}{dt}\right)^2 = g\sin(\alpha)$$

dutá koule

$$K\left(\frac{ds_d}{dt}\right)^2 + m\frac{d^2 s_d}{dt^2} + \frac{1}{R^2}J_d\frac{d^2 s_d}{dt^2} = mg\sin(\alpha) \quad \Leftrightarrow \quad \left(1 + \frac{2(R^5 - r^5)}{5R^2(R^3 - r^3)}\right)\frac{d^2 s_d}{dt^2} + \frac{K}{m}\left(\frac{ds_d}{dt}\right)^2 = g\sin(\alpha)$$

Nyní můžeme zkusit spočítat časový průběh rozdílu drah dvou koulí ve fiktivním experimentu. Mějme dvě koule o průměru $2R$=0.1 m a váze m=2 kg. Tyto koule pustíme po šikmé ploše se sklonem φ=10°. Koeficient aerodynamického odporu je u obou koulí stejný a jeho hodnota je c_x=0.25. Bude nás zajímat prvních 5 s pohybu.

Pro řešení v MATLABu musíme přepsat původní diferenciální rovnici druhého řádu na soustavu dvou rovnic řádu prvého a vyjádřit ve formě m-funkce *f103*.

[78] Odpor valivého tření je závislý na poloměru a je tedy u obou koulí stejný. Budeme-li sledovat rozdíl dráhy obou koulí, pak tento rozdíl na valivém odporu nezávisí.

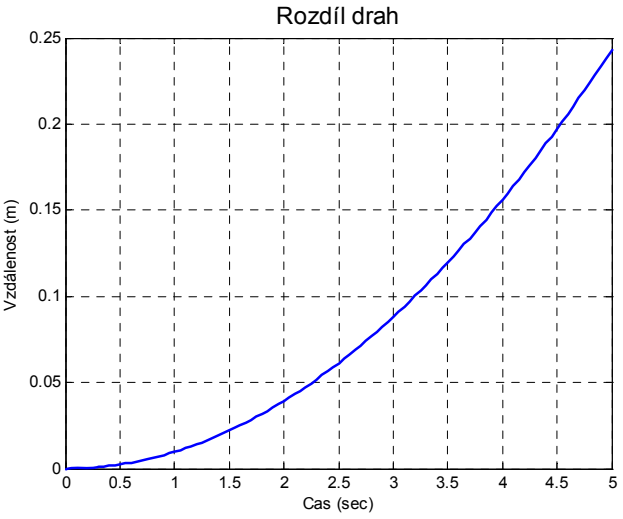
$$k_1 = \frac{K}{m} \qquad k_2 = g\sin(\alpha)$$

$$k_3 = \frac{7}{5} \quad nebo \quad 1 + \frac{2(R^5 - r^5)}{5R^2(R^3 - r^3)}$$

$$\frac{ds}{dt} = v \qquad \frac{dv}{dt} = \frac{k_2 - k_1 v^2}{k_3}$$

Vlastní řešení se provede pomocí skriptu *zf103.m* včetně vykreslení časového průběhu rozdílu drah obou koulí (viz obr. 10-7).

```
function ds=f103(t,s,flag,k1,k2,k3)
% Příklad 10.3 DVĚ KOULE
% dif. rov.pohybu koule na šikmé ploše
%
% ds=f103(t,s,flag,k1,k2,k3)
%
% s=[sk      ds=[dsk/dt
%    vk]         dvk/dt]

ds=zeros(2,1); % priprava
sloupc.vystup.vektoru
ds(1)=s(2);     % derivace je rychlost
ds(2)=(k2-k1*s(2)^2)/k3;
```



obrázek 10-7 Rozdíl dráhy plné a duté koule

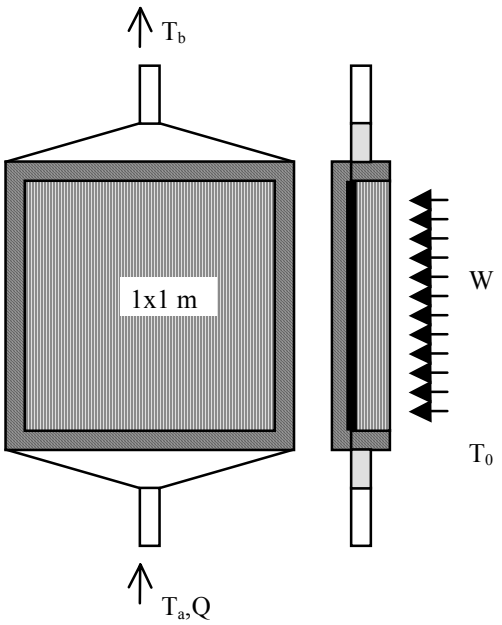
```
% skript zf103.m
% Příklad 10.3 DVĚ KOULE
% zadané parametry
R=0.05;              % poloměr koule
r=0.02;              % poloměr dutiny
cx=0.25;             % koef.odporu vzduchu
m=2;                 % hmotnost koule
alfa=2*pi/360*10;    % úhel plošiny 10
stupnu
% pomocné proměnné
K=cx*pi*R^2;         % odpor koule
k1=K/m;
k2=9.81*sin(alfa);
Jp=7/5;
Jd=1+2/5*(R^5-r^5)/R^2/(R^3-r^3);
% čas, ve kterém nás zajímá výsledek
t=0:0.05:5;
% výpočet pro plnou kouli
[tp,sp]=ode45('f103',t,[0;0],[],k1,k2,Jp)
;
% výpočet pro dutou kouli
[td,sd]=ode45('f103',t,[0;0],[],k1,k2,Jd)
;
plot(t,(sp(:,1)-sd(:,1)))
title('Rozdíl drah')
xlabel('Cas (sec)');
ylabel('Vzdálenost (m)')
```

10.4 Sluneční kolektor

Námět na tuto úlohu vzešel z rozhovoru s jedním známým, který začal vyrábět, kromě jiného, sluneční kolektory. Pravil, že jsou perfektní neboť voda se v nich ohřeje až na 80 °C. Tato informace je sice zajímavá, ale když se nad problémem zamyslíte, naprosto nic nevypovídá o kvalitě vlastního kolektoru[79]. Základní informací by mělo být o kolik stupňů dokáže kolektor o jednotkové ploše ohřát určitý průtok. Ještě lepší by byla informace o účinnosti kolektoru. Jenže pro určení účinnosti je potřeba znát teoretický výkon. A jaký je vlastně teoretický výkon kolektoru ?

Samozřejmě základní veličinou, která tento výkon ovlivňuje, je dopadající výkon slunečního záření. Za ideálních podmínek je asi W=750 $Wm^{-2}s^{-1}$. Dalším faktorem jsou tepelné ztráty. Část obrácená ke slunci nemůže být izolována na 100% a část tepla (úměrná rozdílu teploty okolí a teploty ohřívané kapaliny) bude unikat. Zkusme namodelovat dynamické chování takového ideálního slunečního kolektoru o rozměrech $l \times s$ (délka l=1 m a šířka s=1 m) s vrstvou vody o tloušťce d=1 cm (viz obr.10-8). Sluneční záření prochází beze ztráty skleněnou tabulí se zanedbatelnou tepelnou kapacitou a dopadá (skrz vrstvu vody) na černé dno, kde se beze zbytku adsorbuje a přeměňuje v teplo. Toto teplo ohřívá vodu a ta i sklo. Při spodní hraně vstupuje (po šířce rovnoměrně rozložená) studená voda o teplotě T_a, průtoku Q a na horním konci je teplá voda o teplotě T_b odváděna. Voda se postupně ohřívá od jednoho konce k druhému. Vzhledem k okolní teplotě T_0, dochází na skle k odvodu tepla (s koeficientem prostupu tepla α). Ostatní strany kolektoru jsou ideálně izolovány a k úniku tepla nedochází.

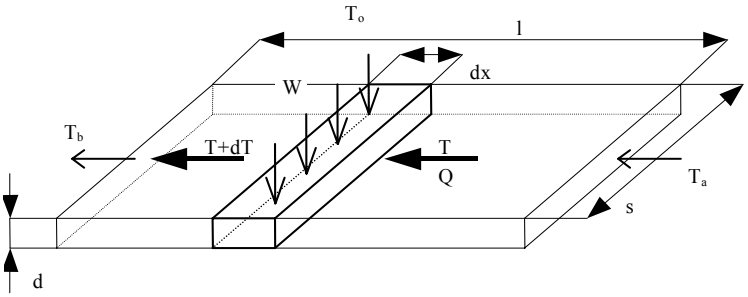
[79] necháme-li hrnec s vodou v autě za horkého letního odpoledne, také se ohřeje na zajímavou teplotu a přesto auto za dobrý sluneční kolektor asi považovat nebudeme



obrázek 10-8 Schéma I pro příklad Sluneční kolektor

Uvnitř kolektoru v podélném směru (směru proudění) nedochází k promíchávání (pístový tok) a ve směru mezi dnem a sklem budeme uvažovat charakteristickou teplotu[80] *T*. Protože uvažujeme, že voda do kolektoru vstupuje i vystupuje rovnoměrně podél příčného směru, budeme předpokládat že i teplota v příčném směru je konstantní. Za takto zavedených předpokladů je charakteristická teplota *T* funkcí pouze času a délky *x*.

Rovnici popisující chování teploty *T* získáme z tepelné bilance objemu kapaliny o délce dx nakreslené na obrázku 10-9. Musí platit, že součet tepla vstupující je roven součtu tepla vystupujícího a akumulace. Je-li *Q* objemový průtok



obrázek 10-9 Schéma II pro příklad Sluneční kolektor

kapaliny, ρ její hustota a c_m měrná tepelná kapacita kapaliny, pak lze uvedenou bilanci vyjádřit rovnicí

$$Q\rho c_m T + W dS = Q\rho c_m (T + dT) + \alpha\, dS (T - T_o) + d\, V \rho c_m \frac{dT}{dt}$$

$$kde \quad dS = s\, dx \quad dV = d\, Sd$$

$$\frac{W}{d\ \rho c_m} = \frac{Q}{sd}\frac{\partial T}{\partial x} + \frac{\alpha}{d\ \rho c_m}(T - T_o) + \frac{\partial T}{\partial t} \quad \Leftrightarrow \quad \frac{W + \alpha T_o}{d\ \rho c_m} = \frac{Q}{sd}\frac{\partial T}{\partial x} + \frac{\alpha}{d\ \rho c_m} T + \frac{\partial T}{\partial t}$$

$$počáteční\ a\ okrajové\ podmínky : T(x = 0, t) = T_a \quad T(x, t = 0) = T_a$$

Získali jsme jednoduchou parciální diferenciální rovnici, kterou je nyní nutné řešit. Numerických řešení je možné použít několik. My zvolíme nejjednodušší metodu, při které derivaci podle délkové proměnné *x* nahradíme podílem diferencí (kolektor rozdělíme na n příčných dílů). Předpokládejme, že teplota uvnitř i-tého dílu je rovná výstupní teplotě T_i a teplota $T_{i=0}=T_a$. Pro zjednodušení dalších zápisů zavedeme substituce a můžeme pro i-tý díl zapsat rovnici

$$a = \frac{W + \alpha T_o}{d\rho c_m} \quad b = \frac{Q}{sd} \quad c = \frac{\alpha}{d\rho c_m} \quad \Delta x = x_i - x_{i-1} = \frac{l}{n} \quad \bar{b} = \frac{b}{\Delta x} = \frac{nQ}{lsd}$$

$$\frac{dT_i}{dt} = a - b\frac{T_i - T_{i-1}}{\Delta x} - cT_i \quad \Leftrightarrow \quad \frac{dT_i}{dt} = a - (\frac{b}{\Delta x} + c)T_i + \frac{b}{\Delta x} T_{i-1}$$

$$\frac{dT_i}{dt} = a - (\bar{b} + c)T_i + \bar{b}\, T_{i-1}$$

Soustavu diferenciálních rovnic, na kterou je nyní řešení parciální diferenciální rovnice převedeno, lze vyjádřit v maticovém tvaru **M*T**+**m**=d**T**/dt, kde **M**(n x n+1), **T**(n+1 x 1) a **m**(n x 1)

[80] místo teploty s rozložením v příčném směru budeme uvažovat střední (nebo charakteristickou) teplotu, která zastupuje ve výpočtech teplotu rozloženou

$$\begin{bmatrix} \bar{b} & -(\bar{b}+c) & 0 & 0 & \dots & 0 & 0 \\ 0 & \bar{b} & -(\bar{b}+c) & 0 & \dots & 0 & 0 \\ 0 & 0 & \bar{b} & -(\bar{b}+c) & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -(\bar{b}+c) & 0 \\ 0 & 0 & 0 & 0 & \dots & \bar{b} & -(\bar{b}+c) \end{bmatrix} * \begin{bmatrix} T_a \\ T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_{n-1} \\ T_n \end{bmatrix} + \begin{bmatrix} a \\ a \\ a \\ \vdots \\ a \\ a \end{bmatrix} = \begin{bmatrix} \frac{dT_1}{dt} \\ \frac{dT_2}{dt} \\ \frac{dT_3}{dt} \\ \vdots \\ \frac{dT_{n-1}}{dt} \\ \frac{dT_n}{dt} \end{bmatrix}$$

Takto připravenou soustavu diferenciálních rovnic již můžeme v MATLABu přímo řešit. Definujme nejprve hodnoty konstant, potom provedeme výpočty pomocných proměnných závislých na zvolených konstantách a dalších volbách. Volba počtu dílů n ovlivňuje jednak přesnost náhrady parciální diferenciální rovnice soustavou diferenčně-diferenciálních rovnic a jednak rozměry pracovních matic i výsledku. Výsledkem řešení bude matice, kde každý sloupec odpovídá časovému vývoji teploty T v každém proužku. Sloupců bude tedy n. Počet řádků odpovídá počtu časových bodů, ve kterých je získáno řešení. Primárně nás zajímá časový vývoj teploty vytékající vody - teplota T v n-tém dílu. Zápis této soustavy rovnic je pak velmi jednoduché (viz m-funkci *f104* v rámečku).

```
function dT=f104(t,T,flag,M,m,Ta)
% Příklad 10.4
%    dT=kolektor(t,T,flag,M,m,Ta)
%
dT=M*[Ta;T]+m;
```

Úlohu budeme řešit pro rozměry kolektoru 1x1 metr. Všechny parametry a posloupnost příkazů je ve skriptu *zaf104.m*. Použité hodnoty fyzikálních konstant (výkon slunečního záření a koeficient prostupu tepla) jsou diskutabilní. Výkon slunečního záření W je sice znám, ale jeho hodnota v konkrétních podmínkách závisí na zeměpisné šířce, denní době a oblačnosti. Stejně tak koeficient prostupu tepla α je závislý na konstrukčním řešení, rychlosti proudění[81] obou medií a také na teplotě[82]. Proto uvedené hodnoty jsou pouze ilustrační.

Výpočet provedeme pro dva průtoky Q_1=1 l/min a Q_2=0.05 l/min. Pro oba průtoky spočítáme matice teplotních rozložení T_1 a T_2. Pro průtok Q_2 je nutné přepočítat matici soustavy M a provést výpočet matice teplotního rozložení T2 znova. Tentokráte v jiném časovém měřítku.

```
% skript zaf104.m
% Příklad 10.4 KOLEKTOR
l=1; s=1;          % rozměry - délka, šířka
n=100; d=l/n;      % počet dělení a délka úseku
ro=998; cm=4180;   % hustota vody a tep.kapacita
alfa=20; W=750;  % koef.prostupu, výkon sl.zař.
To=25;  Ta=25;     % teploty okolí a vody
% pomocné proměnné
a=(W+alfa*To)/d/ro/cm; c=alfa/d/ro/cm;
m=ones(n,1)*a;
% poč.pod. (naplnění vodou o vstupní teplotě)
T0=ones(n,1)*Ta;
% výpočet pro průtok 1 litr za minutu
Q=0.001/60;        % průtok 1 litr za minutu
b=n*Q/l/s/d;   M1=eye(n,n+1)*b;
M2=[zeros(n,1),eye(n,n)]*(b+c); M=M1-M2;
t=0:10:20*60;      % 0-20 minut po 10 sec
[t1,T1]=ode45('f104',t,T0,[],M,m,Ta);
% výpočet pro průtok 0.05 litrů za minutu
Q=0.00005/60;      % průtok 0.05 litrů za minutu
b=n*Q/l/s/d;   M1=eye(n,n+1)*b;
M2=[zeros(n,1),eye(n,n)]*(b+c); M=M1-M2;
t=0:60:3*3600;    % 0-180 minut po 60 sec.
[t2,T2]=ode45('f104',t,T0,[],M,m,Ta);
mer1=[1:6:121];  % každý šestý bod tj. minuta
mer2=[1:10:181]; % každý desátý bod tj. 10 min
x=[0.01:0.01:1]; % poloha sledované teploty
subplot(221); plot(t1/60,T1(:,100));
title('Časový průběh výstupní teploty');
ylabel('Teplota'); xlabel('Čas (min)');
subplot(222); plot(x,T1(mer1,:)');
title('Rozložení teplot v časech po 1 min');
ylabel('Teplota'); xlabel('Vzdálenost (m)');
subplot(223); plot(t2/60,T2(:,100));
title('Časový průběh výstupní teploty');
ylabel('Teplota'); xlabel('Čas (min)');
subplot(224); plot(x,T2(mer2,:)');
title('Rozložení teplot v časech po 10 min');
ylabel('Teplota'); xlabel('Vzdálenost (m)');
```

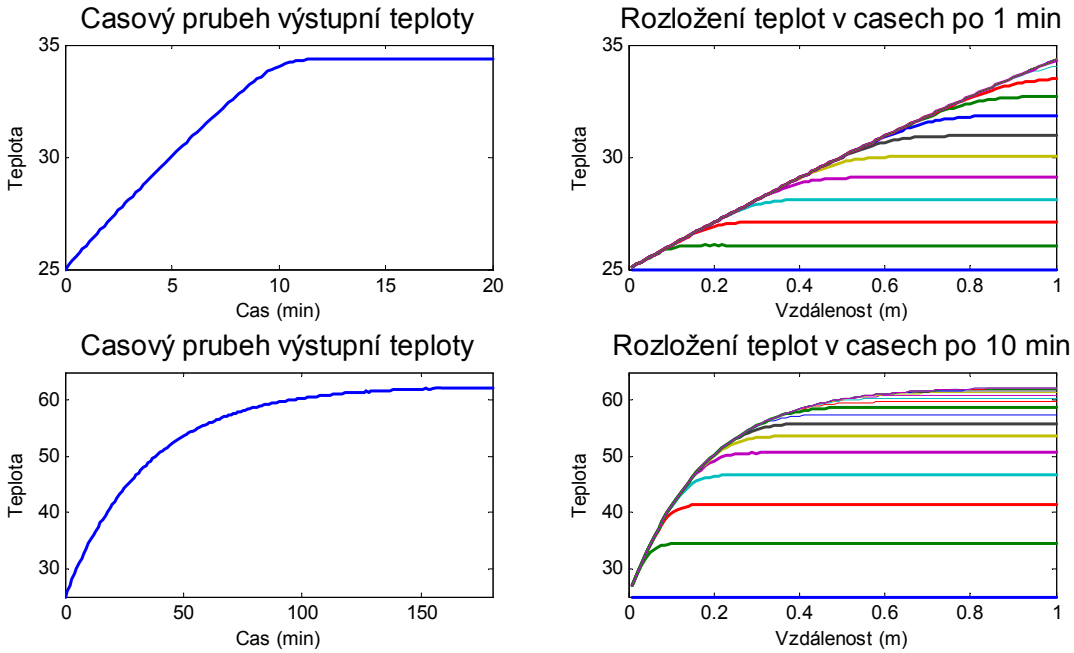
[81] je zřejmé že bude-li kolektor ofukován větrem bude ochlazování výraznější

[82] při vyšších teplotách se mění mechanismus předávání tepla - mimo vedení tepla začíná hrát významnější úlohu i odvod tepla sáláním

```
% skript zbf104.m
% Příklad 10.4 KOLEKTOR
N=50;                % počet bodů
Q2=0.001/60;         % 1 l/min
Q1=0.00005/60;       % 0.05 l/min
Q=Q1:(Q2-Q1)/(N-1):Q2;  % průtoky
mq=m;  T=zeros(N,1); Tt=T;
for i=1:N
   Tt(i)=Ta+W/ro/cm/Q(i);% teplota
   b=n*Q(i)/l/s/d; mq(1)=m(1)+b*Ta;
   M1=[zeros(1,n); eye(n-1,n)*b];
   M2=eye(n,n)*(b+c); M=M1-M2;
   Tp=M\(-mq);  % řešení ust.tepl.
   T(i)=Tp(n);  % konc. teploty
   uc(i)=T(i)/Tt(i); % účinnost
end
Q=Q*60000;        % hodnoty v l/min
subplot(211), plot(Q,Tt,':',Q,T)
legend('teoretická','skutecná')
title('Teoretická a skutecná
výstupní teplota')
ylabel('Teplota [C]')
subplot(212), plot(Q,uc)
title('Ucinnost kolektoru')
xlabel('Průtok [l/min]')
ylabel('Ucinnost')
```

Na závěr se vykreslí výsledky ve formě čtyř grafů v jednom obrázku (10-10). První graf představuje časový průběh výstupní teploty a druhý graf představuje vývoj teplotního rozložení po délce výměníku v časových okamžicích po 1 minutě. Oba pro průtok Q_1. Zbývající dva grafy zobrazují totéž pro průtok Q2 a čas po 10 min.

Vraťme se nyní k otázce položené v úvodu tj. jakou účinnost má námi simulovaný kolektor. Odpověď na ní není jednoduchá neboť účinnost kolektoru (definovaná jako poměr skutečné a teoreticky dosažitelné výstupní teploty) závisí na průtoku. Tepelná účinnost je daná především "mírou izolovanosti" vyjádřené v našem případě koeficientem prostupu tepla. Spočítejme tedy nyní závislost účinnosti na průtoku pro náš případ pomocí skriptu *zbf104.m*. Pro výpočet ustáleného stavu využijeme výše uvedenou soustavu diferenciálních rovnic - v ustáleném stavu jsou derivace nulové. Řešení potom přejde na řešení soustavy n rovnic o n neznámých. Vlastní řešení v MATLABu je uvedeno ve formě skriptu v rámečku. Výsledkem jsou dva grafy na obr. 10-11. První představuje závislost teoretické a vypočítané výstupní teploty kolektoru na průtoku v rozmezí 0.05 až 1 l/min. Druhý představuje účinnost počítanou jako podíl vypočítané a teoretické ustálené výstupní teploty. Pro průtok 1 l/min je účinnost cca 96% a pro průtok 0.05 l/min je účinnost cca 28%. Výsledek odpovídá fyzikální představě – při velkém průtoku je teplota ve výměníku malá a tepelné ztráty (snížení účinnosti) závislé na rozdílu teplot ve výměníku a okolí jsou malé.

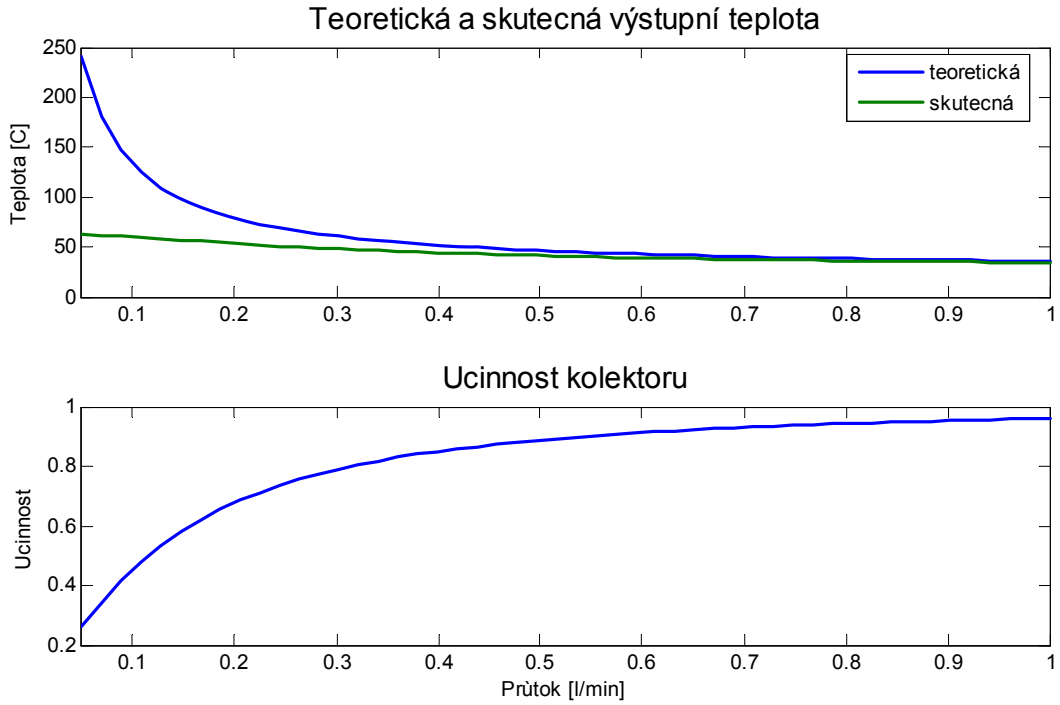


obrázek 10-10 Časový průběh výstupní teploty a rozložení teplot

Zajímavé jsou teoretické teploty při malých průtocích – hodnoty překračují vysoko bod varu vody.

10.5 Titrace slabé kyseliny slabou zásadou (MATLAB)

Jedním z numericky obtížných příkladů, kdy se řešení mění v rozsahu deseti řádů, je výpočet průběhu titrační křivky. Jde o jeden ze standardních chemických problémů, který chemici obvykle počítají pouze přibližně s využitím zjednodušení. Ne proto, že by to neuměli, ale proto, že exaktní řešení vede na polynomiální rovnici čtvrtého stupně s hodnotami koeficientů lišících se i o dvacet řádů.



obrázek 10-11 Účinnost kolektoru

Nejprve co to je titrační křivka. Titrační křivka je průběh koncentrace[83] vodíkových iontů (označovaný obvykle $[H^+]$ vyjádřený v hodnotě pH[84]) v titrační směsi v závislosti na přídavku titračního činidla. Představme si, že máme nádobu naplněnou objemem V_{A0}=0.05 litru kyseliny A o koncentraci c_{A0}=0.05 mol/litr. Do nádoby budeme přidávat zásadu B o koncentraci c_{B0}=0.08 mol/litr. Zajímá nás průběh pH v nádobě v závislosti na celkovém přidaném objemu V_B látky B. Titrace probíhá ve vodném prostředí tj. rozpouštědlem obou látek je voda. Aktuální koncentrace obou látek v závislosti na přidaném objemu V_B je za předpokladu dokonalého míchání a dána vztahem

$$c_A = \frac{c_{A0}V_{A0}}{V_{A0}+V_B} \qquad c_B = \frac{c_{B0}V_B}{V_{A0}+V_B}$$

Schopnost uvolňování vodíkových iontů (disociace) je vyjádřena tzv. disociační konstantou. Přepokládejme, že budeme titrovat kyselinu octovou (K_A = 1.75 10^{-5}) etylaminem (K_B = 4.37 10^{-4}). V tomto případě jde o titraci slabé kyseliny slabou zásadou, což je případ kdy nelze zjednodušení[85] použít.

Jaké jsou tedy vztahy popisující tento případ? Jak kyselina octová tak etylamin disociují pouze do prvního stupně. Disociační konstanty vyjadřují, že součin koncentrací disociovaných částí vztažený na koncentraci nedisociované sloučeniny je konstantní a jsou tedy definovány vztahy

$$K_A = \frac{[H^+][A^-]}{[HA]} \qquad K_B = \frac{[B^+][OH^-]}{[BOH]}$$

Definice disociační konstanty vody je stejná, ale vzhledem k tomu, že voda disociuje velmi málo, předpokládá se, že koncentrace nedisociované vody je jedna a výsledný vztah se v tomto smyslu zjednodušuje.

[83] i toto je zjednodušení reálného chování, správně by se mělo počítat s aktivitami místo koncentracemi iontů

[84] záporně vzatý dekadický logaritmus koncentrace vodíkových iontů tj. pH = -log[H^+]

[85] zjednodušení ve smyslu zavedení určitých předpokladů vedoucích na snížení řádu výsledného polynomu

$$K_{voda}=\frac{[H^+][OH^-]}{[H_2O]}=10^{-14} \quad \approx \quad K_{voda}=[H^+][OH^-]=10^{-14}$$

Vztah mezi molární koncentrací a koncentrací vodíkových (hydroxylových) iontů je dán zákonem zachování hmoty – počet molů látky A i B zůstává bez ohledu na disociaci konstantní (v daném objemu)

$$c_A=[A^-]+[HA] \qquad c_B=[B^+]+[BOH]$$

Poslední rovnicí je rovnice disociace

$$HA+BOH \leftrightarrow H^+ + A^- + B^+ + OH^-$$

Vyjdeme-li nyní z podmínky elektroneutrality (součet záporných a kladných nábojů v roztoku musí být nula) můžeme sestavit výslednou rovnici popisující koncentraci vodíkových iontů v závislosti na aktuální koncentraci obou látek a jejich vlastnostech

$$[H^+]+[B^+]=[A^-]+[OH^-]$$

Z předchozích vztahů můžeme vyjádřit koncentrace iontů kyseliny [A^-], zásady [B^+] a hydroxylových iontů [OH^-] jako $[A^-]=\frac{c_A K_A}{K_A+[H^+]}$ $[B^+]=\frac{c_B K_B}{K_B+[OH^-]}$ $[OH^-]=\frac{K_{voda}}{[H^+]}$

Po dosazení do podmínky neutrality dostaneme rovnici jejímž řešením je koncentrace vodíkových iontů pro dané podmínky

$$[H^+]+\frac{c_B K_B}{K_B+\frac{K_{voda}}{[H^+]}}=\frac{c_A K_A}{K_A+[H^+]}+\frac{K_{voda}}{[H^+]}$$

a po úpravě pak výslednou bikvadratickou rovnici pro koncentraci vodíkových iontů

$$[H^+]^4+$$
$$+[H^+]^3\left\{K_A+\frac{K_{voda}}{K_B}+c_B\right\}+$$
$$+[H^+]^2\left\{K_A(c_B-c_A)+K_{voda}\left(\frac{K_A}{K_B}-1\right)\right\}-$$
$$-[H^+]\left\{K_A K_{voda}\left(\frac{c_A}{K_B}-1\right)+\frac{K_{voda}^2}{K_B}\right\}-$$
$$-K_A\frac{K_{voda}^2}{K_B}=0$$

```
% skript z105.m
% Příklad 10.5 TITRACE
VA0=0.05;                       % počáteční objem
cA0=0.05; cB0=0.08;             % výchozí koncentrace
% disociační konstanty
KA=1.75e-5; KB=4.37e-4; Kv=1e-14;
% přidávaný objem (po kapkách - 3 kapky=1 ml)
VB=0:0.00033:0.1;
x=Kv^2/KB; x1=x*KA; x2=Kv*(KA/KB-1);
for k=1:length(VB),
    cA=cA0*VA0/(VA0+VB(k));  % látka A
    cB=cB0*VB(k)/(VA0+VB(k)); % látka B
    p=[1,KA+Kv/KB+cB,KA*(cB-cA)+x2,...
       -KA*Kv*(cA/KB-1)-x,-x1];
    h=roots(p);              % řešení polynomu
% rozhodnutí o výběru řešení (reálné, kladné)
    if (isreal(h(1))==1)&(h(1)>0), pH(k)=h(1); end
    if (isreal(h(2))==1)&(h(2)>0), pH(k)=h(2); end
    if (isreal(h(3))==1)&(h(3)>0), pH(k)=h(3); end
    if (isreal(h(4))==1)&(h(4)>0), pH(k)=h(4); end
end
pH=-log10(pH); plot(VB,pH)
title('Titrace slabé kyseliny slabou zásadou')
xlabel('objem zásady (ml)'),ylabel('pH'),grid
```

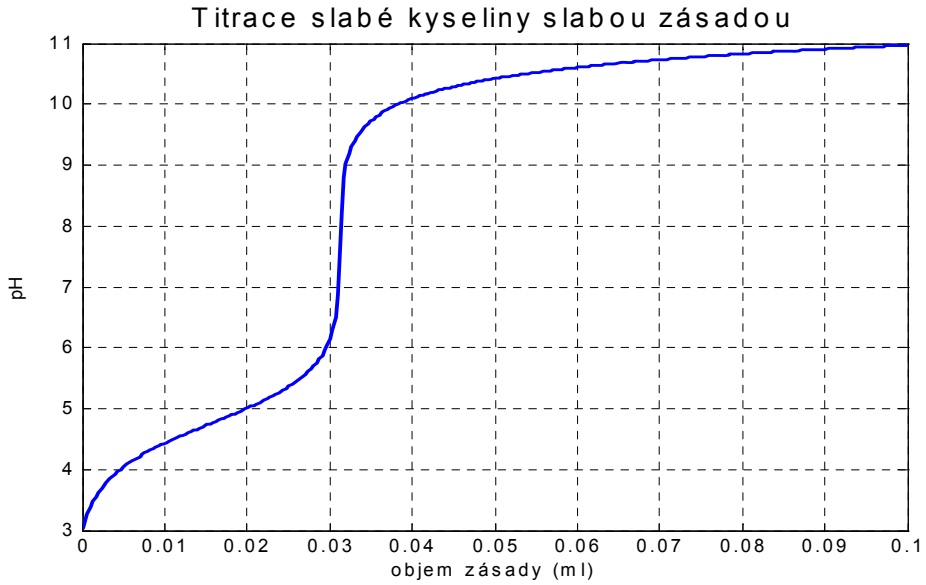
Jde tedy o polynomiální rovnici 4. stupně s proměnnými koeficienty. Situaci komplikuje existence 4 řešení z nichž pouze jedno má fyzikální význam. Fyzikálně přípustné řešení musí být reálné a kladné. Průběh pH zobrazený na obrázku 10-12 pak můžeme získat např. následující posloupností příkazů (zapsané formou skriptu *z105.m*).

Řešení se pohybuje od hodnoty cca 10^{-3} do hodnoty 10^{-11} tj. v rozsahu 8 řádů.
Koeficienty polynomu

$$x^4+bx^3+cx^2+dx+e=0$$

se mění v rozsazích

b ∈ <+1.7500e-005 ~ +0.0533> c ∈ <-8.7500e-007 ~ +6.4162e-007>
d ∈ <-1.9848e-017 ~ -6.4997e-018> e = -4.0046e-030



obrázek 10-12 Titrační křivka

10.6 Chemická reakce

Druhým příkladem z oblasti chemie je typická úloha reakční kinetiky tj. popis časových změn koncentrací reagujících a vznikajících látek při chemické reakci. Uvedený příklad je pouze ilustrační tj. nepopisuje žádnou konkrétní reakci, ale měl by sloužit k získání představy jak se úlohy tohoto typu řeší a jak lze využít možnosti MATLABu.

Do prázdné válcové nádoby o průměru 25 cm nalijeme V_A roztoku látky A o koncentraci x_{A0} a V_B roztoku látky B o koncentraci x_{B0}. Oba roztoky jsou ve stejném inertním rozpouštědle I a mají teplotu okolí T_o=20 ºC. Látky A a B spolu reagují dle rovnice

$$A + 2B \xrightarrow{k_1} C + \Delta H \quad k_1 = e^{\ln(B_1) - \frac{A_1}{273+T}}$$

$$C \xrightarrow{k_2} A + 2B - \Delta H \quad k_2 = e^{\ln(B_2) - \frac{A_2}{273+T}}$$

Za následujících předpokladů

- dokonalé míchání směsi
- nedochází ke skupenským změnám
- neuvažuje se tepelná kapacita zařízení
- konstantní teplota okolí
- výměna tepla s okolím neprobíhá dnem nádoby
- hustota směsi je lineární kombinací hustot složek
- měrná tepelná kapacita směsi je lineární kombinací kapacit složek

určete časový průběh koncentrací všech tří složek a teplotu reakční směsi.

Poznámky k úloze: Jde o vratnou exotermickou reakci tj. vznikající produkty reakce se rozpadají na výchozí látky a při přímé reakci se teplo uvolňuje a při vratné reakci se spotřebovává. Obě reakce probíhají s různou rychlostí danou rychlostní konstantou závislou na teplotě. Dále probíhá výměna tepla s okolím. Kromě toho původní a vznikající látky mají různou hustotu tedy se v průběhu reakce mění i objem reakční směsi (a tudíž i plocha pro přestup tepla).

Hodnoty a rozměry potřebných parametrů jsou shrnuty v následující tabulce.

Označení	Rozměr	Hodnota	Význam
A_1	K	13132	Parametr A (Arheniův vztah) →
B_1	s^{-1}	5.8354e+017	Parametr B (Arheniův vztah) →
A_2	K	7470	Parametr A (Arheniův vztah) ←
B_2	s^{-1}	1.1836e+009	Parametr B (Arheniův vztah) ←
ΔH	J $kmol^{-1}$	1.2e+007	Reakční teplo
M_A	kg $kmol^{-1}$	37	Molekulová hmotnost látky A
$\rho_{A,20}$	kg m^{-3}	700	Hustota látky A
c_A	J K^{-1} kg^{-1}	600	Tepelná kapacita látky A
M_B	kg $kmol^{-1}$	23	Molekulová hmotnost látky B
$\rho_{B,20}$	kg m^{-3}	800	Hustota látky B
c_B	J K^{-1} kg^{-1}	750	Tepelná kapacita látky B
$\rho_{C,20}$	kg m^{-3}	825	Hustota látky C
c_C	J K^{-1} kg^{-1}	800	Tepelná kapacita látky C
M_I	kg $kmol^{-1}$	18	Molekulová hmotnost látky I
$\rho_{I,20}$	kg m^{-3}	1000	Hustota látky I
c_I	J K^{-1} kg^{-1}	800	Tepelná kapacita látky I
α	J s^{-1} K^{-1} m^{-2}	50	Koeficient prostupu tepla z reaktoru do okolí

Sledované proměnné s počátečními hodnotami jsou uvedeny v další tabulce

Označení	Rozměr	Poč. hodnota	Význam
x_{A0}	kg kg^{-1}	0.75	Koncentrace látky A (v objemu V_A)
V_A	m^{-3}	0.004	Objem roztoku látky A
x_{B0}	kg kg^{-1}	0.45	Koncentrace látky B (v objemu V_B)
V_B	m^{-3}	0.003	Objem roztoku látky B
x_{C0}	kg kg^{-1}	0.0	Koncentrace látky C (v objemu V_C)
V_C	m^{-3}	0.0	Objem roztoku látky C
T	°C	T_o	Teplota v reaktoru

Změna koncentrace *x* jednotlivých složek v čase pro zadanou chemickou reakci je popsána následující diferenciální rovnicí

$$\frac{dx_A}{dt} = \frac{1}{2}\frac{dx_B}{dt} = -\frac{dx_C}{dt} = -k_1 x_A x_B^2 + k_2 x_C \quad [\,s^{-1}]$$

V jakých jednotkách je vyjádřena koncentrace je v podstatě jedno. Záleží na tom, co všechno další se bude dopočítávat. Pro náš případ, kdy bude docházet ke změně objemu, je asi nejvhodnější vyjádřit koncentrace jako molární zlomek[86] a sledovat počet molů jednotlivých složek.

$$N = n_A + n_B + n_C + n_I \quad x_{nA} = \frac{n_A}{N} \quad x_{nB} = \frac{n_B}{N} \quad x_{nC} = \frac{n_C}{N}$$

kde n_x jsou počty molů jednotlivých složek a N je celkový počet molů v reakční směsi

Přechod na počet molů pro látku A (n_A) ukazuje následující rovnice

$$\frac{dx_{nA}}{dt} = -k_1 x_{nA} x_{nB}^2 + k_2 x_{nC} \quad \Leftrightarrow \quad \frac{d}{dt}\left(\frac{n_A}{N}\right) = \frac{N - n_A}{N^2}\frac{dn_A}{dt} = -k_1\left(\frac{n_A}{N}\right)\left(\frac{n_B}{N}\right)^2 + k_2\left(\frac{n_C}{N}\right)$$

[86] molární zlomek je definován jako počet molů dané látky ku celkovému počtu molů

Popis chemické reakce pomocí časové změny počtu molů pro všechny tři složky je

$$\frac{dn_A}{dt} = -k_1 \frac{n_A n_B^2}{N(N-n_A)} + k_2 \frac{n_C N}{N-n_A} \qquad \frac{dn_B}{dt} = 2\left[-k_1 \frac{n_A n_B^2}{N(N-n_B)} + k_2 \frac{n_C N}{N-n_B} \right]$$

$$\frac{dn_C}{dt} = k_1 \frac{n_A n_B^2}{N(N-n_C)} - k_2 \frac{n_C N}{N-n_C}$$

Počáteční počet molů složky A,B a I je dán objemem roztoku a jeho koncentrací

$$\underbrace{\frac{n_{A0} M_A}{\rho_A}}_{\text{objem látky A}} + \underbrace{\frac{n_{IA} M_I}{\rho_I}}_{\text{objem látky I}} = V_A \qquad \frac{n_{A0} M_A}{\underbrace{n_{A0} M_A}_{\text{hmotnost látky A}} + \underbrace{n_{IA} M_I}_{\text{hmotnost látky I}}} = x_{A0}$$

konkrétně (počáteční koncentrace x_{A0} a x_{B0} jsou vyjádřeny jako hmotnostní zlomek)

$$n_{A0} = \frac{x_{A0}\rho_I \rho_A V_A}{M_A[\rho_A + (\rho_I - \rho_A)x_{A0}]} \qquad n_{IA} = \frac{(1-x_{A0})\rho_I \rho_A V_A}{M_I[\rho_A + (\rho_I - \rho_A)x_{A0}]}$$

$$n_{B0} = \frac{x_{B0}\rho_I \rho_B V_B}{M_B[\rho_B + (\rho_I - \rho_B)x_{B0}]} \qquad n_{IB} = \frac{(1-x_{B0})\rho_I \rho_B V_B}{M_I[\rho_B + (\rho_I - \rho_B)x_{B0}]}$$

Přestupní plocha je závislá na výšce kapaliny v reaktoru. Výška h je daná aktuálním objemem V (počtem molů) složek a geometrickým tvarem reaktoru (válec o průměru D)

$$V = \frac{n_A M_A}{\rho_A} + \frac{n_B M_B}{\rho_B} + \frac{n_I M_I}{\rho_I} = \pi \frac{D^2}{4} h$$

$$S = \pi \frac{D^2}{4} + \pi D h = \pi \frac{D^2}{4} + \frac{4}{D}\left(\frac{n_A M_A}{\rho_A} + \frac{n_B M_B}{\rho_B} + \frac{n_I M_I}{\rho_I} \right)$$

Tepelná bilance – teplo vyvinuté reakcí je spotřebováno na ohřátí reakční směsi (akumulace) a ztráty do okolí

$$\underbrace{\frac{dn_C}{dt}\Delta H}_{\text{reakční teplo}} = \underbrace{\alpha S(T - T_o)}_{\text{ztráty tepla do okolí}} +$$

$$+ \underbrace{(c_A n_A M_A + c_B n_B M_B + c_I n_I M_I)\frac{dT}{dt}}_{\text{akumulace tepla}}$$

Pro řešení v MATLABu sestavíme m-funkci *f106.m*, které představuje soustavu diferenciálních a algebraických rovnic popisujících danou úlohu. Protože závisí na velkém počtu parametrů byl použit pro řešení v MATLABu poněkud méně vhodný (ale jednoduchý) způsob předání parametrů. První výkonný řádek funkce představuje volání skriptu s názvem *d106.m*. Tento skript obsahuje definici konstant a výpočet pomocných proměnných pro snížení výpočetní náročnosti. Tento způsob byl použit i proto, že bude stejný příklad řešen v SIMULINKu, kde bude skript s definicí parametrů použit. Korektní řešení pro MATLAB by bylo vytvoření struktury parametrů a tuto strukturu jako jeden parametr do funkce předat přes vstupní parametry funkce.

```
function dY=f106(t,Y)
% Příklad 10.6 CHEMICKÁ REAKCE
% ODE popisující chemickou reakci
% Y ... nA      aktuální počet molů látky A
%       nB      aktuální počet molů látky B
%       nC      aktuální počet molů látky C
%       T       aktuální teplota reakční směsi

d106          % definice konstant
nA=Y(1); nB=Y(2); nC=Y(3); T=Y(4);
dY=zeros(4,1);

N=nA+nB+nC+nI;
k1=exp(log(B1)-A1/(273+T));
k2=exp(log(B2)-A2/(273+T));
x=-k1*nA*nB^2/N+k2*nC*N;
dY(1)=x/(N-nA);
dY(2)=2*x/(N-nB);
dY(3)=-x/(N-nC);
h=(nA*MA/roA+nB*MB/roB+nC*MC/roC+nI*MI/roI)/Sr;
jm=cA*nA*MA+cB*nB*MB+cC*nC*MC+cI*nI*MI;
dY(4)=(dY(3)*dH-alfa*(Sr+pi*D*h)*(T-To))/jm;
```

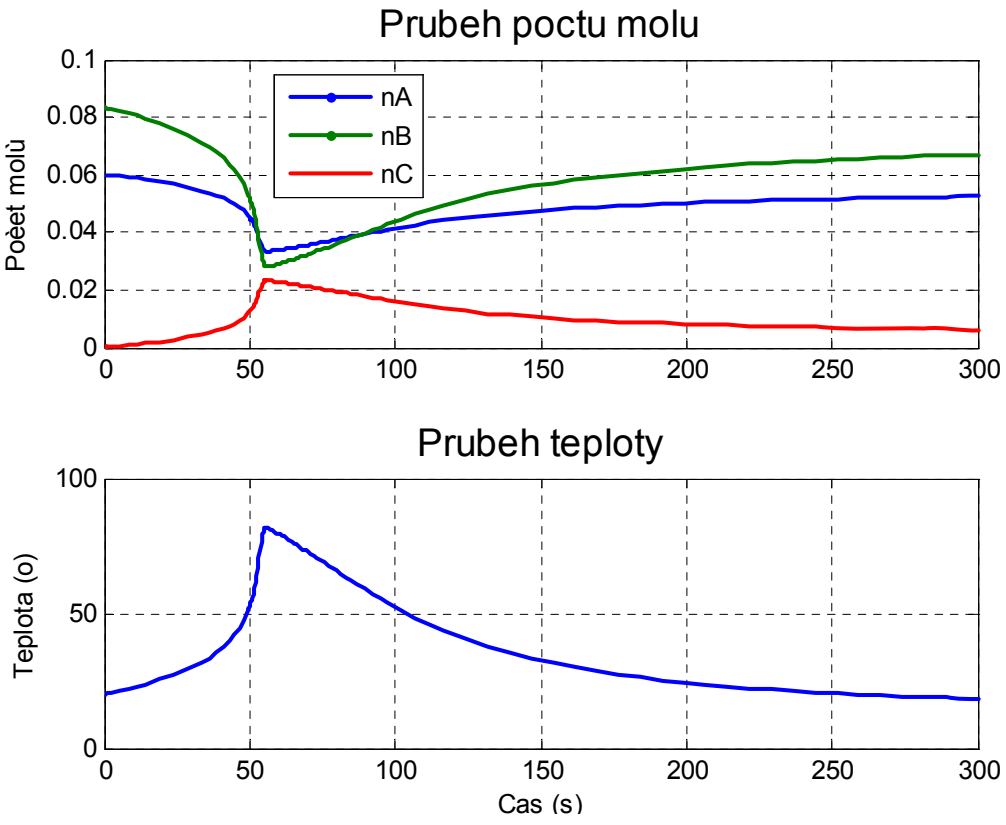
Určení parametrů závislosti rychlostní konstanty na teplotě (Arheniovy rovnice) je ve skriptu řešeno volbou hodnot rychlostní konstanty pro dvě zvolené teploty.

Řešení rovnic včetně vykreslení výsledku lze pak provést příkazy uloženými ve skriptu *zf106.m*.

```
% skript zf106.m
% Příklad 10.6 CHEM. REAKCE
Y0=[nA0;nB0;0;To];
t0=[0,300];
[t,Y]=ode45('f106',t0,Y0);
subplot(211)
plot(t,Y(:,1:3))
legend('nA','nB','nC')
ylabel('Počet molů')
subplot(212)
plot(t,Y(:,4))
xlabel('Cas (s)')
ylabel('Teplota (o)')
```

```
% data d106.m
% Příklad 10.6 CHEMICKÁ REAKCE
D=0.25;      % průměr reaktoru
% rychlostní konstanty při různých teplotách
T1=20;       T2=110;      % teploty
kI1=0.02;    kI2=750;     % r.k. ->
kII1=0.01;   kII2=4;      % r.k. <-
% výpočet konstant Arheniovy rovnice
B1=(kI2/kI1^((273+T1)/(273+T2)))^((273+T2)/(T2-T1));
A1=(273+T1)*(log(B1)-log(kI1));
B2=(kII2/kII1^((273+T1)/(273+T2)))^((273+T2)/(T2-T1));
A2=(273+T1)*(log(B2)-log(kII1));
clear kI1 kII1 kI2 kII2 T1 T2
MA=37;           % molekulová hmotnost látky A
MB=23;           % molekulová hmotnost látky B
MC=MA+2*MB; % molekulová hmotnost látky C
MI=18;           % molekulová hmotnost látky I
dH=12000000;     % reakční teplo
roA=700;roB=800;roC=825;roI=900;     % hustota A,B,C,I
cA=600;cB=750;   % tepelná kapacita látky A a B
cC=800;cI=500;   % tepelná kapacita látky C a I
alfa=100;        % součinitel prostupu tepla
To=20;       % teplota okolí
% počáteční volby
xA0=0.75;    % hmotový zlomek látky A v zásobníku a
xB0=0.45;    % hmotový zlomek látky B v zásobníku b
VA=0.004;    % objem roztoku látky A
VB=0.005;    % objem roztoku látky B
% předpočítané pomocné konstanty
jmA=roA+(roI-roA)*xA0; jmB=roB+(roI-roB)*xB0;
% počet molů I v reaktoru
nI=roI/MI*(roA*VA*(1-xA0)/jmA+roB*VB*(1-xB0)/jmB);
nA0=roI*roA*VA*xA0/MA/jmA;   % počet molů A v reaktoru
nB0=roI*roB*VB*xB0/MB/jmB;   % počet molů B v reaktoru
Sr=pi*D^2/4;                 % plocha reaktoru
```

Na obrázku 10-13 jsou vidět časové průběhy počtu molů všech látek v horním grafu a průběh teploty v dolním grafu. Po smíchání začnou látky reagovat, uvolňuje se reakční teplo, teplota stoupá a reakce se urychluje. Látky zreagují a ustaví se rovnováha při dané teplotě. Jak začíná teplota v důsledku ztrát do okolí klesat, rovnováha se ustaluje při jiném poměru látek. Díky výrazné závislosti rychlostních konstant obou reakcí na teplotě se i rovnovážné počty molů jednotlivých látek mění výrazně.



obrázek 10-13 Chemická reakce

10.7 Ochlazování místnosti

Ukázkou nesmyslného zadání je poslední příklad. Má sloužit jako ukázka toho, jak je důležité uvažovat nad zadáním globálně, abychom pracně neřešili nesmyslný, byť jinak zcela korektně zadaný příklad.

Představte si, že máme dokonale tepelně izolovanou místnost s objemem vzduchu V [m^3]. V místnosti je ustálená teplota T [°C]. Měrná tepelná kapacita vzduchu je c [$Jkg^{-1}K^{-1}$] a jeho hustota ρ [kgm^{-3}]. Aby tedy poklesla teplota v místnosti o 1 °C, je nutné odebrat energii (množství tepla) $E = V \times \rho \times c$ [J]. Místnost je úplně prázdná. Pouze uprostřed stojí normální kompresorová lednička o příkonu W_l [W] s otevřenými dveřmi. Účinnost chlazení je ε = 60% - chladící výkon (odebírané množství tepla za 1 sec) je tedy $W_l \times \varepsilon$ [W].

Před ledničkou je elektrický ventilátor s příkonem W_v [W] pro zajištění cirkulace vzduchu v místnosti. Pro zjednodušení nebudeme uvažovat tepelné kapacity ledničky a ventilátoru.

Vaším úkolem je nyní odpovědět na otázku

"Za jak dlouho poklesne teplota v místnosti o 5 °C ?"

Odpověď bez dalších výpočtů je zřejmá - nikdy. Proč? Lednička sice ochlazuje vnitřní prostor, ovšem odvedené teplo ohřívá vnější tepelný výměník, takže z místnosti se žádné teplo neodvede. Navíc k tomuto procesu je potřeba energii dodávat a veškerá přivedená elektrická energie se mění nakonec v teplo (včetně mechanické energie ventilátoru). Výsledkem je, že teplota v místnosti bude stoupat.

11. Další možnosti MATLABu pod Windows

MATLAB představuje v současné době celý propracovaný systém různých prostředků a lze jej využívat nejen k pouhým numerickým výpočtům. V této kapitole se velmi stručně zmíníme o některých z dalších možností, které jsou nabízeny v momentálně poslední verzi 7.0.4 (Release 14 with Servis Pack 2 z ledna 2005). Nejde o ucelený popis, ale o subjektivní výběr několika věcí, o kterých si autor myslí, že by mohly být pro zájemce o MATLAB zajímavé a považuje za vhodné se o nich zmínit. U některých možností je uveden i seznam funkcí, které tuto zmíněnou oblast podporují.

11.1 Vícerozměrné matice a objektová orientace

Od verze 5 přináší MATLAB principiálně nové možnosti. Kromě základních typů - *char (string), uint8, double* a *sparse*[87] zavádí datové typy **`cell`** (*cell array*) a **`struct`** (*structure array*). Zavedení těchto datových struktur souvisí s přechodem na objekty.

Cell array (pole buněk) je datová struktura odpovídající vícerozměrné matici. Vytvořit proměnnou tohoto typu lze funkcí `cell` nebo pomocí složených závorek {}. Např. třírozměrné pole prázdných prvků vytvoříme příkazem

```
>> C=cell(2,3,4)
C(:,:,1) =  []      []      []
            []      []      []
C(:,:,2) =  []      []      []
            []      []      []
C(:,:,3) =  []      []      []
            []      []      []
C(:,:,4) =  []      []      []
            []      []      []
```

V poli buněk lze kombinovat různé datové typy

```
>> CC={pi 'Ahoj' magic(3); 5 C 7}
CC    =     [3.1416]    'Ahoj'          [3x3 double]
            [     5]    {2x3x4 cell}    [         7]
```

Počet operací použitelných pro proměnné tohoto typu je omezený. Navíc záleží na kombinaci datových typů, zda operace je přípustná se všemi prvky.

```
>> C=ones([3 3 3]);           % vytvoření třírozměrné matice 3x3x3
>> C=C+4;                     % přičtení konstanty
>> C=C+C;                     % součet dvou prvků stejného typu
>> C=C*0.1;                   % násobení konstantou
>> C(2,:,2)=[2 2 2]           % vložení vetoru
C(:,:,1) =  1     1     1
            1     1     1
            1     1     1
C(:,:,2) =  1     1     1
            2     2     2
            1     1     1
C(:,:,3) =  1     1     1
            1     1     1
            1     1     1
>> sin(CC)
   ??? Function 'sin' not defined for variables of class 'cell'.
```

(matice CC obsahuje kromě čísel i text)

[87] tzv. řídká matice tj. matice, která obsahuje významný počet nulových prvků. Tento datový typ používá paměťově úsporný způsob uložení příslušné matice. Všechny funkce jádra lze použít pro operace s řídkými maticemi nebo kombinace řídká a plná matice.

```
>> sin(C)
ans(:,:,1) = 0.8415    0.8415    0.8415
             0.8415    0.8415    0.8415
             0.8415    0.8415    0.8415
ans(:,:,2) = 0.8415    0.8415    0.8415
             0.9093    0.9093    0.9093
             0.8415    0.8415    0.8415
ans(:,:,3) = 0.8415    0.8415    0.8415
             0.8415    0.8415    0.8415
             0.8415    0.8415    0.8415
```

Structure array (pole struktur) odpovídá přímo významu struktury používané v programovacích jazycích. Jednotlivé prvky pole jsou pojmenované. Strukturu lze vytvořit funkcí **struct** nebo naplněním jednotlivých položek. **Pozor!** Proměnné vytvořené funkcí `struct` (osoba1) a naplněním položek (osoba2) nejsou shodné jak by člověk předpokládal

```
>> osoba1=struct('jmeno',{'Jan','Novak'},'narozeni',[26,7,1970])
osoba1 =    1x2 struct array with fields:
            jmeno
            narozeni
>> osoba2.jmeno={'Jan','Novak'}
osoba2 =    jmeno: {1x2 cell}
      >> osoba2.narozeni=[26,7,1970]
osoba2 =    jmeno: {1x2 cell  }
            narozeni: [26 7 1970]
>> whos
  Name         Size         Bytes  Class
  C            3x3x3          216  double array
  CC           2x3            752  cell array
  osoba1       1x2            496  struct array
  osoba2       1x1            472  struct array
Grand total is 106 elements using 1936 bytes
```

Přístup k hodnotě měsíce narození je pak realizován různými příkazy podle typu proměnné

```
>> osoba1(1,2).narozeni(2)
ans =     7
>> osoba2.narozeni(2)
ans =     7
```

Datový typ *structure array* je využíván v objektech. Objekty a objektový přístup je nová vlastnost MATLABu od verze 5.0. Objekty jsou plně využívány systémem Handle Graphics. Výpočetní jádro MATLABu tuto možnost nabízí, ale více využívat se začíná až ve verzi 6. Jedno z prvních standardních využití bylo v Control System Toolboxu verze 4. Zde lze vytvořit tzv. LTI (Linear Time-Invariant) model - objekt kompletně popisující lineární vícerozměrový dynamický systém. Matematický popis chování může být ve formě stavového modelu, přenosové funkce či daný póly a nulami. Objekt dále obsahuje informace o dopravním zpoždění, o tom zda model je spojitý či diskrétní (interval vzorkování), dokonce může obsahovat i názvy vstupů a výstupů. Následující příkazy vytvoří LTI model spojitého systému daného póly, nulami a zesílením s jedním vstupem (označení x), jedním výstupem (označení y) a dopravním zpožděním. Příkaz **`set`** umožňuje nastavit jednotlivé vlastnosti, příkaz **`get`** umožňuje získat hodnoty vlastností - uveden bez označení vlastnosti vypíše vše.

```
>> model=zpk([],[-0.1 -0.2],0.02);  % vytvoř LTI model (zesílení, póly, kořeny)
>> set(model,'InputDelay',0.5);     % nastav vlastnost dopravní zpoždění
>> set(model,'InputName','x','OutputName','y','Notes','spojitý model');
>> model
Zero/pole/gain from input "x" to output "y":
                  0.02
exp(-0.5*s) * ---------------
              (s+0.1) (s+0.2)
```

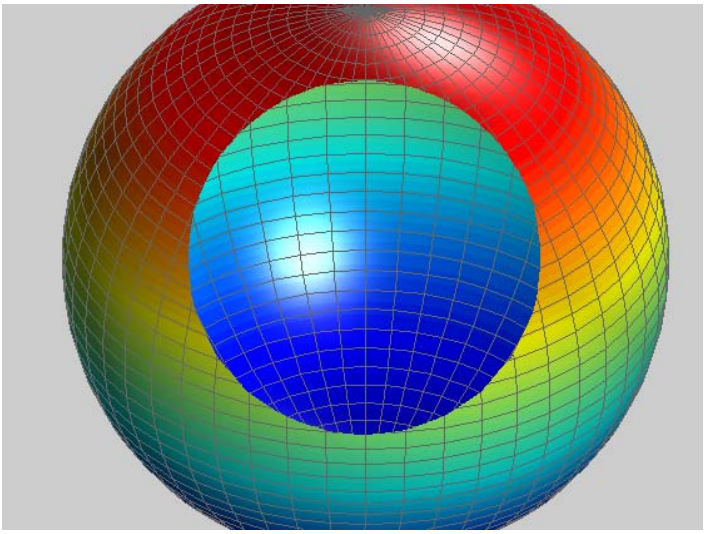
```
>> get(model)
            z: {[0x1 double]}
            p: {1x1 cell}
            k: 0.02
     Variable: 's'
           Ts: 0
      ioDelay: 0
   InputDelay: 0.5
  OutputDelay: 0
    InputName: {'x'}
   OutputName: {'y'}
   InputGroup: {0x2 cell}
  OutputGroup: {0x2 cell}
        Notes: {'spojitý model'}
     UserData: []
```

11.2 Grafické možnosti - systém Handle Graphics

Objektový přístup je využit při práci s grafikou. Dříve zmiňované příkazy pro kreslení (**plot, mesh** atd.) jsou jen zlomkem možností, které MATLAB pro práci s grafikou a pro tvorbu uživatelského rozhraní poskytuje. Jde o nové přepsání (do objektové podoby) systému **Handle Graphics** známého již z verze 4 pro Windows. Tento systém umožňuje nastavování všech možných (i nemožných) vlastností grafických objektů i tvorbu uživatelského rozhraní ve stylu programů Windows. Vlastní návrh se provádí obdobně jako pomocí standardních programovacích „vizuálních“ jazyků jako je např. Visual Basic či Delphi.

```
% skript z112.m
% Příklad 11.2 HANDLE GRAPHICS
sphere(48)          % nakresli kulovou plochu
h=findobj('Type','surface');% najdi objekt
% vlastnosti objektu
set(h,'FaceLighting','phong',...
   'FaceColor','interp',...
   'EdgeColor',[.4 .4 .4],...
   'BackFaceLighting','reverselit')
light('Position',[1 3 2]);  % zdroj světla c.1
light('Position',[-3 -1 3]);% zdroj světla c.2
axis off
material shiny    % nastav odrazové vlastnosti
% vlastnosti objektu camera
set(gca,'CameraPositionMode','manual',...
   'CameraTargetMode','manual',...
   'CameraViewAngleMode','manual');
set(gca,'CameraTarget',[0 0 0]);
set(gca,'CameraPosition',[10 10 10]);
set(gca,'CameraViewAngle',7);
pause(1)
% krouženi kamery okolo koule
N=15; R=sqrt(2*N^2);
for i=1:N,
   z=10*cos(2*pi/N*i);  výpočet x,y,z pozice
   x=R*sin(2*pi/N*i+pi/4);
y=R*cos(2*pi/N*i+pi/4);
   set(gca,'CameraPosition',[x y z]);
   pause(0.1)
end
% pohyb kamery směrem do středu koule
for i=1:N,
   p=N/(i+1)/2;
   set(gca,'CameraPosition',[p p p]);
   set(gca,'CameraViewAngle',7+5*i);% uhel
   pause(0.1)
end
```

Pro aspoň orientační přiblížení možností uveďme jednoduchý příklad (skript v rámečku vpravo). Ten ukazuje jak se s grafickými objekty pracuje na úrovni příkazů. Skript po spuštění nejprve ukazuje pohledy kamery kroužící kolem třírozměrného objektu (koule) s nastavenými vlastnostmi (materiál povrchu, poloha světel atd.) a pak pohledy, když se kamera začne pohybovat směrem do středu koule (včetně pohledu, kdy je vidět současně část vnějšího i část vnitřního povrchu kulové plochy, který na obrázku 11-1).



obrázek 11-1 Ukázka Handle Graphics

11.2.1 Seznam příkazů a funkcí pro práci s grafy a tvorbu uživ. rozhraní

Příkazy a funkce využívané pro tvorbu uživatelského rozhraní jsou v adresáři matlab\graphic a matlab\uitools. Jejich seznam lze získat pomocí příslušných příkazů `help graphics` a `help uitools`.

```
>> help graphics
  Handle Graphics.

  Figure window creation and control.
figure - Create figure window.
gcf    - Get handle to current figure.
clf    - Clear current figure.
shg    - Show graph window.
close  - Close figure.
refresh - Refresh figure.
refreshdata - Refresh data plotted in
figure.
openfig - Open new copy or raise
existing copy of saved figure.

  Axis creation and control.
subplot - Create axes in tiled
positions.
axes   - Create axes in arbitrary
positions.
gca    - Get handle to current axes.
cla    - Clear current axes.
axis   - Control axis scaling and
appearance.
box    - Axis box.
caxis  - Control pseudocolor axis
scaling.
hold   - Hold current graph.
ishold - Return hold state.

  Handle Graphics objects.
figure - Create figure window.
axes   - Create axes.
line   - Create line.
text   - Create text.
patch  - Create patch.
rectangle - Create rectangle,
rounded-rectangle, or ellipse.
surface - Create surface.
image  - Create image.
light  - Create light.
uicontrol - Create user interface
control.
uimenu - Create user interface menu.
uicontextmenu - Create user interface
context menu.

  Handle Graphics operations.
set    - Set object properties.
get    - Get object properties.
reset  - Reset object properties.
delete - Delete object.
gco    - Get handle to current object.
gcbo   - Get handle to current
callback object.
gcbf   - Get handle to current
callback figure.
drawnow - Flush pending graphics
events.
findobj - Find objects with specified
property values.
copyobj  - Make copy of graphics
object and its children.
isappdata - Check if application-
defined data exists.
getappdata - Get value of
application-defined data.
setappdata - Set application-defined
data.
rmappdata - Remove application-
defined data.

  Hardcopy and printing.
print  - Print graph or Simulink
system; or save graph to M-file.
printopt - Printer defaults.
orient - Set paper orientation.

  Utilities.
closereq - Figure close request
function.
newplot - M-file preamble for
NextPlot property.
ishandle - True for graphics handles.

  ActiveX Client Functions (PC Only).
actxcontrol - Create an ActiveX
control.
actxserver - Create an ActiveX server

  See also graph2d, graph3d,
specgraph, winfun.

>> help uitools
Graphical user interface tools.
  MATLAB Version 7.0.4 (R14SP2)
   21-Jan-2005

  GUI functions.
uicontrol - Create user interface
control.
uimenu - Create user interface menu.
ginput - Graphical input from mouse.
dragrect - Drag XOR rectangles with
mouse.
rbbox - Rubberband box.
selectmoveresize - Interactively
select, move, resize, or copy
objects.
waitforbuttonpress - Wait for
key/buttonpress over figure.
waitfor - Block execution and wait
for event.
uiwait - Block execution and wait for
resume.
uiresume - Resume execution of
blocked M-file.
uistack - Control stacking order of
objects.
uisuspend - Suspend the interactive
state of a figure.
```

uirestore - Restore the interactive state of a figure.

GUI design tools.
guide - Design GUI.
inspect - Inspect object properties.
align - Align uicontrols and axes.
propedit - Edit property.

Dialog boxes.
axlimdlg - Axes limits dialog box.
dialog - Create dialog figure.
errordlg - Error dialog box.
helpdlg - Help dialog box.
imageview - Show an image preview in a figure window.
inputdlg - Input dialog box.
listdlg - List selection dialog box.
menu - Generate menu of choices for user input.
movieview - Show movie in figure with replay button.
msgbox - Message box.
pagedlg - Page position dialog box.
pagesetupdlg - Page setup dialog.
printdlg - Print dialog box.
printpreview - Display preview of **figure** to be printed.
questdlg - Question dialog box.
soundview - Show sound in figure and play.
uigetpref - Question dialog box with preference support.
uigetfile - Standard open file dialog box.
uiputfile - Standard save file dialog box.
uigetdir - Standard open directory dialog box.
uisetcolor - Color selection dialog box.
uisetfont - Font selection dialog box
uiopen - Show open file dialog and call OPEN on result.
uisave - Show open file dialog and call SAVE on result.
uiload - Show open file dialog and call LOAD on result.
waitbar - Display wait bar.
warndlg - Warning dialog box.

Menu utilities.
makemenu - Create menu structure.
menubar - Computer dependent default setting for MenuBar property.
umtoggle - Toggle "checked" status of uimenu object.
winmenu - Create submenu for "Window" menu item.

Toolbar button group utilities.
btngroup - Create toolbar button group.
btnresize - Resize button group.
btnstate - Query state of toolbar button group.
btnpress - Button press manager for toolbar button group.
btndown - Depress button in toolbar button group.
btnup - Raise button in toolbar button group.

Preferences.
addpref - Add preference.
getpref - Get preference.
rmpref - Remove preference.
setpref - Set preference.
ispref - Test for existence of preference.

Miscellaneous utilities.
allchild - Get all object children.
clipboard - Copy and Paste strings to and from system clipboard.
edtext - Interactive editing of axes text objects.
findall - Find all objects.
findfigs - Find figures positioned off screen.
getptr - Get figure pointer.
getstatus - Get status text string in figure.
hidegui - Hide/unhide GUI.
listfonts - Get list of available system fonts in cell array.
movegui - Move GUI to specified part of screen.
guihandles - Return a structure of handles.
guidata - Store or retrieve application data.
overobj - Get handle of object the pointer is over.
popupstr - Get popup menu selection string.
remapfig - Transform figure objects' positions.
setptr - Set figure pointer.
setstatus - Set status text string in figure.
uiclearmode - Clears the currently active interactive mode.

11.3 Spolupráce s jinými programy pod Windows

MATLAB může spolupracovat v rámci Windows s jinými programy. Jedna cesta je pomocí DDE a druhá pomocí ActiveX Automation. V obou případech může fungovat buď jako klient (využívá funkce jiné aplikace) nebo jako server (z jiné aplikace jsou využívány jeho služby).

Ukažme použití spolupráce s Excelem pomocí ActiveX a DDE. Na listu Excelu s názvem *zdroj* budou čísla ve čtvercové oblasti 10 x 10 buněk (B7:K16). Cílem je na listu *cil* ve stejné oblasti buněk vytvořit inverzní matici.

Pro případ ActiveX je funkce (napsaná jako funkce v VBA – Visual Basic for Application) spuštěná pomocí tlačítka z Excelu. Pro případ DDE je přepočet aktivován změnou dat ve sledované oblasti. Funkce používající ActiveX přenese data z listu *zdroj* do matice Basicu, spustí MATLAB, přenese data do MATLABu, nechá v MATLABu spočítat inverzní matici, výsledek přenese zpět do Basicu a uloží data do oblasti na listu *cil*. Zdrojový text této funkce je v prvním rámečku na druhé straně. Uživatel vše ovládá z prostředí Excelu pomocí funkce napsané v VBA. MATLAB je využíván jako server a Excel k němu přistupuje jako klient.

V případě použití DDE je situace složitější a prac-

```
' Řešení pomocí ActiveX - funkce ve VBA
Dim Matlab As Object     ' objekt (ActiveX Server)
Dim MI() As Double       ' pro imaginární část (prázdná matice)
Dim MR(1 To 10, 1 To 10) As Double    ' pro reálnou část matice

Private Sub CommandButton1_Click()
    ' přenos dat z listu sešitu do matice
    For r = 1 To 10
       For s = 1 To 10
         MR(r, s) = Sheets("zdroj").Cells(6 + r, 1 + s).Value
       Next s
    Next r
    ' spuštění MATLABu
    Set Matlab = CreateObject("Matlab.Application")
    ' přenos matice BASICu do MATLABu
    Call Matlab.PutFullMatrix("M", "base", MR, MI)
    ' provedení příkazu MATLABu
    Call Matlab.Execute("MI=inv(M)")
    ' přenos matice MATLABu do BASICu
    Call Matlab.GetFullMatrix("MI", "base", MR, MI)
    ' přenos dat z matice na list sešitu
    For r = 1 To 10
       For s = 1 To 10
         Sheets("cil").Cells(6 + r, 1 + s).Value = MR(r, s)
       Next s
    Next r
End Sub
```

```
% skript DDE.m  - vytvoření a ukončení DDE propojení na Excel
% otevření kanálu DDE s aplikací Excel pro soubor DDE.xls
kz = ddeinit('excel','[DDE.xls]zdroj'); % vytvoř spoj (zdroj)
kc = ddeinit('excel','[DDE.xls]cil');   % vytvoř spojení (cil)
zdroj='r7c2:r16c11';                    % oblast B7:K16
data=ddereq(kz,zdroj,[1 1]);
% vytvoření propojení na změnu dat v Excelu
x=ddeadv(kz,zdroj,'DDEfun(kc,data);','data',[1 1]);

% vytvoř okno s tlačítkem, po jeho stisku
% ukončení propojení, ukončení spojení a zavření okna
h=uicontrol('String','Close','Position',[5 5 80 30],...
    'Callback',...
    'x=ddeunadv(kz,zdroj,[1 1]);x=ddeterm(kz);x=ddeterm(kc);close;');
```

```
function DDEfun(kanal,data)
% přenos v textové podobě, český Excel

% - oddělovač čárka, nahradit tečkami

pozice=findstr(data,',');          % najdi pozici čárek
data(pozice)='.';                  % nahraď čárku tečkou
X=str2num(data);                   % konverze řetězec na číslo
X=inv(X);                          % výpočet inverzní matice
% pro zpětný přenos nahradit tečky čárkami, každé číslo zvlášť
[nr,ns]=size(X);                   % rozměry matice
for r=1:nr                         % pro každý řádek
    for s=1:ns                     % pro každý sloupec
        xs=num2str(X(r,s),16);     % číslo na řetězec (16 znaků)
        xs(findstr(xs,'.'))=',';   % nahraď tečku čárkou
        % adresa buňky Ecxelu ve formě řetězce "r1c1"
        cil=strcat('r',int2str(6+r),'c',int2str(1+s));
        xx=ddepoke(kanal,cil,xs,[1 1]);  % zápis dat do cíle
    end
```

nější. Je možné využívat MATLAB zase jako server – řešení by bylo opět převážně na straně Excelu – opět pomocí VBA. Ukažme zde řešení obrácené, kdy Excel vystupuje jako server a řešení je na straně klienta tj. MATLABu. Využijeme možnosti tzv. "horkého propojení" (hot link) s Excelem – funkce **ddeadv**. Toto propojení zajistí automatické volání příkazu MATLABu (callback) při změně dat v propojené oblasti. Pro realizaci je potřeba skript *DDE.m* a funkce *DDEfun* uvedené v rámečcích.

Pro realizaci výše uvedeného příkladu je kromě napsání příslušného skriptu (*DDE.m*) a m-funkce (*DDEfun.m*) ještě obě aplikace připravit. Uživatel musí spustit Excel a otevřít sešit s daty (*DDE.xls*). Potom musí spustit MATLAB, v něm spustit dále uvedený skript, který musí používat jméno sešitu (*DDE.xls*) a listů (*zdroj* a *cil*), které je načteno ve spuštěném Excelu. Nyní se může vrátit do Excelu a v oblasti dat změnit číslo. Automaticky dojde k přepočtu výsledku. Po ukončení výpočtů v Excelu musí v MATLABu nejprve zrušit spojení s Excelem. V rámci tohoto příkladu je nutné řešit následující problémy:

- přenos čísel mezi Excelem a MATLABem – český Excel používá jako oddělovač desetinné části čárku a MATLAB tečku. Řešení na úrovni MATLABu zajišťuje funkce *DDEfun.m* pomocí funkcí pro práci s řetězci.
- zajistit korektní ukončení propojení až po ukončení práce v Excelu a ne při ukončení skriptu. Použité řešení využívá vytvoření tlačítka s vyvoláním funkce při jeho stisku.

11.3.1 Seznam příkazů a funkcí pro spolupráci v prostředí Windows

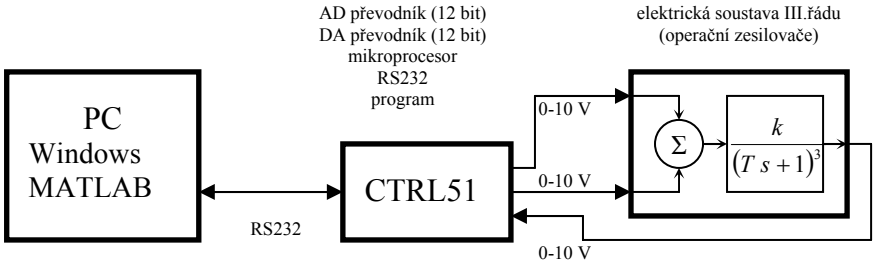
Příkazy a funkce využívané pro tyto účely jsou v adresáři matlab\winfun. Jejich seznam lze získat pomocí příkazu `help winfun`.

```
>> help winfun
  Windows Operating System Interface
           Files (COM/DDE)

  COM Automation Client Functions.
actxcontrol - Create an ActiveX
control.
actxserver - Create an ActiveX
server.
eventlisteners - Lists all events
that are registered.
isevent - True if event of object.
registerevent - Registers events for
a specified control at runtime.
unregisterallevents - Unregister all
events for a specified control at
runtime.
unregisterevent - Unregister events
for a specified control at runtime.
iscom - True if input handle is a
COM/ActiveX object.
isinterface - True if input handle
is a COM Interface.
COM/set - Set a property value on a
COM object.
COM/get - Get COM object properties.
COM/invoke - Invoke method on object
or interface, or display methods.
COM/events - Return list of events
the COM object can trigger.
COM/interfaces - List custom
interfaces supported by a COM
server.
COM/addproperty - Add custom
property to an object.
COM/deleteproperty - Remove custom
property from object.
COM/delete - Delete a COM object.
COM/release - Release a COM
interface.
COM/move - Move and/or resize an
ActiveX control in its parent
window.
COM/propedit - Invoke the property
page.
COM/save - Serialize a COM object to
a file.
COM/load - Initialize a COM object
from a file.

  COM Sample code.
mwsamp - Sample Activex control
creation.
sampev - Sample event handler for
ActiveX server.

  DDE Client Functions.
ddeadv - Set up advisory link.
ddeexec - Send string for execution.
ddeinit - Initiate DDE conversation.
ddepoke - Send data to application.
ddereq - Request data from
application.
ddeterm - Terminate DDE conversation
ddeunadv - Release advisory link.

  Other
winopen - Open a file using the
appropriate Windows command.
winqueryreg - Get information from
the Windows registry.
```

11.4 Podpora sériové linky

Jednou ze zajímavých možností MATLABu od verze 6 je přímá podpora sériové linky. Existuje značné množství měřicích přístrojů podporujících komunikaci prostřednictvím sériové linky. S těmito přístroji lze nyní komunikovat přímo z MATLABu. Jedná se o značně netypické použití MATLABu, proto si dovolím věnovat této záležitosti více prostoru. Ukažme si jednoduchý příklad na ovládání elektrického modelu v uzavřené smyčce v reálném čase. Na tomto příkladě je ukázána práce s objektem typu `serial` reprezentujícím sériovou linkou, spouštění funkce MATLABu od události i synchronizace běhu programu s reálným zařízením.

Uspořádání zařízení je na obrázku 11-2. Ovládanou soustavou je elektrická soustava s dvěma vstupy a jedním výstupem. Tato soustava je třetího řádu s ustálením cca 10 s a zesílením cca 1. Signály jsou připojeny na dva výstupy (kanál 1 a 2) a jeden vstup (kanál 1) jednotky CTRL51. Tato jednotka obsahuje 12 bitový analogo-číslicový (ADC Analog to Digital Convertor) a číslico analogový (DAC Digital to Analog Convertor) převodník, oba s vstupním (výstupním) rozsahem 0-10 V. Činnost jednotky je řízena mikroprocesorem, který kromě obsluhy AD převodníku s multiplexorem a DA převodníku také zajišťuje obsluhu komunikace po sériové lince tj. příjem příkazů a odeslání výsledků.



obrázek 11-2 Uspořádání experimentu pro řízení pomocí MATLABu

Pro spolupráci s jednotkou CTRL stačí čtyři pomocné uživatelské funkce využívající možnosti MATLABu pro práci se sériovou linkou. Základní uživatelskou funkcí je *o_CTRL*, která vytvoří objekt typu serial (**`serial`**), nastaví požadované parametry, napojí objekt na zvolený port (**`fopen`**), nastaví signály na rozhranní RS232 (zápis do vlastností otevřeného objektu), vyčte případné byte ve vstupním bufru (**`fread`**) a vrátí vytvořený objekt navázaný na zvolený port. Ve vlastnostech objektu je nastaveno používání časovače s periodickým spouštěním lokální uživatelská funkce *CtrlTimer*. Tato funkce při svém provedení pouze povýší číslo v položce objektu .UserData určené pro volné použití. Tato položka bude sloužit pro synchronizaci hlavního programu. Druhou uživatelskou funkcí je *c_CTRL*, která zajistí korektní ukončení práce se sériovou linkou tj. ukončí vazbu na hardware (**`fclose`**) a uvolní používaný objekt typu serial (**`delete`**). Zbývající funkce zajistí vlastní komunikaci s CTRL. Uživatelská funkce *r_CTRL* vyšle

```
function s=o_ctrl(port,period)
% zahájení práce s CTRL
%   s=o_ctrl(port,period)
% s      ... vytvořený objekt typu serial
% port   ... port pro komunikaci např. 'COM1'
% period .. perioda časovače

s=serial(port,'BaudRate',9600,'Parity','none',...
    'DataBits',8,'StopBits',1);% vytvoření objektu
s.DataTerminalReady='off';  % DTR do Low
s.InputBufferSize=512;  % velikost vstupního bufru
s.OutputBufferSize=512;% velikost výstupního bufru
s.FlowControl='none';      % bez řízení komunikace
s.TimeOut=5;                        % max.čekání 5 s
s.TimerPeriod=period;             % perioda časovače
s.TimerFcn=@CtrlTimer;% funkce spouštěná časovačem
s.UserData=0;               % příznak synchronizace
fopen(s); % propojení objektu s fyzickým zařízením
pause(0.1)            % čas na HW inicializace CTRL
s.DataTerminalReady='on';     % signal DSR do High
pause(0.5)                          % počkej na CTRL
x=s.BytesAvailable;    % počet byte ve vstup.bufru
if x>0,fread(s,x); end                  % vše vyčti
% lokální funkce -dostupná pouze z o_ctrl

function CtrlTimer(obj,event)
% callback od Timeru
obj.UserData=obj.UserData+1;       % povyš semafor
```

```
function c_ctrl(s)
% ukončení práce s objektem serial

fclose(s)        % odpojení od fyzického zařízení
delete(s)           % zrušení objektu
```

požadavek na čtení (`fwrite`) zvoleného kanálu a počká (`fread`) na odpověď, kterou vrátí jako výsledek, uživatelská funkce *w_CTRL* zajistí vyslání požadavku (`fwrite`) na zápis dané hodnoty na zvolený kanál.

```
function d=r_ctrl(s,k)
% vyčtení kanálu k CTRL
%
%   d=r_ctrl(s,k)
%
% s ... objekt typu serial
% k ... číslo kanálu CTRL
% d ... měřená hodnota

% pošli 1 unsigned byte
fwrite(s,128+k,'uchar');
% vyčti 1 unsigned short int
d=fread(s,1,'uint16');
% převod (12 bit ADC) na V
d=d*10/4095;
```

```
function w_ctrl(s,k,x)
% zápis hodnoty x na kanál k CTRL
%
%   w_ctrl(s,k,x)
%
% s ... objekt typu serial
% k ... číslo kanalu CTRL
% x ... hodnota (0-10)

% pošli 1 unsigned byte (řídicí byte)
fwrite(s,64+k,'uchar');
% pošli 1 unsigned short int (hodnota)
fwrite(s,x/10*4095,'uint16');
```

Použití těchto funkcí včetně synchronizace běhu programu na reálný čas využívající časovač objektu serial ukažme na příkladě změření odezvy elektrické soustavy na vstupní signál ve tvaru sinusoidy s proměnnou frekvencí. Řešení ve formě skriptu je ukázáno v rámečku vpravo. Příkazy jsou v textu komentovány a tak vysvětlení snad vyžaduje pouze synchronizace. Je použit mechanismus nazývaný "synchronizace pomocí semaforu[88]". Ve funkci o_CTRL je nastaveno použití časovače. To znamená, že od okamžiku spojení objektu s hardware (`fopen`) je, nezávisle na tom co dál v MATLABu děláme, periodicky[89] spouštěna zadaná funkce CtrlTimer. Tato funkce povyšuje počítadlo průchodů – "semafor" s numerickou hodnotou. V hlavním programu otestujeme hodnotu semaforu – nulová hodnota signalizuje, že interval ještě nevypršel a je třeba počkat[90]. Nenulová hodnota signalizuje, že interval již vypršel – je nutné zde hodnotu semaforu snížit. Pak provedeme potřebné příkazy a vrátíme se na test semaforu. Je-li hodnota semaforu nenulová uplynula již další perioda (provedení příkazů trvalo déle) a příkazy se provedou ihned znovu. Je-li hodnota semaforu nenulová trvale – trvá provádění potřebných příkazů déle než je nastavená perioda. Konkrétně v našem případě, když bude hodnota semaforu po skončení cyklu větší než 1 znamená to, že byla nastavena příliš krátká perioda.

```
% měření odezvy na generovaný vstupní signál
T=0.4;                  % interval vzorkování
T1=24; T2=12;           % max. a min. perioda
N=180;                  % počet měření
% vytvoření a otevření linky
s=o_ctrl('COM2',T);
%příprava na experiment
w_ctrl(s,1,5);          % kanál 1 nastav na 1 V
w_ctrl(s,2,0);          % kanál 2 nastav na 0 V
y=zeros(N,1);           % vektor výstupů
u=zeros(N,1);           % vektor vstupů
t=zeros(N,1);           % vektor času měření
pause(5)                % počkej 5 sec na ustálení
% synchronizace na následující událost časovače
s.UserData=0;           %zrušení hodnoty semaforu
while s.UserData==0, end;   % čekej na příznak
s.UserData=0;           %zrušení hodnoty semaforu
% začátek experimentu
tic             % začátek kontrolního čas.intervalu
t(1)=toc;               % čas prvního měření
y(1)=r_ctrl(s,1);       % první měření
u(1)=5;
for k=2:N,
    % synchronizace
    while s.UserData==0, end;   % čekej na příznak
    s.UserData=s.UserData-1;    % snížení hod.sem.
    t(k)=toc;           % čas měření (od startu)
    % výpočet vstupního signálu
    u(k)=5+4*sin(2*pi/(T1-(T1-T2)/N/T*t(k))*t(k));
    w_ctrl(s,1,u(k));   % nastav kanál 1
    y(k)=r_ctrl(s,1);   % další měření
end
c_ctrl(s);        % ukonči práci se sériovou linkou
```

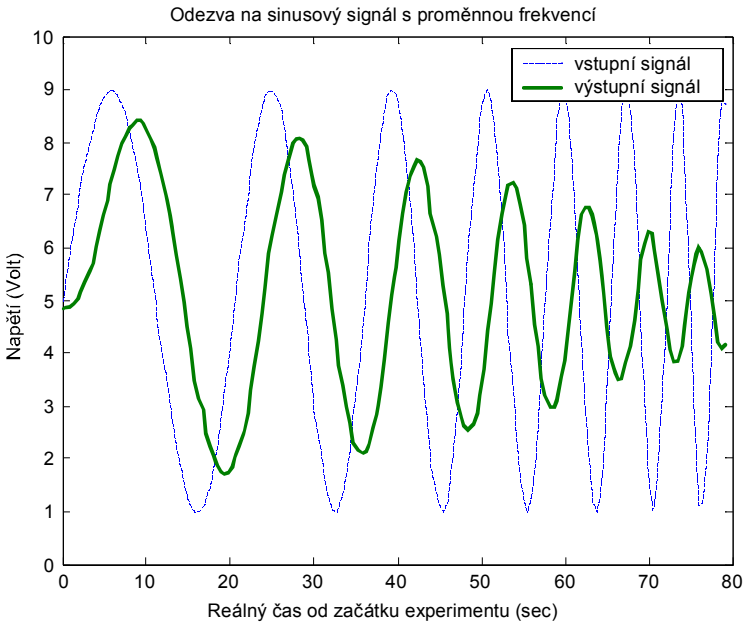
---

[88] semafor může být buď logický nebo numerický. V příkladu je použit numerický semafor protože dovoluje rozpoznat situaci, kdy čas vykonávání jedné sady příkazů překročí nastavenou periodu opakování. Také automaticky zajistí dodržení celkového času v situaci, kdy k překročení periody dojde jen občas, ale v průměru je doba sady příkazů menší než perioda časovače.

[89] ve Windows je slovo periodicky potřeba brát s rezervou. Dodržení periody je přibližné – pro delší periody je v průměru dodrženo. Menší periody se nedodrží (v uvedeném příkladu na Pentiu 150 MHz a Win98 je minimální v průměru dodržená perioda cca 0.4 sec).

[90] např. pomocí konstrukce `while s.UserData == 0, end;`

Použitím příkazu **toc** je zajištěno, že i při "sklouznutí" či nepravidelné periodě je generován průběh vstupního signálu v závislosti na skutečném a ne předpokládaném čase. Na obrázku 11-3 je pro zajímavost uveden průběh vstupního generovaného a výstupního měřeného signálu.



obrázek 11-3 Průběh reálného experimentu

11.5 Generování algoritmů v "C" a vytvoření spustitelného kódu

MATLAB můžeme rozšířit o vlastní algoritmy[91] vytvořené ve standardním programovacím jazyku (standardně je nyní podporován pouze jazyk "C/C++"). Pro platformu Windows jde o moduly historicky nazývané MEX (podle přípony používané v DOSu) i když nyní to jsou v podstatě klasické dynamicky linkované knihovny (.DLL). Kromě překladače jazyka C (Lcc verze 2.4 v R14), který je součástí MATLABu, je možné používat i překladače jiných výrobců – konkrétně pro platformu Windows Borland C++ verze 5.3-6 (tj. Borland C++Builder verze 3.0, 4.0, 5.0 a 6.0) a Microsoft Visual C/C++ verze 6.0, 7.0 a 7.1. Přestal být podporován jazyk FORTRAN a překladače fy WATCOM. Volání zvoleného překladače a vytvoření výsledné DLL knihovny zajišťuje MATLAB Compiler (pro R14SP2 ve verzi 4.2) zpřístupněný příkazem **mcc**.

Kromě této je velmi zajímavá i možnost opačná. Použití výpočetních algoritmů, které používá jádro MATLABu, nebo použití komplexních řešení vytvořených v MATLABu (v podobě m-souborů) ve vlastních programech. První variantu je možné řešit pomocí knihoven MATLAB C/C++ Math Library, která dovoluje používat (volat z vlastního programu psaného v "C" či "C++") stejné funkce jaké jsou používány jádrem MATLABu. Druhou variantu opět řeší MATLAB Compiler. Compiler dovoluje přeložit řešení problému realizované m-souborem (funkcí), který může obsahovat volání jak standardních funkcí jádra tak i uživatelských m-funkcí a výsledný program pak lze samostatně spustit. Do verze MATLABu 5.2 nebylo možné používat v překládaném řešení grafické možnosti MATLABu. Od verze 5.3 je k dispozici i grafická knihovna MATLAB C/C++ Graphics Library umožňující vygenerovat (pomocí opět MATLAB Compiler) spustitelný (na MATLABu) nezávislý program obsahující jak numerické řešení tak i interaktivní grafické uživatelské rozhraní (vytvořené prostředky MATLABu).

Toto vše zní hezky ale jak to funguje prakticky. Vytvoření vlastního programu je sice relativně jednoduché, ale jeho distribuce a používání není tak jednoduché, jak by se na první pohled zdálo. Ukažme konkrétní příklad – mějme dvě funkce jak je uvedeno v rámečcích. Příkazem

```
>> mcc -m main.m kresli.m
```

Necháme vygenerovat soubor main.exe (11 kB) a main.cnf (40 kB velikost podle složitosti původní aplikace desítky kB až jednotky MB). Pro spuštění na počítači kde není nainstalován MATLAB je potřeba instalační program MCRInstaller.exe o velikosti cca

```
function main
k=1;
while k>0,
    disp('0 ... konec')
    disp('1 ... sinus')
    disp('2 ... cosinus')
    k=input('Volba ? :');
    if (k==1) || (k==2), kresli(k), end
    drawnow
end
close(gcf)
```

[91] ať už z důvodů vyšší rychlosti řešení algoritmu, skrytí vlastního algoritmu před jinými uživateli či pro využití možností, které MATLAB neposkytuje (např. využití dat uložených ve speciální databázi)

105 MB (k dispozici po instalaci MATLABu s MATLAB Compilerem). Tento program obsahuje všechny potřebné knihovny pro běh výsledného programu. Pro instalaci jsou potřebná administrátorská práva. Po instalaci se vytvoří složka MATLAB Component se soubory o velikosti 312 MB. Vlastní program po spuštění vyextrahuje z datového souboru main.cnf poměrně složitou adresářovou strukturu (45 kB – 8 složek a 18 souborů, opět v závislosti na rozsahu a složitosti původní aplikace).

```
function kresli(r)
x=-2*pi:4*pi/1000:2*pi;
if r==1,
    y=sin(x);
    t='Sinus';
else
    y=cos(x);
    t='Cosinus';
end
plot(x,y,'r')
axis([-2*pi,2*pi,-1.1,1.1])
title(t)
grid
```

11.6 Něco málo o toolboxech

Jak již bylo zmíněno na začátku MATLAB je tvořen jádrem obsahujícím základní funkce a prostředky. Pro jednotlivé oblasti použití je možné tento základ rozšířit o problémově (na danou oblast) orientované funkce soustředěné do toolboxu (Tbx) - něco jako knihovny. Jde o funkce řešící specializované problémy dané oblasti. S využitím možností systému Handle Graphics dovolují tyto toolboxy využívat značně sofistikované matematické postupy i uživatelům nepříliš zběhlým v matematice. Umožní uživateli jednoduše zadat jeho data, vybrat metodu a spustit výpočet. Přitom mohou tyto funkce komunikovat s uživatelem v pojmech, které jsou v dané oblasti standardní a uživatel nemusí o použitých matematických metodách na které je převedeno řešení jeho problému nic vědět. Příkladem takovéhoto toolboxu je Financial Toolbox (vyžaduje Statistic a Optimization Tbx) určený pro oblast bankovnictví (zpracování a predikce časových řad).

Další toolboxy jsou určeny pro určité oblasti použití

- obecné použití - NAG Foundation Tbx, Symbolic Math Tbx, Partial Differential Equation Tbx, Statistic Tbx, Spline Tbx, Optimization Tbx, Neural Network Tbx, Fuzzy Logic Tbx, Mapping Tbx
- zpracování signálu a obrazu - Wavelet Tbx, System Identification Tbx, Signal Processing Tbx, High-Order Spectral Analysis Tbx, Frequency Domain Identification Tbx, Data Acquistion Tbx, Quantized Filtering Tbx, Communications Tbx
- návrhu řízení - Control System Tbx, Robust Control Tbx, Model Predictive Control Tbx, Nonlinear Control Design Tbx, LMI Control Tbx, μ-Analysis and Synthesis Tbx, Quantitative Feedback Theory Tbx

Kromě výše uvedených toolboxů fy MathWorks existují ještě toolboxy jiných výrobců, které jsou součástí standardní nabídky (oficiálně podporované) a desítky dalších, které jsou vytvořeny obvykle na univerzitách pro řešení problematiky v určité oblasti.

Některé toolboxy obsahují vlastní výkonné programy a příkazy v syntaxi MATLABu k nim zprostředkovávají přístup. Do této kategorie patří např. podpora pro návrh a tvorbu algoritmů určených pro digitální signálové procesory (DSP) - Fixed-Point Blockset, možnost použití DSP přímo v SIMULINKovém schématu (DSP Blockset) atd. Samostatnou zmínku zasluhují dva produkty z této kategorie - Symbolic Math Toolbox a Real Time Toolbox.

11.6.1 Symbolic Math Toolbox

Toolbox pro symbolické výpočty je typickým příkladem toolboxu, součástí kterého je jiné výpočetní jádro - v tomto případě (Release 14, 2005) jádro MAPLE 8 od fy Waterloo Maple, Inc. Opět jsou využívány výhody objektového přístupu. Ukázka použití funkcí Symbolic Math Toolbox je také v kapitole druhé. Ukažme si několik komentovaných příkladů

```
>> x=sym('x');                          % vytvoření objektu symbolické proměnné x
```

výpočty s definovaným počtem platných cifer

```
>> vpa(erf(0.5),40)                     % výpočet chybové funkce erf na 40 cifer
ans =       .52049987781304651868197197472909465432 17
M=[0.1 -1.2; 4.3 1.1];
>> vpa(inv(M),20)                       % inverze matice na 20 cifer
ans = [ .20872865275142314991, .22770398481973434535]
      [-.81593927893738140417, .18975332068311195446e-1]
```

maticové operace

```
>> A=[sin(x),-sin(x);cos(x),sin(x)] % vytvoření matice funkcí
A =   [sin(x), -sin(x)]
      [cos(x),  sin(x)]
>> det(A)                          % výpočet determinantu (symbolicky)
ans = sin(x)^2+cos(x)*sin(x)
>> inv(A)                          % výpočet inverzní matice (symbolicky)
ans = [              1/(sin(x)+cos(x)), 1/(sin(x)+cos(x))]
      [ -cos(x)/sin(x)/(sin(x)+cos(x)), 1/(sin(x)+cos(x))]
>> poly(A)                         % charakteristický polynom matice
ans = t^2-2*t*sin(x)+sin(x)^2+sin(x)*cos(x)
```

řešení algebraických rovnic $\frac{1}{x}+y=a \quad x+\frac{1}{y}=b$

```
>> X=solve('1/x+y=a','x+1/y=b','x,y');   % řešení soustavy algebraických rovnic
X =   x: [2x1 sym]
      y: [2x1 sym]
>> X.x                             % výpis řešení pro x
ans = [ 2*(-1+1/2*a*b+1/2*(a^2*b^2-4*a*b)^(1/2))*b/(a*b+(a^2*b^2-4*a*b)^(1/2))]
      [ 2*(-1+1/2*a*b-1/2*(a^2*b^2-4*a*b)^(1/2))*b/(a*b-(a^2*b^2-4*a*b)^(1/2))]
>> xx=simple(X.x);                 % zjednodušení výrazu
>> xx
xx =  [ (-2+a*b+(a*b*(a*b-4))^(1/2))*b/(a*b+(a*b*(a*b-4))^(1/2))]
      [ (-2+a*b-(a*b*(a*b-4))^(1/2))*b/(a*b-(a*b*(a*b-4))^(1/2))]
```

integrace a derivace

```
>> int(x^2*sin(x))                 % integrace analyticky řešitelná  ∫ x^2 sin(x) dx
ans = -x^2*cos(x)+2*cos(x)+2*x*sin(x)
>> int(exp(-x^2))                  % integrace analyticky neřešitelná ∫ e^(x^-2) dx
ans = 1/2*pi^(1/2)*erf(x)          % řešení pomocí chybové fce erf(x)
>> diff(x^2*sin(x^2))              % derivace funkce  d/dt (x^2 sin^2(x))
ans = 2*x*sin(x^2)+2*x^3*cos(x^2)
```

diferenciální rovnice $\frac{d^2}{dt^2}y+a\frac{d}{dt}y+b=\sin(t) \quad \frac{d}{dt}y\Big|_{t=0}=a \quad y(t=0)=b$

```
>> r=dsolve('D2y+a*Dy+b=sin(t)','Dy(0)=a','y(0)=b')
r =  -(b*t*a^3+a*b*t+cos(t)*a^3-b*a^2-b+a^2*sin(t))/a^2/(a^2+1)+(a*b+a+1)/a-
     (b*a^2+b+a+a^4+a^2)/a^2/(a^2+1)*exp(-a*t)
```

```
>> r=dsolve('Dy*y+a*y^4=b','y(0)=c')      y\frac{d}{dt}y+ay^4=b \quad y(t=0)=c
r =  -1/a*(a*tanh(2*t*(a*b)^(1/2)+atanh(c^2*a/(a*b)^(1/2)))*(a*b)^(1/2))^(1/2)
```

integrální transformace

```
>> f=fourier(exp(-x^2))            % Fourierova transformace
f =   pi^(1/2)*exp(-1/4*w^2)
>> ifourier(f)                     % zpětná Fourierova transformace
ans = 1/2*4^(1/2)*exp(-x^2)
>> a=sym('a');                     % symbolická proměnná a
>> l=laplace(x^2*exp(-a*x))        % Laplaceova transformace
l =   2/(s+a)^3
>> ilaplace(l)                     % zpětná Laplaceova transformace
ans = t^2*exp(-a*t)
```

Příkazy a funkce používané v Symbolic Math Tbx jsou v adresáři toolbox\symbolic. Jejich seznam lze získat pomocí příkazu `help symbolic`.

```
>> help symbolic

  Symbolic Math Toolbox
  Version 3.1.2 (R14SP2) 21-Jan-2005

  Calculus.
diff  - Differentiate.
int   - Integrate.
limit - Limit.
taylor - Taylor series.
jacobian - Jacobian matrix.
symsum - Summation of series.

  Linear Algebra.
diag  - Create or extract diagonals.
triu  - Upper triangle.
tril  - Lower triangle.
inv   - Matrix inverse.
det   - Determinant.
rank  - Rank.
rref  - Reduced row echelon form.
null  - Basis for null space.
colspace - Basis for column space.
eig   - Eigenvalues and eigenvectors.
svd   - Singular values and singular
vectors.
jordan - Jordan canonical (normal)
form.
poly  - Characteristic polynomial.
expm  - Matrix exponential.
mldivide - \  matrix left division.
mpower - ^  matrix power.
mrdivide - /  matrix right division.
mtimes - *  matrix multiplication.
transpose - .' matrix transpose.
ctranspose - '  matrix complex
conjugate transpose.

  Simplification.
simplify - Simplify.
expand - Expand.
factor - Factor.
collect - Collect.
simple - Search for shortest form.
numden - Numerator and denominator.
horner - Nested polynomial
representation.
subexpr - Rewrite in terms of
subexpressions.
coeffs - Coefficients of a
multivariate polynomial.
sort  - Sort symbolic vectors or
polynomials.
subs  - Symbolic substitution.

  Solution of Equations.
solve - Symbolic solution of algebraic
equations.
dsolve - Symbolic solution of
differential equations.
finverse - Functional inverse.
compose - Functional composition.

   Variable Precision Arithmetic.
vpa   - Variable precision arithmetic.
digits - Set variable precision
accuracy.

  Integral Transforms.
fourier - Fourier transform.
laplace - Laplace transform.
ztrans  - Z transform.
ifourier - Inverse Fourier transform.
ilaplace - Inverse Laplace transform.
iztrans - Inverse Z transform.

  Conversions.
double - Convert symbolic matrix to
double.
single - Convert symbolic matrix to
single precision.
poly2sym - Coefficient vector to
symbolic polynomial.
sym2poly - Symbolic polynomial to
coefficient vector.
char  - Convert sym object to string.
int8  - Convert to signed 8-bit
integers.
int16 - Convert to signed 16-bit
integers.
int32 - Convert to signed 32-bit
integers.
int64 - Convert to signed 64-bit
integers.
uint8 - Convert to unsigned 8-bit
integers.
uint16 - Convert to unsigned 16-bit
integers.
uint32 - Convert to unsigned 32-bit
integers.
uint64 - Convert to unsigned 64-bit
integers.

  Symbolic Operations.
sym   - Create symbolic object.
syms  - Short-cut for constructing
symbolic objects.
findsym - Determine symbolic
variables.
pretty - Pretty print a symbolic
expression.
latex - LaTeX representation of a
symbolic expression.
texlabel - Produces the TeX format
from a character string.
```

ccode - C code representation of a symbolic expression.
fortran - FORTRAN representation of a symbolic expression.

Arithmetic and Algebraic Operations.
plus - + addition.
minus - - subtraction.
uminus - - negation.
times - .* array multiplication.
ldivide - \ left division.
rdivide - / right division.
power - .^ array power.
abs - Absolute value.
ceil - Ceiling.
conj - Conjugate.
colon - Colon operator.
fix - Integer part.
floor - Floor.
frac - Fractional part.
mod - Mod.
round - Round.
quorem - Quotient and remainder.
imag - Imaginary part.
real - real part.
exp - Exponential.
log - Natural logarithm.
log10 - Common logarithm.
log2 - Base-2 logarithm.
sqrt - Square root.
prod - Product of the elements.
sum - Sum of the elements.

Logical Operations.
isreal - True for real array
eq - Equality test.
ne - Inequality test.

Special Functions.
besseli - Bessel function, I.
besselj - Bessel function, J.
besselk - Bessel function, K.
bessely - Bessel function, Y.
erf - Error function.
sinint - Sine integral.
cosint - Cosine integral.
zeta - Riemann zeta function.
gamma - Symbolic gamma function.
gcd - Greatest common divisor.
lcm - Least common multiple.
hypergeom - Generalized hypergeometric function.
lambertw - Lambert W function.
dirac - Delta function.
heaviside - Step function.

Trigonometric Functions.
acos - Inverse cosine.
acosh - Inverse hyperbolic cosine.
acot - Inverse cotangent.
acoth - Inverse hyperbolic cotangent.
acsc - Inverse cosecant.
acsch - Inverse hyperbolic cosecant.
asec - Inverse secant.
asech - Inverse hyperbolic secant.
asin - Inverse sine.
asinh - Inverse hyperbolic sine.
atan - Inverse tangent.
atanh - Inverse hyperbolic tangent.
cos - Cosine function.
cosh - Hyperbolic cosine.
cot - Cotangent.
coth - Hyperbolic cotangent.
csc - Cosecant.
csch - Hyperbolic cosecant.
sec - Secant.
sech - Hyperbolic sechant.
sin - Sine function.
sinh - Hyperbolic sine.
tan - Tangent function.
tanh - Hyperbolic tangent.

String handling utilities.
isvarname - Check for a valid variable name (MATLAB Toolbox).
vectorize - Vectorize a symbolic expression.
disp - Displays a sym as text.
display - Display function for syms.
eval - Evaluate a symbolic expression

Pedagogical and Graphical Applications.
rsums - Riemann sums.
ezcontour - Easy to use contour plotter.
ezcontourf - Easy to use filled contour plotter.
ezmesh - Easy to use mesh (surface) plotter.
ezmeshc - Easy to use combined mesh/contour plotter.
ezplot - Easy to use function, implicit, and parametric curve plotter
ezplot3 - Easy to use spatial curve plotter.
ezpolar - Easy to use polar coordinates plotter.
ezsurf - Easy to use surface plotter.
ezsurfc - Easy to use combined surface/contour plotter.
funtool - Function calculator.
taylortool - Taylor series calculator.

Demonstrations.
symintro - Introduction to the Symbolic Toolbox.
symcalcdemo - Calculus demonstration.
symlindemo - Demonstrate symbolic linear algebra.
symvpademo - Demonstrate variable precision arithmetic

```
symrotdemo - Study plane rotations.
symeqndemo - Demonstrate symbolic
equation solving.

  Access to Maple. (Not available with
Student Version.)
maple - Access Maple kernel.
mfun  - Numeric evaluation of Maple
functions.
mfunlist - List of functions for MFUN.
mhelp - Maple help.
procread - Install a Maple procedure.
(Requires Extended Toolbox.)
```

11.6.2 Real Time Toolbox

Příkladem úzce specializovaného tbx je Real Time Toolbox, který byl vytvořen českou firmou HUMUSOFT, distributorem MATLABu pro Českou republiku. Jde o jednodušší variantu (a levnější) Data Acquisition Toolbox. Je určen pouze pro platformu Windows/Intel a předpokládá použití akviziční karty. Relativně jednoduše propojuje MATLAB a SIMULINK s reálným světem. Umožňuje sběr dat v reálném čase, jejich bezprostřední zpracování příkazy MATLABu či modelem SIMULINKu a okamžitou realizaci vypočtených hodnot (ovládání výstupních signálů).

Funkce nabízené Real Time Toolboxem ve verzi 3.12 (pro MATLAB R14) lze získat (po instalaci z jedné diskety) příkazem `help rt`.

```
>> help rt
  Real Time Toolbox.
  Version 3.12  8-June-2004
  Copyright (c) 1992-2004
   HUMUSOFT s.r.o.

  Object definition
rtload - Load hardware driver
rtunload  - Unload all hardware
drivers
rtdef  - Define timer or history
variable
rtclear - Clear timers and/or history
variables
rtlink  - Link timers with history
variables
rtunlink - Unlink timers or history
variables

  Object control
rtstart - Start timers
rtstop - Stop timers
rtrd  - Read timer or history
variable
rtwr  - Write timer or history
variable
rtflush - Flush history variables
rtscan - Fast scan of input channel
rtwave  - Generate wave on output
channel
rtin  - Direct asynchronous input
from hardware
rtout - Direct asynchronous output to
hardware

  Kernel control
rtwho -  List hardware drivers,
timers, history variables, etc.
rtscript - Store commands created by
GUI into file
rtslice - Get or set real time kernel
timeslice period
rtperf  - Get performance of Matlab
with the real time kernel

  Auxiliary functions (not to be
called directly)
rteval_ - Evaluate Real Time Toolbox
command
rbutton - Toggle radio buttons
uinumchk - Check GUI numeric entry
for validity
```

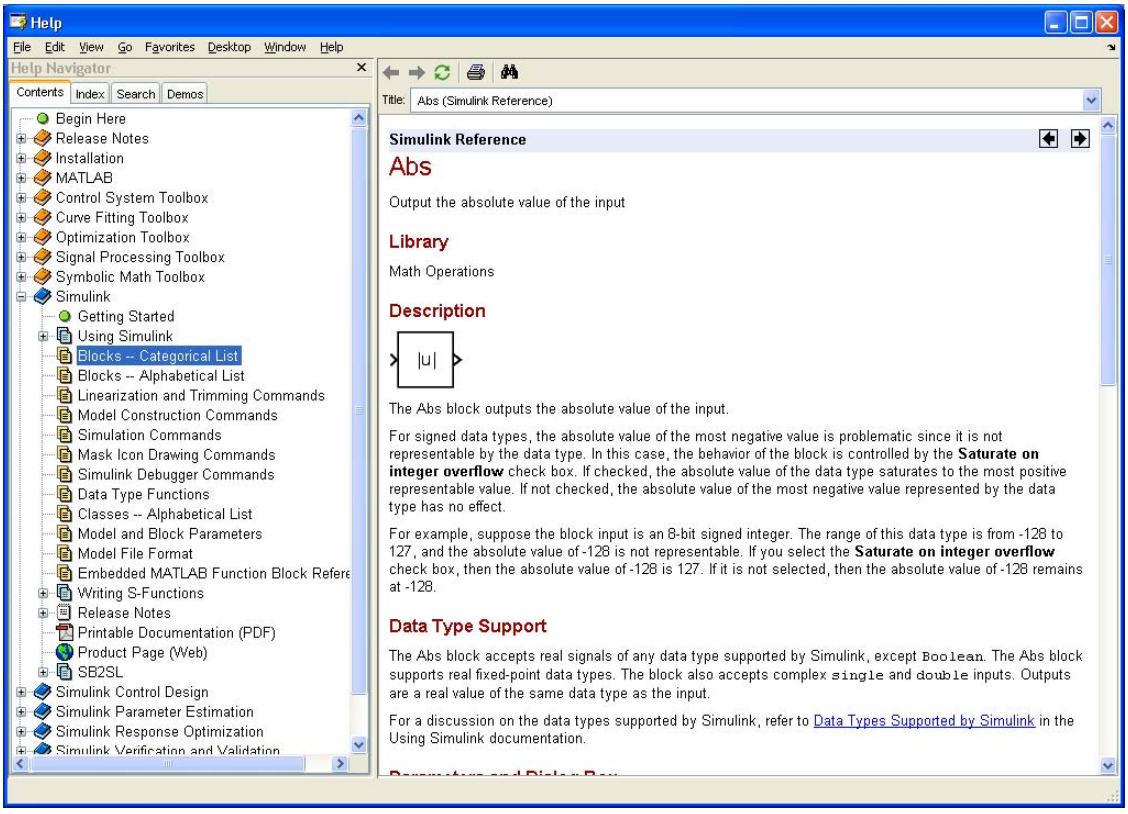
Poznámka: Pod pojmem akviziční karta se myslí rozšiřující deska do sběrnice počítače vybavená obvykle možností měřit a generovat několik analogových signálů. Většinou také umožňuje připojení dalších typů signálů. Fa HUMOSOFT nabízí dvě relativně levné varianty těchto karet – jednodušší AD512 pro sběrnici ISA a velmi vybavenou (z hlediska druhů signálů) MF614 pro sběrnici PCI. Blíže viz http://www.humusoft.cz/datacq/indexcz.htm.

12. Co je SIMULINK ?

Rozšíření MATLABu (nadstavba) SIMULINK je k dispozici až od verze 4. Využívá grafických možností hostitelské platformy (Windows případně X Windows). Způsob zápisu modelu i celkový způsob práce je značně odlišný od práce s vlastním MATLABem, který je orientován na řádkové příkazy. To svádí k tomu, používat SIMULINK "samostatně", nezávisle na znalosti MATLABu. I tento přístup je možný, ale vystačíme s ním pouze pro nejjednodušší využití. Pro efektivní využití a při řešení složitějšího problému je znalost MATLABu prakticky nezbytná.

Jak se vyvíjí MATLAB vyvíjí se i SIMULINK. Vzhledem k tomu, že je SIMULINK poměrně novou součástí MATLABu jsou i změny častější a významnější. V podstatě každá verze má jiné uspořádání bloků do složek základní knihovny a též dialogové obrazovky některých bloků se poněkud mění. V nových verzích se často rozšiřují možnosti původních bloků a také přibývají bloky se zcela novými funkcemi. Toto je třeba mít na mysli při čtení těchto skript – neleknout se, když po spuštění SIMULINKu vypadají obrazovky trochu jinak než je zde uvedeno. Ve verzi SIMULINKu 4.1 došlo k principiální změně – zavedení maticových[92] proměnných do SIMULINKu. Kromě toho byly podstatným způsobem rozšířeny možnosti práce[93] se subsystémy. Ve verzi SIMULINKu 6.2 (R14SP2) byly rozšířeny možnosti výpočtů v pevné řádové čárce.

SIMULINK má sjednocený systém nápovědy s MATLABem. Propracovaná on-line nápověda je přístupná buď tlačítky help v dialogu na jednotlivých blocích nebo standardně přes položku Help v hlavním menu všech oken SIMULINKu (viz obr. 12-1). Nápověda je ve formátu HTML a využívá hypertextové odkazy, stromovou strukturu obsahu, abecední seznam a vyhledávání. Tuto formu nápovědy používal SIMULINK dříve než MATLAB.

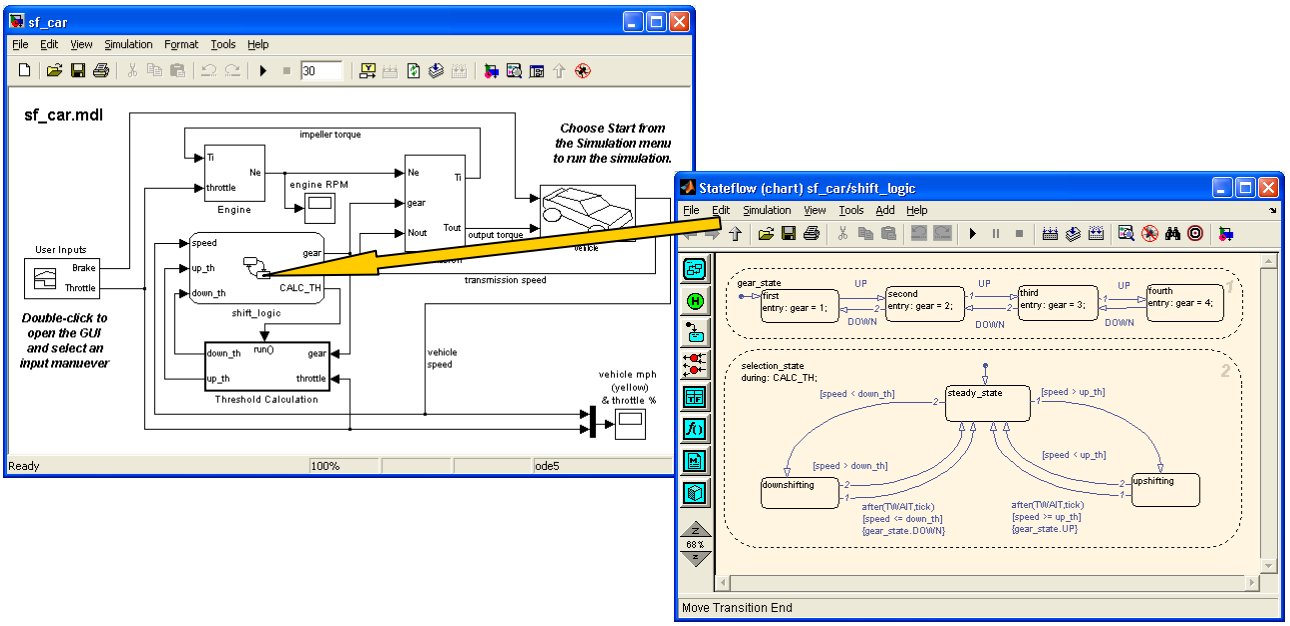


obrázek 12-1 Příklad nápovědy v SIMULINKu

[92] do této doby bylo možno sice pracovat s vektorem, ale pouze ve smyslu jednorozměrného (1-D array) pole tj. bez možnosti vektorových operací

[93] zejména ve smyslu přepínaných bloků – to umožňuje např. přirozeným způsobem realizovat ekvivalent rozhodovacího bloku if ... else ... end

SIMULINK je především určen na časové řešení (simulaci) chování dynamických systémů. Lze s jeho pomocí určit časové průběhy výstupních veličin (a všech ostatních) v závislosti na časovém průběhu veličin vstupních a počátečním stavu. Popis soustavy může být značně rozsáhlý a složitý. Může obsahovat i algebraické rovnice a vzorkované (diskrétní) výpočty. Ačkoliv SIMULINK poskytuje nástroje na řešení diskrétních událostí není primárně vhodný na řešení úloh kde se přepínají (nebo větví) odlišná řešení podle nastalých situací. Na tyto logické úlohy je určen další nástroj (nadstavba či rozšíření SIMULINKu) STATEFLOW. STATEFLOW nelze používat samostatně bez SIMULINKu. Funkce tohoto prostředku vychází z teorie konečných automatů a má i svůj editor a způsob zápisu. Ukázka modelu SIMULINKu kombinovaného s blokem STATEFLOW je na obrázku 12-2. Jde o demo simulující chování automatické převodovky.



obrázek 12-2 Okno STATEFLOW

Je potřeba si uvědomit, že SIMULINK nové metody či principiálně jiné možnosti řešení než MATLAB nenabízí. Co nabízí je jednoduchý přístup k propracovaným[94] metodám MATLABu pro časové řešení soustavy nelineárních diferenciálních rovnic, prostředky pro relativně snadný zápis problému (vytvoření modelu) a vizualizaci výsledků a elegantní řešení mnoha problémů, které při simulacích vznikají.

Vzhledem k třídě problémů, na řešení kterých je SIMULINK určený, se u uživatele předpokládají alespoň základní znalosti z oblasti matematického modelování (speciálně dynamických systémů) a simulace.

Přístup k návrhu zápisu problému silně připomíná návrh zapojení pro analogový počítač. Co zaujme na první pohled, je grafický způsob zápisu - v terminologii SIMULINKu nazývaný **model**. Z nabídky příslušné knihovny se myší přetahují výkonné **bloky** a pak se myší pospojují odpovídající vstupy a výstupy (**signály**). Při tvorbě složitějších modelů se využívá možnosti část modelu sdružit do dalšího bloku (**subsystému**[95]) a ty opět standardně kopírovat a spojovat. Jako zdroj signálů se používají nejčastěji signály z knihovních bloků generujících základní typy signálů s volitelnými parametry. Lze též používat připravená data ze souboru nebo z matic připravených v pracovním prostoru MATLABu. Ve spolupráci s Real Time Toolboxem či Data Acquisition Toolboxem (a měřící kartou) lze jako vstupní data použít i přímo měřené hodnoty v reálném čase. Výsledek simulace - časový průběh řešení - se zobrazuje nejčastěji graficky pomocí standardních bloků - zobrazení typu osciloskop (Scope) či XY graf. Další možností je výstup do proměnných pracovního prostoru MATLABu či ukládání do diskového souboru. Opět je možná přímá realizace signálů prostřednictvím příslušného hardware a Real Time Toolboxu či Data Acquisition Toolboxu.

---

[94] pod pojmem propracované je myšleno, že je nabízeno více metod, které jsou jak robustní (numericky stabilní) tak efektivní (rychlé řešení)

[95] na subsystém je možné se dívat přibližně jako na funkci (přesněji jako na makro)

Podíváme-li se na problematiku simulace chování dynamických systémů podrobněji, zjistíme, že simulace není jen[96] numerické řešení soustavy nelineárních diferenciálních rovnic (i když je to asi nejdůležitější část). V konkrétním případě musíme řešit několik dalších problémů - od výběru vhodné metody řešení pro daný problém a zahrnutí počátečních stavů, přes volbu kroku řešení (souvisí s požadovanou přesností a rychlostí výpočtů), problém nespojitých nelinearit ("zahuštění" bodů řešení v okolí nespojitosti), synchronizaci výpočtu na zadaný čas, problém algebraických smyček až po problém závislostí daných tabulkou. Právě automatické (i s možností manuální volby) řešení některých[97] problémů a podporu pro řešení dalších[98] SIMULINK nabízí.

Kromě vlastního grafického prostředí se standardními knihovnami (a případných rozšíření o knihovny z toolboxů) je SIMULINK podporován i příkazy a funkcemi dostupnými z prostředí MATLABu. Přehled příkazů je uveden na konci této kapitoly. Hotový model je možné využívat ve výpočtech v MATLABu. Naopak, používání funkcí MATLABu (ať už standardních či uživatelských) v modelu SIMULINKu, je poněkud komplikovanější.

Stejně jako v MATLABu lze i v SIMULINKu vytvářet uživatelské funkce. Tyto funkce se pak chovají jako nové bloky SIMULINKu s uživatelem definovaným chováním. Nabízí se několik možností řešení tohoto problému od nejjednodušší varianty pomocí bloků `Fcn` a `MATLAB Fcn`, přes blok `Embedded MATLAB Function` až po nejsložitější řešení v podobě S-funkcí. S-funkce je možné vytvářet jako textové soubory (obdoba tvorby uživatelských m-funkcí) nebo v jazyce "C" či FORTRANu. Nicméně vytvoření S-funkce již není triviální záležitostí.

SIMULINK je podobně jako MATLAB interpret (principiálně pomalý). Zatímco u MATLABu to tolik nevadí (předpokládá se spíše vývoj algoritmu) tak u SIMULINKu se model obvykle opakovaně spouští a nechává vypočítávat s různými hodnotami vstupních parametrů. V tomto případě již rychlost výpočtu může být zdržující. Máme-li instalován Real Time Workshop (případně Simulink Accelerator) je možné vytvořit to, čemu se v prvních verzích SIMULINKu říkalo akcelerovaný model. V podstatě jde o převod modelu do zdrojového tvaru programovacího jazyka s následným překladem do strojového kódu. V této podobě je průběh výpočtu modelu podstatně rychlejší.

Ve spojení s dalšími speciálními toolboxy lze vygenerovat strojový kód odpovídající SIMULINKovému modelu pro některé signálové procesory (DSP). Výsledný strojový kód se pak do tohoto procesoru přenese a celý výpočet pak probíhá mimo PC. Např. firma dSPACE GmbH (www.dspaceinc.com) takovýto toolbox nabízí. Základem je deska[99] do sběrnice PC osazená vlastním procesorem. Ze standardního modelu vygenerovaný program může využívat vstupy a výstupy této karty a třeba zpracovávat signály v reálném čase (bez účasti procesoru a operačního systému PC) nebo jen využívat výpočetní výkon signálového procesoru. SIMULINK je pak (během výpočtu) využíván jen jako vizualizační prostředek dovolující zobrazovat průběh výpočtu případně zasahovat do výpočtu změnou některých parametrů. Toto řešení je využíváno při vývoji programů do mikroprocesorových aplikací např. automobilovém či leteckém průmyslu (rapid control prototyping).

SIMULINK lze ovládat i z příkazového řádku MATLABu. Není to sice příliš obvyklá cesta, ale je užitečné o ní vědět. Použitelné příkazy jsou ukázány v následujícím výpisu nápovědy.

---

[96] proto také vzniklo několik nových programovacích jazyků specializovaných na oblast simulace

[97] automatická volba kroku, nespojité nelinearity, synchronizace výpočtu, tabulkou zadané závislosti

[98] nalezení ustáleného stavu, linearizace ve vybraném bodě, řešení algebraických smyček

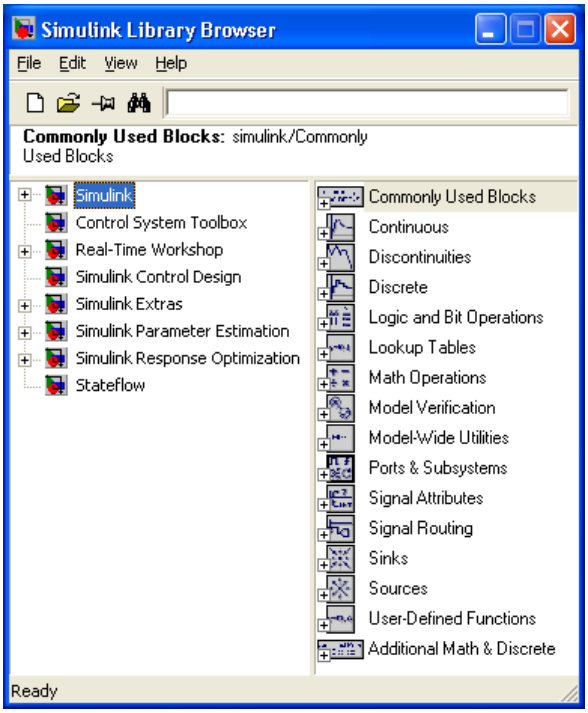
[99] např. DS1104 R&D Controller Board postavená na jádru procesoru PowerPC 603e / 250 MHz, která obsahuje 4×ADC 16 bit, 4×ADC 12 bit, 8×DA 16 bit, 2× incremental encoder interfaces, 20-bits of digital I/, Serial interface, Three-phase PWM outputs plus 4 single PWM outputs, 14 bits of digital I/O

```
>> help simulink

  Simulink
Version 6.2 (R14SP2) 21-Jan-2005

  Simulation.
sim    - Simulate a Simulink model.
sldebug - Debug a Simulink model.
simset - Define options to SIM Options structure.
simget - Get SIM Options structure

  Linearization and trimming.
linmod   - Extract linear model from continuous-time system.
linmod2  - Extract linear model, advanced method.
dlinmod  - Extract linear model from discrete-time system.
trim     - Find steady-state operating point.

  Model Construction.
close_system  - Close open model or block.
new_system    - Create new empty model window
open_system   - Open existing model or block.
load_system   - Load existing model without making model visible.
save_system   - Save an open model.
add_block     - Add new block.
add_line      - Add new line.
delete_block  - Remove block.
delete_line   - Remove line.
find_system   - Search a model.
hilite_system - Hilite objects within a model.
replace_block - Replace existing blocks with a new block.
set_param     - Set parameter values for model or block.
get_param     - Get simulation parameter values from model.
add_param     - Add a user-defined string parameter to a model.
delete_param  - Delete a user-defined parameter from a model.
bdclose       - Close a Simulink window.
bdroot        - Root level model name.
gcb           - Get the name of the current block.
gcbh          - Get the handle of the current block.
gcs           - Get the name of the current system.
getfullname   - get the full path name of a block
slupdate      - Update older 1.x models to 3.x.
addterms      - Add terminators to unconnected ports.
boolean       - Convert numeric array to boolean
slhelp        - Simulink user's guide or block help.

  Masking.
hasmask       - Check for mask.
hasmaskdlg    - Check for mask dialog.
hasmaskicon   - Check for mask icon.
iconedit      - Design block icons using ginput function.
maskpopups    - Return and change masked block's popup menu items.
movemask      - Restructure masked built-in blocks as masked subsystems.

  Library.
libinfo       - Get library information for a system.

  Diagnostics.
sllastdiagnostic - Last diagnostic array.
sllasterror   - Last error array.
sllastwarning - Last warning array.
sldiagnostics - Get block count and compile stats for a model.

  Hardcopy and printing.
frameedit     - Edit print frames for annotated model printouts.
print         - Print graph or Simulink system; or save graph to M-file.
printopt      - Printer defaults.
orient        - Set paper orientation.

  See also blocks and simdemos.
```

13. Jak se v SIMULINKu pracuje

Základní podmínkou pro spuštění SIMULINKu je spuštěný MATLAB.

Z příkazové řádky SIMULINK spustíme příkazem **simulink** nebo tlačítkem Simulink na nástrojové liště okna MATLABu. Objeví se okno se standardními knihovnami[100] v levém sloupci a seznamem prvků vybrané knihovny ve sloupci pravém (obr. 13-1). Stručný popis vybraného prvku je v horní části okna. Základní knihovnou je knihovna Simulink rozdělená dále na skupiny funkcí podle jejich zaměření. Ostatní knihovny, které jsou na obrázku 13-1, souvisí s nainstalovanými toolboxy.

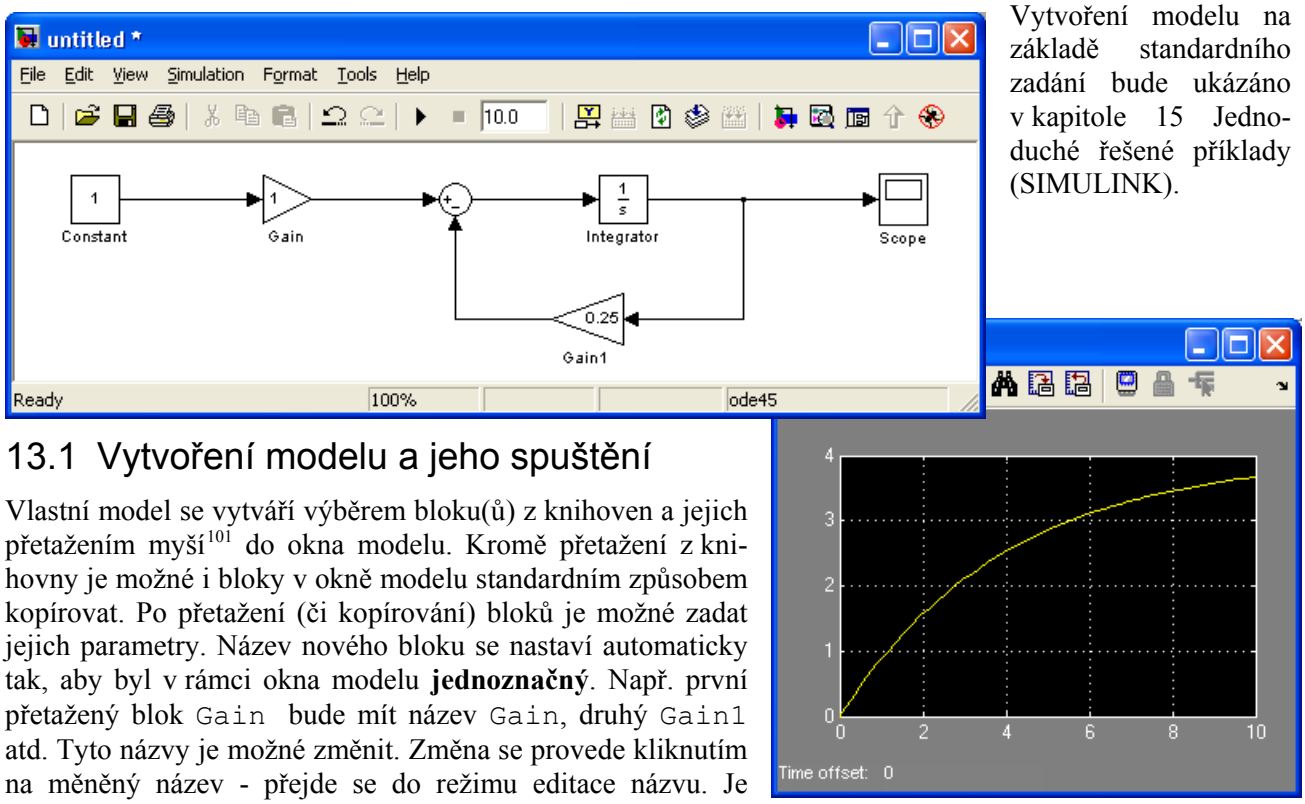


obrázek 13-1 Okno prohlížeče knihoven (Library Browser)

Nyní můžeme buď otevřít (*File->Open*) existující model SIMULINKu (standardní přípona .mdl) nebo otevřít nové okno (*File->New->Model*) pro zápis modelu. Na dalším obrázku (obr. 13-2) je již v okně modelu zápis matematického modelu soustavy I. řádu popsaného rovnicí

$$\frac{dx}{dt} - 0.25x = 0.5\,.$$

Výsledek je zobrazován pomocí bloku osciloskopu (blok `Scope`). Poklepáním myší na bloku osciloskopu se otevře samostatné okno, ve kterém je vykreslován v průběhu simulace časový průběh příslušného signálu. Vytvoření modelu na základě standardního zadání bude ukázáno v kapitole 15 Jednoduché řešené příklady (SIMULINK).



obrázek 13-2 Okno modelu a osciloskopu

13.1 Vytvoření modelu a jeho spuštění

Vlastní model se vytváří výběrem bloku(ů) z knihoven a jejich přetažením myší[101] do okna modelu. Kromě přetažení z knihovny je možné i bloky v okně modelu standardním způsobem kopírovat. Po přetažení (či kopírování) bloků je možné zadat jejich parametry. Název nového bloku se nastaví automaticky tak, aby byl v rámci okna modelu **jednoznačný**. Např. první přetažený blok `Gain` bude mít název `Gain`, druhý `Gain1` atd. Tyto názvy je možné změnit. Změna se provede kliknutím na měněný název - přejde se do režimu editace názvu. Je potřeba dbát na to, že názvy musí být v rámci okna modelu

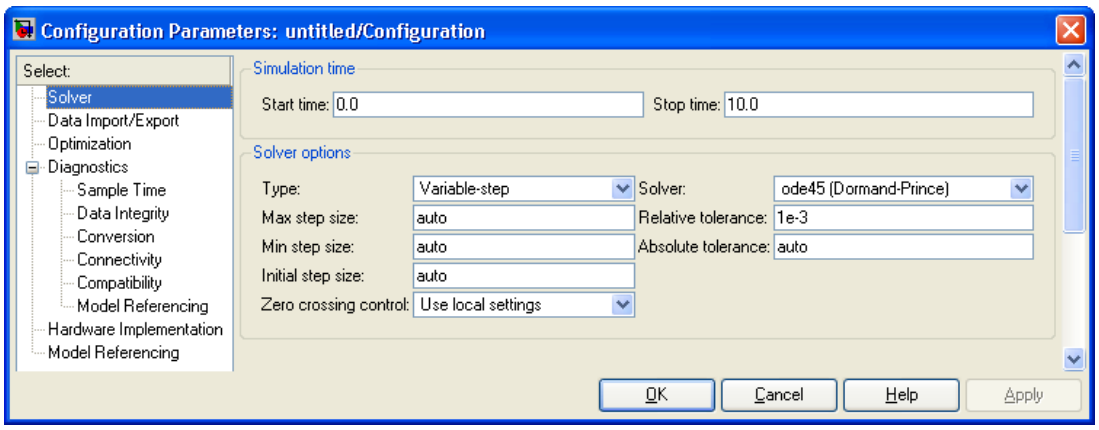
[100] všechny obrázky se vztahují k SIMULINKu verze 6.2

[101] pojem přetažení myší znamená, že blok v knihovně nezmizí., ale ve skutečnosti se vytvoří tzv. link tj. vazba na knihovní blok (viz kapitola 14.3)

jednoznačné. Když toto pravidlo nedodržíme, SIMULINK nedovolí stejný název vytvořit.

Dále je potřeba pospojovat odpovídající vstupy a výstupy bloků – někdy se tato spojení nazývají signály podle propojení bloků analogového počítače. Výstup jednoho bloku může být připojen na libovolný počet vstupů jiných bloků nebo na vstup tohoto bloku[102]. Spojení se provede myší tak, že se kurzor myši umístí na značku vstupu (či výstupu) bloku a při stisknutém levém tlačítku se kurzor přemístí na výstup (vstup) jiného bloku a tlačítko se pustí. V případě většího požadovaného počtu pravoúhlých zalomení je možné taženou čáru ukončit bez ukončujícího připojení a z tohoto místa táhnout další čáru. V libovolném místě okna modelu je možné umístit komentář - vysvětlující text. Dvojklikem na požadovaném místě se dostaneme do režimu editace a můžeme zapsat požadovaný text.

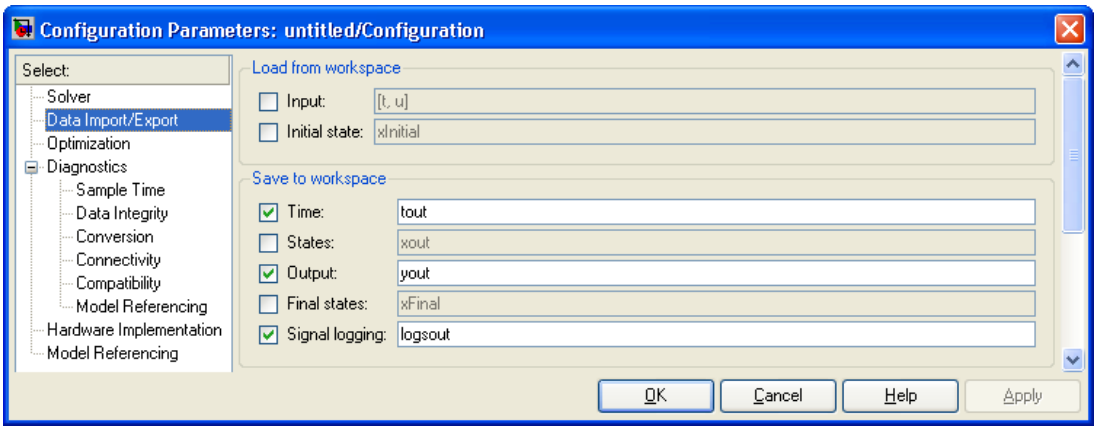
Po vytvoření modelu nastavíme základní parametry simulace. Po otevření nabídky *Simulation* z hlavního menu okna modelu a výběru položky *Configuration parameters* (nebo *Ctrl+E*) se objeví okno parametrů simulace s položkami *Solver, Data Import/Export , Optimization a Diagnostic*. Pod položkou **Solver** je možné nastavit čas simulace (*Start time* a *Stop time*), volbu řešitele (použitou metodu řešení) a volby týkající se automatické velikosti kroku. Implicitní volby jsou na obr. 13-3.



obrázek 13-3 Okno parametrů simulace - Solver

Volby pod položkou **Data Import/Export** se týkají možností napojení na pracovní prostor MATLABu. Je možné předepsat, že se mají přebírat počáteční stavy a průběhy vstupních veličin z proměnných v pracovním prostoru MATLABu. Dále že vybrané vypočtené veličiny se mají ukládat do předepsaných proměnných MATLABu – viz obrázek 13-4.

V položce **Optimization** jsou volby týkající se výkonu simulace a generování kódu z modelu. Volby pod položkou **Diagnostics** umožňují nastavit, které z kontrolovaných druhů chyb či událostí mají vyvolat hlášení a na jaké úrovni. Možné úrovně hlášení jsou *None* - nekontroluje, *Warning* – vypíše se



obrázek 13-4 Okno parametrů simulace – Data Export/Import

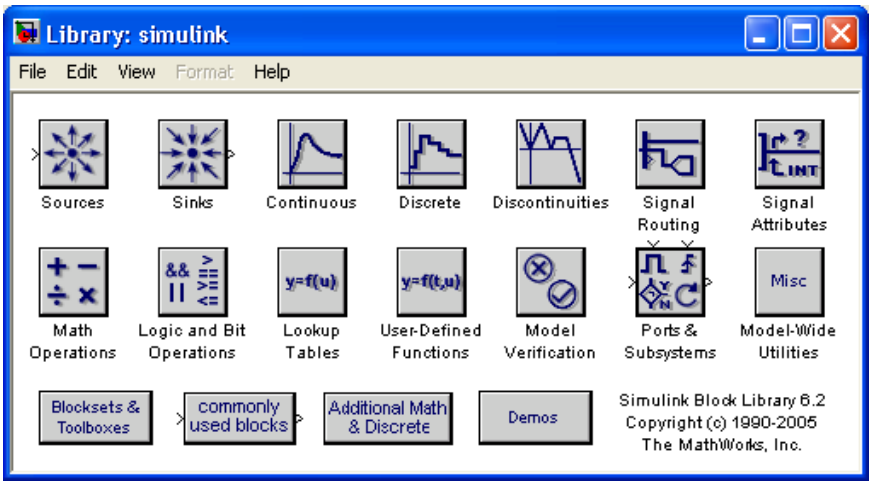
[102] obráceně to přirozeně neplatí – nelze přivést na jeden vstup dva či více různých výstupů. Stejně jako u analogového počítače nemůžeme přivést dva různé zdroje napětí na jeden vstup – tam by se spojily a jaké by bylo výsledné napětí? Zatímco u analogového počítače toto teoreticky provést lze, SIMULINK mám to nedovolí.

do příkazové řádky MATLABu hlášení, ale výpočet pokračuje, *Error* – vypíše se hlášení a výpočet se zastaví. Některé chyby se kontrolují ve fázi přípravy modelu na spuštění výpočtu a některé se mohou vyskytnout během výpočtu.

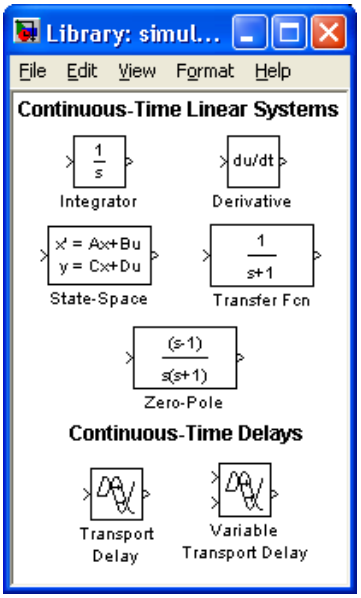
Nyní je již možné příkazem *Simulation* - *Start* (nebo *Ctrl+T*) spustit simulaci. V případě, že nebyl zařazen žádný blok pro vizualizaci průběhu výsledku se pouze v stavovém řádku okna modelu ukazuje průběh času výpočtu. V případě, že použijeme některý zobrazovací blok je potřeba dvojklikem na bloku otevřít zobrazovací okno. Pokud to neuděláme (ať už před spuštěním simulace nebo po skončení) nedojde k zobrazení sledovaných veličin. Pozor na to, že zobrazovací okno má implicitně nastavené určité měřítko a když jsou naše výsledky výrazně jiné, nemusíme nic vidět. V tomto případě je třeba manuálně změnit měřítko v zobrazovacím okně např. pomocí ikony s dalekohledem nastavíme automatické měřítko (zobrazí se vše).

13.2 Standardní knihovny SIMULINKu

Každá knihovna obsahuje jednotlivé bloky, které lze myší přetáhnout do okna modelu. Většina bloků dovoluje zadat hodnoty parametrů. Hodnoty lze zadat buď jako konstanty nebo i pomocí názvů proměnných, které jsou umístěny v pracovním prostoru MATLABu. Tento způsob dovoluje inicializaci hodnot pomocí naplnění hodnot proměnných např. skriptem v MATLABu před spuštěním modelu v SIMULINKu. Bloky jsou v knihovnách seskupeny podle oblasti použití. Základní knihovna Simulink obsahuje 16 skupin bloků (viz obr. 13-5 – knihovna v UNIXové podobě).



obrázek 13-5 Knihovna Simulink



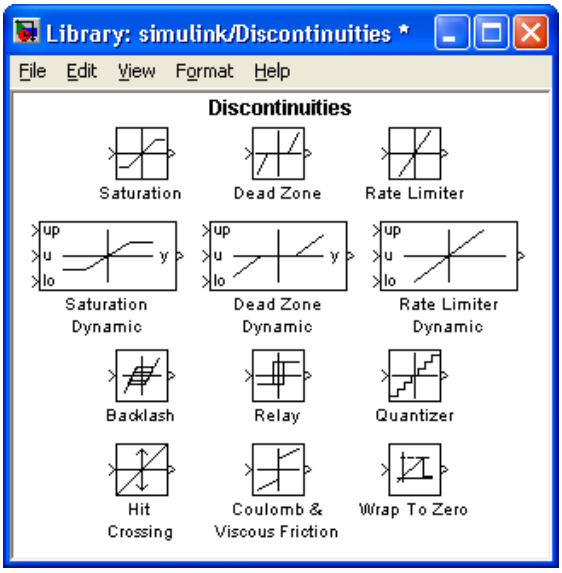
obrázek 13-6 Knihovna Simulink - Continuous

Skupina ***Commonly Used Blocks*** obsahuje výběr (kopie) 22 nejčastěji používaných bloků. Jejich popis je uveden v dalším textu, který je jednotlivým skupinám věnován.

Skupina ***Continuous*** (obr. 13-6) obsahuje 7 nejdůležitějších bloků sloužících pro vytvoření modelů z diferenciálních rovnic. Prvky z této skupiny tvoří obvykle základ spojitého dynamického modelu. Jaké prvky to jsou? Mezi základní patří blok `Integrator` (numerická integrace vstupního signálu). Další bloky z této skupiny jsou `Derivative` (numerická derivace vstupního signálu), `State-Space` (vyjádření soustavy lineárních diferenciálních rovnic prvního řádu v maticovém zápisu – stavový model), `Transfer Fcn` (vyjádření lineární diferenciální rovnice v přenosovém tvaru), `Zero-Pole` (vyjádření lineární diferenciální rovnice pomocí kořenů jmenovatele – pólů, kořenů čitatele – nul a zesílení přenosové funkce), `Transport Delay` (spojité dopravní zpoždění), `Variable Transport Delay` (proměnné spojité dopravní zpoždění).

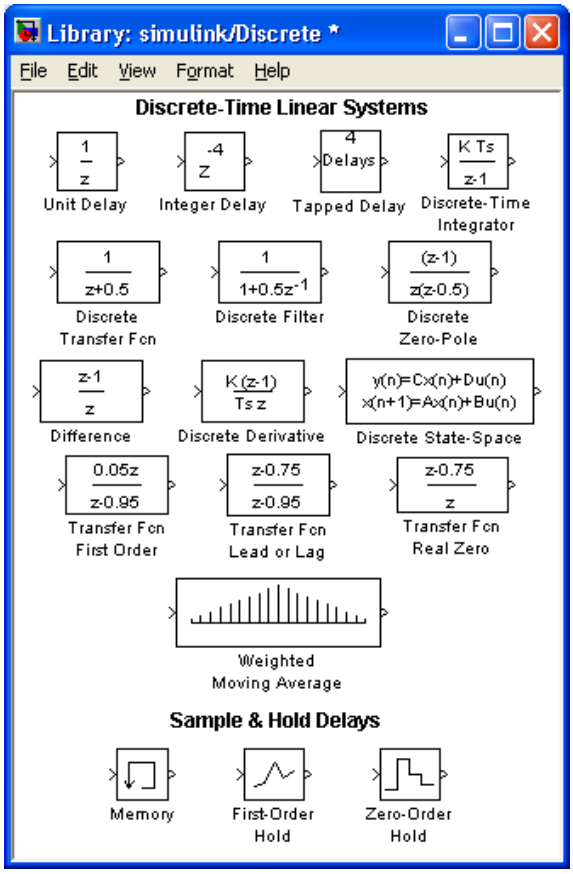
Další skupinou je ***Discontinuities*** (obr. 13-7). Tato skupina zahrnuje bloky typických nespojitostí. Často používaným blokem z této skupiny je blok `Saturation` (omezení signálu mezi zadanou dolní a horní mez). Dalšími bloky jsou `Dead Zone` (oblast s nulovým výstupem) a `Rate Limiter` (omezení rychlosti změny signálu), `Backlash` (blok mechanické vůle), `Relay` (přepínání mezi dvěmi konstantami) a

`Quantizer` (analogově-číslicový převodník), `Coulomb & Viscous Fiction` (blok Coulombického a viskózního tření), `Wrap To Zero` (vynuluje výstup, jestliže vstup překročí zadaný limit). Blok `Hit Crossing` detekuje přechod signálu přes zadanou hodnotu (u řešitelů s proměnným krokem výpočtu vynutí výpočet poblíž tohoto bodu).



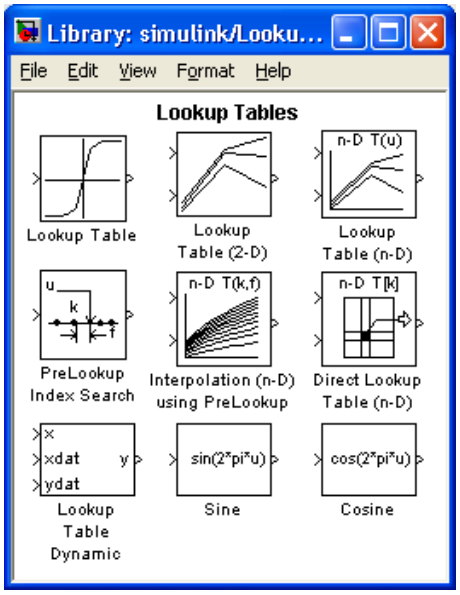
obrázek 13-7 Knihovna Simulink - Discontinuities

Ve skupině ***Discrete*** (obr. 13-8) jsou bloky určené k podobnému účelu jako bloky ze skupiny ***Continous***, ale používané v diskrétní oblasti. Jde o těchto 17 bloků - `Unit Delay` (zpoždění signálu o jednu periodu vzorkování), `Integer Delay` (zpoždění signálu o N period vzorkování), `Tapped Delay` (zpoždění signálu o N period vzorkování a předání všech zpožděných hodnot na výstupu bloku), `Discrete-Time Integrator` (ekvivalent spojitého integrátoru v diskrétní oblasti), `Discrete Transfer Fcn` (popis lineární diferenční rovnice ve formě diskrétní přenosové funkce v z), `Discrete Filtr` (popis lineární diferenční rovnice ve formě diskrétní přenosové funkce v z^{-1}), `Discrete Zero-Pole` (diskrétní přenosová funkce vyjádřená pomocí kořenů jmenovatele a čitatele), `Difference` (výpočet diference jako rozdílu dvou po sobě jdoucích hodnot), `Discrete Derivative` (diskrétní derivace), `Discrete State-Space` (vyjádření pomocí diskrétního stavového modelu), `Transfer Fcn First Order` (popis lineární diferenční rovnice 1. řádu), `Transfer Fcn Lead or Lag` (popis lineární diferenční rovnice ve formě lead-lag modelu), `Transfer Fcn Real Zero` (popis lineární diferenční rovnice s reálnou nulou a bez pólu), `Weighted Moving Average` (vážený klouzavý průměr vstupu), `Memory` (zpoždění o jeden integrační krok). Bloky `First-Order Hold` (tvarovač prvního řádu) a `Zero-Order Hold` (tvarovač nultého řádu) určují průběh výstupního signálu mezi intervaly vzorkování.



obrázek 13-8 Knihovna Simulink - Discrete

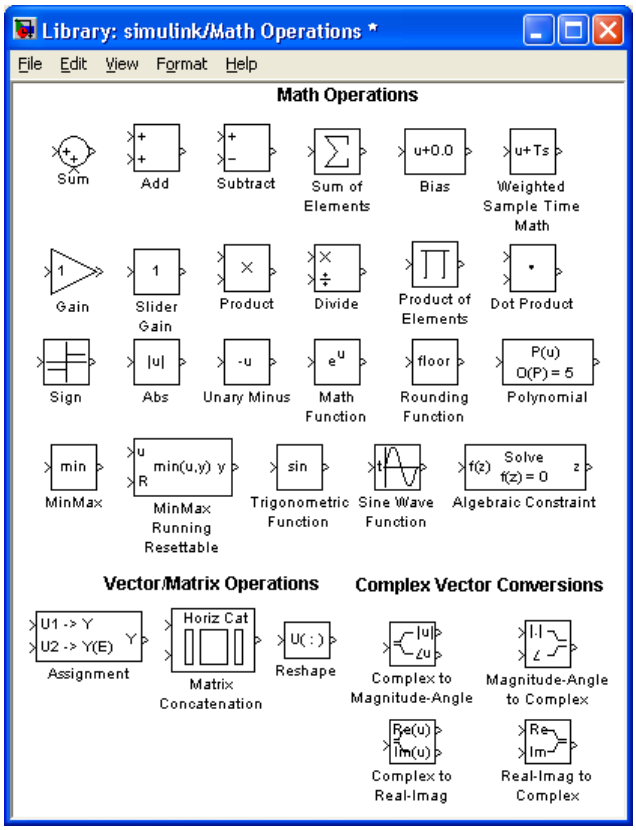
Skupina ***Lookup Tables*** (obrázek 13-9) obsahuje 9 bloků. Bloky `Lookup Table`, `Lookup Table (2-D)`, `Lookup Table (n-D)`, `PreLookup Index Search`, `Interpolation (n-D) using PreLookup` a `Direct Lookup Table (n-D)` nabízejí různé možnosti vyhledávacích tabulek a interpolaci mezi hodnotami tabulkového zadání průběhu. Funkce `Sine` a `Cosine` implementují funkce sinus a cosinus



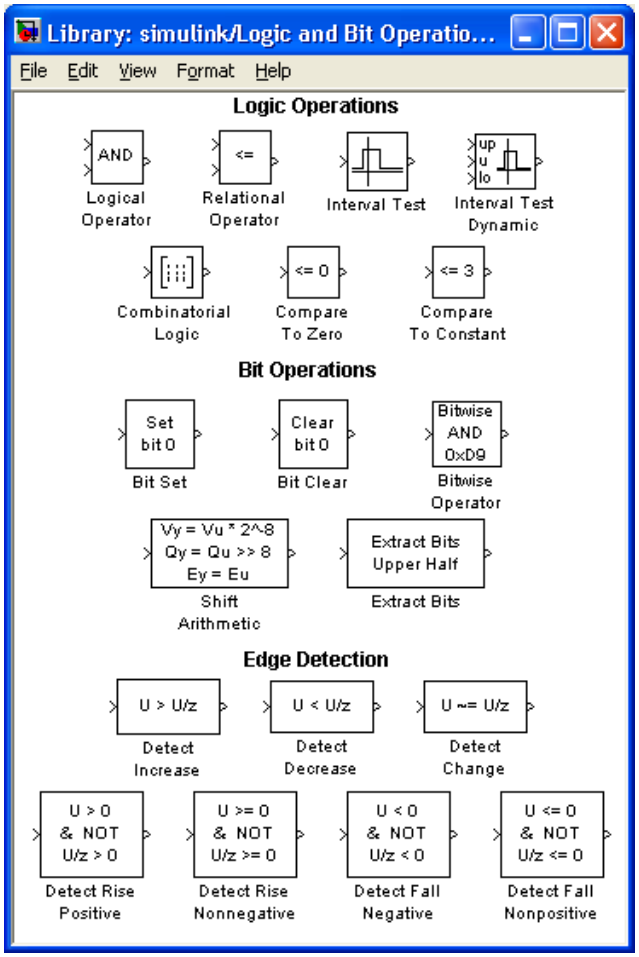
obrázek 13-9 Knihovna Simulink - Lookup Tables

v pevné řádové čárce (s využitím vyhledávací tabulky).

Skupina ***Logic and Bit Operations*** (obrázek 13-10) obsahuje 19 bloků logických a bitových operací. Bloky `Logical Operator` a `Relational Operator` umožňují použít logické a relační oprerátory. Blok `Interval Test` nastaví na výstup 1 (PRAVDA) pokud hodnota leží v daném intervalu. Blok `Combinatorial Logic` umožní aplikovat pravdivostní tabulku. Bloky `Compare to Zero` a `Compare to Constant` porovnají hodnotu vstupu s nulou nebo zadanou konstantou. Bloky `Bit Set` a `Bit Clear` umožní nastavit nebo vynulovat zadaný bit vstupu (integer). Blok `Bitwise Operator` provede zadanou bitovou operaci se vstupem. Bloky `Shift Arithmetic` a `Extrakt Bits` umožňují aritmetické posuvy a získání vybraných bitů z celočíselného vstupu. Podskupina bloků **Edge Detection** umožňují



obrázek 13-11 Knihovna Simulink - Math Operations

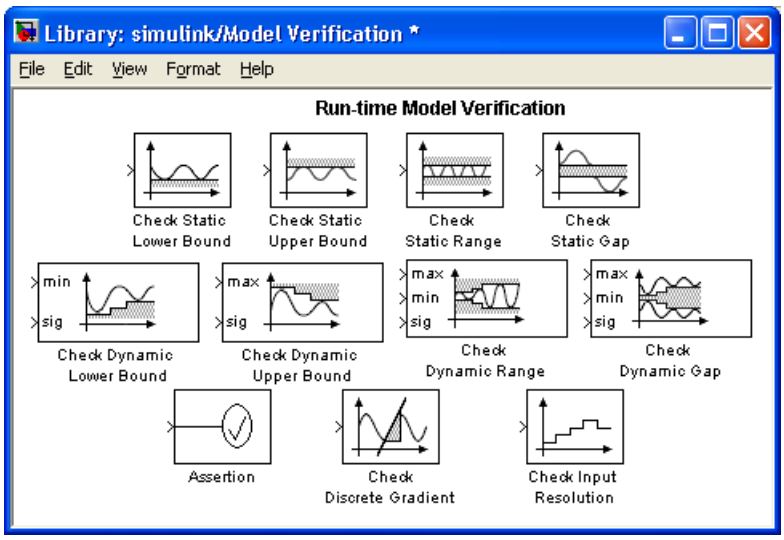


obrázek 13-10 Knihovna Simulink - Logic and Bit Operations

detekovat různé změny ve vstupu (náběžnou, sestupnou hranu a obojí).

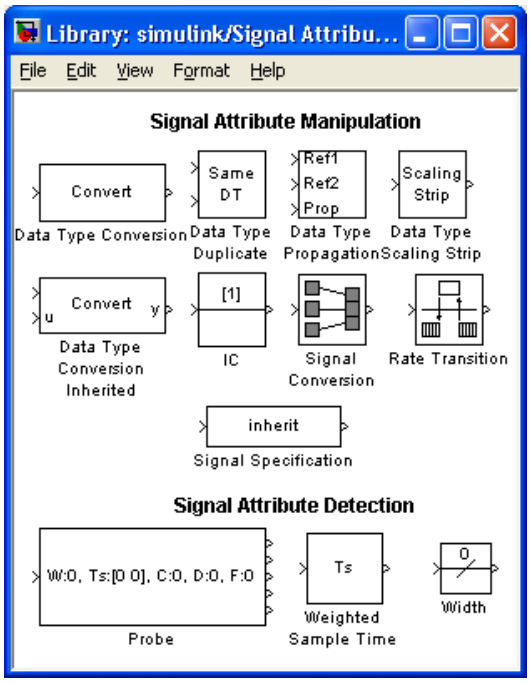
Bloky ze skupiny ***Math Operations*** (obr. 13-11) jsou asi používané nejčastěji. Tuto skupinu tvoří 30 bloků, pomocí kterých se realizuje zápis algebraické časti modelu. Blok `Sum` (součet vstupních signálů) je v knihovně ve 4 modifikacích jako `Sum`, `Add`, `Substract` a `Sum of Elements`. Blok `Bias` umožní přičíst ke vstupnímu signálu konstantu a blok `Weighted Sample Time Math` provede přičtení nebo vynásobení vstupního signálu hodnotou intervalu vzorkování nebo vzorkovací frekvencí. Blok `Gain` provede vynásobení vstupního signálu konstantou (prvek po prvku nebo maticově). Variantou bloku Gain je blok `Slider Gain`, který se liší tím, že hodnotu skalární konstanty lze během výpočtu měnit pomocí posuvníku v samostatném okně. Na první pohled podobné bloku `Gain` jsou bloky `Product` (násobení nebo dělení vstupů – prvek po prvku nebo maticově) a `Dot Product` (skalární součin). Ve všech případech jde sice o násobení, ale v případě `Gain` násobení konstantou, ve zbývajících případech jde o násobení signálů, jejichž hodnoty se v průběhu výpočtu mění. Bloky `Divide` a `Product of Elements` jsou variantami bloku `Product`. Blok `Sign` vrací znaménko vstupu, `Abs` jeho absolutní hodnotu a blok `Unary minus` jej neguje. Blok `Math Function` umožňuje zadat různé matematické

funkce (z maticových operací je zde například transpozice). Blok `Rounding Function` provede různá zaokrouhlení. Blok `Polynomial` přepočítá vstupní signál pomocí zadaného polynomu (jako funkce MATLABu *polyval*). Blok `Minimax` vrací maximální nebo minimální hodnotu vstupu. Jeho modifikace `Minimax Running Resettable` umožňuje výstup resetovat na zadané počáteční hodnoty. Blok `Trigonometric Function` počítá trigonometrické funkce vstupu. Blok `Sine Wave` počítá sinus z externího signálu nebo simulačního času. Blok `Algebraic Constraint` počítá výstup tak, aby jeho vstup byl nulový (musí existovat vazba mezi jeho výstupem a vstupem). Blok `Assignment` umožňuje zapsat na zadanou pozici vektoru nebo matice. Blok `Matrix Concatenation` sloučí vstupy horizontálně nebo svisle do jednoho signálu. Blok `Reshape` mění rozměry vstupního signálu (vytvoří 1-D pole, řádkový nebo sloupcový vektor nebo matici daných rozměrů). Bloky `Complex to Magnitude-Angle`, `Magnitude-Angle to Complex`, `Complex to Real-Imag`, `Real-Imag to Complex` jsou konverze mezi různými vyjádřeními komplexních čísel.

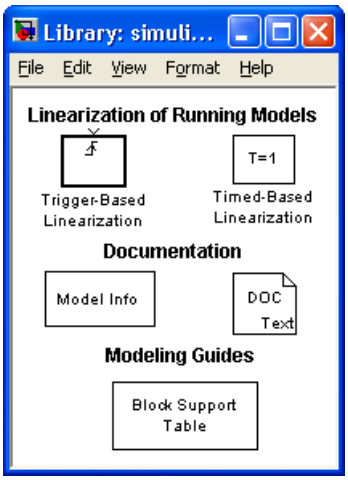


obrázek 13-12 Knihovna Simulink – Model Verification

Ve skupině ***Model Verification*** (obr.13-12) jsou bloky pro testování signálů. Pokud vstupní signál bloků `Check Static Lower Bound`, `Check Static Upper Bound`, `Check Static Range` a `Check Static Gap` překročí dolní mez, horní mez, interval nebo naopak do zadaného intervalu patří, je simulace zastavena a zobrazeno chybové hlášení. U bloků obsahujících v názvu `Dynamic` jsou limity dymamicky měněny vstupními signály. Blok `Assertion` testuje zda není některý prvek signálu nula. Blok `Check Discrete Gradient` kontroluje, zda absolutní hodnota rozdílu dvou po sobě jdoucích vzorků diskrétního signálu je menší než zadaná horním mez. Blok `Check Input Resolution` testuje, zda má vstupní signálů dané rozlišení.



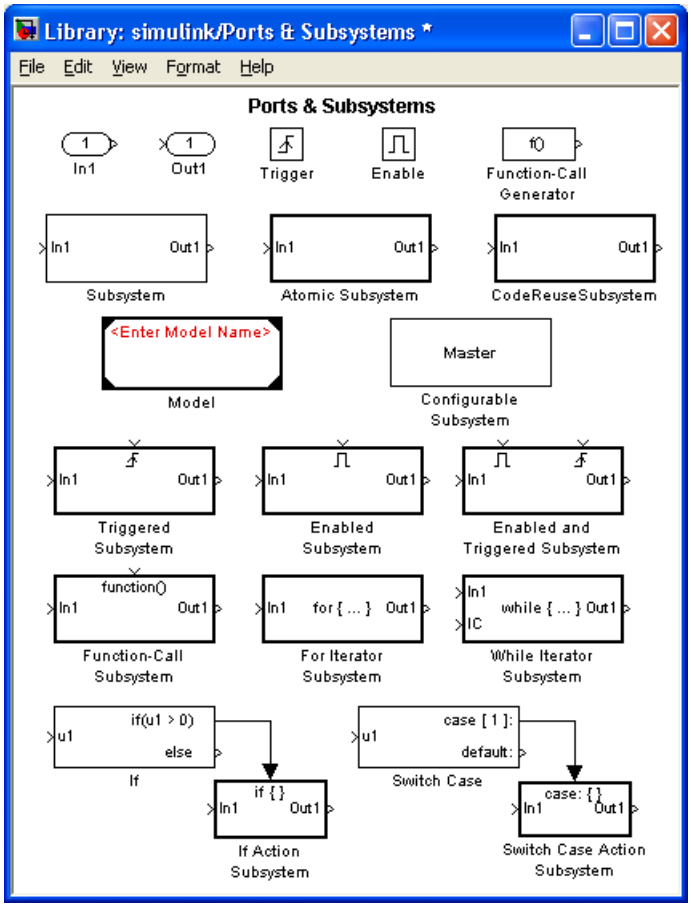
obrázek 13-13 Knihovna Simulink – Signal Attributes



obrázek 13-14 Knihovna Simulink – Model-Wide Utilities

Ve skupině ***Model-Wide Utilities*** (obr. 13-13) jsou bloky `Trigger-Based Linearization` a bloky `Timed-Based Linearization` umožňující linearizaci modelu (je spuštěna externím signálem nebo v zadaný čas simulace). Blok `Model Info` zobrazuje vybrané informace o modelu, blok `DocBlock` slouží pro vytvoření textové dokumentace modelu a blok `Block Support Table` u bloků základní knihovny zobrazí informace o podpoře datových typů při generování kódu.

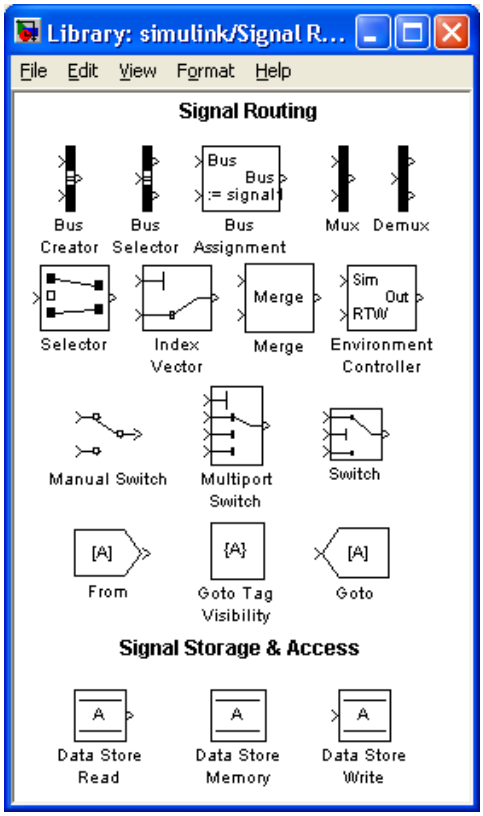
Ve skupině ***Signal Attributes*** (obr.13-14) jsou bloky



obrázek 13-15 Knihovna Simulink – Ports & Subsystems

související s vlastnostmi signálů. Blok `Data Type Conversion` převádí různé datové typy a umožňuje normování signálu. Pomocí bloku `Data Type Duplicate` lze testovat, zda mají všechny signály stejný datový typ. Blok `Data Type Propagation` nastaví typ a normování signálu podle informací z referenčních signálů. Blok `Data Type Scaling Strip` odstraní normování signálu s pevnou řádovou čárkou a přizpůsobí datový typ (zachování informace s minimálním počtem bitů). Blok `Data Type Conversion Inherited` převádí datový typ a normování druhého signálu podle prvního vstupu. Pomocí bloku `IC` lze nastavit počáteční hodnotu signálu. `Blok Signal Conversion` převádí signály na nové typy bez změny jejich hodnot. Blok `Rate Transition` souvisí s volbou mechanismu přenosu dat mezi dvěmi systémy s různým vzorkováním. Blok `Signal Specification` nastaví požadované rozměry, interval vzorkováni, datový typ, numerický typ a další vlastnosti signálu. Blok `Probe` má na výstupu a také zobrazuje během simulace vlastnosti signálu jako šířku, interval vzorkování, rozměr a další. Blok `Weighted Sample Time` umožňuje různé aritmetické operace mezi vstupním signálem a intervalem vzorkování. Blok `Width` vrací šířku vstupního signálu.

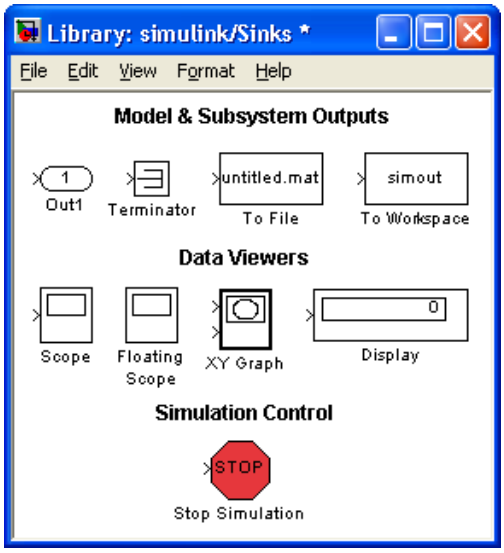
Skupina ***Ports & Subsystems*** (obr.13-15) obsahuje bloky související s tvorbou subsystémů a dovolující do SIMULINKového modelu zahrnout standardní programové konstrukce. Jde o řešení opakování či rozhodování v modelu (`For Iterator Subsystem`, `While Iterator Subsystem`, `If Action Subsystem`, `Switch Case Action Subsystem`). V souvislosti s možností používání maticových operací umožňují tyto bloky realizovat např. složitější diskrétní regulační algoritmy přímo na úrovni modelu. Bloky `In1` a `Out1` jsou vstupní a výstupní bod pro subsystémy. Bloky `Trigger` a `Enable` umožní vytvořit `Triggered` a `Enabled Subsystems` (subsystém spouštěný hranami nebo hladinou signálu). Blok `Function-Call Generator` je modifikací bloku `Trigger` a umožňuje vykonání bloku `Function-Call Subsystem`, který je jinak volaný z S-Funkce. Blok `Atomic Subsystem` je vykonáván celý najednou odděleně od ostatních bloků stejné úrovně (vlastnost *Treat as Atomic Unit* ovlivní pořadí vykonávání bloků). Blok `Model` umožní vložit celý model SIMULINKu jako blok. Blok `Configurable Subsystem` představuje pomocí dialogu vybraný blok z uživatelské knihovny.



obrázek 13-16 Knihovna Simulink – Signal Routing

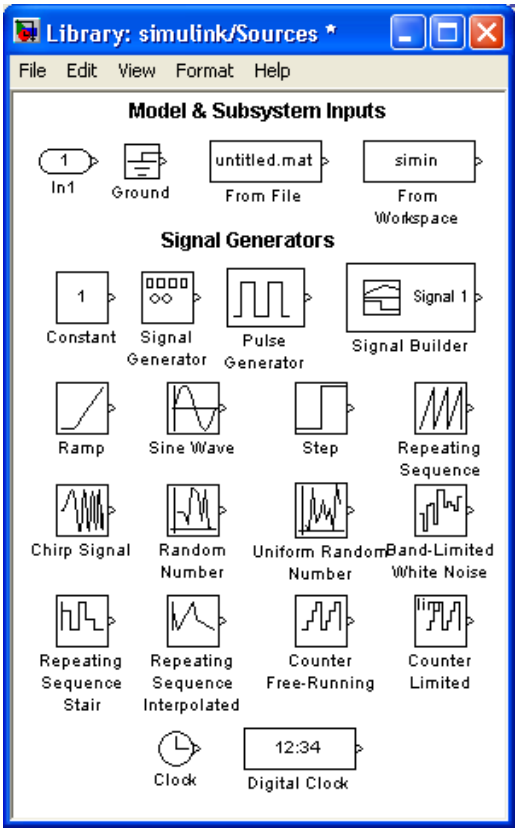
Ve skupině ***Signal Routing*** (obr.13-16) jsou bloky potřebné pro vytváření a práci se sběrnicovými signály, přepínání signálů a bloky pro propojování bez nutnosti kreslení signálů. Bloky `Bus`

Creator a Mux vytvoří sběrnici (bus). Pomocí blok Bus Selector lze lze vytvářet další sběrnice nebo získat signály. Blok Demux rozloží buďto vektor na skaláry nebo sběrnici na signály. Blok Bus Assignment dovolí přiřadit signálu ve sběrnici jiný signál. Pomocí bloku Selector lze z vektoru nebo matice vybrat požadované prvky. Blok Index Vector vybere z vektoru požadovaný prvek. Blok Merge sloučí více signálů v jeden. Blok Environment Controller souvisí s generováním kódu ze SIMULINKu. Pomocí bloku Manual Switch lze přepínat mezi dvěmi signály „ručně“ kliknutím myši. Blok Multiport Switch přepíná signály podle zaokrouhlené hodnoty řídicího signálu. U bloku Switch dochází k přepnutí signálu na základě vyhodnocení logické podmínky pro řídicí signál. Pomocí bloků From, Goto Tag Visibility a Goto lze spojovat bloky bez kreslení signálů. Podobnou funkci mají bloky Data Store Read, Data Store Memory a Data Store Write (ovšem dochází k zápisu hodnot do datové paměti a k jejich následnému čtení).



obrázek 13-17 Knihovna Simulink – Sinks

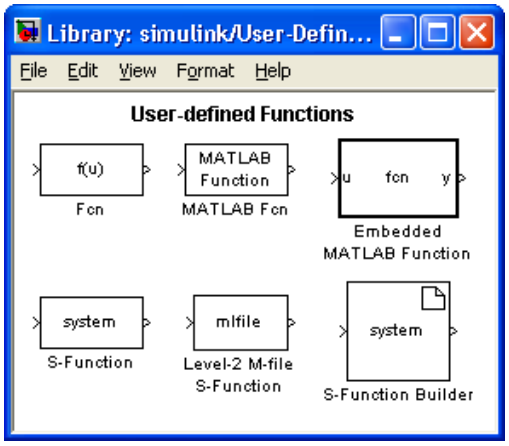
Důležitou skupinou je skupina ***Sinks*** (obr. 13-17) obsahující bloky pro výstup výsledků. Asi nejpoužívanějším blokem je blok Scope případně Floating Scope. Oba zobrazují ve formě grafu časový průběh vstupních signálů – podobně jako zapisovač či osciloskop. Pro případ, že je potřeba vynést průběh dvou signálů vůči sobě, je určen blok XY Graph. Zobrazení hodnoty signálu v číselné podobě umožňuje blok Display. Užitečný je blok To Workspace, který dovoluje zaznamenat do proměnné v pracovním prostoru MATLABu průběh signálu(ů) pro následné zpracování. Obdobně funguje blok To File s tím rozdílem, že se hodnoty zaznamenají do souboru na disku. V případě, že potřebujeme ukončit výpočet modelu při dosažení zvolené hodnoty některého signálu, použijeme blok Stop Simulation. Blok Out (podobně jako blok In v knihovně Sources) je užitečný v případě tvorby subsystémů. Blok Terminator se používá pro korektní ukončení signálu, který není nikam přiveden (model by byl sice funkční, ale objevilo by se hlášení typu Warning).



obrázek 13-18 Knihovna Simulink – Sources

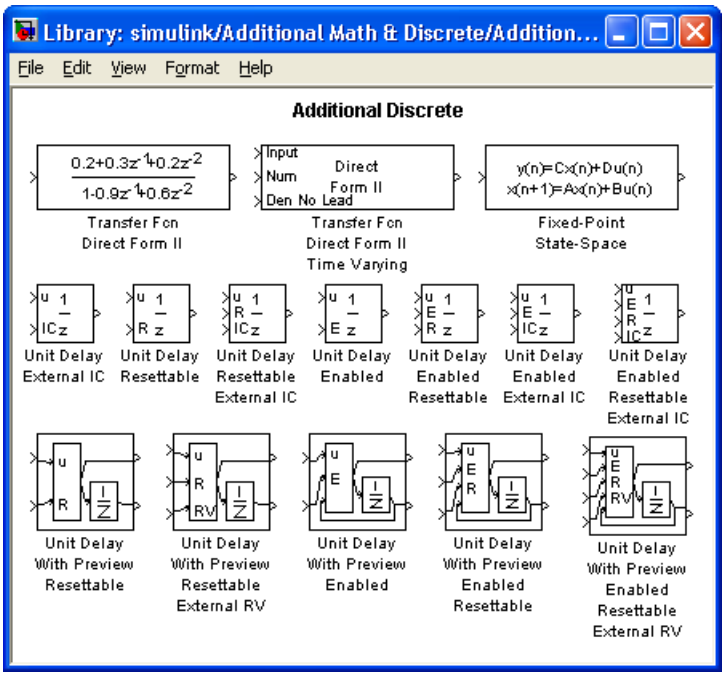
Další důležitou skupinou je skupina ***Sources*** (obr. 13-18). Zde jsou soustředěny bloky používané jako vstupy a zdroje signálů. Nejčastěji používané jsou bloky Constant (zdroj signálu časově neproměnného) a Step (skoková změna signálu v definovaném čase). Pokud potřebujeme mít k dispozici explicitně čas výpočtu použijeme blok Clock nebo Digital Clock (diskrétní verze). Blok In1 byl zmiňován v předchozím odstavci v souvislosti s blokem Out. Blok Ground má podobný význam jako blok Terminator – v tomto případě definovat hodnotu nepřipojeného vstupu. Bloky From Workspace a From File umožňují načíst průběh signálu z pracovního prostotu MATLABu či souboru na disku. Průběh signálu musí být doplněn časovým údajem. SIMULINK zajistí automatickou interpolaci pro odhad hodnoty signálu v časech, které nejsou v datech k dispozici, ale probíhá v nich výpočet modelu. Další bloky vytvářejí časově proměnný signál definovaných vlastností. Ať už jde o signály

popsané matematickou funkcí – bloky `Sine Wave`, `Signal Generator` a `Chirp Signal`[103], nebo standardní tvary signálů – `Pulse Generator`, `Ramp` a `Repeating Sequence`, `Repeating Sequence Stair`, `Repeating Sequence Interpolated`, či různé typy náhodných signálů – bloky `Band-Limited White Noise`[104], `Random Nunber` a `Uniform Random Number`. Blok `Signal Generator` umožní vytvořit signál graficky (po částech lineární). Bloky `Counter Free Running` a `Counter Limited` představují přetékající čítač s daným počtem bitů a čítač s daným horním limitem.



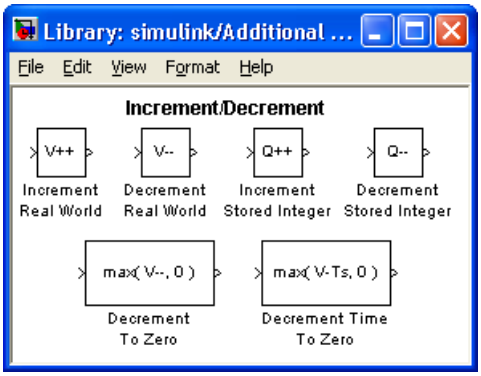
obrázek 13-19 Knihovna Simulink – User-Defined Functions

Ve skupině ***User-Defined Functions*** (obr. 13-19) jsou bloky umožňující vytvářet uživatelské funkce. Blok `Fcn` dovoluje zadat v syntaxi MATLABu výraz určující, jak se má vstupní signál přepočítat na výstupní (jeho vstupem může být skalár nebo vektor, výstup je vždy skalár) a `MATLAB Fcn` dovoluje s určitými omezeními volat funkce MATLABu (funkce musí mít jeden vstup a jeden výstup). Blok `Embedded MATLAB Function` umožňuje zapsat funkci MATLABu přímo v modelu SIMULINKu (může mít více vstupů i výstupů). Blok `S-Function` je uživatelský blok v jazyce C, M (level-1), Fortran a Ada (musí odpovídat standardu S-funkce). Jak se vyvíjel SIMULINK vznikla novější verze `Level-2 M-file S-Function`. Blok `S-Function Builder` umí vytvořit S-funkci z kódu v jazyce C.



obrázek 13-20 Knihovna Simulink – Aditional Discrete

Skupina bloků ***Aditional Math & Discrete*** obsahuje 2 položky. V položce ***Aditional Discrete*** (obr. 13-20) jsou dvě modifikace bloku diskrétní přenosové funkce (přímý přenos a přenos s časově proměnnými koeficienty). Ostatní bloky jsou různé modifikace bloku `Unit Delay` (s externími počátečními podmínkami, s resetem, aktivovaný, `With Preview` znamená že na jednom jeho výstupu je také aktuální hodnota). V položce ***Aditional Math: Incerement – Decrement*** (obr. 13-21) jsou bloky umožňující povýšit nebo snížit hodnotu signálu o 1 (rozlišují mezi reálnou hodnotou a uloženou celočíselnou hodnotou signálu).



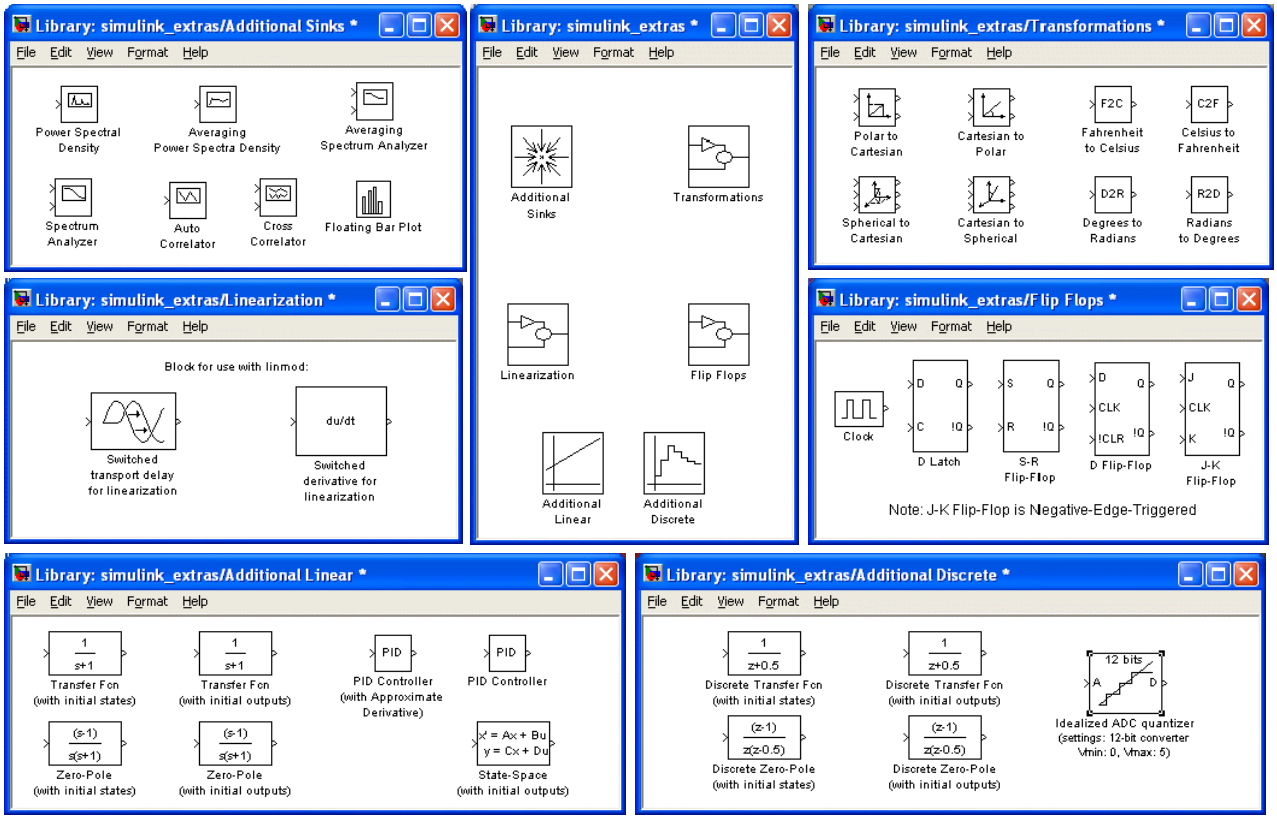
obrázek 13-21 Knihovna Simulink – Aditional Math: Incerement – Decrement

Skupina bloků ***Simulink Extras*** (obr. 13-22) obsahuje 6 položek. V položce ***Aditional Discrete*** jsou rozšířeny možnosti bloků `Discrete Transfer Fcn` a `Discrete Zero-Pole` o

[103] signál s časově proměnnou frekvencí

[104] jde o aproximaci bílého šumu tj. náhodného signálu, který má stejný výkon na všech frekvencích. Tento signál je prakticky nerealizovatelný a proto se používají různé aproximace. Tento blok zajišťuje signál, který v určitém rozsahu frekvencí má přibližně stejný výkon

zadání počátečních podmínek (jako stavy nebo kombinací vstupů a výstupů). Blok `Idealized ADC quantizer` představuje idealizovaný lineární AD převodník. V položce ***Aditional Linear*** jsou obdobná rozšíření bloků `Transfer Fcn`, `Zero-Pole` a `State-Space` o možnost zadat počáteční podmínky. V této položce jsou také dvě verze PID regulátoru. Položka ***Additional Sinks*** obsahuje bloky pro výpočet a vizualizaci výkonové spektrální hustoty, spektra, vzájemné korelace a blok pro sloupcový graf. V položce ***Flip Flop*** jsou bloky simulující činnost klopných obvodů a blok pro časování. Položka ***Linearization*** obsahuje bloky, kterými se mají nahradit bloky derivace a dopravního zpoždění v případě, že je model linearizován funkcí *linmod*. Poslední položka ***Transformations*** obsahuje přepočty mezi různými souřadnicovými systémy (mezi kartézskými, polárními a sférickými souřadnicemi, teplotou ve stupních Celsia a Fahrenheita, úhlovými stupni a radiány).



obrázek 13-22 Knihovna Simulink – Simulink Extras – Additional Discrete, Additional Linear, Additional Sinks, Transformations, Flip flop, Linearization

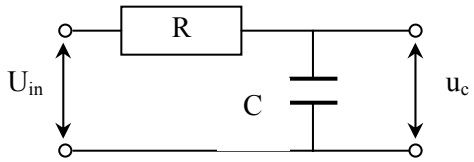
14. Subsystémy a knihovny

Důvody pro tvorbu subsystémů a uživatelských knihoven jsou obecně stejné jako důvody pro psaní podprogramů a vytváření knihoven v klasickém programování. Subsystémy umožňují přehledný zápis složitého modelu a zjednodušují tvorbu modelu v případě, že se sestává z více stejných nebo podobných částí. Také usnadnění ladění složitých modelů může být nezanedbatelným důvodem pro rozdělení modelu na samostatně odladitelné subsystémy. Rozhodnutí zda v modelu použít subsystém(y) je plně na tvůrci modelu. Vždy by se měl používat zdravý rozum. Jeden extrém je vytvořit celý složitý model v jednom okně zabírajícím velikost deseti obrazovek a druhý extrém je vytvářet desítky subsystémů zahrnujících třeba jen dva bloky. Z hlediska přehlednosti se zdá rozumné uvažovat o použití subsystému v okamžiku, kdy se model už nemůže vejít do okna velikosti obrazovky. Tvorba knihoven (seskupení třeba i maskovaných subsystémů obvykle souvisejících s řešením nějakého problému) umožňuje oddělit obecně použitelnou část programu od části specializované na řešení konkrétního problému.

14.1 Subsystémy

Subsystém obvykle zahrnuje určitý logický celek (spolu související operace), který si s okolím vyměňuje relativně omezené množství informací. Tato výměna informací je realizována prostřednictvím vstupních a výstupních proměnných (signálů).

Subsystém v SIMULINKu vytvoříme tak, že sestavíme standardně model a označíme vstupní a výstupní signály pomocí bloků `In` a `Out`. Tyto bloky můžeme pojmenovat (názvy se přenesou do označení vstupních a výstupních signálů bloku, který vznikne po vytvoření subsystému). Po označení vstupů a výstupů vybereme[105] všechny bloky, které mají být v subsystému, a volbou *Edit->Create Subsystem* subsystém vytvoříme. V případě, že chceme vytvořit subsystém z části existujícího modelu, stačí tuto část označit (včetně signálů) a zvolit vytvoření subsystému (bloky vstupu a výstupu se do subsystému přidají automaticky). Další možností jak vytvořit subsystém je přetáhnout blok subsystému z knihovny SIMULINKu. Blok subsystému lze dvojklikem otevřít do samostatného okna a dále upravovat.

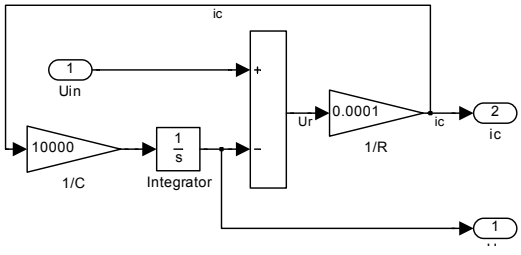


obrázek 14-1 Schéma RC členu

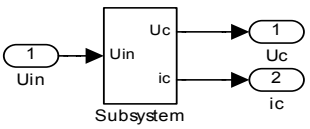
Ukažme si příklad vytvoření subsystému popisujícího chování elektrického RC členu. Základní zapojení zobrazené na obr. 14-1 je popsáno rovnicemi:

$$U_{in} = u_R + u_C \quad kde \quad u_R = Ri_c \qquad \frac{du_C}{dt} = \frac{1}{C} i_c \qquad U_{in} = Ri_c + \frac{1}{C}\int_0^t i_c\,dt \qquad i_c = \frac{1}{R}\left(U_{in} - \frac{1}{C}\int_0^t i_c\,dt\right)$$

Zvolíme-li kapacitu C = 100 μF a odpor R = 10 kΩ, můžeme zapsat model (včetně označení vstupu a výstupu tj. připravený na vytvoření subsystému) např. jak je uvedeno na obr. 14-2. Označíme-li nyní všechny bloky a vytvoříme subsystém, dostaneme nový blok s jedním vstupem a dvěma výstupy jak je ukázáno na obr. 14-3.



obrázek 14-2 Subsystém - vnitřní schéma



obrázek 14-3 Subsystém - vnější popis

Chceme-li nyní použít kopie tohoto subsystému s různými hodnotami parametrů, musíme nové

[105] výběr provedeme buďto výběrem obdélníkové oblasti myší (kurzor myši přesuneme do rohu oblasti stiskneme a držíme levé tlačítko myši a ukazatel přesuneme do protilehlého rohu oblasti, po uvolnění tlačítka myši by měly být všechny bloky v oblasti vybrány) nebo stiskem klávesy Shift a klikáním levým tlačítkem myši na blocích, které chceme vybrat. Všechny bloky v daném okně lze vybrat kombinací kláves Ctrl+A.

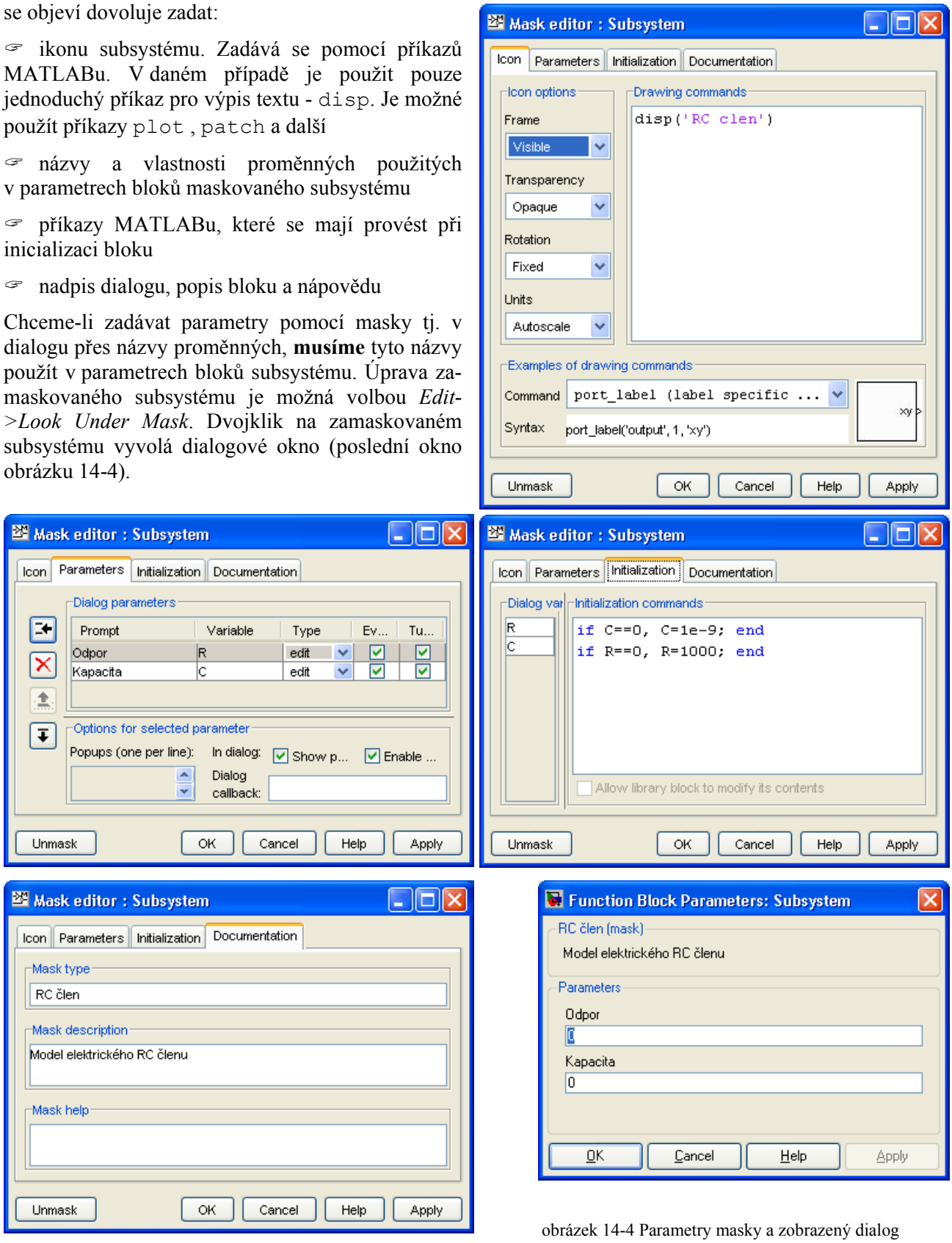
kopie editovat (změnit číselné hodnoty uvnitř subsystému). SIMULINK nabízí takzvané maskování subsystému, které nám umožní zadat hodnoty proměnných pomocí dialogu.

14.2 Maska subsystému

Maskování subsystému je dostupné po vybrání subsystému pod volbou *Edit->Mask Subsystem*... V okně jež se objeví dovoluje zadat:

- ikonu subsystému. Zadává se pomocí příkazů MATLABu. V daném případě je použit pouze jednoduchý příkaz pro výpis textu - `disp`. Je možné použít příkazy `plot` , `patch` a další
- názvy a vlastnosti proměnných použitých v parametrech bloků maskovaného subsystému
- příkazy MATLABu, které se mají provést při inicializaci bloku
- nadpis dialogu, popis bloku a nápovědu

Chceme-li zadávat parametry pomocí masky tj. v dialogu přes názvy proměnných, **musíme** tyto názvy použít v parametrech bloků subsystému. Úprava zamaskovaného subsystému je možná volbou *Edit->Look Under Mask*. Dvojklik na zamaskovaném subsystému vyvolá dialogové okno (poslední okno obrázku 14-4).

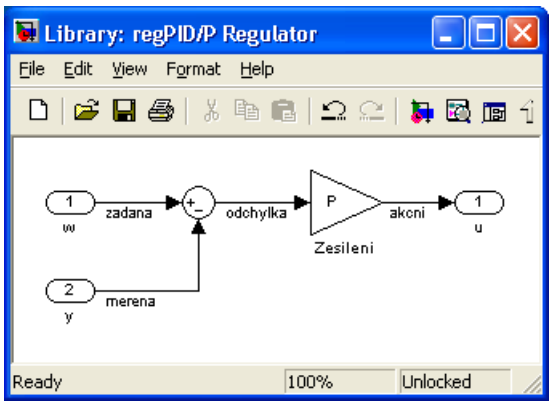


obrázek 14-4 Parametry masky a zobrazený dialog

14.3 Knihovny

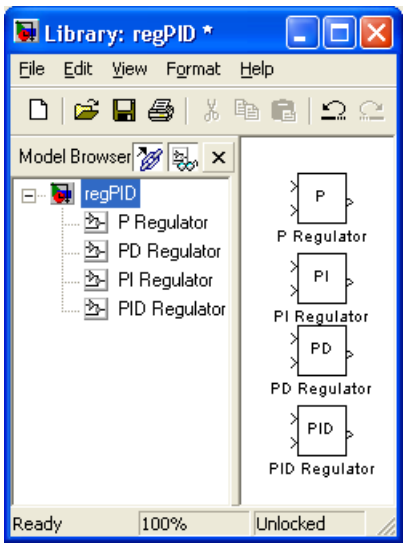
Další rozšíření možností subsystémů představuje knihovna. Vytvářet knihovnu (sadu bloků) má význam v případě, že předpokládáme, že tyto bloky budeme opakovaně používat v různých modelech. Uložení bloků ve formě knihovny umožňuje jejich jednoduché přetahování z knihovny do otevřeného modelu a automatickou aktualizaci knihovních bloků použitých v modelu v případě, že byla knihovna upravena. Další vlastností je ochrana bloku použitého v modelu - knihovní modul není možné v modelu upravovat bez zrušení vazby na knihovnu.

Ukažme si jednoduchý příklad vytvoření knihovny, která bude obsahovat čtyři varianty ideálního spojitého PID regulátoru (P, PI, PD a PID). Knihovnu vytvoříme volbou *File->New->Libr*ary, která založí nové okno pro tvorbu bloků knihovny. Než budeme moci používat bloky z nové knihovny, musí být nejprve uložena (nastaví se ochrana). Pokud otevřeme již existující knihovnu, je chráněna proti nechtěným změnám. Před prováděním změn je nutné ochranu zrušit *Edit->Unlock Library*. Uzavřením okna knihovny se ochrana opět nastaví. Když zkopírujeme blok z existující knihovny je vytvořena vazba na příslušný knihovní blok a nelze jej upravovat bez zrušení vazby na knihovnu (příkazem *Edit->Link Option->Disable Link*). Vazba může být opět obnovena (příkazem *Edit->Link Option->Restore Link* je vyvolán dialog, který nabídne buďto použití původního knihovního bloku nebo aktualizaci knihovního bloku na základě provedených změn) nebo úplně zrušena příkazem *Edit->Link Option->Break Link*. Aktualizace bloků modelu dle stavu v knihovně se provádí při načtení modelu nebo při použití příkazu *Edit->Update Diagram*.

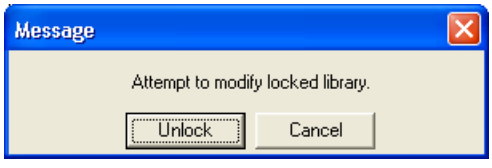


obrázek 14-5 Blok knihovny – P regulator

Vraťme se k našemu příkladu. Sestavíme model pro nejjednodušší regulátor P (obr. 14-5), označíme ho jako subsystém a zamaskujeme ho - vytvoříme masku pro zadání parametru P. Knihovnu uložíme např. pod názvem ***regPID***. To, že pracujeme s knihovnou se projeví v okamžiku, kdy budeme chtít tento P regulátoru jako základ dalšího bloku. Otevřeme-li zkopírovaný blok pro úpravy (*Edit->Look Under Mask*) zjistíme, že otevřeme vazbu (link) na původní blok. Pokud chceme vytvořit nový nezávislý blok, musíme zrušit vazbu příkazem *Edit->Link Option->Disable Link* a následně *Edit->Link Option->Break Link*.. Po zrušení vazby můžeme po příkazu *Edit->Look Under Mask* upravit schéma bloku a po příkazu *Edit->Edit Mask* také masku nového bloku. Takto připravíme všechny bloky. Výsledné okno knihovny pak obsahuje čtyři bloky jak je ukázáno na obr. 14-6.



obrázek 14-6 Uživatelská knihovna



obrázek 14-7 Dialog – povolení odemknutí knihovny pro úpravy

Příznak *Unlocked* v pravém dolním rohu indikuje, že jsme stále v režimu úprav knihovny. Po uzavření okna a novém otevření bude příznak nastaven na *Locked*. Před jakoukoliv úpravou nově otevřené knihovně je potřeba knihovnu odemknout příkazem *Edit->Unlock library* (nebo povolením v dialogu, který je při pokusu o provedení změny vyvolán – obr. 14-7). Bloky z takto připravené knihovny můžeme používat v dalších aplikacích.

15. Jednoduché řešené příklady (SIMULINK)

15.1 Zadání příkladů

Sestavte simulační schémata pro řešení následujících diferenciálních rovnic[106]. Vytvořte maskované subsystémy, tak aby bylo možné parametry a počáteční podmínky zadávat pomocí dialogu masky.

a) $y'' + a^2 y = 0, \quad a = 2\pi/5, \quad y(0) = 0, \quad y'(0) = 0.5$

b) $y'' + 2ay' + b^2 y = 0, \quad a = 1, \quad b = 2, \quad y(0) = 1, \quad y'(0) = -0.5$

c) $y' + y \cdot \cos(x) = e^{-\sin(x)}, \quad y(0) = 0$

d) $y'' + 4xy' + (4x^2 + 2)y = 0, \quad y(0) = 1, \quad y'(0) = -10$

e) $\frac{d^3y}{dt^3} + a_2 \frac{d^2y}{dt^2} + a_1 \frac{dy}{dt} + a_0 y = x(t), \quad a_0 = a_1 = a_2 = 1, \quad y(0) = 0, \quad y'(0) = 1, \quad y''(0) = 0$

f)
$$x' = -5x - 2y, \quad x(0) = 2$$
$$y' = x - 7y, \quad y(0) = -1$$

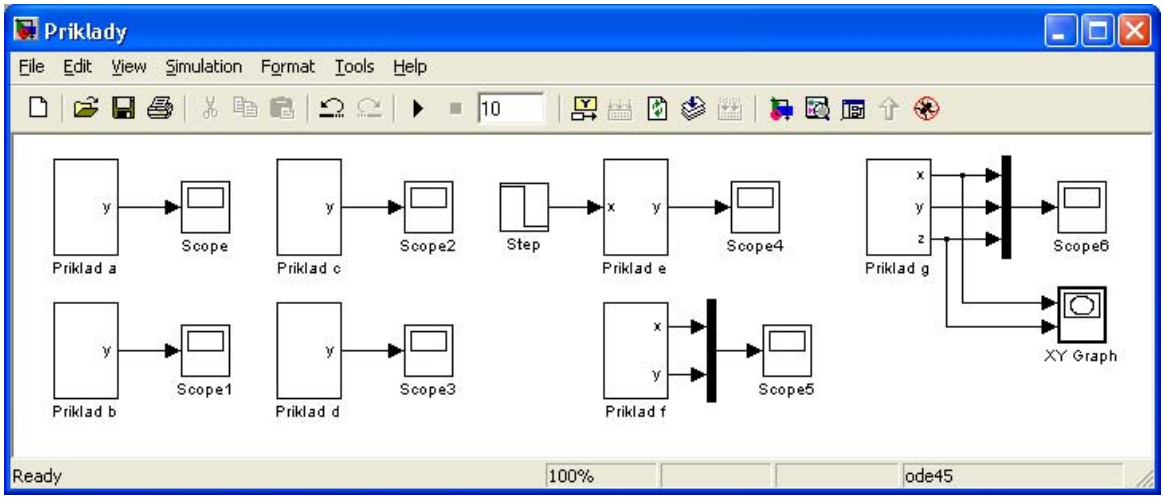
$$\frac{dx}{dt} = s(y - x), \quad s = 10, \quad x(0) = 7.69$$

g) $\frac{dy}{dt} = rx - y - xz, \quad r = 126.52, \quad y(0) = 15.61$

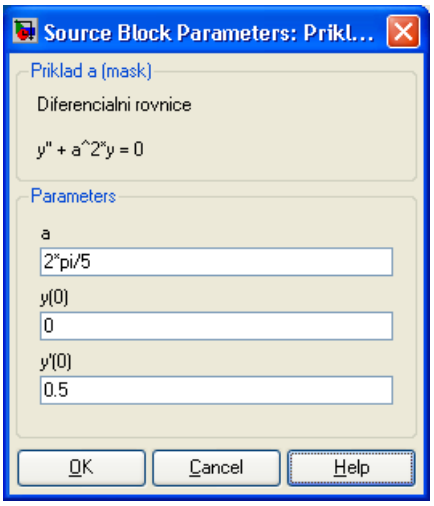
$$\frac{dz}{dt} = xy - bz, \quad b = 8/3, \quad z(0) = 90.39$$

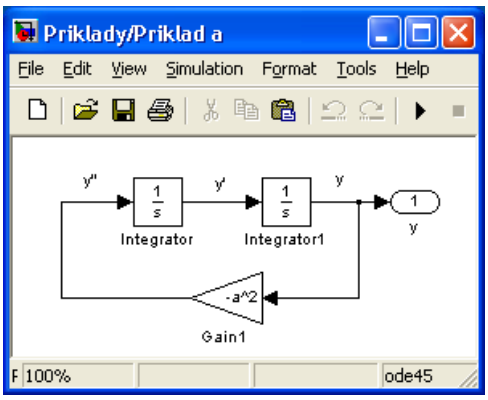
15.2 Řešení příkladů

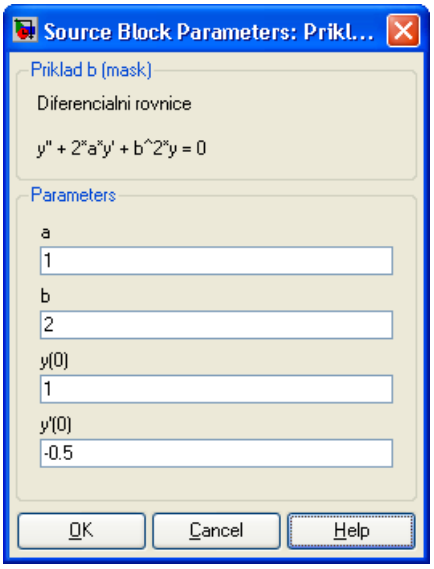
Existují různé postupy jak dospět k simulačnímu schématu. Jednou z možností je vyjádřit z rovnice nejvyšší derivaci proměnné veličiny a jejím postupným integrováním (průchodem signálu blokem integrátoru) získat nederivovanou veličinu (řešení). Na následujícím obrázku jsou subsystémy pro jednotlivé příklady spolu s bloky pro zobrazení výsledku simulace. Na dalších obrázcích jsou dialogy a simulační schémata maskovaných subsystémů.

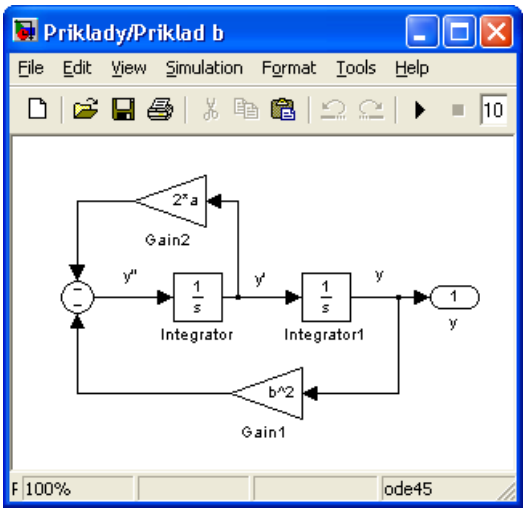


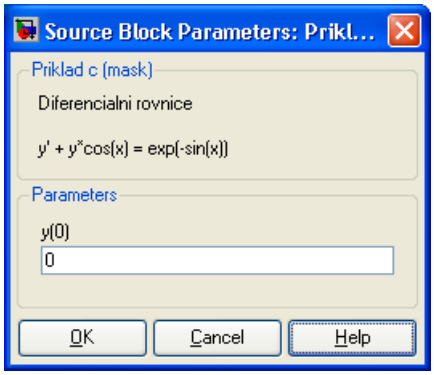
[106] Všimněte si formálního zápisu rovnic. Ze zápisu derivace y' není jasné, podle které proměnné derivujeme (V SIMULINKu má nezávisle proměnná vždy význam času). U příkladů c) a d) je nezávisle proměnná označena jako x. V příkladu e) je proměnná x obecnou funkcí času (konkrétně je použit jednotkový skok). Další otázkou je interpretace výsledku řešení. U příkladu f) lze je vynášet proměnné x a y proti sobě nebo jako funkce času. Obdobně je tomu i u příkladu g).

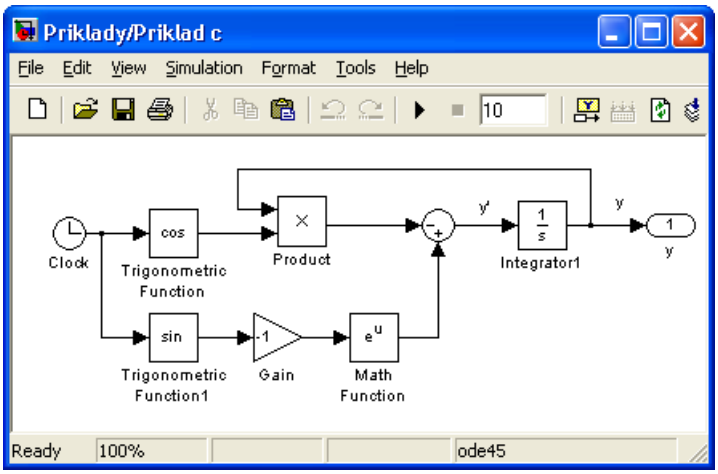

Source Block Parameters: Prikl...
Priklad a (mask)
Diferencialni rovnice
y" + a^2*y = 0
Parameters
a
2*pi/5
y(0)
0
y'(0)
0.5
OK
Cancel
Help

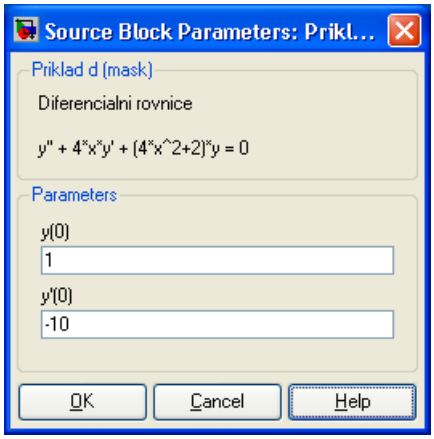

Priklady/Priklad a
File Edit View Simulation Format Tools Help
y"
1/s
Integrator
y'
1/s
Integrator1
y
1
y
-a^2
Gain1
100%
ode45

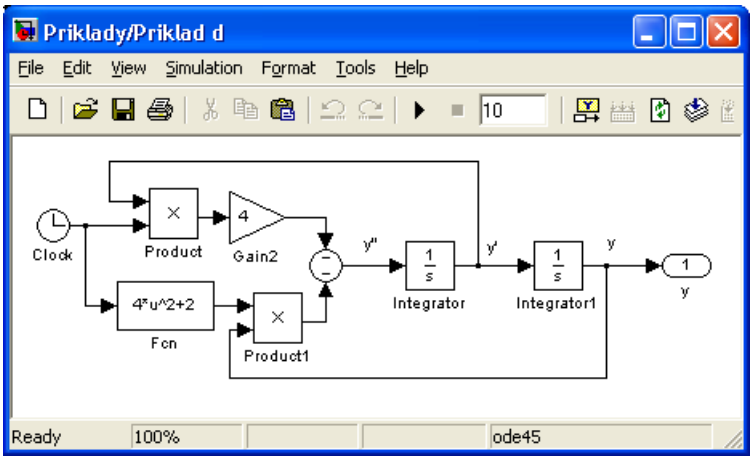

Source Block Parameters: Prikl...
Priklad b (mask)
Diferencialni rovnice
y" + 2*a*y' + b^2*y = 0
Parameters
a
1
b
2
y(0)
1
y'(0)
-0.5
OK
Cancel
Help

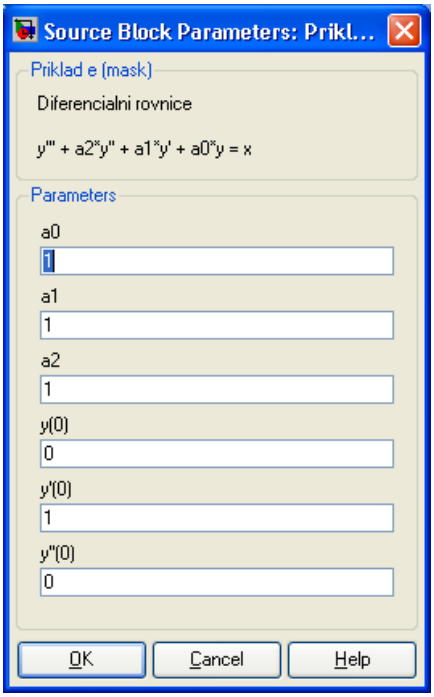

Priklady/Priklad b
File Edit View Simulation Format Tools Help
10
2*a
Gain2
y"
1/s
Integrator
y'
1/s
Integrator1
y
1
y
b^2
Gain1
100%
ode45

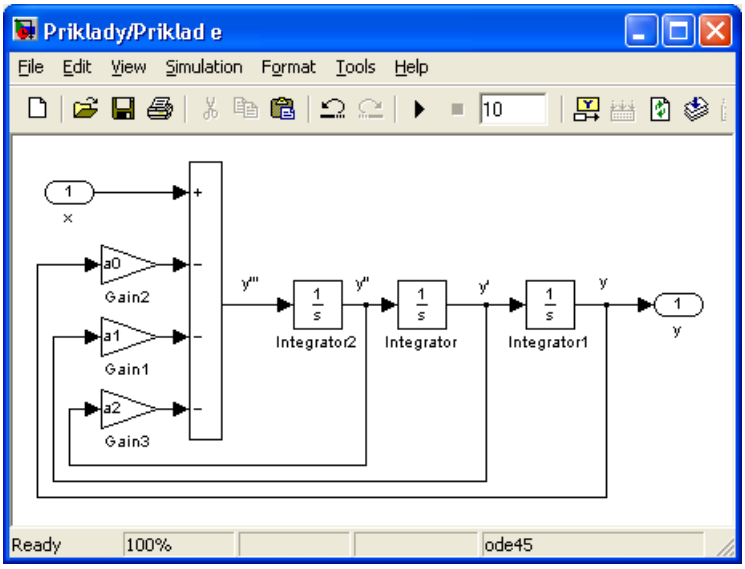

Source Block Parameters: Prikl...
Priklad c (mask)
Diferencialni rovnice
y' + y*cos(x) = exp(-sin(x))
Parameters
y(0)
0
OK
Cancel
Help

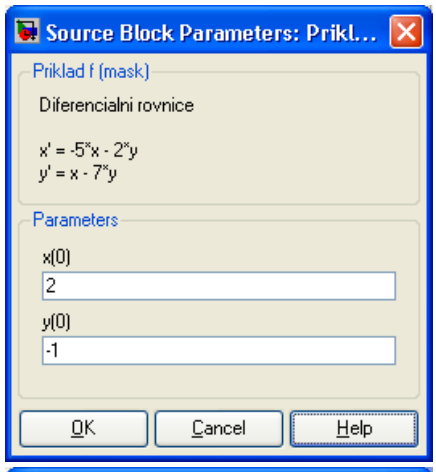

Priklady/Priklad c
File Edit View Simulation Format Tools Help
10
Clock
cos
Trigonometric Function
Product
y'
1/s
Integrator1
y
1
y
sin
Trigonometric Function1
-1
Gain
e^u
Math Function
Ready
100%
ode45

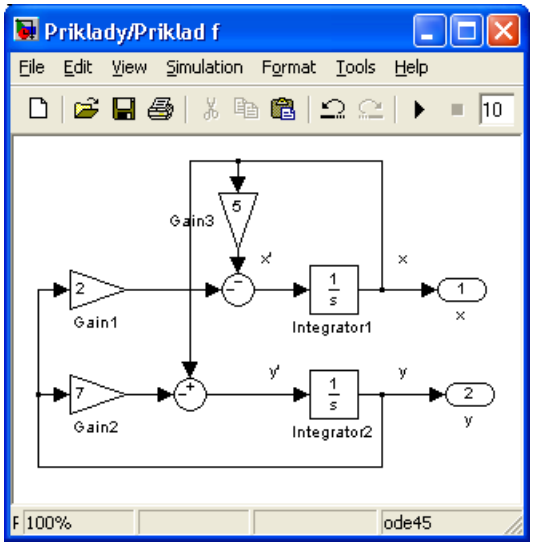

Source Block Parameters: Prikl...
Priklad d (mask)
Diferencialni rovnice
y" + 4*x*y' + (4*x^2+2)*y = 0
Parameters
y(0)
1
y'(0)
-10
OK
Cancel
Help

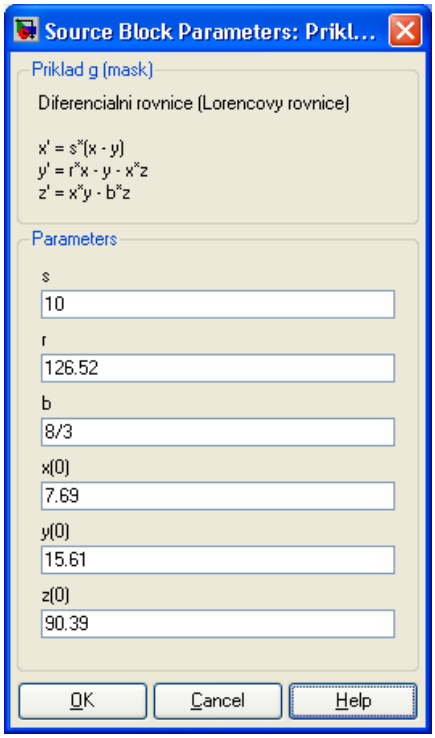

Priklady/Priklad d
File Edit View Simulation Format Tools Help
10
Clock
Product
4
Gain2
y"
1/s
Integrator
y'
1/s
Integrator1
y
1
y
4*u^2+2
Fcn
Product1
Ready
100%
ode45

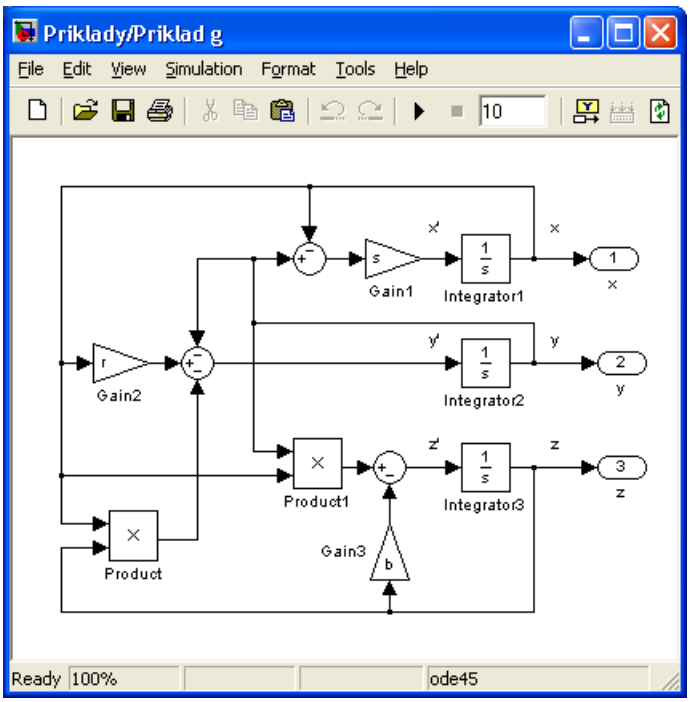

Source Block Parameters: Prikl...
Priklad e (mask)
Diferencialni rovnice
y''' + a2*y'' + a1*y' + a0*y = x
Parameters
a0
1
a1
1
a2
1
y(0)
0
y'(0)
1
y''(0)
0
OK
Cancel
Help


Priklady/Priklad e
File Edit View Simulation Format Tools Help
10
Gain2
Gain1
Gain3
Integrator2
Integrator
Integrator1
Ready
100%
ode45


Source Block Parameters: Prikl...
Priklad f (mask)
Diferencialni rovnice
x' = -5*x - 2*y
y' = x - 7*y
Parameters
x(0)
2
y(0)
-1
OK
Cancel
Help


Priklady/Priklad f
File Edit View Simulation Format Tools Help
10
Gain3
Gain1
Gain2
Integrator1
Integrator2
100%
ode45


Source Block Parameters: Prikl...
Priklad g (mask)
Diferencialni rovnice (Lorencovy rovnice)
x' = s*(x - y)
y' = r*x - y - x*z
z' = x*y - b*z
Parameters
s
10
r
126.52
b
8/3
x(0)
7.69
y(0)
15.61
z(0)
90.39
OK
Cancel
Help


Priklady/Priklad g
File Edit View Simulation Format Tools Help
10
Gain1
Gain2
Gain3
Product
Product1
Integrator1
Integrator2
Integrator3
Ready
100%
ode45

16. Komplexní řešené příklady (SIMULINK)

V kapitole 13 jsou ukázány základní prvky, způsob jejich použití a spuštění simulace výsledného modelu. V kapitole 14 jsou uvedeny postupy jak vytvořit nové uživatelské bloky (subsystémy) a případně uživatelské knihovny. Používání subsystémů je užitečné v případě složitého modelu a tvorba knihoven se vyplatí při předpokládaném častém používání v budoucnu. V této kapitole si ukážeme jak vytvořit schéma modelu ze zadání a několik příkladů na použití nejdůležitějších bloků. Studium řešených příkladů představuje efektivní cestu jak se naučit se SIMULINKem pracovat.

16.1 Sestavení modelu - výtok z nádrže

Použité bloky: `Integrator, Gain, Math Function, Mux, Pulse Generator, Scope, Sum`

Problém: nelineární funkce (odmocnina), počáteční stav integrátoru, omezení (saturace)

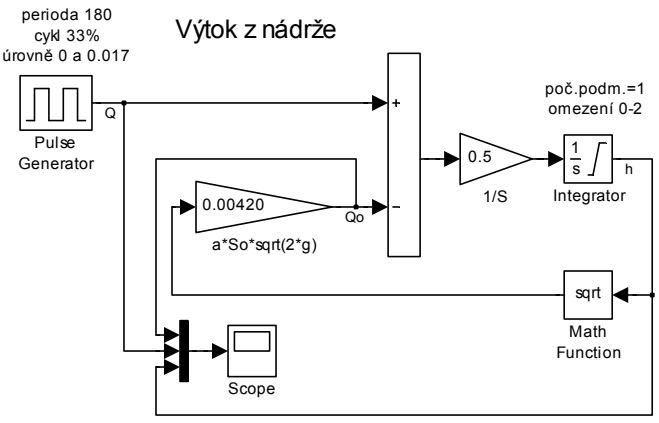
Mějme následující úlohu. Z nádoby o průřezu S = 2 m^2 a výšce H = 2 m vytéká voda (ρ = 1000 kg m^{-3}) otvorem ve dně o průřezu S_o = 0.001 m^2. Hodnota hydrodynamického součinitele je α=0.94. Výška hladiny na počátku sledovaného období je h = 1 m. Zjistěte, jak se bude měnit v čase výška hladiny h a vytékající množství Q_o. když nebude vždy 1 minutu nic přitékat (Q = 0) a pak 2 minuty bude přitékat množství 0.017 m^3 s^{-1}.

Z hmotové bilance systému $\underbrace{\rho Q}_{\text{vstupující hmota}} = \underbrace{\rho Q_o}_{\text{vystupující hmota}} + \underbrace{\frac{d(\rho S h)}{dt}}_{\text{akumulace hmoty}}$, Torriceliho vztahu pro výtokovou rychlost (s korekcí na skutečnou rychlost pomocí hydrodynamického součinitele α, g je tíhové zrychlení) $v_o = \alpha\sqrt{2gh}$ a souvislosti mezi hmotovým průtokem a rychlostí $\rho Q_o = \rho v_o S_o$ snadno sestavíme výslednou nelineární diferenciální rovnici prvního řadu pro závislost výšky hladiny h na přitékajícím množství Q

$$\mathbf{Q} = \alpha S_o\sqrt{2g\mathbf{h}} + S\frac{d\mathbf{h}}{dt}$$
$$h(t=0) = 1 \quad h \in <0,2>$$

Pro potřeby snadného převodu do schématu modelu tuto rovnici upravíme na ekvivalentní tvar implicitní (neznámá h se nachází na obou stranách) integrální rovnice

$$\mathbf{h} = \int_0^t \frac{1}{S}\left(\mathbf{Q} - \underbrace{\alpha S_o\sqrt{2g}}_{konstanta}\sqrt{\mathbf{h}}\right)dt$$



obrázek 16-1 Model Výtok z nádrže

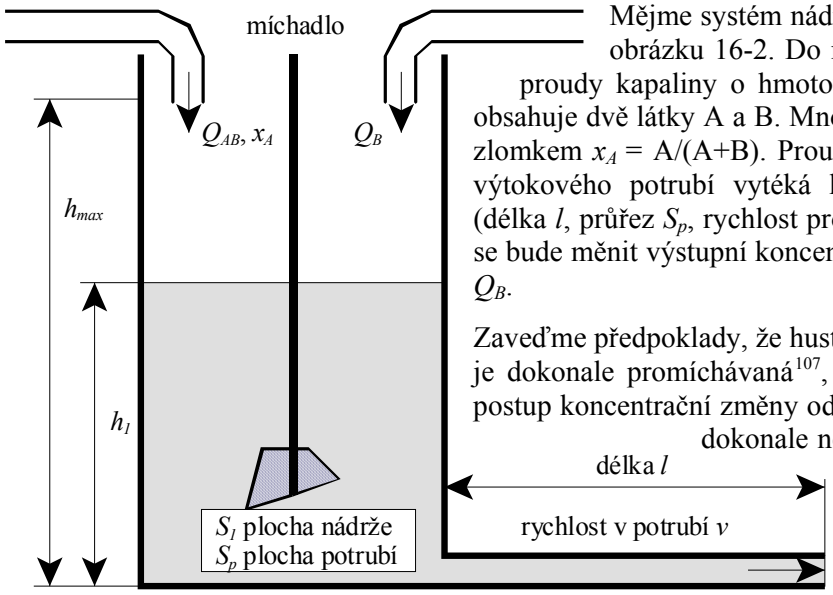
Tato rovnice již můžeme přímo přepsat do schématu modelu SIMULINKu. Potřebujeme jeden integrátor, jeden součet, dvě zesílení, funkci odmocniny a generátor vstupního signálu. Výstupem integrátoru bude hledaný časový průběh funkce h. Jeho vstup je tvořen součtem dvou signálů (blok `Sum`) vyděleným konstantou S (blok `Gain`). Jeden signál představuje časový průběh průtoku Q (blok `Pulse Generator`). Druhý, se znaménkem mínus, je tvořen odmocninou (blok `Sqrt`) z hledané funkce h vynásobené opět konstantou (blok `Gain`). Počáteční stav i omezení hledané funkce h - výšky hladiny - je zajištěno přímo nastavením parametrů integrátoru. Stejně tak obdélníkový průběh vstupního signálu - průtoku Q - je přímo generován knihovním blokem s příslušně nastavenými parametry. Vizualizace výsledku je zajištěna blokem `Scope` (osciloskop). Aby se všechny tři průběhy zobrazovaly v jednom grafu, jsou pomocí bloku `Mux` sloučeny do jednoho signálu (vektoru), který je veden do bloku `Scope`. Toto řešení nemusí být optimální

v případě, že sledované signály jsou značně odlišné. Potom je potřeba měnit měřítko pro zobrazení jednotlivých průběhů. Nyní se musíme rozhodnout jak dlouho budeme systém sledovat. Pokud budeme sledovat po tři periody opakování vstupního signálu nastavíme dobu sledování v parametrech simulace na dobu 540 s. Potom je již možné řešení spustit. Např. kromě již zmiňovaných způsobů i tlačítkem (s trojúhelníčkem) na nástrojové liště. Jestliže chceme vývoj řešení pozorovat, dvojklikem na bloku `Scope` otevřeme okno osciloskopu před spuštěním simulace. V opačném případě otevřeme okno až po skončení simulace.

16.2 Koncentrace v nádrži s dlouhým potrubím

Použité bloky: `Integrator, Gain, Product, Scope, Sine Wave, Sum, Variable Transport Delay`

Problém: součin proměnných, obecná funkce (převrácená hodnota), proměnné dopravní zpoždění, počáteční stav integrátoru, omezení (saturace)



obrázek 16-2 Schéma příkladu Koncentrace v nádrži s dlouhým potrubím

Mějme systém nádrž s odtokem dlouhým potrubím podle obrázku 16-2. Do nádrže (průřez S_1) jsou přiváděny dva proudy kapaliny o hmotových průtocích Q_{AB} a Q_B. Proud AB obsahuje dvě látky A a B. Množství látky A je vyjádřeno hmotovým zlomkem x_A = A/(A+B). Proud B obsahuje pouze látku B. Z konce výtokového potrubí vytéká kapalina o výsledné koncentraci x_o. (délka l, průřez S_p, rychlost proudění v). Naším úkolem je zjistit, jak se bude měnit výstupní koncentrace x_o při změnách průtoku kapaliny Q_B.

Zaveďme předpoklady, že hustoty ρ látek A a B jsou stejné, že nádrž je dokonale promíchávaná[107], že proudění v potrubí je laminární a postup koncentrační změny odpovídá pístovému toku, že kapalina je dokonale nestlačitelná a v celém prostoru potrubí je průměrná průtočná rychlost v. Za výše uvedených předpokladů lze sestavit celkovou hmotovou bilanci nádrže a hmotovou bilanci látky A.

$$Q_{AB} + Q_B = \underbrace{\rho S_p v}_{\text{odtok potrubím}} + \frac{d\left(\rho S_1 h_1\right)}{dt}$$

$$Q_{AB} x_A = \rho S_p x_1 v + \frac{d\left(\rho S_1 h_1 x_1\right)}{dt}$$

Výpočet výtokové rychlosti je zcela jiný než v předchozím příkladě. Tam se předpokládá, že se potenciální energie kapaliny (hydrostatický tlak) změní na kinetickou energie vytékající kapaliny (Torriceliho vztah). V tomto případě (významná hmotnost kapaliny v potrubí) je potřeba uvažovat zrychlení tj. další diferenciální rovnice. Průměrnou průtočnou rychlost v zjistíme z rovnováhy sil. Síla daná tlakovým rozdílem na obou koncích potrubí a jeho průřezem působí na hmotu kapaliny v potrubí a urychluje ji

$$\underbrace{S_p \Delta p}_{F} = \underbrace{S_p l \rho \frac{dv}{dt}}_{ma}$$

Rozdíl tlaků na obou koncích potrubí závisí na výšce sloupce kapaliny v nádrži a na ztrátách při proudění v potrubí. Teoretické ztráty při proudění v potrubí popisuje univerzální vzorec Darcy-Weisbachův ve tvaru

$$\Delta p_z = \lambda \frac{l}{d} \rho \frac{v^2}{2}$$

[107] tj. koncentrace uvnitř nádrže je stejná jako koncentrace na jejím výstupu (začátku potrubí)

kde pro laminární proudění lze určit parametr λ jako funkci Reynoldsova čísla *Re*

$$\lambda = \frac{64}{\mathrm{Re}} \quad \mathrm{Re} = \frac{vd}{\gamma} \quad \gamma = \frac{\eta}{\rho}$$

Potom rozdíl tlaků na obou koncích potrubí je dán hydrostatickým tlakem zmenšeným o tlakovou ztrátu díky proudění a je popsán následujícím vztahem

$$\Delta p = \underbrace{\rho g h_1}_{\text{rozdíl výšek}} - \underbrace{vl\frac{8\eta\pi}{S_p}}_{\text{tlaková ztráta}}$$

Koncentrace látky A vystupující z potrubí odpovídá vstupní koncentraci časově posunuté (o dopravní zpoždění) o hodnotu τ. Tato hodnota je závislá na délce potrubí *l* a rychlosti *v* podle vztahu τ = l/v. Výsledná soustava rovnic je

$$\frac{d\mathbf{h_1}}{dt} = \frac{Q_{AB}+Q_B}{\rho S_1} - \frac{S_p}{S_1}\mathbf{v} = 0.0005\left(Q_{AB}+Q_B\right) - 0.000157\mathbf{v}$$

$$\frac{d\mathbf{v}}{dt} = \frac{g}{l}\mathbf{h_1} - \frac{8\eta\pi}{\rho S_p}\mathbf{v} = 1.962\mathbf{h_1} - 0.08\mathbf{v}$$

$$\frac{d\mathbf{x_1}}{dt} = \frac{Q_{AB}x_A}{\rho S_1 \mathbf{h_1}} - \frac{Q_{AB}+Q_B}{\rho S_1}\frac{\mathbf{x_1}}{\mathbf{h_1}} = 0.0005\left[\frac{Q_{AB}x_A}{\mathbf{h_1}} - \frac{\mathbf{x_1}}{\mathbf{h_1}}\left(Q_{AB}+Q_B\right)\right]$$

$$x_o(t) = x_1(t-\tau) \quad \tau = l/\mathbf{v}$$

Zvolme konkrétní hodnoty parametrů jak je uvedeno v tabulce

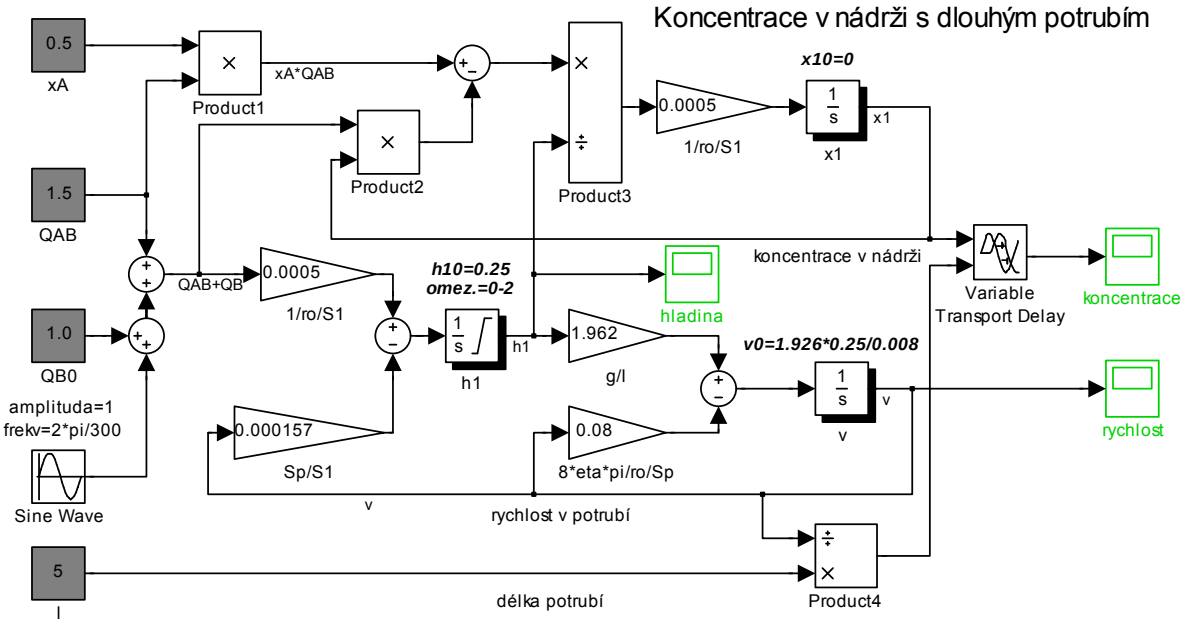
Označení	Hodnota	Rozměr	Význam
Q_{AB}	1.5	kg s^{-1}	hmotový průtok proudu AB
Q_B	0.0-2.0	kg s^{-1}	hmotový průtok proudu B
x_A	0.5	kg kg^{-1}	hmotový zlomek látky A
S_1	2	m^2	průřez nádrže 1
S_p	0.0003141	m^2	průřez propojovacího potrubí (průměr 2 cm)
l	5	m	délka propojovacího potrubí
ρ	1000	kg m^{-3}	hustota látek A a B (voda)
η	0.001	Pa s	dynamická viskozita

V následujícím modelu (obrázek 16-3) je vstupní průtok Q_B představován časově proměnným signálem popsaným rovnicí

$$Q_B(t) = 1.0 + 1.0\sin\left(\frac{2\pi}{300}t\right)$$

tj. mění se s periodou 300 s v rozmezí 0-2. Počáteční hodnoty proměnných h_1, x_1 a v (výchozí hodnoty integrátorů) jsou nastaveny do stavu odpovídajícího výšce hladiny 0.25 m a nulové koncentraci látky A v nádrži. Odpovídající ustálená rychlost proudění je v = 1.962×0.25/0.008. Dobu experimentu je vhodné nastavit na dobu aspoň 600 s (dvě periody vstupního signálu). Sledovány – pomocí bloků `Scope` jsou koncentrace x_o na konci potrubí, rychlost v v potrubí a výška h_1 v nádrži.

V modelu na obrázku 16-3 jsou zvýrazněny bloky `Constant` definující hodnoty signálů, které by se při opakovaných výpočtech pravděpodobně měnily. Při porovnání SIMULINKového modelu s matematickým modelem je vhodné se orientovat podle integrátorů (zvýrazněny stínováním), které odpovídají diferenciálním rovnicím. Celý model je nelineární – vypočítávaný signál (koncentrace x_1) je závislý na podílu



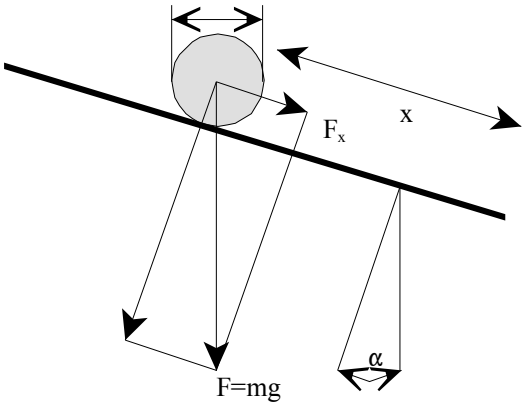
obrázek 16-3 Model Koncentrace v nádrži s dlouhým potrubím

vypočítávaných signálů (realizace blokem Product3). Nejzajímavějším blokem je blok `Variable Transport Delay`, který zajišťuje proměnný časový posun výstupního signálu vůči vstupnímu – závisí na vypočítávané rychlosti proudění.

16.3 Kulička na tyči (Ball on Beam)

Použité bloky: `Abs`, `Dead Zone`, `Gain`, `Hit Crossing`, `Integrator`, `Math Function`, `Memory`, `Product`, `Repeating Sequence`, `Scope`, `Sign`, `Sum`, `Switch`, `To Workspace`, `Trigonometric Function`

Problém: externí reset a počáteční podmínky integrátoru, součin proměnných, přepínač, indikace průchodu nulou, zápis do proměnné MATLABu



obrázek 16-4 Schéma příkladu Kulička na tyči

Na univerzitách technického směru je možné se setkat s laboratorním modelem "Kulička na tyči (Ball on Beam)". Tento model je často používán v laboratořích předmětu Teorie automatické regulace. Je tvořen kuličkou odvalující se po tyči. Tyč je uprostřed uchycena a je možné ji naklánět o určitý úhel na obě strany. Popišme časovou změnu polohy x kovové kuličky o poloměru r a hustotě ρ v závislosti na časové změně úhlu α naklonění tyče.

Sestavení pohybové rovnice vychází z rovnováhy sil. Toto sestavení bylo popsáno v kapitole 10.3 a zde uvedeme pouze základní a výsledné vztahy.

$$\underbrace{K\left(\frac{dx}{dt}\right)^2}_{odpor} + \underbrace{m\frac{d^2x}{dt^2}}_{posuvný\ pohyb} + \underbrace{\frac{1}{r}J\frac{d^2\varphi}{dt^2}}_{točivý\ pohyb} = \underbrace{mg\sin(\alpha)}_{působící\ síla}$$

$$J = \frac{8}{15}\pi r^5\rho \quad x = r\varphi \quad m = \rho\frac{4}{3}\pi r^3 \quad K = c_x\pi r^2$$

Tato rovnice popisuje ideální chování. Ve skutečnosti je pohyb ovlivňován statickým třením o hodnotě F_{st}. Jde o sílu, která působí proti síle pohyb vyvolávající jen pokud je rychlost nulová. Jinak tento odpor nepůsobí. Definujme její velikost pomocí úhlu α_{st}, o který je potřeba tyč naklonit, aby se kulička začala pohybovat tj. $F_{st} = mg\sin(\alpha_{st})$.

Další komplikací je to, že tyč je obvykle konečné délky a na koncích jsou dorazy. Na těchto dorazech dochází k speciální variantě tlumeného rázu (odrazu kuličky). Pro náš příklad budeme předpokládat, že tento ráz se projeví změnou orientace vektoru rychlosti kuličky (rychlost změní znaménko). Protože ráz není ideální dojde i ke ztrátám, které zahrneme poměrným snížením hodnoty rychlosti tj. $v = -\beta v$.

Zapišme výslednou soustavu rovnic (včetně převodu diferenciální rovnice druhého řádu na soustavu dvou rovnic prvního řádu) pro tyč délky 0.5 m a železnou kuličku (ρ = 7800 kg m^{-3}) o poloměru r = 0.01 m. Poloha 0 bude ve středu tyče tj. dorazy budou ve vzdálenosti a = ±0.25 m. Ztráty při odrazu budou 25% tj. β = 0.75. Úhel pomocí kterého definujeme sílu statického tření nechť je α_s = 2°.

Je potřeba zajistit, aby síly odporů působily vždy proti síle vyvolávající pohyb - proto je použita funkce sign.

$$\frac{dx}{dt} = v \qquad poloha$$

$$v = -\beta v \quad pro \quad x = -a \vee x = +a \quad odraz\ v\ poloze \pm a$$

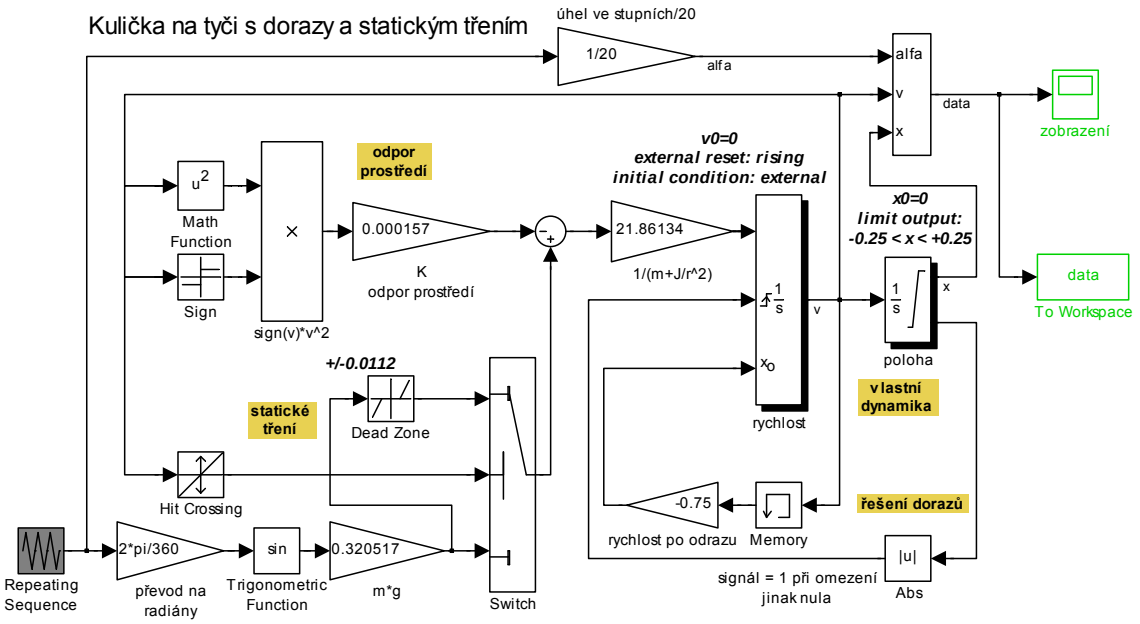
$$\frac{dv}{dt} = \begin{cases} \dfrac{mg\sin(\alpha) - sign(v)\times F_{st} - sign(v)\times Kv^2}{m + J/r^2} & v = 0 \wedge |F_x| > F_{st_s} \\ 0 & v = 0 \wedge |F_x| \le F_{st} \\ \dfrac{mg\sin(\alpha) - sign(v)\times Kv^2}{m + J/r^2} & v \neq 0 \end{cases}$$

$$mg = 0.320517 \quad \frac{1}{m + J/r^2} = 21.861942 \quad K = 0.00015708$$

$$F_{st} = mg\sin(\alpha_{st}) = 0.0111859 \quad a = 0.25 \quad \beta = 0.75 \quad c_x = 0.5$$

Sestavení modelu v SIMULINKu je vhodné provést ve třech krocích. Nejprve je vhodné sestavit model odpovídající základním diferenciálním rovnicím (včetně odporu prostředí). Zde je pouze nelinearita (blok `Sign`) pro zajištění správného smyslu působící síly odporu prostředí. Druhý krok představuje řešení statického tření a ve třetím kroku přidat řešení odrazů

Pro řešení statického tření použijeme tři bloky - `Dead Zone`, `Hit Crossing` a `Switch`. První blok udržuje výstup na nule pro zadaný rozsah vstupního signálu a druhý indikuje průchod vstupního signálu nulou. Blok indikace průchodu nulou vytváří na svém výstupu řídicí signál pro blok přepínače (Switch), který přepíná plnou hodnotu působící síly a hodnotu zmenšenou o sílu statického tření. Jako vstupní signál je použit pilový (trojúhelníkový) signál ve významu úhlu natočení ve stupních s amplitudou ±15° a periodou 10 s začínající z nuly. Tento signál je generován blokem `Repeating Sequence`, ve kterém jsou zadány vektor časů [0 2.5 7.5 10] a vektor hodnot signálu [0 15 -15 10] v těchto okamžicích. SIMULINK automaticky zajistí interpolaci výstupního signálu mezi zadanými časovými okamžiky.



obrázek 16-5 Model Kulička na tyči

Při řešení situace na dorazech je využita možnost externího resetu[108] integrátoru rychlosti (volba *External reset: rising*[109] a *Initial condition source: external*) a indikace omezení integrátoru polohy (volba *Show saturation port*). Dosažení polohy levého (pravého) dorazu je indikováno hodnotou –1 (+1) druhého výstupního signálu integrátoru polohy. Z tohoto signálu je vytvořena absolutní hodnota (blok `Abs`), aby došlo ke změně z hodnoty 0 na hodnotu 1 při každém dorazu, a tento signál ovládá externí reset integrátoru rychlosti. Řídicí událostí je vzestupná hrana tohoto signálu. Při resetu se nastaví nová počáteční hodnota integrátoru rychlosti zadávaná opět externě. Protože se používá hodnota závislá na výstupu integrátoru, je nutné[110] použít blok `Memory`. Tento blok udržuje v paměti hodnotu signálu v minulém integračním kroku. Omezení na výstupu integrátoru polohy kromě možnosti indikace dorazu též zajišťuje aby se nám "kulička zakutálela za dorazy". Výsledný model v SIMULINKu je na obrázku 16-5.

Model uvedený na obr. 16-5 není zcela správný. Pro uvedené hodnoty parametrů sice poskytuje správné výsledky, ale např. při výrazně tlumených odrazech na koncích nebo pomalejší změně natočení tyče by už výsledky neodpovídaly skutečnosti. Problém je v tom, že sice integrátor polohy díky saturaci omezí polohu kuličky, ale integrace rychlosti (její nárůst) pokračuje, ač by měla zůstat rychlost nulová. Výsledkem je, že kulička zůstane na dorazu déle než ve skutečnosti. Korektní řešení úlohy je samozřejmě možné. Jde o řešení logických stavů (např. pomocí bloků `Enabled Subsystem`). Zápis řešení by značně znepřehlednil celý model a proto je řešení ponecháno ve tvaru uvedeném na obr. 16-5.

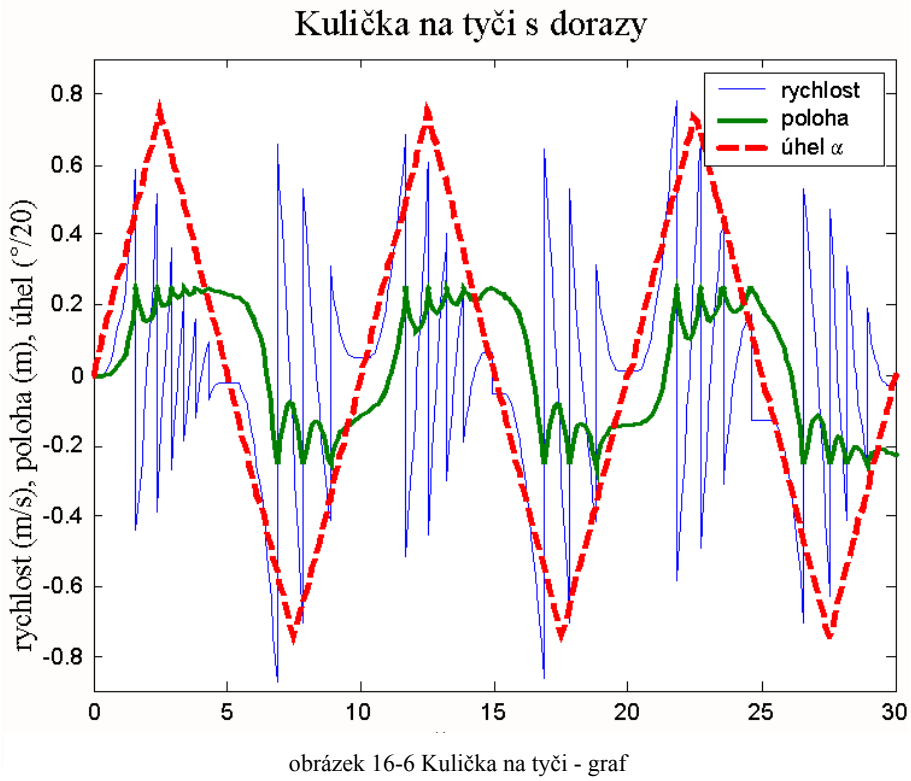
Existence dorazů ovlivňuje dominantním způsobem časový průběh polohy. Zobrazme pro uvedené hodnoty parametrů (pro které dává model korektní výsledky) a start z nulových podmínek časový průběh polohy kuličky během prvních třiceti sekund (tří period) trvání pilového vstupního signálu (úhel naklonění tyč se během 10 s mění od -15° do +15° a zpět). Na obr. 16-6 je vidět, kromě průběhu vstupního signálu (úhlu natočení) a průběhu rychlosti, časový průběh polohy kuličky. Jasně je vidět odskakování na dorazech umístěných v rozmezí ±0.25 m od středu tyče.

---

[108] reset znamená vnucení nové hodnoty do integrátoru. Jde o podobnou záležitost jako při startu výpočtu kdy jde o zahájení výpočtu z počátečních podmínek. O resetu se tedy mluví v případě, že výpočet již nějakou dobu běží. Pak je důležité přesně definovat událost, na základě které se reset provede.

[109] možnosti jsou na vzestupnou hranu (rising), sestupnou hranu (falling), na obě hrany (either) nebo na úroveň (level) řídicího signálu

[110] je možné i řešení s využitím stavového výstupu integrátoru rychlosti (volba *Show state port*). Použité řešení je víceméně proto, aby se ukázalo použití bloku `Memory`.



obrázek 16-6 Kulička na tyči - graf

Vykreslení časového průběhu sledovaných signálů je realizováno v prostředí MATLABu pomocí příkazu **plot** a editace výsledného grafu. Přenos okna osciloskopu (blok `Scope`) do textového editoru např. MS Word je možný pouze kopírováním obrazovky. To je sice možné, ale problémem je černé pozadí obrázku, které při tisku může dělat problémy. Jednodušší je uložit výsledky pomocí bloku `To Workspace` do proměnné[111] a výsledné zpracování provést v MATLABu.

16.4 Průtokový ohřívač

Použité bloky: `In, Integrator, Gain, Mux, Out, Product, Repeating Sequence, Scope, Sign, Slider Gain, Sum, Switch, Terminator, Zero-Order Hold`

Problém: subsystém, součin proměnných, generování šířkově modulovaných pulsů, změna hodnoty parametru během simulace

Tento příklad ukazuje řešení úlohy ověření chování matematického modelu průtokového ohřívače[112]. Máme průtokový ohřívač s elektrickým ohřevem po celé délce (skleněná trubka ovinutá odporovým drátem a z vnější strany izolovaná). Teplota vinutí je ovládána spínáním napájecího napětí šířkově modulovanými pulsy (PWM) tj. příkon je buď 0 nebo E (J s^{-1}). Při sestavení modelu uvažujme 4 teploty – teplotu kapaliny ^{A}T, teplotu skleněné stěny ^{B}T, teplotu vinutí ^{C}T a teplotu izolační vrstvy ^{D}T. Všechny teploty jsou „charakteristické“ tj. v dané vrstvě v radiálním směru stejné. Takovýto systém je popsán 4 parciálními diferenciálními rovnicemi. Rozdělme tento systém na N stejných dílů a předpokládejme že charakteristická

[111] v parametrech bloku je možné nastavit tři typy výstupní proměnné – Structure with Time, Structure a Matrix. První dva typy využívají strukturované uložení informace. Použijeme-li např. první typ (s vloženým časovým údajem) lze pak vykreslit průběh např. příkazem **plot(data.time,data.signals.values)**. Pokud chceme pracovat se standardním typem použijeme typ matice. Pak musíme získat ještě čas, ve kterém byly hodnoty získány. Jednou z možností je využít možnost ukládání času výpočtu do proměnné. Zapnutí této možnosti a případné nastavení názvu proměnné (implicitně tout) je volbou *Simulation->Configuration parameters: Data Import (Save to Workspace:Time)* viz též obrázek 13-4 v kapitole 13.

[112] např. proměnné dopravní zpoždění při změně průtoku, jiné zesílení v různých pracovních bodech apod.

teplota vrstvy je stejná v celém dílu. Takto získáme soustavu 4xN diferenciálních rovnic aproximujících chování průtokového ohřívače.

Jeden díl (i-tý) průtokového ohřívače je popsán následující soustavou rovnic (tučným písmem jsou označeny časově proměnné veličiny)

$$\frac{d^{A}\mathbf{T}_i}{dt} = A_1\mathbf{Q}\left({}^{A}\mathbf{T}_{i-1} - {}^{A}\mathbf{T}_i\right) - \underbrace{\left(2\alpha_A A_2 + \alpha_{AB}A_3\right)}_{a_1}{}^{A}\mathbf{T}_i + \underbrace{\alpha_A A_2}_{a_2}\left({}^{A}\mathbf{T}_{i-1} + {}^{A}\mathbf{T}_{i+1}\right) + \underbrace{\alpha_{AB}A_3}_{a_3}{}^{B}\mathbf{T}_i$$

$$\frac{d^{B}\mathbf{T}_i}{dt} = \underbrace{\frac{\alpha_{AB}B_2}{P_B}}_{b_{11}}{}^{A}\mathbf{T}_i - \underbrace{\frac{\alpha_{BC}B_1 + \alpha_{AB}B_2}{P_B}}_{b_2}{}^{B}\mathbf{T}_i + \underbrace{\frac{\alpha_{BC}B_1}{P_B}}_{b_3}{}^{C}\mathbf{T}_i$$

$$\frac{d^{C}\mathbf{T}_i}{dt} = \frac{\alpha_{BC}C_3}{P_C}{}^{B}\mathbf{T}_i - \frac{\alpha_{CD}C_2 + \alpha_{BC}C_3}{P_C}{}^{C}\mathbf{T}_i + \frac{\alpha_{CD}C_2}{P_C}{}^{D}\mathbf{T}_i + \frac{C_1}{P_C}\mathbf{E}_i$$

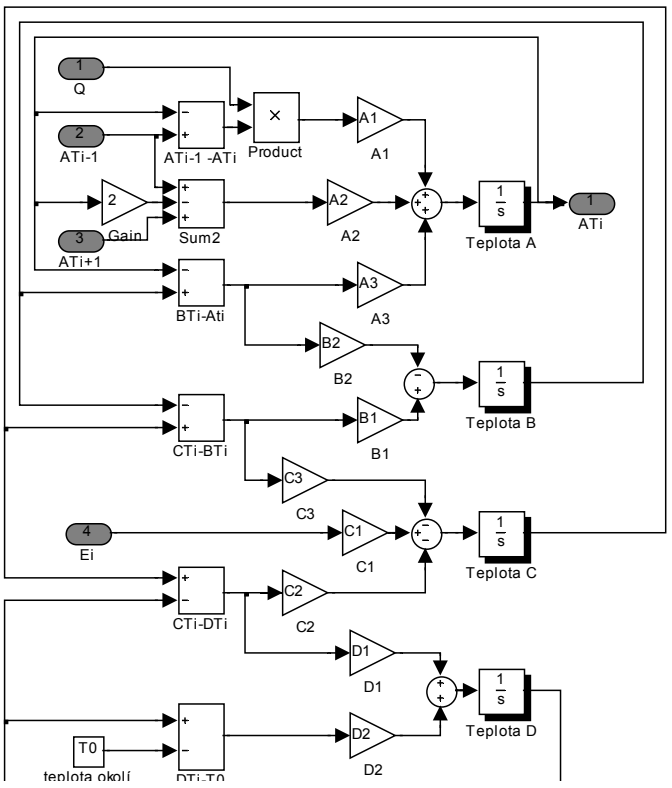
$$\frac{d^{D}\mathbf{T}_i}{dt} = \underbrace{\frac{\alpha_{CD}D_1}{P_D}}_{d_1}{}^{C}\mathbf{T}_i - \underbrace{\frac{\alpha_{CD}D_1 + \alpha_D D_2}{P_D}}_{d_2}{}^{D}\mathbf{T}_i + \underbrace{\frac{\alpha_D D_2}{P_D}}_{d_3}T_0$$

kde

A_n, B_n C_n a D_n	parametry vycházejí z geometrického uspořádání
P_x	parametry zahrnují hustotu, tepelnou kapacitu a objem vrstvy x,
α_A	koeficient přestupu tepla v kapalině v axiálním směru
α_{xy}	koeficienty přestupu tepla mezi vrstvou x a y
T_0	teplota okolí
E_i	příkon i-tého dílu

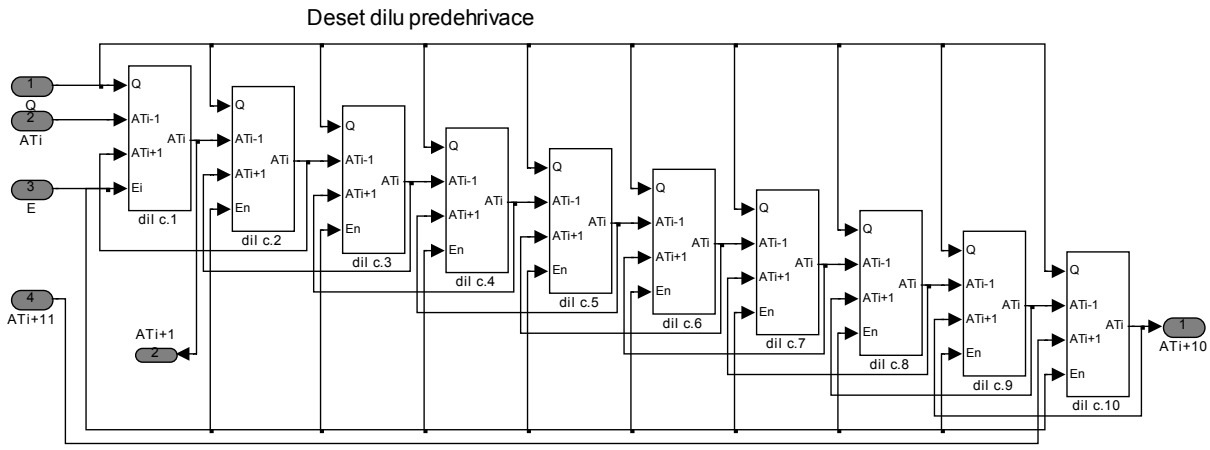
Sestavme odpovídající simulační model. Využijeme možnost použít názvy proměnných MATLABu v parametrech bloků. Simulační model sestavme pro dělení na 50 dílů. Výsledný model (jeho simulační schéma) je již značně rozsáhlý a proto využijeme možnost tvorby subsystémů. Protože jednotlivé základní subsystémy (čtveřice diferenciálních rovnic tvořících popis jednoho dílu) jsou stejné, ušetří nám jejich použití jednak práci s jejich vícenásobnou tvorbou a jednak je výsledné schéma podstatně přehlednější. Model (subsystém) odpovídající těmto rovnicím lze vyjádřit například tak, jak je ukázáno na obr. 16-7.

V modelu jako parametry bloků figurují přímo názvy proměnných vytvořených pro tyto účely připraveným skriptem. Tento skript musí být samozřejmě spuštěn před zahájením simulace. Model je označen jako subsystém tj. vstupy modelu jsou teplota vody předcházejícího dílu ${}^{A}T_{i-1}$, následujícího dílu ${}^{A}T_{i+1}$, průtok Q a příkon připadající na tento díl E_i a výstupem modelu je teplota vody tohoto dílu ${}^{A}T_i$ (jsou vedeny z/do bloků `In/Out`).

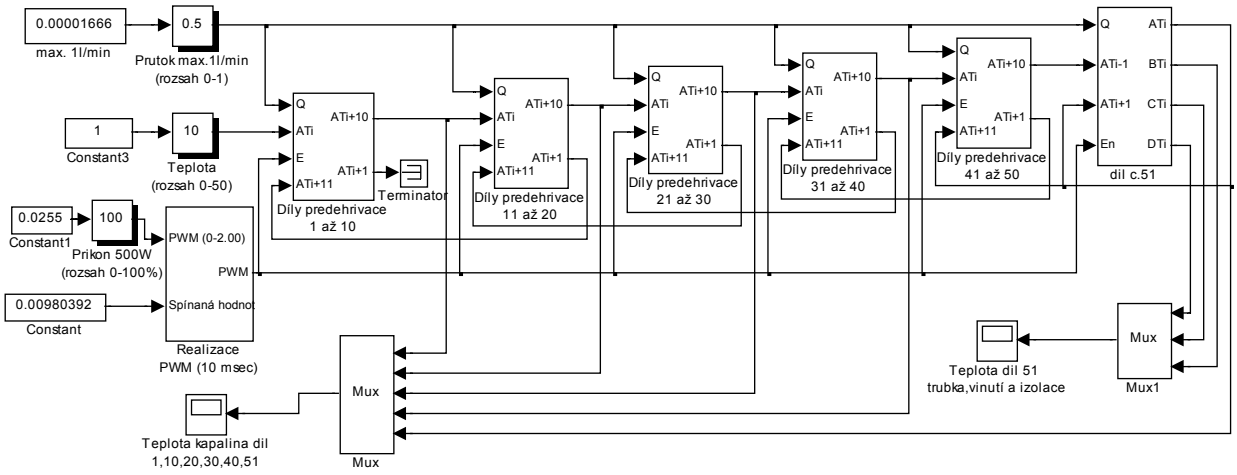


obrázek 16-7 Průtokový ohřívač – i-tý díl

Těchto subsystémů bude potřeba 50. Proto vytvoříme ještě jeden subsystém sdružující 10 těchto dílčích modelů (obr. 16-8). V tomto modelu je dobře vidět propojení vstupů a výstupů jednotlivých základních dílů. Signál průtok vstupuje do všech dílů současně stejně jako signál o aktuálním příkonu (musí mít význam příkonu do jednoho dílu). Hodnota výstupní teploty vstupuje do dílu předcházejícího i následujícího.

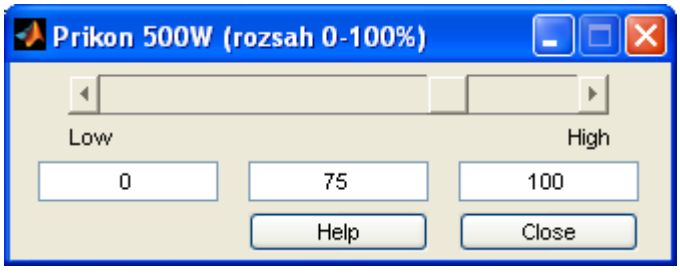


obrázek 16-8 Průtokový ohřívač - deset dílů



obrázek 16-9 Průtokový ohřívač - celkový model

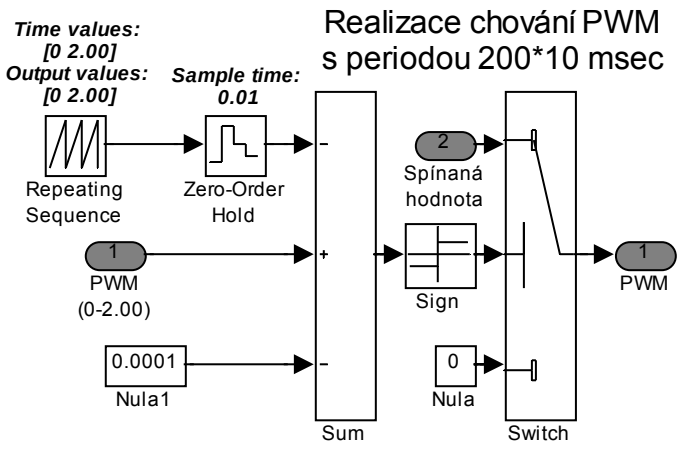
Celkový model pak můžeme znázornit pomocí schématu na obr. 16-9. Zde je přidán ještě jeden modifikovaný díl na konci, aby bylo možné jednoduše sledovat chování teplot všech kapacit jednoho dílu. Do grafů se zobrazují výstupní teploty vody z 10, 20, 30, 40 a 51 dílu a teploty všech kapacit posledního dílu. Protože druhý výstup z subsystému prvních deseti dílů není dále využit je připojen na blok `Terminator`, aby nebylo po spuštění výpočtu modelu generováno varovné hlášení o nezapojeném signálu.



obrázek 16-10 Okno Slider Gain

Pro možnost změny vstupních hodnot během průběhu simulace, která v tomto případě již trvá delší dobu, jsou do modelu zahrnuty bloky `Slider Gain` (označené vrženým stínem) umožňující interaktivní změnu zesílení. Okno, pomocí kterého je možné změny zesílení v zadaném rozsahu provádět, je zobrazeno na obrázku 16-10. Toto okno se objeví po dvojkliku na blok v modelu a je možné během výpočtu měnit hodnotu zesílení buď pomocí myši posuvníkem v rámci mezí nebo přímo číselně v prostředním zadávacím poli.

V modelu průtokového ohřívače je použit ještě blok subsystému vytvářející šířkově modulované pulsy podle hodnoty vstupního signálu. Princip činnosti tohoto subsystému zobrazeného na obrázku 16-11 není tak přímočarý a proto ho nyní přiblížíme. Do subsystému vstupují dva vstupní signály. První (*PWM*) představuje šířku pulsu v hodnotě 0-2.00 v sekundách a druhý (*Hodnota*), který představuje spínanou hodnotu příkonu. Výstupem je periodický obdélníkový signál (*PWM*), který nabývá hodnotu 0 nebo *Hodnota.* Trvání nenulové části signálu na úrovni Hodnota je 0 až 2.00 s (dané hodnotou vstupního signálu *PWM*). Tento puls se s periodou 2.00 s opakuje. Popsaného chování je dosaženo tak, že blok `Repeating Sequence` vytváří pilový signál o periodě 2.00 s a amplitudě 2.00 (nastaveno v parametrech). Výstup tohoto bloku je vzorkován po 10 ms v bloku `Zero-Order Hold`. Odečtení vyvzorkovaného signálu od vstupního signálu a průchod přes blok `Sign` vytváří komparátor, jehož výstupní signál ovládá přepínač dvou úrovní výstupního signálu. SIMULINK se již sám postará o synchronizaci výpočtů tj. výpočet v časech odpovídajících času změny výstupního signálu.



obrázek 16-11 Subsystém šířkově modulovaných pulsů

16.5 Vstup a výstup dat

Použité bloky: `From Workspace, Clock, Rounding Function, Memory, Relational Operator, Hit Crossing, Switch, Zero-Order Hold, Transfer Fcn`

Problém: načtení dat, přenosová funkce, vzorkování (diskretizace), čas simulace, zaokrouhlení, logické porovnání, přepínač

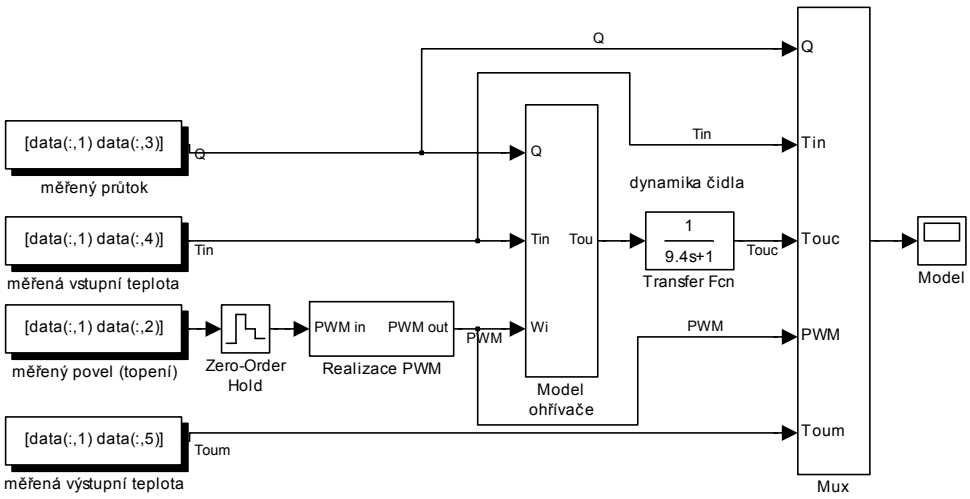
V této kapitole budeme pokračovat v příkladu z kapitoly 16.4. Je zde ukázáno načtení dat do modelu a řešení netriviálního problému vynucení okamžiku výpočtu SIMULINKu na základě načtené hodnoty.

Máme tedy matematický model průtokového ohřívače ve formě modelu SIMULINKu a máme data naměřená na reálném zařízení. Tj. soubor s časovým průběhem (měření po 500 ms) tří vstupních veličin (vstupní teplota, průtok a relativní příkon) a jedné výstupní veličiny (výstupní teploty). Zajímá nás porovnání skutečné výstupní teploty a výstupní teploty vypočtené z modelu. Jako vstupní signály do modelu je potřeba použít naměřené hodnoty (měřená výstupní teplota je použita pouze pro zobrazení). Jedna z možných realizací modelu, který náš požadavek splňuje, je na obrázku 16-12. V modelu je zahrnuta pomocí bloku `Transfer Fcn` známá dynamika teplotního čidla (kapacita I.řádu, jednotkové zesílení a časová konstanta 9.4 s).

Data je možné načítat buď přímo ze souboru na disku (blok `From File`) nebo z matice v pracovním prostoru MATLABu (`From Workspace`) - bloky z knihovny ***Sources***. V případě souboru na disku musí tento obsahovat matici minimálně o dvou řádcích. První řádek obsahuje vzrůstající hodnoty času, druhý - případně další řádky - obsahují hodnoty proměnné v uvedeném čase. Hodnoty proměnných v časových okamžicích výpočtu SIMULINKu jsou automaticky interpolovány. Určitou nevýhodou čtení ze souboru je to, že formát souboru musí být binární formát MATLABu (.mat).

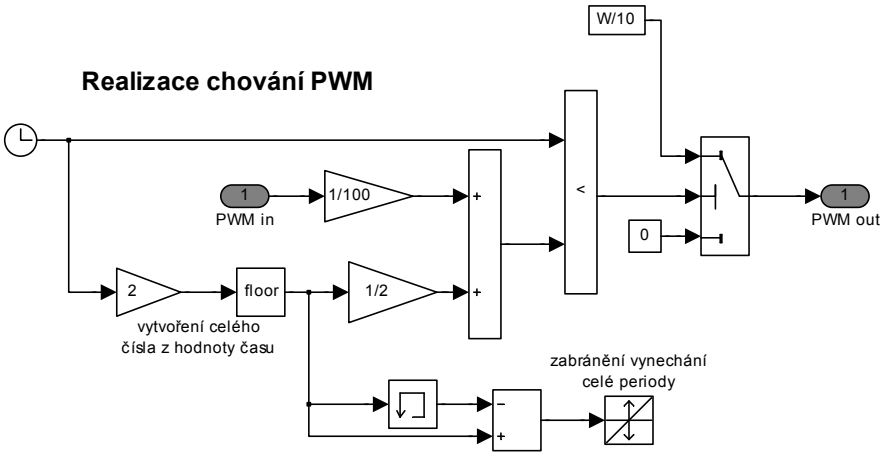
V modelu na obr.16-12 je jako zdroj dat určena matice *data*. Tato matice v pracovním prostoru MATLABu (zpřístupněná blokem `From Workspace` – zvýrazněné bloky) je tvořená pěti sloupci. Matice musí obsahovat v prvním sloupci (jiná organizace oproti předchozímu případu čtení ze souboru) opět hodnoty vzrůstajícího času. V druhém sloupci a dalších jsou příslušná data. Před spuštěním modelu je potřeba naplnit matici *data* změřenými hodnotami tak, aby první sloupec obsahoval čas měření, druhý sloupec informaci o

příkonu, třetí sloupec průtok, čtvrtý sloupec vstupní teplotu a pátý sloupec výstupní teplotu. Všechny sloupce jsou stejně dlouhé a odpovídají času měření v řádku prvním. Čas měření začíná od nuly.



obrázek 16-12 Čtení dat z matice MATLABu

V našem případě činí problémy formát informace o příkonu. Příkon je řízen šířkově modulovaný pulsy (PWM) o periodě 2000 ms. Hodnoty na zařízení jsou měřeny každých 500 ms synchronně s periodou PWM. Informace o aktuálním příkonu pak vypadá tak, že je zaznamenána aktuální hodnota čítače trvání úrovně topení, který je po 10 ms dekrementován. Maximální hodnota čítače je 200 (topení trvale zapnuto). V tomto případě budou měřené hodnoty příkonu tvořeny opakující se čtveřicí čísel 200,150,100 a 50. Pokud bude požadavek na topení na úrovni 95% maximálního výkonu budou se opakovat čtveřice 190,140, 90,40. V případě požadavku topení na 20% budou opakující se čísla 40,0,0,0. Pro jednoduché zjištění průměrné hodnoty příkonu v rámci periody stačí vyčíst hodnotu na začátku každé periody - toto stačí pro vykreslení časového průběhu pro zobrazení výsledků. Pro simulaci je potřeba rekonstruovat skutečný časový průběh příkonu tj. určit okamžik vypnutí topení.



obrázek 16-13 Vynucení okamžiku výpočtu

K tomuto slouží subsystém `Realizace PWM` (obr. 16-11). Sestavení tohoto bloku již není triviální záležitostí. V subsystému jsou použity bloky `Clock` (čas výpočtu) z knihovny ***Sources***, `Rounding Function` (floor zaokrouhlení na celé číslo), `Memory` (paměť hodnoty v minulém okamžiku výpočtu), `Relational Operator` (logické porovnání) a `Hit Crossing` (indikace[113] průchodu signálu zadanou hodnotou). Nejzajímavější částí subsystému je dolní "slepá" větev s posledně jmenovaným blokem `Hit Crossing`. Jejím úkolem je zajistit, aby výpočet modelu proběhl v okamžiku vypnutí či zapnutí topení.

[113] funkce bloku není tak jednoduchá, jak by se na první pohled zdálo. Výpočet modelu je vždy diskrétní tj. po určitém časovém kroku (který se během výpočtu obvykle mění). Proto pravděpodobnost, že hodnota sledovaného signálu v okamžiku výpočtu je právě zadaná hodnota, je minimální. V jednom kroku výpočtu bude řekněme pod a v následujícím nad zadanou hodnotou. V tomto případě použití tohoto bloku zajistí mimořádnou změnu časového kroku výpočtu a opakování posledního výpočtu v čase, ve kterém bude mít sledovaný signál zadanou hodnotu.

16.6 Titrace slabé kyseliny slabou zásadou (SIMULINK)

Použité bloky: `Algebraic Constraint, Dot Product, Gain, Math Function, Product, Ramp, Sum, XY Graph, Zero-Order Hold`

Problém: algebraická smyčka, maskovaný subsystém, zobrazení závislosti dvou signálů

Protože tato skripta jsou vytvářena na fakultě chemickotechnologické, nezapomeňme na příklad z oblasti chemie. Jde o stejný příklad jako byl řešen v kapitole 10.5 (Titrace slabé kyseliny slabou zásadou). Na tomto příkladě ukážeme použití řešení algebraické smyčky v SIMULINKu, použití maskovaného subsystému s inicializačními výpočty a kreslení grafu průběhu dvou signálů vůči sobě.

Využijeme tedy zadání i rovnic příkladu kapitoly 10.5 a zde pouze shrneme zadání a uvedeme odvozené vztahy jako základ pro sestavené modelu v SIMULINKu. Následující rovnice popisují časový průběh vývoje koncentrace vodíkových iontů v dokonale míchané nádobě naplněné objemem V_{A0} = 0.05 litru kyseliny A (kyselina octová K_A = 1.75 10^{-5}) o koncentraci c_{A0} = 0.05 mol/litr. Do nádoby budeme přidávat zásadu B (etylamin K_B = 4.37 10^{-4}) o koncentraci c_{B0}=0.08 mol/litr. Látka B bude přidávána po kapkách (objem kapky je 0.3 ml) s frekvencí 1 kapka za sekundu. Zajímá nás průběh pH v nádobě v závislosti na celkovém přidaném objemu V_B látky B. Titrace probíhá ve vodném prostředí to znamená, že rozpouštědlem obou látek je voda ($K_V = 10^{-14}$).

aktuální koncentrace obou látek v nádobě:

$$c_A = \frac{c_{A0}V_{A0}}{V_{A0}+V_B} \qquad c_B = \frac{c_{B0}V_B}{V_{A0}+V_B}$$

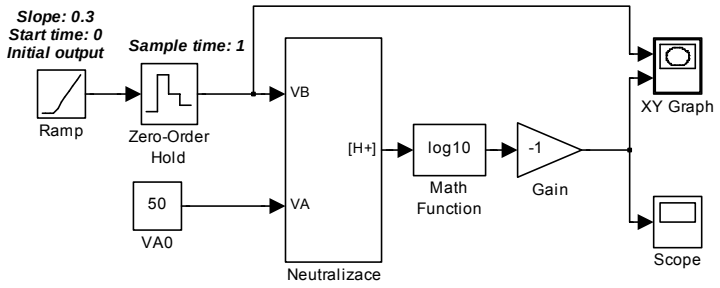
koncentrace vodíkových iontů:

$$[H^+]^4 + \left\{K_A + \frac{K_v}{K_B} + c_B\right\}[H^+]^3 + \left\{K_A(c_B - c_A) + K_v\left(\frac{K_A}{K_B} - 1\right)\right\}[H^+]^2 - \left\{K_A K_v\left(\frac{c_A}{K_B} - 1\right) + \frac{K_v^2}{K_B}\right\}[H^+] - K_A\frac{K_v^2}{K_B} = 0$$

výpočet pH:

$$\mathrm{pH} = -\log [H^+]$$

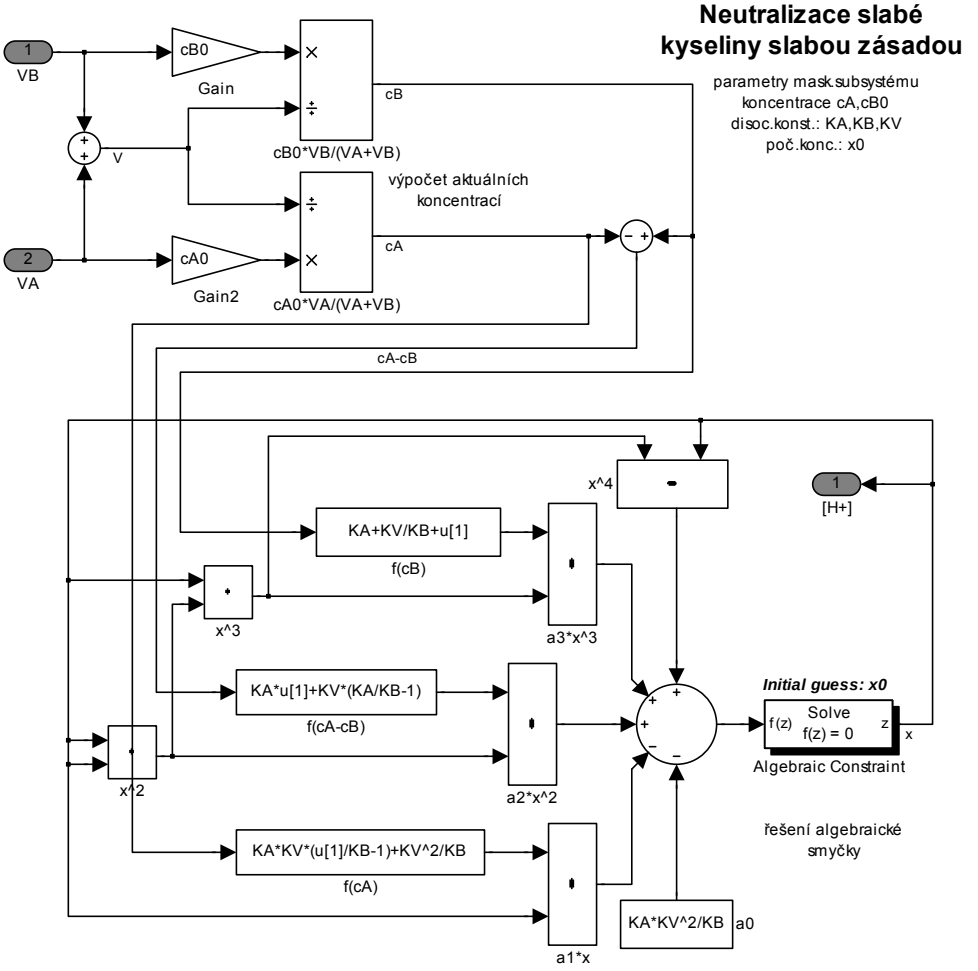
Oddělme při řešení výpočet koncentrace a rovnováhy[114] (subsystém Neutralizace) od časového průběhu dávkování látky B. Na obrázku 16-14 je celkový model znázorňující řešení dávkování. Hodnoty objemů jsou zadávány v mililitrech. Simulace přidávání po kapkách je řešena pomocí bloků `Ramp` a `Zero-Order Hold`. První blok vytváří signál jehož hodnota v čase definovaně roste (0.3 za 1 sec) a druhý blok hodnotu tohoto signálu vždy na 1 sekundu "podrží". Hodnota koncentrace vodíkových iontů se přepočte na pH a v bloku `XY Graph` se vykreslí průběh pH vůči "nakapanému" objemu látky B. Zároveň je tentýž signál vynášen v bloku `Scope` vůči času. Blok `XY Graph` se chová jinak než blok Scope. Mimo jiné nemá automatické měřítko tj. je potřeba v jeho dialogovém okně (objeví se po dvojkliku na blok) měřítka osy x i y nastavit ručně a vlastní okno s grafem se objeví po spuštění výpočtu automaticky.



obrázek 16-14 Titrace - celkový model

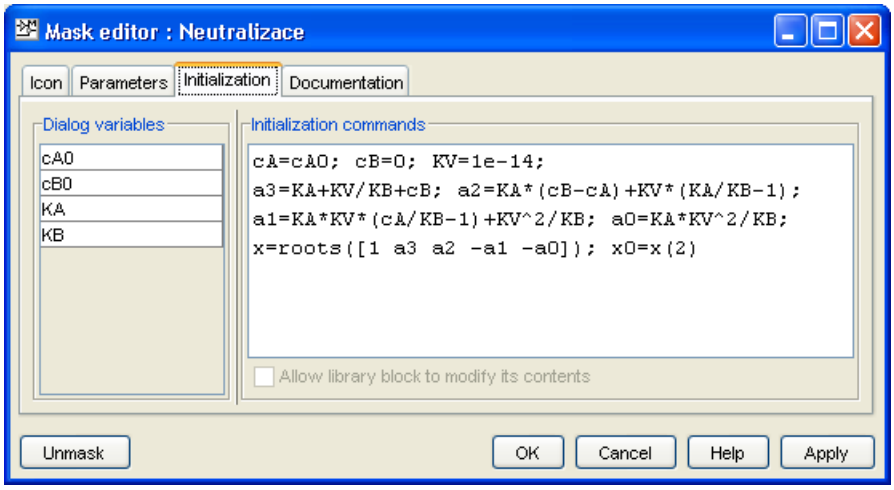
[114] v tomto případě "pouze" řešení algebraických rovnic bez časových závislostí

Výpočet koncentrace vodíkových iontů provedeme formou maskovaného subsystému např. v podobě uvedené na obr. 16-15. Při realizaci výše uvedených rovnic v SIMULINKu tvoří problém pouze bikvadratická rovnice popisující závislost koncentrace vodíkových iontů na aktuálních koncentracích obou látek a vlastnostech obou látek a rozpouštědla vyjádřených jejich disociačními konstantami. Z hlediska simulace jde o algebraickou smyčku. SIMULINK umožňuje řešit tento problém blokem `Algebraic Constraint`. Vstupem tohoto bloku je funkční hodnota závislá na výstupu bloku. V průběhu simulace je v každém kroku simulačního výpočtu vyhledána taková hodnota výstupu bloku, aby vstupní funkční hodnota byla nulová.



obrázek 16-15 Titrace - subsystém Neutralizace

Velký význam v těchto případech hraje volba počáteční hodnoty výstupu bloku při začátku simulace. Konkrétně v našem případě, kdy bikvadratická rovnice může mít čtyři reálná řešení, špatná volba počáteční hodnoty způsobí, že SIMULINK nedokáže model řešit. Na druhou stranu vhodné volby počátečních hodnot zajistí automatický výběr správného řešení v případě, že algebraická smyčka má více řešení. Jak však zajistit správnou počáteční hodnotu pro různé parametry. Tento problém můžeme vyřešit[115] použitím maskovaného subsystému. Subsystém je tu použit ne kvůli opakovanému využití, ale právě z důvodu využití

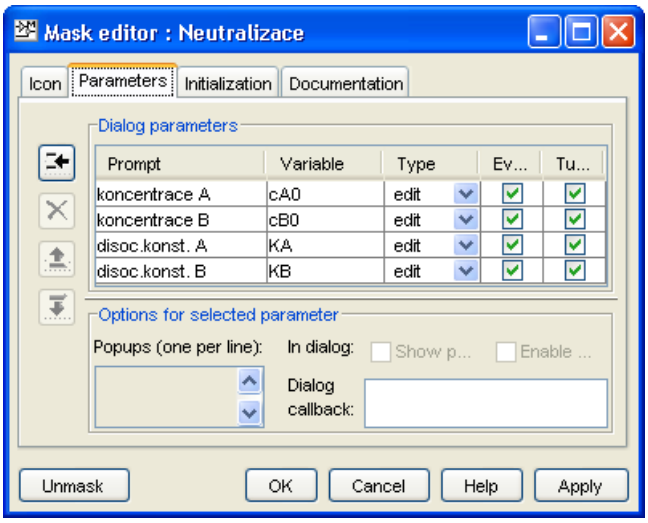


obrázek 16-16 Editor masky subsystému – záložka inicializace

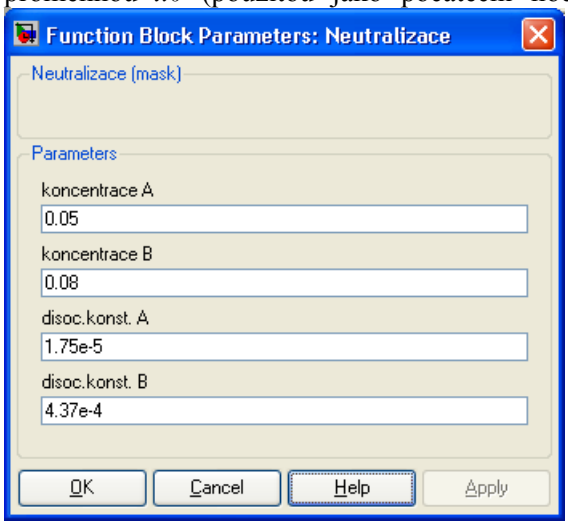
[115] jiné řešení by mohlo být pomocí parametrů zadávaných přes proměnné v pracovním prostoru MATLABu plněných skriptem, ve kterém by proběhl výpočet počáteční koncentrace. Když se zamyslíme nad funkcí maskovaného subsystému vidíme, že je to v podstatě stejný mechanismus zpřístupněný ale přímo z prostředí SIMULINKu.

možností masky provést v rámci inicializace pomocné výpočty

Na obrázku 16-16 je ukázáno dialogové okno editoru masky – záložka *Initialization*. V části *Initialization commands* jsou zapsány (standardní) příkazy MATLABu, které se provedou při startu výpočtu modelu, ve kterém je příslušný maskovaný blok použit. Příkazy používají hodnoty zadané v dialogovém okně bloku (obrázek 16-18) případně spočítané pomocné proměnné. Poslední příkaz plní proměnnou *x0* (použitou jako počáteční hodnota



obrázek 16-17 Editor masky subsystému – záložka parametry



obrázek 16-18 Maska subsystému

bloku `Algebraic Constraint`) správným[116] řešením ze čtyř možných. Názvy všech proměnných (jak definovaných v dialogu, tak vytvořených při inicializaci) mohou použity v parametrech bloků subsystému. Pozor – v blocích maskovaného subsystému nelze používat názvy proměnných definovaných v pracovním prostoru MATLABu, používají se pouze proměnné definované v masce.

Jak již bylo řečeno při popisu řešení příkladu v MATLABu jde o numericky náročný výpočet. To se projeví při řešení v SIMULINKu. Zatímco křivka závislosti pH na přídavku látky B je při řešení v MATLABu hladká (obrázek 10-12), křivka odpovídající řešení v SIMULINKu je v části řešení pro vyšší hodnoty pH "roztřepaná" – stav signalizující problémy při numerickém řešení.

16.7 Průběžná identifikace (diskrétní maticové operace v SIMULINKu)

Použité bloky: `Assigment, Displey, Mux, Product, Pulse genarator, Reshape, Selector, Sum, To Workspace, Transfer Fcn, Unit Delay, Zero-Order Hold`

Problém: maticové operace, kombinace spojitého a diskrétního systému, vzorkování, zobrazení číselné hodnoty, maskovaný subsystém, rozšířené možnosti zobrazení informací o modelu

Spojité maticové operace rozložené na jednotkové operace šlo samozřejmě realizovat vždy. Od SIMULINKu verze 4.1 je možné použití signálů typu matice v standardních blocích. V předchozích verzích bylo možné používat pouze signály označené jako 1-D tj. signál sdružující více jednoduchých signálů, ale bez možnosti s nimi provádět matematické operace. Ukažme nové možnosti na příkladě identifikace parametrů lineární diferenční rovnice pomocí rekurentní metody nejmenších čtverců s exponenciálním zapomínáním.

Nejprve velmi stručně o co jde. Mějme nějakou lineární jednorozměrnou soustavu - jeden vstup *x(t)* a jeden výstup *y(t)* - jejíž dynamické chování (souvislost mezi časový průběhem vstupu a výstupu) je popsáno diferenciální rovnicí

[116] které řešení je správné je nutné určit zkušebním výpočtem (nejlépe v MATLABu)

$$y^{(n)} + \overline{a}_{n-1} y^{(n-1)} + \ldots + \overline{a}_1 y' + \overline{a}_0 y = \overline{b}_m x^{(m)} + \overline{b}_{m-1} x^{(m-1)} + \ldots + \overline{b}_1 x' + \overline{b}_0 x \quad n \geq m$$

kde $x = f_x(t)$ *vstupní časově proměnný signál*

$y = f_y(t)$ *výstupní časově proměnný signál*

Pokud budeme sledovat vstupní a výstupní signál pouze ve ekvidistantních časových okamžicích tj. signál vyvzorkujeme s intervalem vzorkování T dostaneme místo spojitých časových průběhů pouze vektory nespojitých (diskrétních) hodnot příslušných signálů. Zavedeme-li předpoklad, že vstupní signál se mění pouze v intervalech vzorkování (tj. v čase mezi intervaly vzorkování je konstantní) lze souvislost mezi vstupním a výstupním signálem sledovaným pouze v intervalech vzorkování zapsat pomocí diferenční rovnice (v praktičtějších zpětných posunech) jako

$$y(k) + a_1 y(k-1) + a_2 y(k-2) + \ldots + a_n y(k-n) = b_0 u(k) + b_1 u(k-1) + \ldots + b_n u(k-n)$$

kde $y(k) = y(t = kT)$ *popřípadě* $u(k-n) = u\big(t = (k-n)T\big)$

nebo to samé ve formě vektorového zápisu jako

$$\mathbf{A} \times \mathbf{y} = \mathbf{B} \times \mathbf{u}$$

kde $\mathbf{A} = [1, a_1, \cdots, a_n]$ *řádkový vektor parametrů a*

$\mathbf{B} = [0, b_1, \cdots, b_n]$ *řádkový vektor parametrů b*

$\mathbf{y}(k) = [y(k), y(k-1), \cdots, y(k-n)]^T$ *sloupcový vektor výstupních dat*

$\mathbf{u}(k) = [u(k), u(k-1), \cdots, u(k-n)]^T$ *sloupcový vektor vstupních dat*

Popis dynamického chování soustavy ve formě diferenční rovnice je obvykle základním požadavkem pro návrh řízení této soustavy. Jak ale tento popis - hodnoty parametrů *a* a *b* - získat? Jeden způsob předpokládá, že máme k dispozici dostatečně dlouhý časový průběh (dostatečný počet vzorků) vstupního a výstupního signálu, který zpracováváme jednorázově jako celek – tzv. of-line metody. Druhý způsob předpokládá, že máme odhad parametrů, který na základě aktuální hodnoty (vzorku) vstupního a výstupního signálu opravíme – tzv. on-line metody. Touto druhou metodou identifikace parametrů lineární diferenční rovnice se budeme zabývat.

Zápis algoritmu aktualizace odhadu parametrů diferenční rovnice na základě změřené hodnoty vstupu *u(k)* a výstupu soustavy *y(k)* v čase t_k=kT ve vektorové podobě je relativně jednoduchý. Též jeho implementace v MATLABu není složitá. Avšak v SIMULINKu realizovat tento algoritmus bylo dosud možné pouze na úrovni S-funkce. Od verze 4.1 je možné úlohy tohoto typu řešit na úrovni základních bloků. Než uvedeme vlastní algoritmus zapíšeme diferenční rovnici ještě ve formě[117] vektorového zápisu

$$y(k) = \mathbf{x}(k) \times \boldsymbol{\Theta}(k)$$

kde $\mathbf{x}(k) = [-y(k-1), \cdots, -y(k-n), u(k-1), \cdots, u(k-n)]$ *řádkový vektor dat*

$\boldsymbol{\Theta}(k) = [a_1, \cdots, a_n, b_1, \cdots, b_n]^T$ *sloupcový vektor parametrů*

Vlastní algoritmus průběžné identifikace parametrů lineární diferenční rovnice rekurentní metodou nejmenších čtverců s exponenciálním zapomínáním[118] lze zapsat ve čtyřech krocích, které se provádějí v každém vzorkovacím okamžiku

---

[117] při tomto zápisu se předpokládá, že hodnota y(k) nezávisí na hodnotě u(k) tj. koeficient b_0=0. Tento předpoklad je prakticky pro všechny reálné dynamické soustavy splněn.

[118] smyslem exponenciálního zapomínání – závislého na parametru $\lambda \in \langle 0,1 \rangle$ - je volitelně "zapomínat" historii a tím dovolit identifikaci, aby sledovala měnící se parametry soustavy

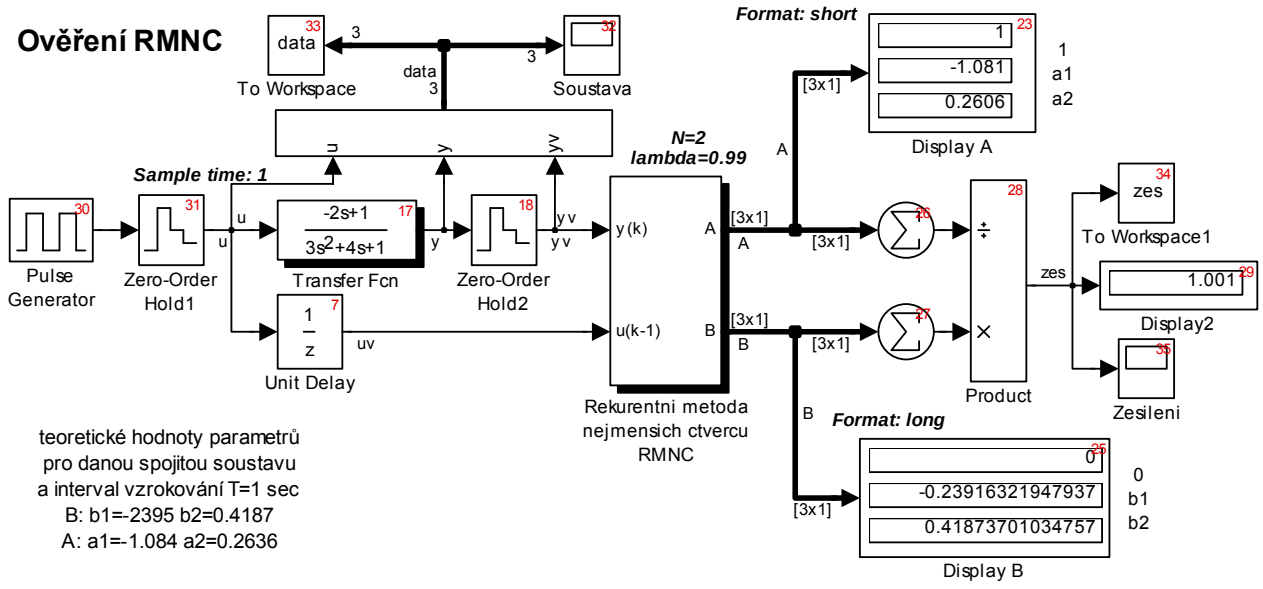
1. aktualizace datového vektoru **x**(k) daty y(k-1), u(k-1)
2. $\varepsilon(k) = y(k) - \mathbf{x}(k)\hat{\theta}(k-1)$ chyba predikce
3. $\begin{cases} \mathbf{m}(k) = \dfrac{\mathbf{P}(k-1)\mathbf{x}^T(k)}{\lambda + \mathbf{x}(k)\mathbf{P}(k-1)\mathbf{x}^T(k)} \\ \mathbf{P}(k) = \lambda^{-1}\left[\mathbf{P}(k-1) - \mathbf{m}(k)\mathbf{x}(k)\mathbf{P}(k-1)\right] \end{cases}$ aktualizace kovarianční matice P
4. $\hat{\theta}(k) = \hat{\theta}(k-1) + \mathbf{m}(k)\varepsilon(k)$ oprava odhadu parametrů

Základním problémem implementace uvedeného algoritmu v SIMULINKu jsou aktualizace datového vektoru ***x*** (rozměr 1×2n), sloupcového vektoru parametrů $\boldsymbol{\Theta}$ (rozměr 2n×1) a čtvercové kovarianční matice ***P*** (rozměr 2n×2n). Na obrázku 16-20 je (snad) dostatečně komentovaný maskovaný subsystém realizující uvedený algoritmus v obecné podobě. Pod pojmem obecné je zde myšleno, že v dialogu masky subsystému lze volit řád (tj. počet určovaných parametrů) diferenční rovnice. Uvedený subsystém je zobrazen pro volbu N=2 tj. čtyři identifikované parametry. Jsou tak využity možnosti zobrazení *Format->Port/Signal Displays->Wide nonscalar lines* (zvýraznit signály modelu, které vektorové), *Format->Signal dimensions* (přidat k vektorovým signálům modelu čísla označující rozměry vektoru) a *Block Displays->Execution order* (k blokům modelu přidat pořadové číslo vykonavání posloupnosti bloků). Všechny tyto možnosti usnadňují jednak čitelnost modelu a jednak hledání chyb. Jednotlivé subsystémy (označené výraznými čísly) představují přímo odpovídající řádky algoritmu.

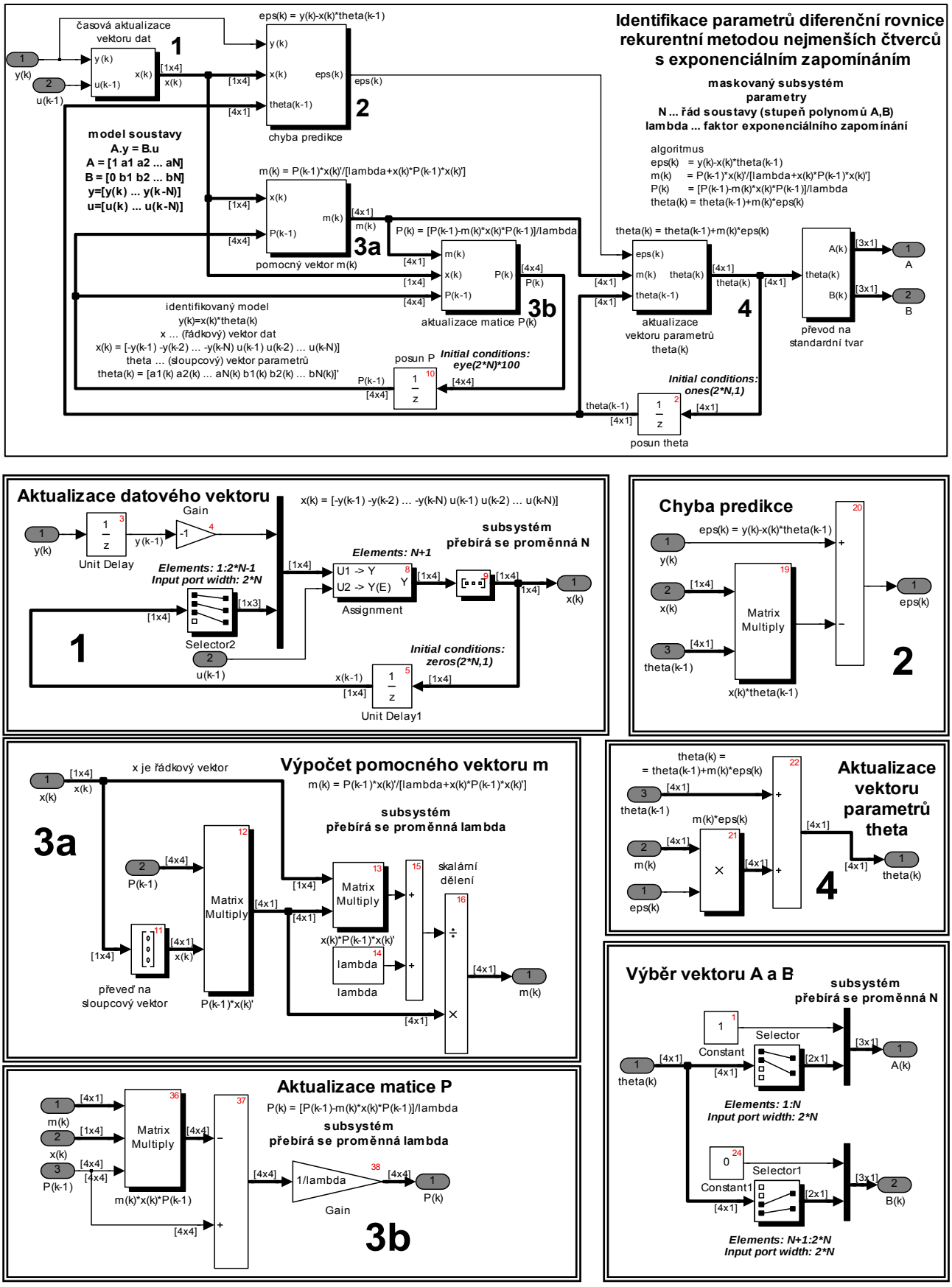
Bloky s vnitřním označením Matrix Multiply jsou standardní bloky `Product`, s nastaveným parametrem *Multiplication: Matrix*. Blok `Selector` dovoluje vybrat část vektoru a ve spolupráci s blokem `Mux` a `Assignment` (zápis do vybrané části vektoru) realizuje v subsystému **1** posun dat ve vektoru (časová aktualizace dat).

Uvedený model odpovídá možnostem nabízeným ve verzi SIMULINKu 4.1 (MATLAB 6.1). V současné verzi SIMULINKu 6.2 (MATLAB 7.0.4) jsou již integrovány další bloky, které dovolují jednodušší řešení tohoto příkladu.

Na obrázku 16-19 je simulační model pro odzkoušení subsystému rekurentní metody průběžné identifikace parametrů zvolené diferenční rovnice. Číselné hodnoty odpovídají stavu po ukončení simulace. Je vidět použití bloku `Display` pro signál typu vektor i kombinaci spojité části realizované blokem `Transfer Fcn` a diskrétních výpočtů.



obrázek 16-19 Průběžná identifikace - celkový model

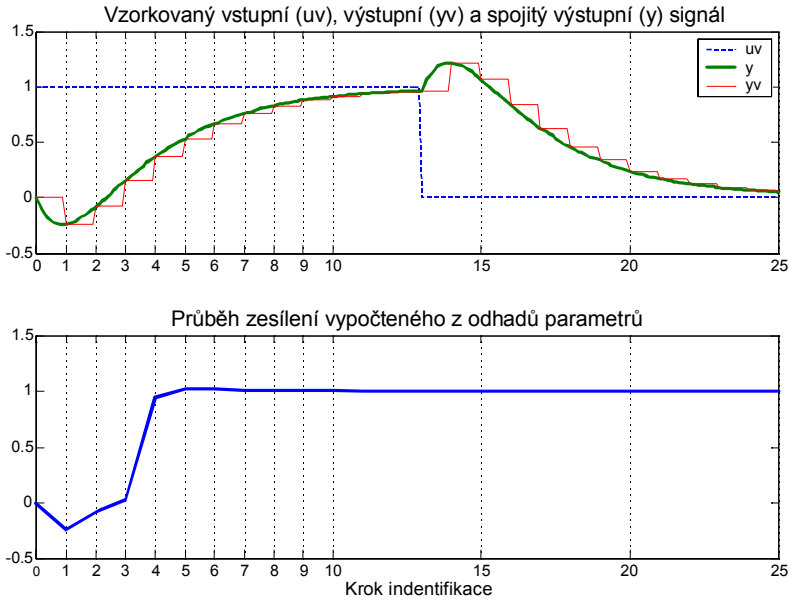


obrázek 16-20 Průběžná identifikace - maskované subsystémy

Použití subsystému Průběžné identifikace je ukázáno v celkovém modelu na obrázku 16-19. Použitá spojitá soustava je II. řádu a je definována pomocí bloku `Transfer Fcn`. Jako vstupní signál vybuzující soustavu je použit obdélníkový signál (blok `Pulse generator`), který je před vstupem do soustavy vyvzorkován (blok `Zero-Order Hold`) s periodou T=1 sec. Signál na výstupu soustavy je vzorkován stejným způsobem. Oba vzorkované signály a spojitý výstup soustavy jsou zobrazovány a ukládány do proměnné *data*. Vzorkované signály jsou vedeny do subsystému průběžné identifikace – vstupní signál je ještě o jeden krok vzorkování zpožděn v bloku `Unit Delay`. Průběžně aktualizované odhady parametrů jsou v číselné podobě zobrazovány v blocích `Display`. Jeden blok má v parametrech nastaven formát zobrazení short a druhý formát long. Na základě těchto signálů je dále počítáno zesílení identifikované diferenční rovnice (má být 1), které je zobrazováno i ukládáno do proměnné *zes*.

Stav (hodnoty odhadu parametrů v blocích `Display`) na obrázku 16-20 je po ukončení výpočtu v trvání 25 sekund tj. 25 kroků identifikace. Vypočítané odhady parametrů se blíží teoretickým hodnotám, které lze získat např. následujícími příkazy MATLABu (funkce z Control System Tbx)

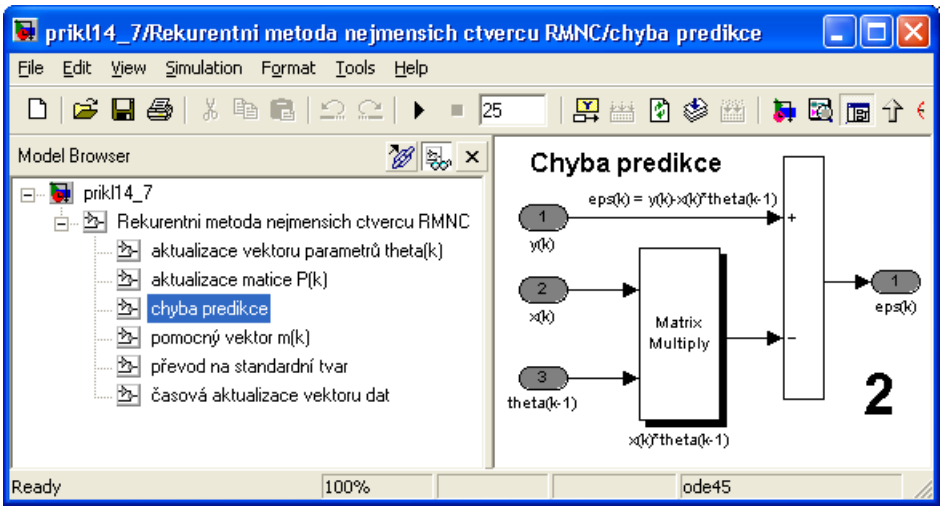
```
>> B=[-2 1]; A=[3 4 1];
>> s=tf(B,A)

        Transfer function:
          -2 s + 1
        ---------------
        3 s^2 + 4 s + 1

>> sd=c2d(s,1,'zoh')

       Transfer function:
         -0.2395 z + 0.4187
       ----------------------
       z^2 - 1.084 z + 0.2636
        Sampling time: 1
```



obrázek 16-21 Průběh signálů průběžné identifikace

Pro zajímavost ukažme ještě průběhy sledovaných signálů – obrázek 16-21. Kromě toho jak vypadá vyvzorkovaný signál je na obrázku také vidět vývoj vypočítávaného zesílení z odhadů parametrů.

Při složitějších modelech už může být problematické udržet přehled o noření jednotlivých subsystémů a hledání kde se který nachází. V této situaci může pomoci *Model browser*, rozšířené okno modelu o část zobrazující strom subsystémů. Tuto volbu zapneme pomocí *View->Model browser option->Model browser* a *View->Model browser option->Show masked subsystems*. Pro model průběžné identifikace je okno "prohlížeče modelu" ukázáno na obrázku 16-22. Kromě stromu subsystémů může ukazovat ještě vazbu knihovních bloků – volba *View->Model browser option ->Show library links*. Uživatelská knihovna subsystémů (bloků) není v tomto příkladě použita takže není nic vidět.



obrázek 16-22 Okno modelu a Model browser

16.8 Dostřel děla (SIMULINK)

Použité bloky: `Displey, Gain, Hit Crossing, Integrator, Product, Sign, Stop Simulation, Sum, To Worspace, XY Graph`

Použité funkce: `fminbnd, sim`

Problém: spolupráce MATLAB a SIMULINK, zastavení výpočtu modelu, zobrazení závislosti dvou signálů, sdílení proměnných v různých pracovních prostorech

Kapitolu 16 uzavřeme příkladem, který je jiným způsobem řešení prvního příkladu kapitoly 10. Na tomto příkladu si ukážeme spolupráci mezi MATLABem a SIMULINKem. V této chvíli by měl mít čtenář[119] představu na jaký typ úloh je vhodný MATLAB a na jaký SIMULINK. V některých případech se však část řešení dělá dobře v MATLABu a další část v SIMULINKu. Mezi tyto úlohy patří i úloha řešení maximálního dostřelu děla. Vlastní popis pohybu dělové koule (dynamika) je typická úloha SIMULINKu. Na druhou stranu vyhledání náměru pro maximální dostřel je typická optimalizační úloha, kterou řešit v MATLABu je už pro nás hračka. Ukažme tedy jak využít obou nástrojů tak, aby každý dělal to na co je určen.

Zopakujme stručně o co jde. Z děla umístěného na pahorku ve výšce h_0 nad okolním terénem je pod úhlem α vystřelena železná koule o poloměru r s úsťovou rychlostí v_0. Určete takový náměr (úhel), při kterém je za daných podmínek maximální dostřel. Při výpočtu uvažujte, že odpor prostředí je úměrný (konstanta c_x) ploše průřezu střely S a kvadrátu rychlosti v. Výsledné rovnice popisující dráhu střely (viz příklad v kapitole 10.1) jsou

$$\frac{dh_y}{dt} = v_y \qquad h_y(t=0) = h_0$$

$$\frac{dv_y}{dt} = -c_x \frac{3}{4\rho r} sign(v_y) v_y^2 - g \qquad v_y(t=0) = v_0 \sin(\alpha)$$

$$\frac{dh_x}{dt} = v_x \qquad h_x(t=0) = 0$$

$$\frac{dv_x}{dt} = -c_x \frac{3}{4\rho r} sign(v_x) v_x^2 \qquad v_y(t=0) = v_0 \cos(\alpha)$$

kde
- h_y výška střely nad povrchem
- v_y rychlost střely ve směru osy y
- h_x vzdálenost střely od ústí hlavně v ose x
- v_x rychlost střely ve směru osy x

Odpovídající model v SIMULINKu využívající proměnné *k, x_0, y_0, v_0* a *alfa* z pracovního prostoru MATLABu je na obrázku 16-23. Výpočet modelu je ukončen (blok `Stop Simulation`) v okamžiku průchodu signálu *hy* (výška střely) nulou tj. v okamžiku dopadu. Vynucení okamžiku výpočtu v hodnotě nula a indikaci tohoto stavu zajišťuje blok `Hit Crossing`. Model je doplněn zobrazováním výšky střely v závislosti na vzdálenosti (blok `XY Graph`), číselným zobrazováním výšky *hy* a vzdálenosti *hx* a ukládáním poslední (jedné) hodnoty vzdálenosti *hx* do proměnné *delka*.

Takto připravený model můžeme využít v příkazech MATLABu. Připravme si skript s definicí parametrů modelu a funkci využívající připravený SIMULINKový model k určení dostřelu pro náměr zadaný v parametrech funkce.

Vlastní řešení pak získáme příkazem

```
>> aopt=fminbnd('p16_8delo',30,60)
aopt =   31.2704
```

[119] pokud se nějaký, který to dočetl až sem, vůbec najde. Ti, kteří tuhle stránku otevřeli náhodou, se nepočítají.

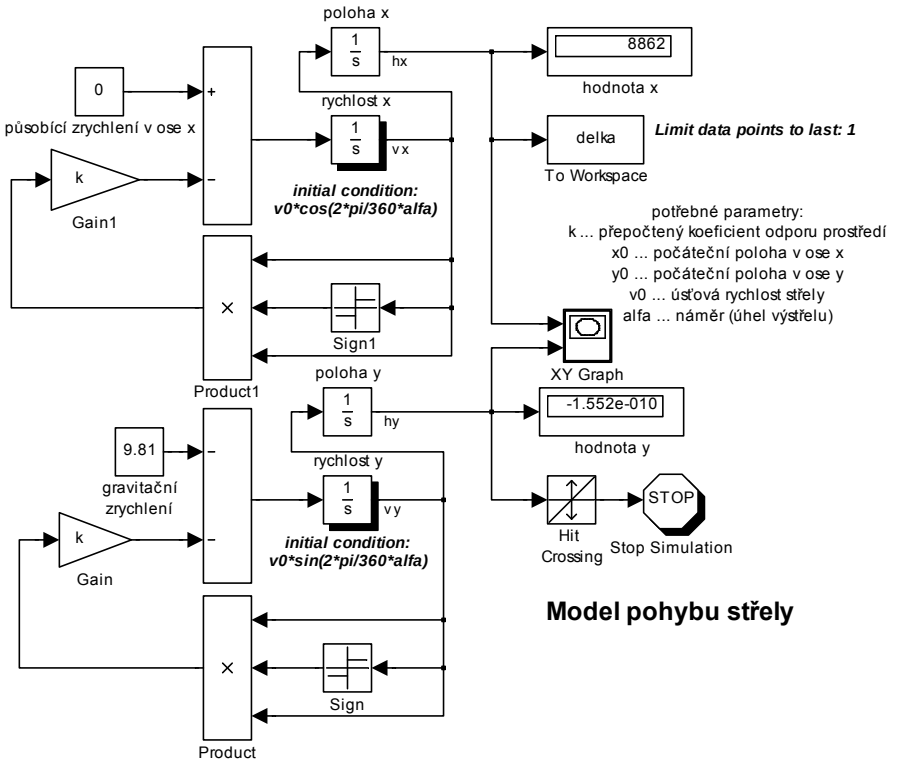
prikl16_8delo.m

```
function d=prikl16_8delo(a)
% dostřel d na základě úhlu
% voláním modelu SIMULINKu

global alfa
alfa=a;
sim('prikl16_8');
d=-delka;
```

prikl16_8d.m

```
% skript s parametry - Model pohybu střely

y0=10;       % výška ústí hlavně
x0=0;        % poloha ústí hlavně v ose x
v0=1200;     % úsťová rychlost střely
r=0.075;     % poloměr střely
ro=7800;     % hustota materiálu střely
cx=0.26;     % součinitel aerodyn. odporu

global alfa
alfa=45;     % úhel osy hlavně

k=3*cx/4/ro/r;    % přepočítaný odpor
```

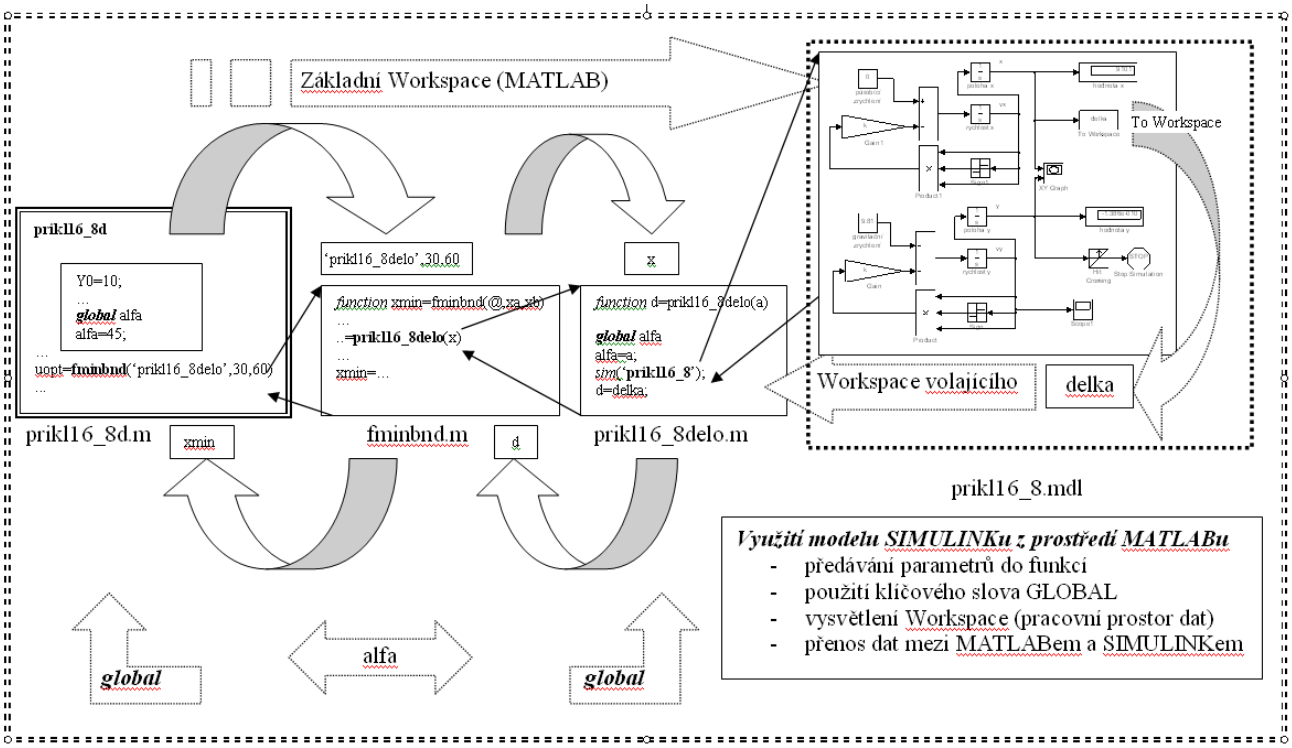
Dospěli jsme k přibližně stejnému výsledku jako v příkladě 10.1 (α=31.1597°), ale srovnáme-li dobu výpočtu i komplikovanost řešení vidíme, že toto řešení je rychlejší i jednodušší. Navíc v případě složitějšího dynamického modelu je sestavení v SIMULINKu jednodušší a přehlednější. Na druhou stranu je sestavení celkového řešení poněkud náročnější. Vyžaduje mít přehled o předávání parametrů mezi jednotlivými částmi řešení. Model v SIMULINKu přebírá parametry z pracovního prostoru MATLABu (*Basic Workspace*) ale blok `To Workspace` vrací proměnné do pracovního prostoru funkce, která model SIMULINKu zavolala. Spojení mezi proměnnými v různých pracovních prostorech zajišťuje označení proměnné jako globální. Od MATLABu verze 5 klíčové slovo **`global`** zajišťuje spojení pouze mezi proměnnými v pracovních prostorech funkcí, ve kterých je klíčové slovo `global` použit. Pro náš příklad je na obrázku 16-24 pokus o znázornění jak proměnné v jednotlivých pracovních prostorech souvisí.



obrázek 16-23 Dostřel děla - celkový model

Na úrovni příkazového řádku (dvojitý rámeček vpravo) je zavolán skript prikl16_8d.m, ve kterém jsou definovány potřebné proměnné pro model SIMULINKu (prikl16_8.mdl). Tyto proměnné se nacházejí v základním prostoru MATLABu (*Basic Workspace*). Proměnná *alfa* je navíc označena jako globální. Dále je z příkazového řádku zavolána standardní funkce MATLABu **`fminbnd`** s parametry 'prikl16_8delo' (název uživatelské funkce), 30 (dolní mez) a 60 (horní mez hledané hodnoty náměru). Tyto parametry jsou standardním[120] mechanismem předány do funkce fminbnd. Tato spočítá nějakou hodnotu úhlu *x* a pomocí volání funkce prikl16_8delo s touto hodnotou úhlu jako parametrem se zeptá na odpovídající dostřel. Funkce prikl16_8delo standardním mechanismem přijme hodnotu úhlu v proměnné *a* a chtěla by pomocí SIMULINKu (funkce **`sim`** s názvem modelu) spočítat dostřel. Avšak model si bere hodnotu úhlu z proměnné *alfa* v Basic Workspace, který je od pracovního prostoru funkce oddělen. Abychom dostali

[120] standardní mechanismus je naznačen širokou šipkou



hodnotu úhlu (máme ji v proměnné *a*) do proměnné *alfa* musíme vytvořit propojení s příslušným pracovním prostorem pro přístup k proměnné *alfa*. Toho dosáhneme uvedení klíčového slova **`global`** ve funkci prikl16_8delo. Nyní můžeme aktualizovat hodnotu proměnné *alfa* a nechat spočítat model. Ten je napsán tak, aby vypočtený dostřel (pro aktuální hodnoty parametrů) uložil do proměnné *delka*. Nyní by měla přijít otázka – do kterého pracovního prostoru ? Na rozdíl od parametrů bloků, které se berou ze základního pracovního prostoru, zápis se provádí do pracovního prostoru funkce, který výpočet modelu vyvolal. Proměnná *delka* je ve funkci prikl16_8delo, tedy dostupná, přebere se vypočtená hodnota (otočí se znaménku neb funkce `fminbnd` hledá minimum) a tato se předá jako výstupní parametr funkce. Tuto hodnotu převezme funkce `fminbnd`, zpracuje a vypočte nový odhad náměru, pro který si opět nechá naší funkcí spočítat nový dostřel. Toto se opakuje dokud funkce fminbnd neuzná, že už to stačí a nevrátí konečný výsledek na příkazový řádek, který ji zavolal. Celé řešení je pozorovatelně rychlejší než řešení uvedené v příkladu kapitoly 10.1.

Globální proměnná tedy funguje v MATLABu jako selektivní propojení mezi několika pracovními prostory (těmi, ve kterých je použita definice s klíčovým slovem `global`), které jsou jinak zcela odděleny.

Seznam literatury

Seznam literatury obsahuje publikace týkající se MATLABu či SIMULINKu, které jsou relativně dostupné (ať už v tištění či elektronické podobě) a v českém či slovenském jazyce. Upozorňuji na nevyrovnanost kvality elektronických odkazů a zejména na jejich nejistou časovou platnost.

Publikace týkající se MATLABu a SIMULINKu v češtině či slovenštině v tištěné podobě

(seznam převzat z http://www.humusoft.cz/matlab/knihy.htm 14.4.2005)

B. Šulc, M. Vítečková. **Teorie a praxe návrhu regulačních obvodů**. Vydavatelství ČVUT, 333 stran, 187 obrázků ISBN 80-01-03007-5
Miloš Sedláček, Radislav Šmíd. **MATLAB v měření**. 204 s. ISBN 80-01-02851-8
Mikuláš Huba, Peter Hubinský, Katarína Žáková. **Teória systémov**. 418 s. ISBN 80-227-1820-3
Monika Bakošová, Miroslav Fikar, Ľuboš Čirka. Základy **automatizácie**. 154 s. ISBN 80-227-1831-9
Róbert BARTKO - Marián MILLER. **MATLAB I. - algoritmizácia a riešenie úloh**. 285 s. ISBN 80-968337-3-1
Mária Franeková. **Modelovanie komunikačných systémov v prostredi MATLAB, Simulink a Communication**. Žilinská univerzita v Žiline, 2003, ISBN 80-8070-027-3
Karel Zaplatílek - Bohuslav Doňar. **MATLAB tvorba uživatelských aplikací**. 216s. ISBN 80-7300-133-0
Karel Zaplatílek - Bohuslav Doňar. **MATLAB pro začátečníky**. 143 s. ISBN 80-7300-095-4
V. Stejskal, M. Okrouhlík. **Kmitání s MATLABem**. 376 s. ISBN 80-01-02435-0
Zdeněk Smékal, Robert Vich. **Číslicové filtry.** 220 s. ISBN 80-200-0761-X
Petr Noskievič. **Modelování a identifikace sys**témů. 276 s. ISBN 80-7225-030-2
Vladimír Bobál, Josef Böhm, Roman Prokop, Jaromír Fessl. **Praktické aspekty samočinně se nastavujících regulatorů: algoritmy a implementace**. 242 s. ISBN 80-214-1299-2
G. Hulkó, M. Antoniová, C. Belavý, J. Belanský, J. Szuda, P. Végh. **Modelovanie, riadenie a návrh systémov s rozloženými parametrami s demonštráciami v prostredí MATLAB**. Vydavateľstvo STU, 1998, 266 stran, - Softcover - ISBN 80-227-1052-0 (slovensky), ISBN 80-227-1083-0 (anglicky)
Štefan Kozák, Slavomír Kajan. **MATLAB-SIMULINK 1 - učebnice zaměřená na hlavní modul MATLABu**. 125 s. ISBN 80-227-1213-2
Štefan Kozák. **MATLAB-SIMULINK 2 - učebnice zaměřená na Control System Toolbox**. 141 s. ISBN 80-227-1235-3

Základní informace o užívání MATLABu v elektronické podobě v češtině (odkazy platné k 14.4.2005)

http://mech.fsik.cvut.cz/doc_cz.html
http://artax.karlin.mff.cuni.cz/~beda/cz/matlab/primercz/matlab-primer.html
http://cmp.felk.cvut.cz/~pisa/Public/ST_matlab.html
http://k315.feld.cvut.cz/vyuka/matlab/
http://cmp.felk.cvut.cz/~pisa/Public/ST_matlab.html
http://uprt.vscht.cz/majerova/matlab/

Publikace týkající se MATLABu a SIMULINKu v angličtině v tištěné i elektronické podobě

V. Stejskal, P. Dehombreux, A. Eiber, R. Gupta, M. Okrouhlik. **Mechanics with Matlab**
http://fsinet.fsid.cvut.cz/en/U2052/mechmat1.html (platné 14.4.2005)

Přehled literatury z různých oblastí využívající MATLAB či SIMULINK (**dostupná na serveru fy MathWorks – MATLAB Based Books for use with MATLAB, MATLAB Toolboxes, and SIMULINK - 808 položek k 14.4.2005)**

http://www.mathworks.com/support/books/index.jsp

Rejstřík

G

H

I

K

L

M

N

O

P

R

S

Š

T

U

V