MATLAB
Simulink

úvod do použití

Libor Kupka

TENTO PROJEKT JE SPOLUFINANCOVÁN
EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM
ROZPOČTEM ČESKÉ REPUBLIKY

Matlab & Simulink

úvod do použití

Libor Kupka

Obsah

Předmluva

Projekt CZ.04.1.03/3.1.15.1/0005 operačního programu EU Rozvoj lidských zdrojů, opatření 3.1, je zaměřen na podporu výuky mechatroniky a automatizace na středních odborných školách. V projektu preferujeme názorné způsoby výuky ve spojení s praktickými učebními pomůckami. Z nejrůznějších důvodů, především finančních a prostorových, se však nemůže nikdy podařit všechny potřebné znalosti a dovednosti demonstrovat žákům s pomocí fyzikálních pomůcek. K dispozici má však dnes už každá střední odborná škola počítačové učebny vybavené kvalitními počítači a zpravidla propojené v síti. Této skutečnosti jsme se rozhodli v rámci programu využít.

S využitím výpočetní techniky lze demonstrovat jevy probíhající v mechatronických systémech, propočítávat potřebné souvislosti a výstupní parametry a nakonec i simulovat fyzické části mechatronických systémů. Na modelech lze i experimentovat. Učitelům umožní tento postup podrobně seznámit žáky s dalšími vlastnostmi mechatronických systémů, které nejsou efektivně realizovatelné na fyzikálních učebních pomůckách. Nákladově bude škola zatížena prakticky jen při nákupu potřebného software a při vyškolení odborných učitelů.

Zevrubné diskusi byl podroben výběr vhodného software pro účely výuky. Nakonec jsme se přiklonili k použití prostředí MATLAB z několika důvodů:

1. MATLAB je univerzální prostředek s rozsáhlou nadstavbou podporující např. simulaci dynamických systémů (Simulink) s dobrou grafikou.
2. MATLAB je rozšířen na všech technických vysokých školách a při dalším studiu se absolventi středních odborných škol v menší či větší míře s tímto programovým prostředím setkají. Znalosti s využitím MATLABu jim mohou významně prospět při adaptaci na vysokoškolských pracovištích.
3. MATLAB je využitelný i v řadě dalších předmětů vyučovaných na středních školách, především v matematice a fyzice.

Abychom usnadnili učitelům středních škol používání prostředí MATLAB včetně Simulinku, připravili jsme v rámci projektu předloženou příručku. Ing. Libor Kupka nás s přehledem pedagogického pracovníka seznamuje se základy využívání MATLABu, s používáním grafiky a nakonec i Simulinku. Metodické návody jsou demonstrovány na příkladech. Příklady jsou voleny tak, aby byly přímo využitelné pro výuku mechatroniky.

Přál bych si, aby MATLAB našel cestu do středních škol, aby pomáhal při výuce středoškolské mládeže a aby příručka Ing. Kupky v tomto průniku sehrála významnou roli. Pak budeme moci konstatovat, že projekt naplňuje své programové cíle.

Doc. Ing. Ladislav Maixner, CSc.
manažer projektu

1 Úvod

Předkládaná kniha si klade za cíl seznámit čtenáře s ovládáním a se základními funkcemi prostředí MATLAB – Simulink. Vzhledem k rozsahu MATLABu jsou podrobně probírány pouze některé jeho části. Jsou to zejména části, které nejsou příliš složité a jejichž znalost je pro práci v tomto prostředí nezbytná. Uživatel – začátečník je postupně seznamován se základy práce s MATLABem a s jeho nadstavbovým prostředím Simulink. Text je hojně doplněn řešenými příklady.

Kniha nemá v žádném případě plnit funkci manuálu. Tam, kde to autor považoval za vhodné, je ale pro zlepšení přehledu uživatele doplněna o výčty všech použitelných funkcí a jejich vlastností (obvykle ve formě tabulek). Text je také doplněn o základy použití třírozměrné grafiky, která na středních školách zpravidla není vyučována. Lze na ní ale ukázat vynikající grafické možnosti MATLABu a rozšířit tak rozhled středoškolského učitele.

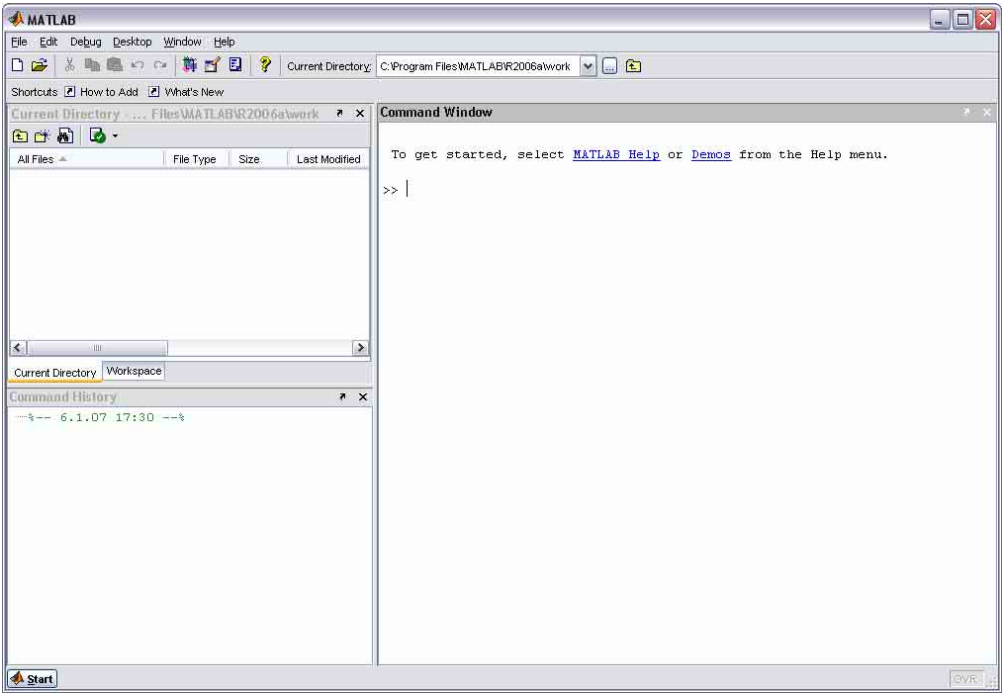
Systém MATLAB, vyvinutý v roce 1984 firmou The Mathworks, Inc. v USA, je výkonné prostředí určené pro vědecké a inženýrské výpočty a vizualizaci dat. Název MATLAB vznikl z anglického *MATrix LABoratory*. MATLAB integruje numerickou analýzu, maticové výpočty a grafiku do uživatelsky příjemného prostředí, ve kterém se řešené problémy zapisují podobně jako v matematice – tedy bez obtíží klasického programování. V prostředí MATLAB lze řešit široký okruh problémů spojených především s matematikou, fyzikou, měřením a zpracováním dat a grafikou, ale i mnoho dalších.

Nadstavbové prostředí MATLABu Simulink slouží k modelování a simulaci dynamických systémů. Simulink využívá rozsáhlých grafických možností operačního systému Windows. Uživateli poskytuje možnost snadno a rychle vytvářet modely ve formě blokových schémat. Tato schémata jsou značně podobná zapojením pro analogový počítač. Hierarchická struktura modelů umožňuje vytvářet i velmi složité systémy, jejichž jednotlivé části lze zahrnout do přehledné struktury subsystémů a to prakticky bez omezení počtu bloků. Simulink využívá algoritmy MATLABu pro numerické řešení diferenciálních rovnic. To je i jeden z důvodů, proč je třeba se nejdříve seznámit se základním prostředím MATLAB. Pro efektivní využití Simulinku při řešení složitějších problémů je znalost MATLABu prakticky nezbytná.

2 Základy práce v prostředí MATLAB

V této části se seznámíme s desktopem prostředí MATLAB, jeho možnostmi a základním nastavením. Ukážeme si, jak se orientovat v rozsáhlé nápovědě a využívat četná dema, která jsou v prostředí MATLAB k dispozici a také jak využívat technickou podporu MATLABu na internetových stránkách výrobce. Dále bude vysvětlen princip práce v příkazovém řádku a zápis jednoduchého výrazu a proměnné. Probrány budou také některé zvláštní typy proměnných a konstant, zápis komplexních čísel a základy práce s řetězci a strukturami.

Pro efektivní práci a osvojení si základních přístupů při práci v prostředí MATLAB je vhodné mít toto prostředí spuštěné. Obvykle je příslušná ikona na pracovní ploše počítače. Není-li tato ikona na ploše, je možné MATLAB spustit prostřednictvím souboru **matlab.exe**, který je umístěn ve složce C:/Program Files/MATLAB/R2006a/bin/win32. Toto umístění je samozřejmě platné pouze pro verzi MATLABu R2006a. Máme-li k dispozici verzi jinou, název složky je pozměněn, název souboru je ale stejný. Záleží samozřejmě také na okolnosti, zda uživatel ponechal při instalaci implicitně nabízenou složku. Popisovanou verzi MATLABu je vhodné provozovat pod operačním systémem Windows XP. Pokud má uživatel nainstalovánu jinou (starší) verzi operačního systému Windows, měl by použít nižší verzi MATLABu.

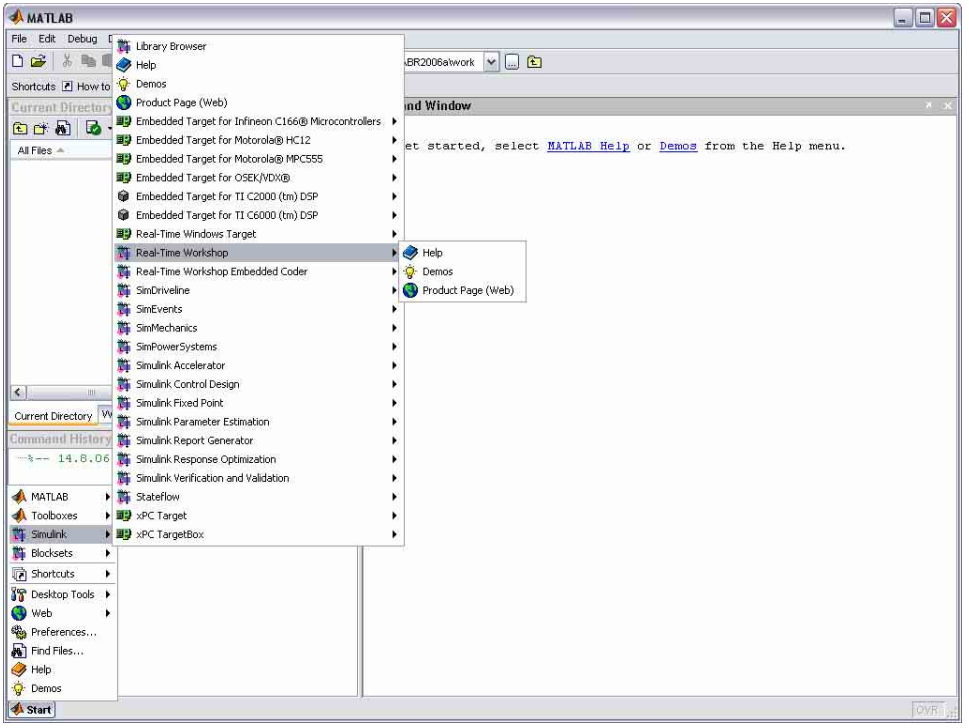


Obr. 2.1: Pracovní plocha prostředí MATLAB R2006a

Verze MATLABu R2006a je oproti předcházejícím verzím (zejména oproti stále velmi často používané verzi 5) poněkud odlišná. Po spuštění MATLABu se otevře pracovní plocha (tzv. desktop, obr. 2.1), která je složená ze tří oken. Základním oknem je *Command Window* a dále jsou otevřena také okna *Workspace* a *Command History*. Okno *Workspace* může být pomocí myši přepnuto na *Current Directory* – kliknutím na záložku ve spodní části.

Kombinace jednotlivých oken v desktopu je libovolná, s každým oknem lze pracovat samostatně a do desktopu lze zakomponovat také další okna. Uživatelské nastavení může být provedeno v položce **Desktop Layout** menu **Desktop** v horní liště základního okna. Původní uspořádání oken lze vrátit pomocí menu **Desktop → Desktop Layout → Default**.

Způsob práce v desktopu názorně popisuje několik animovaných demo programů v sekci *Desktop Tools and Development Environment*. Jejich spuštění je možné prostřednictvím okna *Command Window* příkazem **demo** nebo z roletového menu **Help**. Další možností je volba položky **Demos** z menu tlačítka START.



Obr. 2.2: Pracovní plocha prostředí MATLAB R2006a – použití tlačítka START

Okno ***Command Window*** je hlavní a nejdůležitější částí pracovní plochy (desktopu). V tomto okně uživatel zapisuje jednotlivé příkazy a je v něm zobrazena odezva a různá systémová hlášení MATLABu.

V okně ***Command History*** se zobrazují všechny příkazy, zapsané uživatelem v hlavním okně *Command Window*. Chce-li uživatel použít již jednou zapsaný příkaz (nebo i celou sekvenci příkazů), stačí jej v tomto okně jednoduše nalistovat a poklepáním znovu aktivovat.

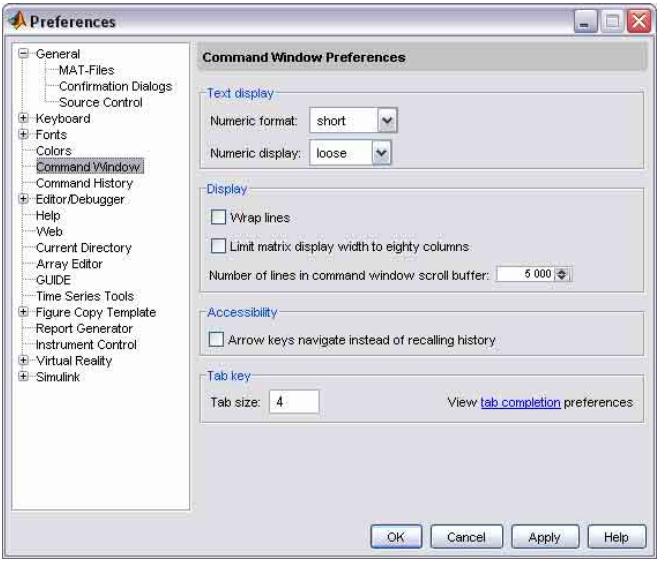
Je také možné jej myší přetáhnout do hlavního okna. Přímo v hlavním okně *Command Window* lze již jednou zapsané příkazy znovu vyvolat pomocí kurzorových šipek nahoru a dolů.

Okno ***Workspace*** je při prvním spuštění MATLABu prázdné. Veškeré proměnné, které v průběhu práce uživatel nadefinuje, se zobrazují v tomto okně. Okno obsahuje vždy úplný seznam všech typů proměnných. Poklepáním myší na některé z nich je možné zobrazit detailní informace (rozměr, struktura, atd.).

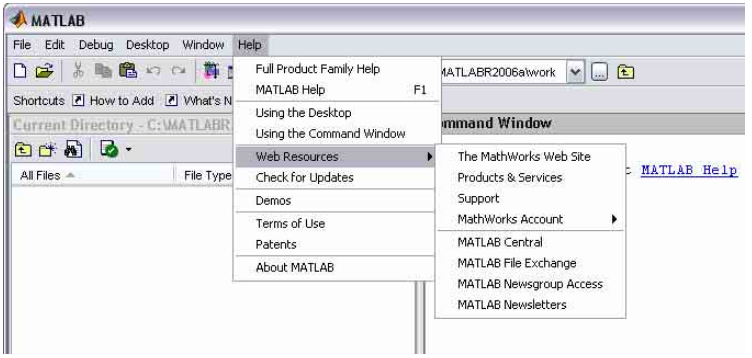
V okně ***Current Directory*** je zobrazen seznam souborů v aktuální složce. Poklepáním myší na některém ze souborů je možné jej otevřít. Způsob otevření souboru je závislý na jeho typu, např. soubory s příponou MAT jsou otevírány ve vestavěném editoru. Aktuální složka (*Current Directory*) je zobrazena nad oknem *Command Window*. Aktuální složku je samozřejmě možné měnit, implicitně je nastavena do složky C:/Program Files/MATLAB/R2006a/Work.

Od MATLABu verze 6 se v levé spodní části pracovní plochy, podobně jako v operačním systému Windows, objevuje tlačítko START (obr. 2.2). S jeho pomocí je možné spouštět řadu vestavěných aplikací, používat nápovědu, otevírat internetové stránky apod. Ukončit práci s prostředím MATLAB je možné standardním způsobem, prostřednictvím nabídky **File → Exit MATLAB (Ctrl + Q)** nebo kliknutím myší na křížek vpravo nahoře. Další možností jak ukončit MATLAB je zápis příkazu **quit** a jeho potvrzení klávesou ENTER.

Některé vlastnosti a vzhled prostředí MATLAB je možné nastavit pomocí dialogového okna *Preferences*, k němuž lze přistoupit přes menu **File → Preferences…** Prostřednictvím tohoto dialogu (obr. 2.3) je možné nastavit vlastnosti pro okna *Command Window*, *Command History* a *Current Directory*. Dále také některé vlastnosti vestavěného textového editoru a Simulinku. Přes základní menu v horní liště hlavního okna je možné jednoduše přistupovat i k systému nápovědy a také k součástem MATLABu dostupným prostřednictvím internetu (obr. 2.4). V horní liště okna v tomto případě zvolíme menu **Help** resp. jeho položku **Web Resources**.



Obr. 2.3: Dialogové okno *Preferences* pro nastavení vlastností prostředí



Obr. 2.4: Přístup k nápovědě a k součástem MATLABu dostupným na Internetu

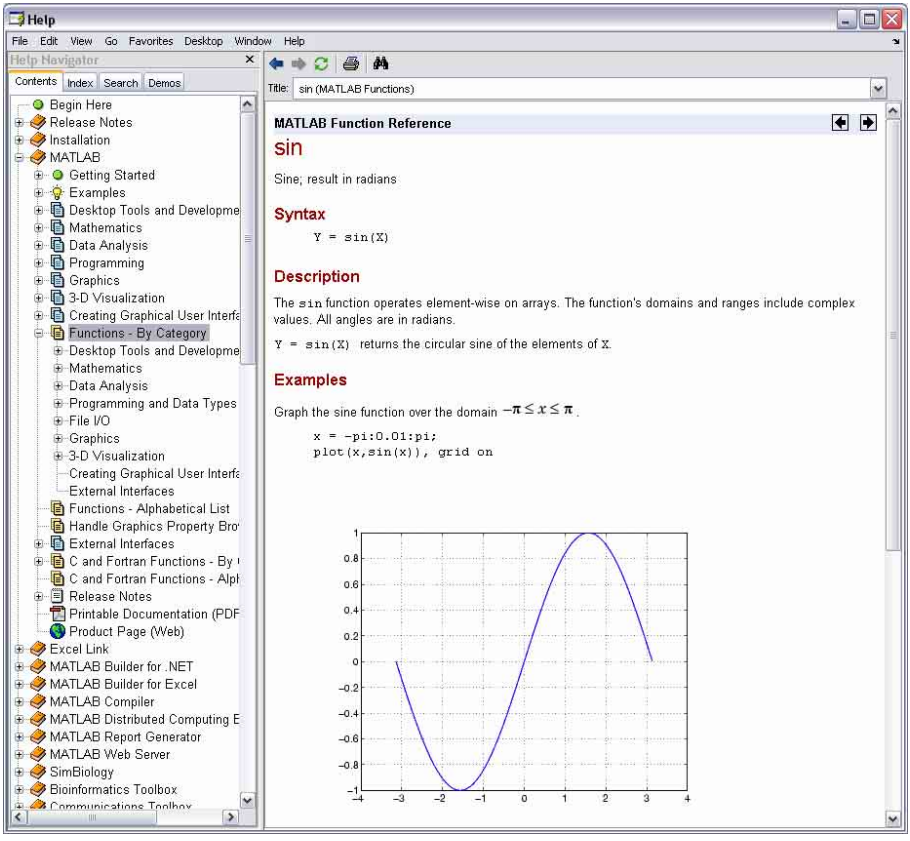
Obr. 2.5: Internetová podpora MATLABu

Další možností jak přistupovat k nápovědě je zápis příkazu **help** v hlavním okně *Command Window*. Pokud příkaz zadáme a potvrdíme bez argumentu, dojde k vypsání kompletního seznamu dostupných položek nápovědy. K jednotlivým položkám je možné přistupovat např. kliknutím na příslušné heslo. Podrobný popis použití nápovědy je možné vyvolat zápisem příkazu **help help**. Užitečné je využití odkazů *See also*, které představují příkazy související s původně zadaným příkazem. Je také možné vyvolat podrobnější textovou verzi nápovědy zobrazovanou v okně *Help*, které lze mimochodem využít také i jako prohlížeč internetových

stránek (jednoduchá obdoba běžně používaných prohlížečů, např. Internet Explorer, Firefox, Opera, atd.). Přechod k této verzi nápovědy je možný opět kliknutím na heslo v části *Reference page in Help browser*. Jako příklad si ukážeme dostupné varianty nápovědy k funkci sinus.

```
Command Window
>> help sin
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded functions or methods (ones with the same name in other
       help sym/sin.m

    Reference page in Help browser
       doc sin

>>
```

Obr. 2.6: Nápověda k příkazu **sin**



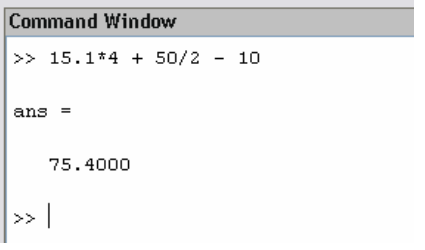
Obr. 2.7: Podrobnější verze nápovědy k příkazu **sin**

Aby bylo možné využívat okno *Help*, je samozřejmě nutné mít nainstalované soubory nápovědy. Tyto soubory jsou typu HTML a poskytují skutečně efektivní způsob práce se systémem nápovědy. V levé horní části okna *Help* může uživatel také vybrat záložku *Index*. Do příkazové řádky lze pak zapisovat celé příkazy nebo pouze jejich části. MATLAB automaticky doplní syntaxi a vyhledá podobná hesla. Dále je možné využít záložky *Search* a *Contents*. V záložce *Demos* lze vyhledat vhodné demo. Všechna dostupná dema obsahují i použité příkazy příp. celé skripty, které můžeme samozřejmě využít ve vlastních programech, a ukázky různých variant zobrazení výsledků, např. ve formě grafu.

2.1 Práce v příkazovém řádku

Jak již bylo uvedeno, okno *Command Window* je základním oknem prostředí MATLAB. Slouží k zadávání příkazů, spouštění skriptů (m-souborů) a jednotlivých funkcí MATLABu, výpisu proměnných a případně k vyvolání nápovědy pomocí příkazu **help**. Mezi základní příkazy patří:

helpwin – vyvolání okna *Help* pro nápovědu,
help <*funkce*> – nápověda k zadané funkci v okně *Command Window*,
doc <*funkce*> – podrobnější nápověda k zadané funkci v okně *Help*,
pwd – vypíše aktuální adresář včetně úplné cesty,
dir, ls – výpis obsahu aktuálního adresáře,
format – nastavení výstupního zobrazovaného formátu v příkazovém okně,
lookfor – vyhledává zadané klíčové slovo ve všech nápovědách,
workspace – zobrazí okno pracovního prostoru,
cd – mění pracovní adresář nebo vypíše aktuální včetně cesty,
mkdir – vytváří nový adresář,
rmdir – odstraňuje adresář,
copyfile – kopíruje soubory nebo adresáře,
movefile – přesouvá soubory nebo adresáře,
what – vrací seznam specifikovaných souborů v aktuálním adresáři,
type – vypíše obsah M-souboru,
whos – výpis proměnných (textová kopie Workspace),
who – zjednodušený výpis proměnných (pouze jejich název),
clear <*proměnná*>, **clear all** – vymazání proměnné resp. všech proměnných,
web – zobrazí HTML soubor nebo zadanou HTTP adresu.



Obr. 2.8: Zápis jednoduchého výrazu a zobrazení výsledků

Výpis všech příkazů je možný po zadání **help matlab/general**. Prompt v příkazovém okně, který představuje připravenost MATLABu k činnosti, je >>. Po zápisu jednoduchého výrazu a potvrzení klávesou ENTER je v okně zobrazen výsledek, který se ukládá do proměnné **ans**, se kterou lze dále pracovat. Příkladem je zápis jednoduchého výrazu na obr. 2.8.

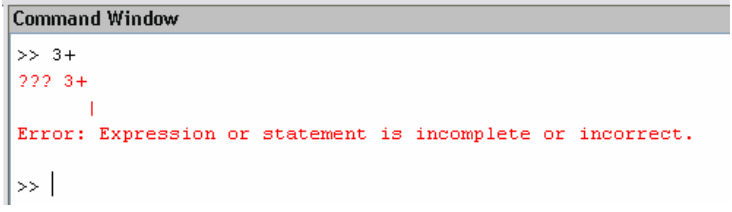
Využití implicitní proměnné **ans** není ale příliš vhodné, protože do **ans** může být uložena jakákoliv hodnota. Vhodnější je přiřazování hodnot do proměnných. Na místo desetinné čárky zapisujeme tečku. Výsledek je zobrazen ve formátu s přesností na čtyři desetinná místa. Základní matematické operace a jejich zápis jsou v tab. 2.1. Výrazy se vyhodnocují zleva doprava s následující prioritou: umocňování, násobení a dělení, sčítání a odčítání. Prioritu lze měnit použitím závorek.

Tab. 2.1: Základní matematické operace v MATLABu

Operace	Symbol	Příklad
sčítání, a + b	+	3+22
odčítání, a – b	–	90–54
násobení, a · b	*	3.14*0.85
dělení, a ÷ b	/ nebo \	56/8 = 8\58
umocňování, a^b	^	2^8

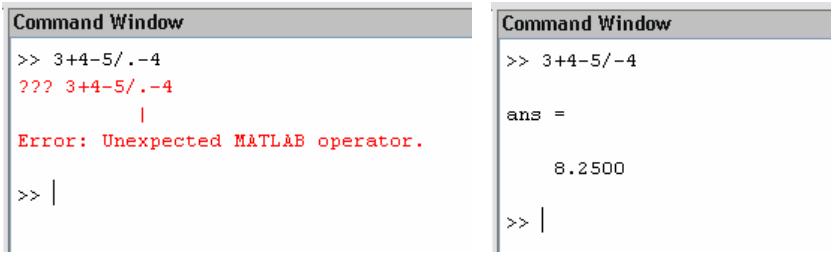
Jak již bylo řečeno, provedené příkazy se ukládají do historie. Jednotlivé příkazy lze pak jednoduše vybírat pomocí kurzorových šipek ↑ a ↓ a nebo lze historii příkazů sledovat v okně *Command History*. Mazání zvoleného příkazu, pokud jej nechceme použít, lze provést stiskem klávesy ESCAPE. Historii lze i vypnout resp. opětovně zapnout pomocí příkazu **diary**. Je také možné historii příkazů zapsat do definovaného souboru (více **help diary).** Implicitně se historie zapíše do souboru „diary“ ve složce C:/Program Files/MATLAB/R2006a/Work, pokud bylo zadáno **diary on**. Vlastní zápis je proveden až při ukončení činnosti MATLABu.

Pokud je výraz, který chceme řešit, zapsán chybně nebo zápis není úplný, dojde po potvrzení klávesou ENTER k vypsání chybového hlášení. Příklad takového hlášení je na obr. 2.9. V tomto případě nebylo zadání výrazu úplné, chybí zápis druhého sčítance. MATLAB zřetelně označil pozici chyby. Chybová hlášení jsou velmi dobrou vlastností MATLABu, umožňují snadné vyhledání chyby i v případě složitých výrazů či celých skriptů.



Obr. 2.9: Příklad chybového hlášení

Výše popsaný nejjednodušší způsob práce v příkazovém řádku MATLABu připomíná práci s kalkulačkou. Pokud budeme postupně zadávat další výrazy, okno *Command Window* se bude průběžně zaplňovat řešenými výrazy, výsledky a případně i chybovými hlášeními. Po zaplnění celého okna dojde k posunu horních řádků směrem nahoru, mimo oblast okna. Obsah okna je samozřejmě možné i smazat pomocí příkazu **clc** a zvýšit tak přehlednost.



Obr. 2.10: Označení chyby v řešeném výrazu, správný zápis

2.2 Proměnné v MATLABu

Proměnná v prostředí MATLAB se může skládat až z 31 znaků, další jsou ignorovány. V názvu proměnné se rozlišují malá a velká písmena (je case sensitive), jména proměnných musí začínat písmenem a nesmí obsahovat tečku. Následuje seznam některých speciálních proměnných a řídicích znaků, viz tab. 2.2 a 2.3.

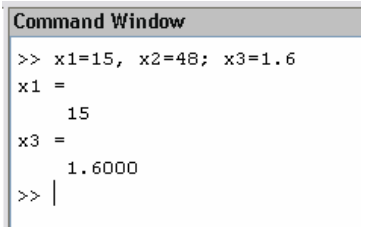
Tab. 2.2: Speciální proměnné

Proměnná	Popis a použití
ans	implicitní proměnná používaná k zápisu výsledků neprovede-li uživatel přiřazení do proměnné
beep	zapnutí nebo vypnutí zvuku, výpis aktuálního nastavení **beep [on |off], s = beep**
eps	přesnost výpočtu na daném počítači
realmax	největší možné kladné reálné číslo
realmin	nejmenší možné kladné reálné číslo
pi	Ludolfovo číslo π
i nebo **j**	imaginární složka komplexního čísla, **i** = **j** = $\sqrt{-1}$
Inf (inf)	nekonečno (např. 1/0)
NaN (nan)	Not–a–Number (např. 0/0)
computer	vypíše typ počítače (obvykle PCWIN)
version	vypíše verzi MATLABu (např. v textu popisovaná verze 7.2.0.232 (R2006a))
clock	vypsání aktuálního času, je obvykle v následujícím formátu 1.0e+003 * (2.0060 0.0080 0.0150 0.0140 0.0070 0.0509)
date	aktuální datum v počítači (např. 15-Aug-2006)
nargin	počet vstupních argumentů funkce, **nargin(*funkce*)**
nargout	počet výstupních argumentů funkce, **nargout(*funkce*)**
varargin	proměnné vstupující do funkce (pole buněk, více **help varargin**)
varargout	proměnné vstupující do funkce (pole buněk, více **help varargout**)

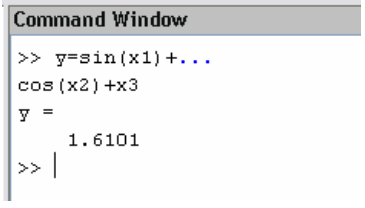
Tab. 2.3: Nejpoužívanější řídicí znaky

Znak	Popis
. (tečka)	desetinná tečka, oddělovač proměnných ve struktuře
, (čárka)	oddělovač příkazů na řádku (výpis není potlačen) nebo prvků
; (středník)	potlačení výpisu po přiřazení do proměnné, volání funkce nebo m-souboru, oddělovač prvků ve sloupci či celých řádků u matic
: (dvojtečka)	rozsah hodnot nebo indexů ve vektoru nebo matici
% (procento)	oddělení komentáře na řádce
… (tři tečky)	rozdělení dlouhého řádku
!	spuštění systémových příkazů (např. **! dir** výpis adresáře v okně MATLABu nebo **! dir &** přímo v okně emulátoru DOSu cmd.exe)
[ ]	ohraničení obsahu definovaného vektoru nebo matice
{ }	ohraničení obsahu definované buňky
( )	použití při indexování vektorů nebo matic, obsahy
' (apostrof)	ohraničení textové proměnné, transpozice matice

Jednotlivé příkazy lze na řádku zapisovat za sebe a navzájem je oddělovat čárkou (**,**) nebo středníkem (**;**). Čárka, na rozdíl od středníku, nepotlačuje výpis (obr. 2.11). Pokud je příkaz příliš dlouhý, lze jej rozdělit na více řádků použitím tří teček (**…**), viz obr. 2.12.



Obr. 2.11: Oddělování příkazů na řádku



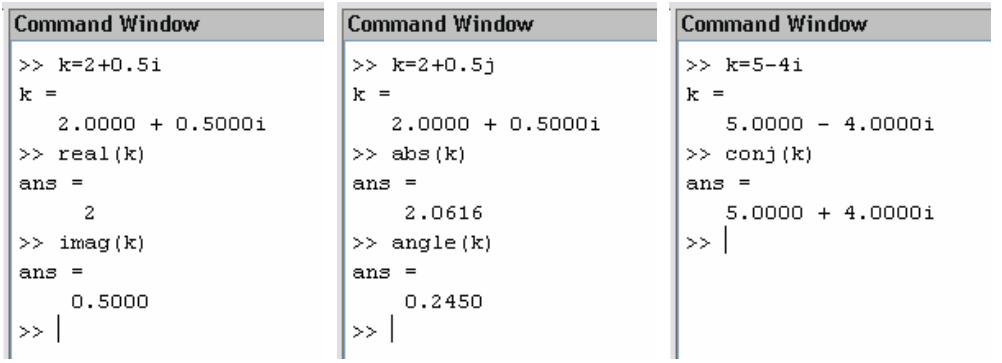
Obr. 2.12: Rozdělení dlouhého příkazu

Při práci s proměnnými je možné, že uživatel ztratí přehled kolik a pod jakými jmény jich již nadefinoval. K výpisu aktuálního stavu lze použít příkazy **who** nebo **whos**. Druhý příkaz z uvedených provede podrobnější výpis seznamu proměnných. Kromě jejich názvu, bude vypsán také jejich rozměr (`1x1` značí proměnnou, jejíž obsah je pouze jedno číslo – skalár) a struktura (např. `8 double array` značí počet obsazených bajtů (Bytes) v paměti a použitou přesnost. Proměnné, které již nechceme používat, lze vymazat a ušetřit tak místo v paměti.

K tomuto účelu slouží příkaz **clear**. Všechny definované proměnné lze vymazat zadáním **clear all**, jednotlivou proměnnou pak zadáním **clear** *<proměnná>*. Příkaz **clear** samozřejmě nabízí více možností, jako je např. mazání definovaných globálních proměnných apod., více zadáním **help clear** nebo **doc clear**.

2.3 Práce s komplexními čísly

Zadání komplexního čísla se v příkazovém řádku provádí pomocí symbolu **i** nebo **j**. Samostatně lze zobrazit reálnou a imaginární část komplexního čísla pomocí příkazů **real** a **imag**, vypočítat absolutní hodnotu (délku průvodiče v komplexní rovině) pomocí příkazu **abs** a úhel mezi průvodičem a reálnou osou pomocí příkazu **angle**. Je také možné vypočítat číslo komplexně sdružené (konjugované) pomocí příkazu **conj**. Na obr. 2.13 jsou uvedeny příklady práce s komplexními čísly.



Obr. 2.13: Příklady práce s komplexními čísly

2.4 Formát zobrazování čísel

MATLAB pracuje ve dvojnásobné přesnosti (double precision). Desetinná čísla jsou zobrazována s desetinnou tečkou, implicitně na 4 desetinná místa (formát short). Formát zobrazování čísel nesouvisí s přesností výpočtu. Změnu formátu čísel při výpisu výsledku na obrazovku a také případné potlačení mezer mezi řádky umožňuje příkaz **format**:

format short – implicitní formát čísla na 5 číslic (4 desetinná místa),

format compact – potlačuje volný řádek v příkazovém okně,

format loose – přidá volný řádek v příkazovém okně.

Tab. 2.4: Možné formáty zobrazení

Příkaz	Příklad – formát čísla π (pi)	Popis
format short	3.1416	5 číslic (4 desetinná místa)
format short eng	3.1416e+000	5 číslic + exponent (násobek tří)
format short e	3.1416e+000	minimálně 5 číslic + exponent
format short g	3.1416	nejlepší z **short** nebo **short e**
format long	3.14159265358979	15 číslic (14 desetinných míst)
format long eng	3.14159265358979e+000	15 číslic + exponent (násobek tří)
format long e	3.14159265358979e+000	15 číslic + exponent
format long g	3.14159265358979	nejlepší z **long** nebo **long e**
format hex	400921fb54442d18	hexadecimální (šestnáctkový) zápis
format bank	3.14	2 desetinná místa
format +	+	kladné (+), záporné (–) nebo 0
format rat	355/113	racionální přiblížení
format debug	Structure address = 2dfd360 m = 1 n = 1 pr = 1d1b760 pi = 0 3.1416	informace o vnitřním uložení čísla s konečným zobrazením **short g**

2.5 Matematické funkce

MATLAB obsahuje celou řadu matematických a dalších funkcí, které jsou rozděleny do základních skupin. Úplný seznam lze získat pomocí příkazu **help**. Mezi základní skupiny funkcí (efektivní je vypsání jednotlivých funkcí pomocí **help** <*skupina*>) patří:

help elfun – přehled elementárních matematických funkcí,

help specfun – přehled speciálních matematických funkcí.

Dále je také možné zobrazit přehled příkazů pro všeobecné použití, přehled matematických operátorů, relačních operátorů, atd.

help general – přehled příkazů všeobecného použití,

help ops – přehled všech typů operátorů,

help arith – přehled aritmetických operátorů,

help relop – přehled relačních operátorů.

Tab. 2.5: Trigonometrické funkce

Funkce	Popis
sin	sinus, argument v radiánech
sind	sinus, argument ve stupních
sinh	hyperbolický sinus
asin	inverzní sinus, výsledek v radiánech
asind	inverzní sinus, výsledek ve stupních
asinh	inverzní hyperbolický sinus
cos	kosinus, argument v radiánech
cosd	kosinus, argument ve stupních
cosh	hyperbolický kosinus
acos	inverzní kosinus, výsledek v radiánech
acosd	inverzní kosinus, výsledek ve stupních
acosh	inverzní hyperbolický kosinus
tan	tangens, argument v radiánech
tand	tangens, argument ve stupních
tanh	hyperbolický tangens
atan	inverzní tangens, výsledek v radiánech
atan2	inverzní tangens (čtyři kvadranty)
atand	inverzní hyperbolický tangens, výsledek v radiánech
atanh	inverzní hyperbolický tangens, výsledek ve stupních
cot	kotangens, argument v radiánech
cotd	kotangens, argument ve stupních
acot	inverzní kotangens, výsledek v radiánech
acotd	inverzní kotangens, výsledek ve stupních
acoth	inverzní hyperbolický kotangens

Tab. 2.6: Exponenciální funkce

Funkce	Popis
exp	exponenciální funkce
log	přirozený logaritmus
log10	dekadický logaritmus
log2	logaritmus při základu 2
pow2	mocnina při základu 2
sqrt	druhá odmocnina
nextpow2	nejbližší vyšší mocnina při základu 2

Tab. 2.7: Komplexní funkce

Funkce	Popis
abs	absolutní hodnota nebo modul
angle	fázový úhel
conj	komplexně sdružená hodnota
imag	imaginární část
real	reálná část
isreal	test pro reálná pole
cplxpair	setřídění komplexně sdružených párů

Tab. 2.8: Funkce pro zaokrouhlování

Funkce	Popis
fix	zaokrouhlování směrem k nule, **fix**(–0.4) = 0
floor	zaokrouhlování směrem k – ∞, **floor**(–0.4) = –1
ceil	zaokrouhlování směrem k + ∞, **ceil**(0.4) = 1
round	zaokrouhlování k nejbližšímu celému číslu, **round**(0.4) = 0, **round**(0,7) = 1
mod	funkce modulo
rem	zbytek po celočíselném dělení, **rem**(13,5) = 3
sign	funkce signum (znaménková funkce), **sign**(5) = 1, **sign**(0) = 0, **sign**(–5) = –1

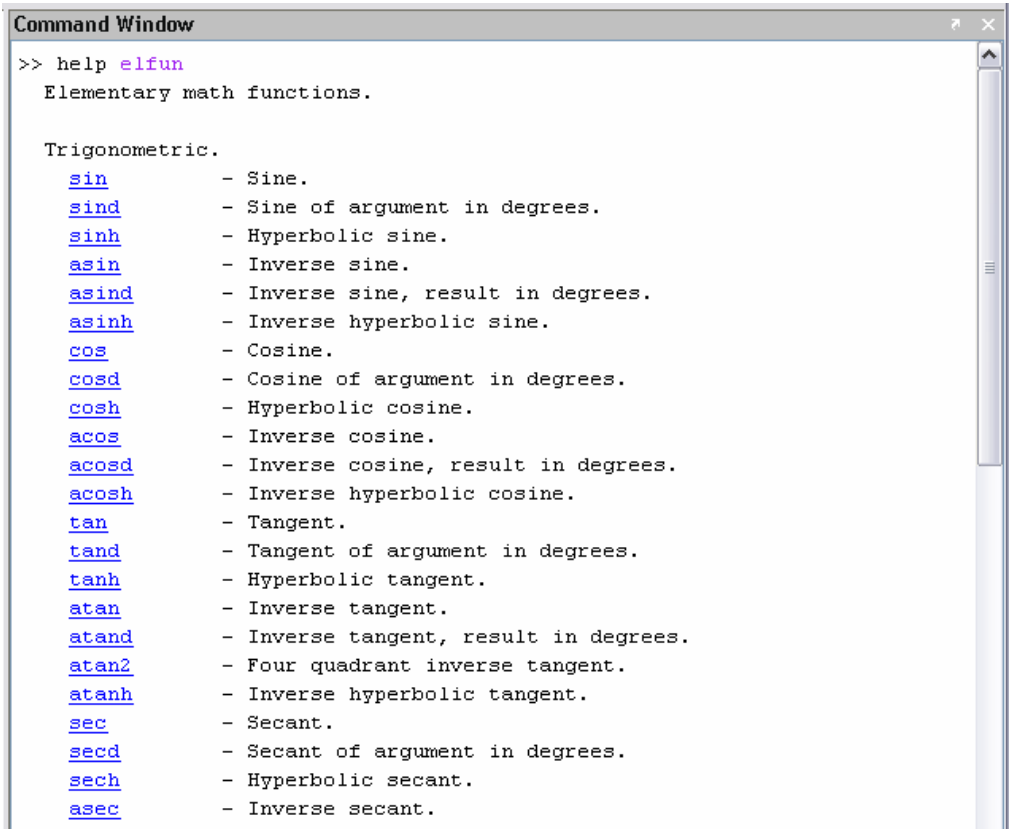
Tab. 2.9: Některé speciální matematické funkce

Funkce	Popis
airy	Airyho funkce
besselj	Besselova funkce prvního druhu
bessely	Besselova funkce druhého druhu
besselh	Besselova funkce třetího druhu (Hankelova funkce)
beta	funkce beta
betaln	logaritmus funkce beta
ellipke	úplný eliptický integrál
erf	chybová funkce
erfinv	inverzní chybová funkce
gamma	funkce gama
gammaln	logaritmus funkce gama
cross	vektorový součin
dot	skalární součin

Tab. 2.10: Funkce teorie čísel

Funkce	Popis
factor	rozklad na prvočísla, **factor**(26) = [2 13]
isprime	testování prvočísel, **isprime**(13) = 1, **isprime**(4) = 0
primes	generování prvočísel, **primes**(10) = [2 3 5 7]
gcd	největší společný dělitel, **gcd**(18,12) = 6
lcm	nejmenší společný násobek
rat	racionální aproximace, **rat**(0.7595) = 1 + 1/(–4 + 1/(–6 + 1/(–3 + 1/(–25))))
rats	racionální přiblížení, **rats**(0.7595) = 1519/2000
perms	všechny možné permutace, **perms**(1:2) = [1 2; 2 1]
nchoosek	všechny kombinace N nad K

Na následujících obrázcích jsou ukázky operací s některými elementárními funkcemi, princip zápisu složitějších výrazů a způsob využívání nápovědy.



Obr. 2.14: Použití nápovědy, výpis části skupiny elementárních funkcí

```
>> sqrt(49)
ans =
     7
>>
```

```
>> exp(-0.5)
ans =
    0.6065
>>
```

```
>> log10(1000)
ans =
     3
>>
```

```
>> sind(30)
ans =
    0.5000
>>
```

```
>> (cos(pi/2)+pow2(6))^2-cot(pi/4)*125+tand(45)
ans =
        3972
>>
```

Obr. 2.15: Operace s některými základními funkcemi

```
>> fix(-0.4)
ans =
     0
>> floor(-0.4)
ans =
    -1
>>
```

```
>> ceil(0.4)
ans =
     1
>>
```

```
>> round(0.4)
ans =
     0
>> round(0.7)
ans =
     1
>>
```

```
>> sign(0.5)
ans =
     1
>> sign(-0.5)
ans =
    -1
>>
```

Obr. 2.16: Některé funkce pro zaokrouhlování

```
>> factor(26)
ans =
     2    13
>> isprime(13)
ans =
     1
>> isprime(4)
ans =
     0
>>
```

```
>> primes(10)
ans =
     2     3     5     7
>> gcd(18,12)
ans =
     6
>> rats(0.7595)
ans =
   1519/2000
>>
```

```
>> rat(0.7595)
ans =
1 + 1/(-4 + 1/(-6 + 1/(-3 + 1/(-25))))
>> perms(1:2)
ans =
     2     1
     1     2
>>
```

Obr. 2.17: Některé funkce teorie čísel

2.6 Práce s řetězci znaků

Doposud byl uživatel seznamován s proměnnými, kterým byly přiřazovány pouze čísla. Velmi často je ale třeba přiřadit do proměnné i text, tedy řetězec znaků. Text je nutné umístit mezi apostrofy (' '), viz obr. 2.18. Určení délky řetězce je obdobné jako u vektorů a matic, pomocí příkazů **length** nebo **size** (podrobněji bude probráno později).

```
Command Window
>> s='Toto je retezec'
s =
Toto je retezec
>> length(s)
ans =
    15
>> size(s)
ans =
     1    15
>>
```

Obr. 2.18: Definování a určení délky řetězce

Převod řetězce do ASCII kódu je možný pomocí příkazu **abs** nebo **double**, zpětný převod z ASCII kódu na text pak pomocí příkazu **char** (obr. 2.19).

```
Command Window
>> u=abs(s)
u =
  Columns 1 through 11
    84   111   116   111    32   106   101    32   114   101   116
  Columns 12 through 15
   101   122   101    99
>> char(u)
ans =
Toto je retezec
>>
```

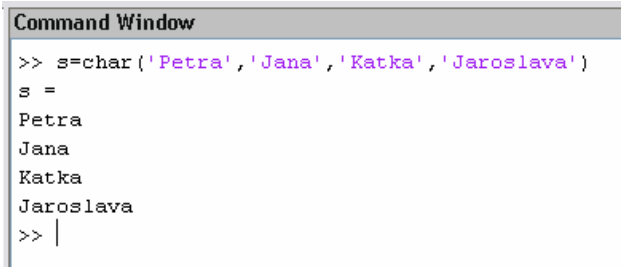
Obr. 2.19: Převod řetězce do ASCII kódu a zpět

```
Command Window
>> s=['cervena';'zelena ';'modra  ']
s =
cervena
zelena
modra
>> s=['cervena';'zelena';'modra']
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.

>>
```

```
Command Window
>> s=['cervena ','zelena ','modra ']
s =
cervena zelena modra
>> s=['cervena','zelena','modra']
s =
cervenazelenamodra
>>
```

Obr. 2.20: Zadávání pole řetězců

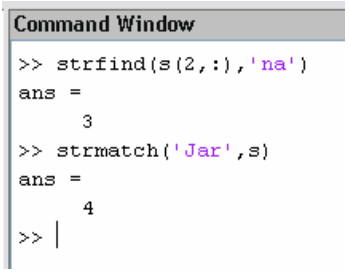
Je také možné vytvářet pole řetězců, jednak standardním způsobem pomocí závorek [ ], viz obr. 2.20, nebo lze znaková pole efektivněji definovat příkazem **char**. Pomocí tohoto příkazu lze provádět i další operace, např. umísťovat text pod sebe do řádků (obr. 2.21).



Obr. 2.21: Zadávání pole řetězců pomocí příkazu **char**

Velikost pole definovaného na obr. 2.21 zjistíme příkazem **size**. Definované textové pole má 4 řádky s maximální délkou řetězce 9 znaků (zde je na posledním řádku), tedy výsledkem po zadání příkazu **size(s)** bude vektor [4 9].

Hledání textu v jednoduchém řetězci zajišťuje funkce **strfind**. Pro hledání v poli řetězců je nutné nejprve určit řádek či sloupec, ve kterém budeme hledat. Postup je obdobný jako při práci s vektory a maticemi. Tato problematika bude podrobněji probírána později, v odstavci 3. K vyhledávání řádku, který začíná zadaným textem můžeme použít příkaz **strmatch**. Na obr. 2.22 je nejprve hledán v dříve definovaném řetězci (obr. 2.21) text ′na′ v celém druhém řádku (indexace (**2, :**). Výsledkem je číslo 3, tj. hledaný text začíná na 3. pozici. Dále je hledán řádek začínající textem ′Jar′. Výsledkem hledání je číslo 4, tj. 4. řádek definovaného pole řetězců.

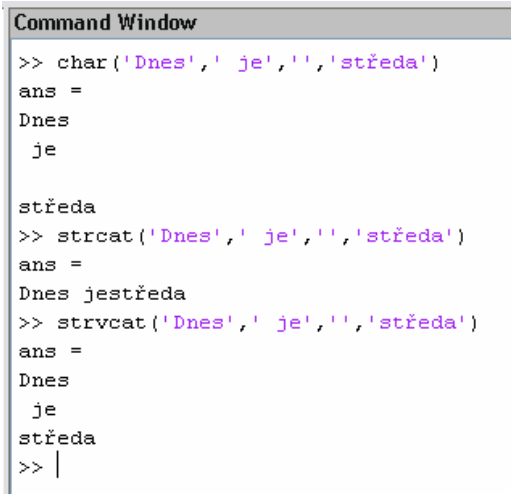


Obr. 2.22: Hledání v poli řetězců

K definování pole řetězců (resp. k spojování jednotlivých řetězců do jednoho) lze mimo příkaz **char** použít také příkazy **strcat** a **strvcat**. Příkaz **strcat** vypíše jednotlivé prvky pole vedle sebe a narozdíl od **char** ignoruje prázdné prvky. Příkaz **strvcat** vypíše prvky pole pod sebe a rovněž ignoruje prázdné prvky. Příklad použití obou uvedených příkazů je na obr. 2.23.

Prázdné pole v seznamu prvků se definuje bez mezery, tedy (**…, ′′, …**), nikoliv (**…, ′ ′, …**). Ekvivalentem pro zápis pole řetězců příkazem **strvcat** jsou hranaté závorky **[ ]** a použití středníků, podmínkou správné funkce je však stejná délka prvků, zatímco příkaz **strvcat** mezery přidá sám. Např. **strvcat(′pes′, ′kočka′, ′a′, ′husa′)** je ekvivalentní následujícímu zápisu **[′pes ′, ′kočka′, ′ a′, ′husa ′]**. Pokud jsou jednotlivé prvky při použití příkazu

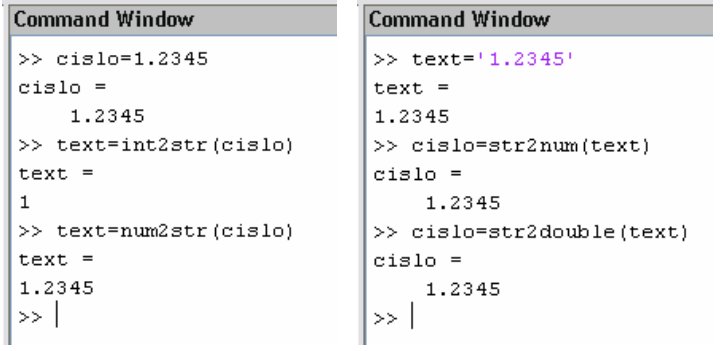
strcat také poli řetězců, je nutné, aby měly stejný počet řádků. Příkaz **strvcat** toto omezení nemá.



Obr. 2.23: Spojování řetězců

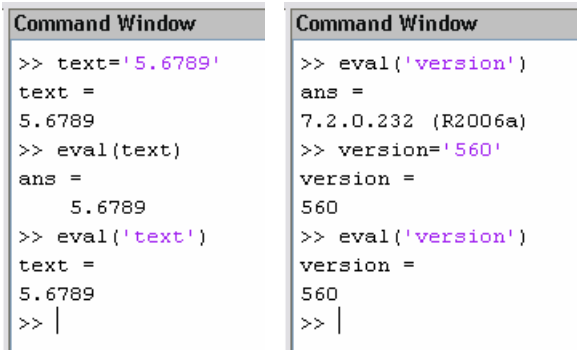
2.7 Převody mezi čísly a řetězci

Převod číselné hodnoty je možno realizovat pomocí příkazů **int2str** a **num2str** (obr. 2.24). Příkaz **int2str** nejprve převáděné číslo zaokrouhlí na celé číslo a poté teprve převede na řetězec. K převodu textu na číslo lze pak použít příkazy **str2num** a **str2double**.



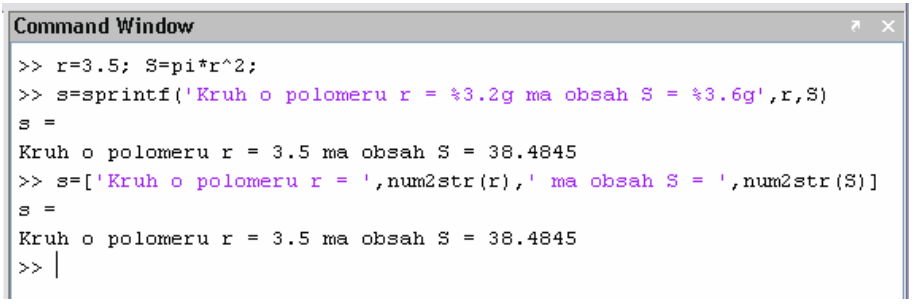
Obr. 2.24: Vzájemné převody mezi čísly a řetězci

Příkaz **str2num** používá při konverzi funkci **eval**. Příkaz **eval** řetězec zpracuje, ale pokud je obsah proměnné nějakou funkcí MATLABu, provede tuto funkci běžným způsobem. Je proto bezpečnější pro převod použít funkci **str2double**. Jinými slovy, funkce **eval** provede obsah řetězce zadaného v argumentu stejným způsobem jako by byl zadán z příkazové řádky nebo v m-skriptu (obr. 2.25).



Obr. 2.25: Použití funkce **eval**

Další možností je použití příkazu **sprintf**, ekvivalentem je zápis pomocí závorek **[ ]** s využitím příkazu **num2str** (obr. 2.26).

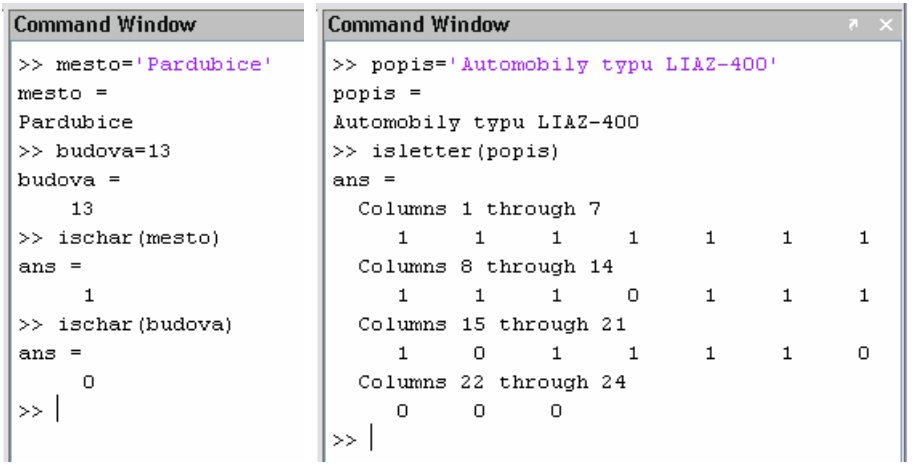


Obr. 2.26: Použití příkazu **sprintf**

Ukázky použití formátování čísel pomocí příkazu **sprintf** jsou uvedeny v tab. 2.11. Mezi další možnosti práce s textovými řetězci patří vytváření polí buněk (tzv. cell arrays), jakýchsi vícerozměrných struktur, zjišťování, zda je proměnná typu text (char) nebo číselná (double) pomocí příkazu **ischar** a zjišťování, na kterých pozicích jsou písmena a na kterých jiné znaky (čísla, mezery, pomlčky, atd.) pomocí příkazu **isletter**.

Tab. 2.11: Formátování při převodu čísel na text

Příkaz	Příklad výsledku
sprintf(‚%.0e‘, pi)	**3e+000**
sprintf(‚%.1e‘, pi)	**3.1e+000**
sprintf(‚%.0f‘, pi)	**3**
sprintf(‚%.1f‘, pi)	**3.1**
sprintf(‚%.0g‘, pi)	**3**
sprintf(‚%.5g‘, pi)	**3.1416**
sprintf(‚%8.5g‘, pi)	** 3.1416**
sprintf(‚%3d‘, 8)	** 8**
sprintf(‚%.3d‘, 8)	**008**



Obr. 2.27: Použití příkazů **ischar** a **isletter**

V tab. 2.12 jsou uvedeny další možné řetězcové funkce, které má uživatel k dispozici. Při práci s řetězci je také možné definovat buňková pole řetězců. Řetězce v jednotlivých buňkách mohou mít libovolnou délku. Většina řetězcových funkcí (např. funkce **char**) pracuje jak s řetězcovými poli, tak s buňkovými poli řetězců. Mezi nejdůležitější příkazy patří **cellstr**, **iscellstr** a **deal**. Popis těchto příkazů by již přesahoval rozsah tohoto textu, příklady použití je možné naleznout v nápovědě, např. zadáním **doc deal**.

Tab. 2.12: Přehled funkcí pro práci s řetězci

Funkce	Popis
deblank(S)	odstraní koncové mezery z řetězce S
ischar(S)	vrací True (1), je-li S textový řetězec
isletter(S)	vrací True (1), je-li S písmeno
isspace(S)	vrací True (1), je-li S bílá mezera
findstr(S1,S2)	vrací místa, kde se kratší řetězec objevuje v delším
strcmp(S1,S2)	porovnává dva řetězce
strncmp(S1,S2,n)	porovnává prvních n znaků dvou řetězců
strjust(S)	zarovnává znakové pole S
strmatch(S1,S2)	vrací indexy řádků S2, které začínají S1
strrep(S1,S2,S3)	nahrazuje všechny výskyty S2 v S1 řetězcem S3
strtok(S1,D)	vrací z S1 znaky, dokud nenajde oddělovač D
lower(S)	převádí S na malá písmena
upper(S)	převádí S na velká písmena

2.8 Buňková pole a struktury

Buňková pole v MATLABu jsou pole, jejichž prvky jsou buňky. Každá buňka v buňkovém poli může obsahovat libovolný datový typ MATLABu, např. číselné pole, text, objekt, jiné buňkové pole či strukturu. Buňkové pole může být vytvořeno přiřazovacím příkazem, používáme složené závorky **{ }**, nebo alokováním pole funkcí **cell**. Příklad vytvoření buňkového pole je na obr. 2.28.

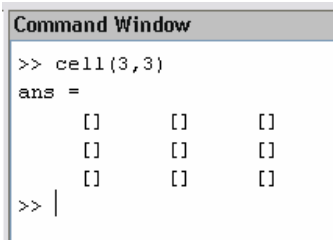
```
Command Window
>> B(1,1)={[1 2 3;4 5 6;7 8 9]}
B =
    [3x3 double]
>> B(1,2)={[inf nan pi eps]}
B =
    [3x3 double]    [1x4 double]
>> B(2,1)={'Textový řetězec'}
B =
          [3x3 double]    [1x4 double]
    'Textový řetězec'              []
>> B(2,2)={eye(3)}
B =
          [3x3 double]    [1x4 double]
    'Textový řetězec'    [3x3 double]
>>
```

Obr. 2.28: Vytvoření buňkového pole

Zobrazení buňkového pole lze provést pomocí **celldisp** (obr. 2.29), prázdnou strukturu např. o rozměru 3×3 vytvoříme příkazem **cell(3,3)**, viz obr. 2.30.

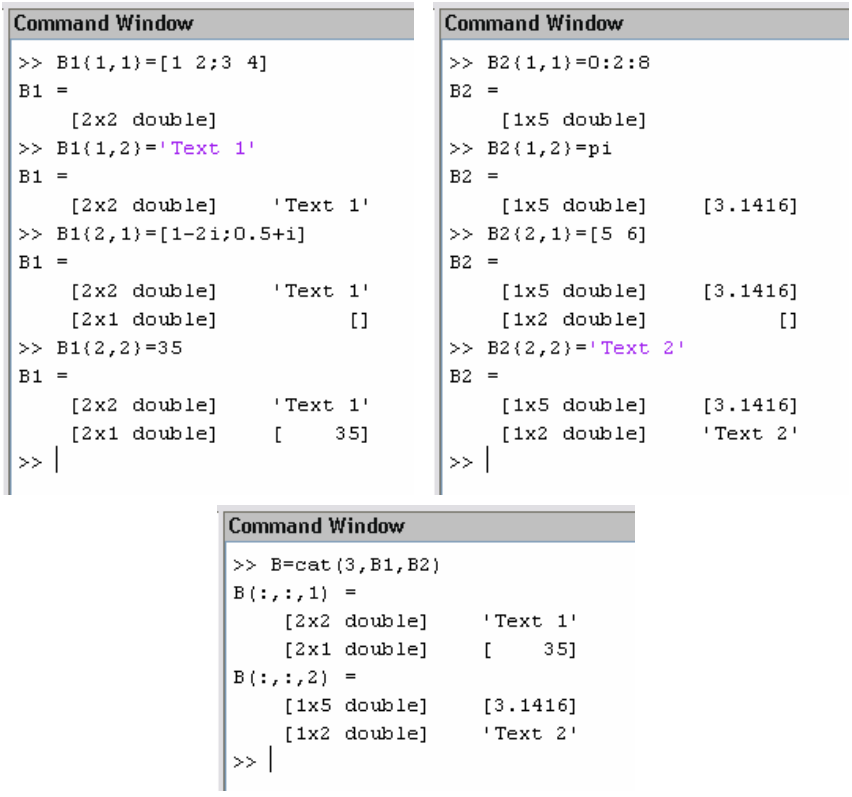
```
Command Window
>> celldisp(B)
B{1,1} =
     1     2     3
     4     5     6
     7     8     9
B{2,1} =
Textový řetězec
B{1,2} =
       Inf       NaN    3.1416    0.0000
B{2,2} =
     1     0     0
     0     1     0
     0     0     1
>>
```

Obr. 2.29: Zobrazení buňkového pole



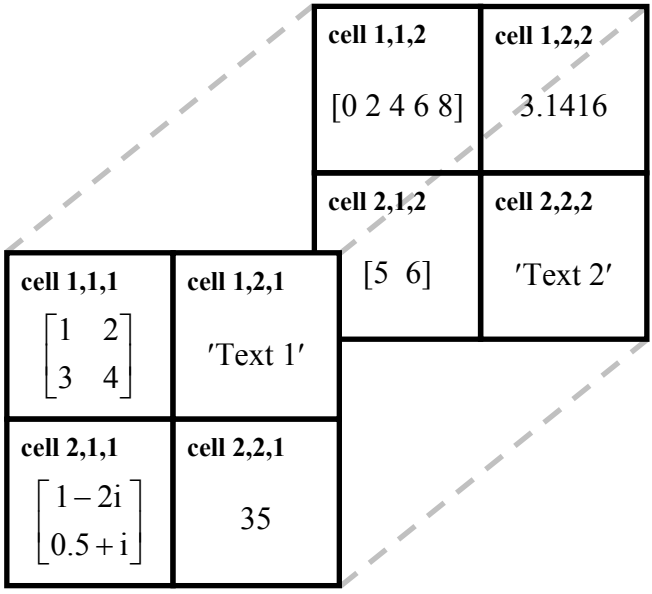
Obr. 2.30: Vytvoření prázdného buňkového pole

Mezi další užitečné příkazy při práci s buňkami a buňkovými poli patří **num2cell**, **iscell**, **iscellstr** a **deal**. Význam a použití prvních dvou je obdobné jako u příkazů v předcházejících odstavcích. Pomocí příkazu **cat** je možné vytvořit i vícerozměrné pole buněk. Ukázkový příklad vytvoření takového pole je na obr. 2.31, jeho struktura pak na obr. 2.32.



Obr. 2.31: Vytvoření vícerozměrného buňkového pole

Struktury jsou podobné buňkovým polím, neboť také dokáží seskupit data různého typu do jediné proměnné. Namísto adresování prvků číslem jsou však prvky struktury adresovány jménem (položkou). Struktura může být vytvořena přiřazovacím příkazem nebo funkcí **struct**. Jednotlivé položky struktury se zpřístupňují buď přímou adresací (přes tečku) nebo pomocí funkce **getfield**.



Obr. 2.32: Schéma vytvořeného vícerozměrného buňkového pole

Na obr. 33 je postup vytváření struktury *student*, která obsahuje položky: *jmeno*, *prijmeni*, *vek*, *bydliste* a *rocnik*. Názvy jednotlivých položek proměnné *student* (za tečkou) mají stejná omezení jako názvy klasických proměnných, tedy mohou obsahovat maximálně 31 znaků a musí začínat písmenem. Stejně tak jsou rozlišována malá a velká písmena. Do vytvořené struktury můžeme přidávat další prvky a lze ji později rozšířit i o další položky (např. o položky nazvané *rocnik* a *prospech*, viz obr. 2.34).

```
Command Window
>> student.jmeno='Jan'
student =
    jmeno: 'Jan'
>> student.prijmeni='Novak'
student =
       jmeno: 'Jan'
    prijmeni: 'Novak'
>> student.vek=17
student =
       jmeno: 'Jan'
    prijmeni: 'Novak'
         vek: 17
>> student.bydliste='Praha'
student =
       jmeno: 'Jan'
    prijmeni: 'Novak'
         vek: 17
    bydliste: 'Praha'
>>
```

```
Command Window
>> student(2).jmeno='Pavel';
>> student(2).prijmeni='Prokop';
>> student(2).vek=18;
>> student(2).bydliste='Brno';
>> student
student =
1x2 struct array with fields:
    jmeno
    prijmeni
    vek
    bydliste
>>
```

```
Command Window
>> student(1)
ans =
       jmeno: 'Jan'
    prijmeni: 'Novak'
         vek: 17
    bydliste: 'Praha'
>> student(2)
ans =
       jmeno: 'Pavel'
    prijmeni: 'Prokop'
         vek: 18
    bydliste: 'Brno'
>>
```

Obr. 2.33: Vytvoření struktury *student* a přidání dalšího prvku

```
Command Window
>> student(1).rocnik=1;
>> student(1).prospech='prospěl';
>> student(2).rocnik=2;
>> student(2).prospech='prospěl s vyznamenáním';
>> student(1)
ans =
       jmeno: 'Jan'
    prijmeni: 'Novak'
         vek: 17
    bydliste: 'Praha'
      rocnik: 1
    prospech: 'prospěl'
>> student(2)
ans =
       jmeno: 'Pavel'
    prijmeni: 'Prokop'
         vek: 18
    bydliste: 'Brno'
      rocnik: 2
    prospech: 'prospěl s vyznamenáním'
>>
```

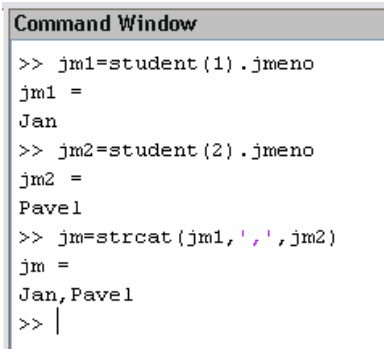
Obr. 2.34: Rozšíření struktury *student* o další položky

Jména prvků struktury lze zjistit příkazem **fieldnames**, což je např. nutné při použití struktury jako parametru volané funkce. Odstranit položku ve struktuře lze příkazem **rmfield** (obr. 2.35). Přiřazení položky struktury (i vícerozměrné) proměnné lze provést způsobem uvedeným na obr. 2.36. Strukturu lze také vytvořit použitím příkazu **struct**. Při práci se strukturou můžeme použít i dynamický přístup (obr. 2.37), případnou indexaci provedeme standardním způsobem probíraným již dříve. Rozměr struktury zjistíme příkazem **size**. Pomocí příkazu **struct** je také možné vnořovat další struktury do již existujících (obr. 2.38).

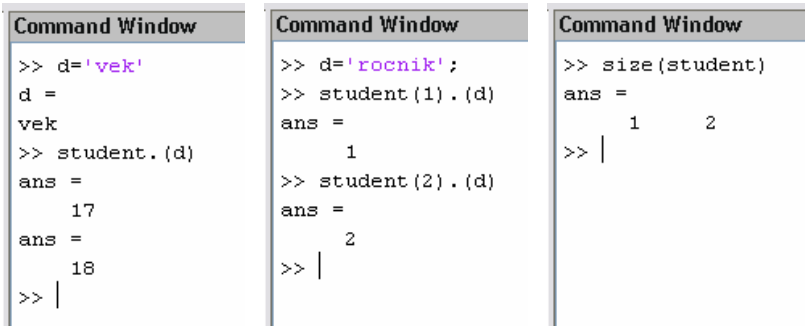
```
Command Window
>> pol=fieldnames(student)
pol =
    'jmeno'
    'prijmeni'
    'vek'
    'bydliste'
    'rocnik'
    'prospech'
>>
```

```
Command Window
>> student1=rmfield(student,pol(6))
student1 =
1x2 struct array with fields:
    jmeno
    prijmeni
    vek
    bydliste
    rocnik
>>
```

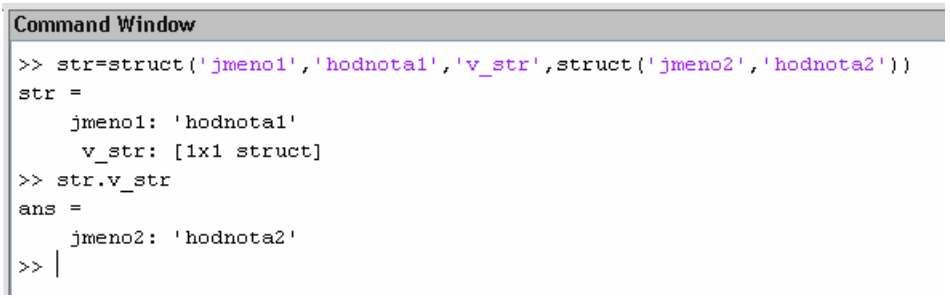
Obr. 2.35: Vypsání názvů položek, odstranění položky ze struktury



Obr. 2.36: Přiřazení položky struktury proměnné



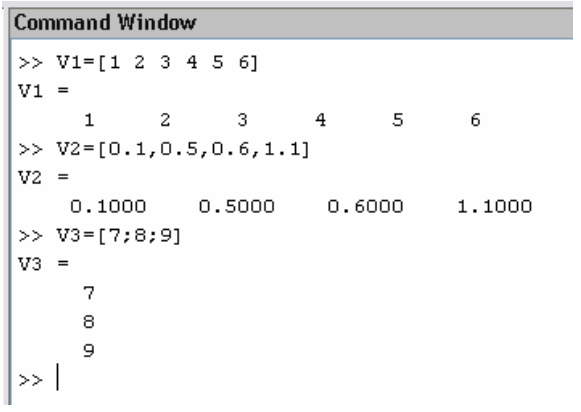
Obr. 2.37: Dynamický přístup k položce struktury



Obr. 2.38: Vytvoření struktury pomocí **struct** a vnoření další

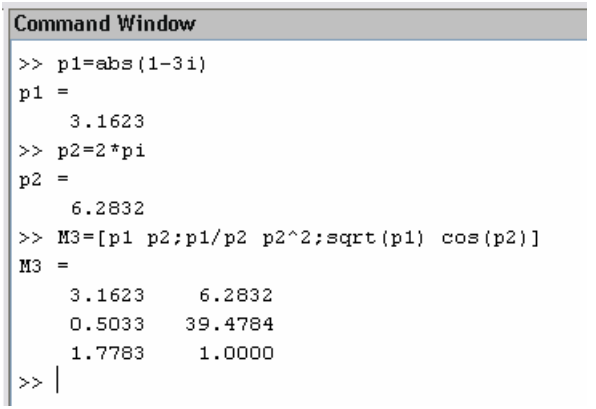
3 Práce s vektory a maticemi

Při seznamování se s prostředím MATLAB je třeba zvláštní pozornost věnovat maticím, na nichž je činnost MATLABu založena. Z tohoto důvodu si v následujícím textu podrobněji ukážeme právě práci s vektory a maticemi. Zejména pak možnosti jejich definice, nejrůznější operace, které může uživatel s vektory a maticemi provádět, a také jejich indexování a princip přístupu k jednotlivým prvkům.



Obr. 3.1: Vytváření vektorů a matic v MATLABu

Jazyk MATLABu neobsahuje žádný příkaz pro nastavení dimenze nebo typu matice. Potřebnou paměť alokuje MATLAB automaticky až do velikosti využitelné na konkrétním počítači. Jednoduchou matici lze zadat výčtem prvků, který je vložen do hranatých závorek. Prvky matice mohou být libovolné výrazy MATLABu. Matice se zapisují po řádcích, které jsou odděleny středníkem. Jednotlivé prvky každého řádku jsou pak odděleny mezerou nebo z důvodu větší přehlednosti mohou být odděleny i čárkou. Několik ukázek definice menších vektorů a matic je na obr. 3.1 a 3.2.

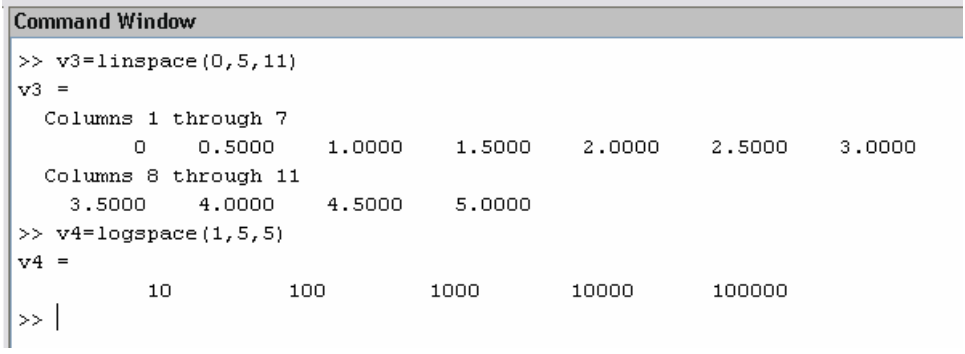


Obr. 3.2: Další možnosti při vytváření matic

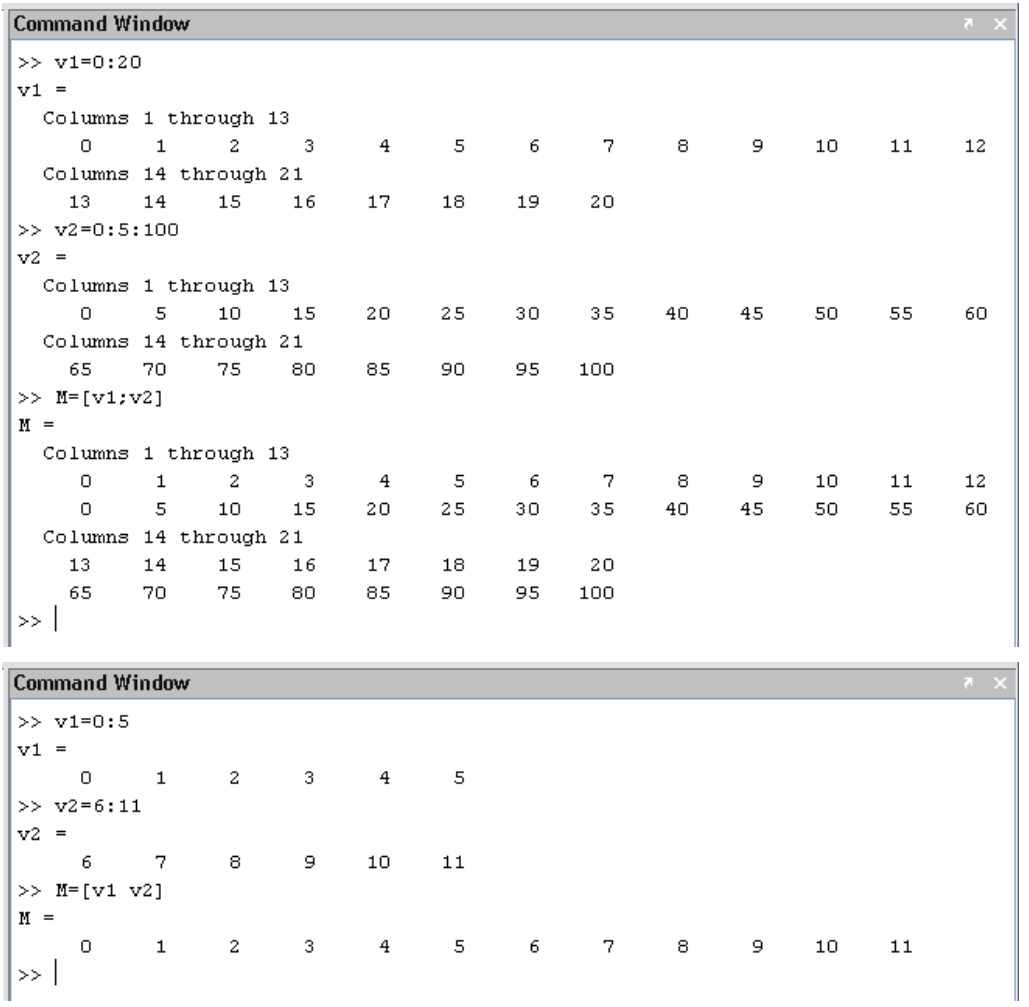
Existují samozřejmě také další možnosti jak vektory či matice definovat. Jednoduchým způsobem lze vytvořit také dlouhé vektory a matice. S výhodou můžeme použít dvojtečku (**:**). Lze použít i příkazy **linspace** a **logspace**. Základní techniky jsou shrnuty v tab. 3.1, některé příklady na obr. 3.3 a 3.4. MATLAB v těchto případech rozdělí výpis vektoru na více řádků.

Tab. 3.1: Základní techniky vytváření vektorů a matic

Technika sestavení	Popis
x = [2 2*pi sqrt(2) 2–3j]	Vytvoří řádkový vektor **x** ze zvolených prvků.
x = X1:X2	Vytvoří řádkový vektor **x** od **X1** do **X2** s krokem **1**
x = X1:K:X2	Vytvoří řádkový vektor **x** od **X1** do **X2** s krokem **K**
x = linspace(X1,X2,N)	Vytvoří řádkový vektor **x** od **X1** do **X2** mající **N** prvků
x = logspace(X1,X2,N)	Vytvoří řádkový vektor **x** s logaritmickým rozložením od $\mathbf{10^{X1}}$ do $\mathbf{10^{X2}}$ mající **N** prvků



Obr. 3.3: Vytváření dlouhých vektorů – příkazy **linspace** a **logspace**



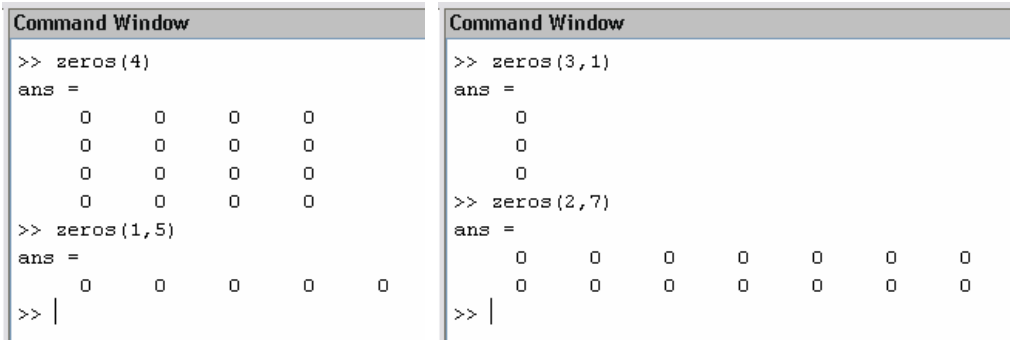
Obr. 3.4: Vytváření dlouhých vektorů a sdružování do matic

3.1 Speciální typy vektorů a matic

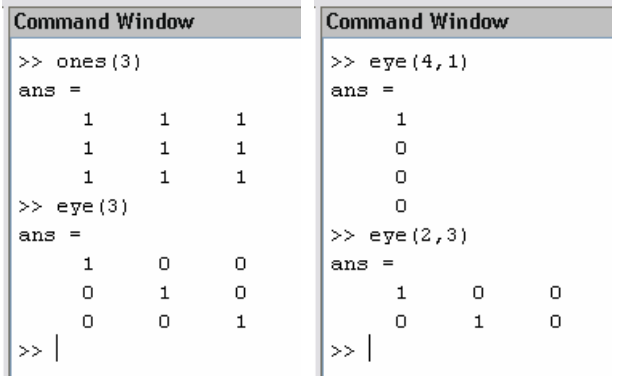
V MATLABu je také možné využít speciální typy matic. Pomocí příkazů **zeros a ones** lze definovat vektor nebo matici samých nul resp. samých jedniček. Pomocí **eye** lze definovat jednotkovou matici. Možnosti definice takových matic jsou na obr. 3.5 a 3.6.

Mezi další pak patří příkaz **magic**, který vytvoří tzv. magický čtverec. Jde o čtvercovou matici, u které je součet každého řádku, každého sloupce a hlavní diagonály stejný (obr. 3.7). Zajímavý typ matice vznikne použitím příkazu **pascal**. Výsledná matice je symetrická pozitivně definitní matice tvořená prvky Pascalova trojúhelníka (obr. 3.7). Pascalovu matici lze

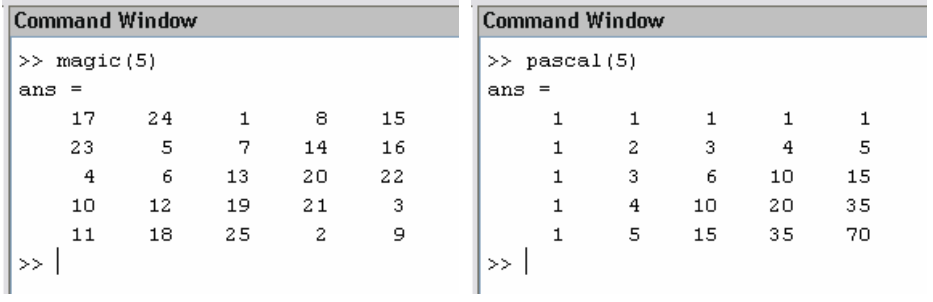
modifikovat a vytvořit tak speciální symetrické matice o velikosti **n**, zapíšeme **pascal(n,1)** nebo **pascal(n,2)**. První z uvedených vytvoří dolní trojúhelníkovou matici, jenž vznikne na základě Choleského faktorizace Pascalovy matice. Druhým případem je horní trojúhelníková matice (symetrie je ale podle vedlejší diagonály), která vnikne permutací prvků a transpozicí matice **pascal(n,1)**. Výsledkem je tedy, na rozdíl od **zeros**, **ones** a **eye**, vždy symetrická matice. Tvorba takovýchto matic již přesahuje rozsah tohoto textu. Pro případné další studium je možné použít nápovědy – **doc pascal** a **doc chol**. Transpozice matic a vektorů bude probrána později.



Obr. 3.5: Speciální typy matic – **zeros**

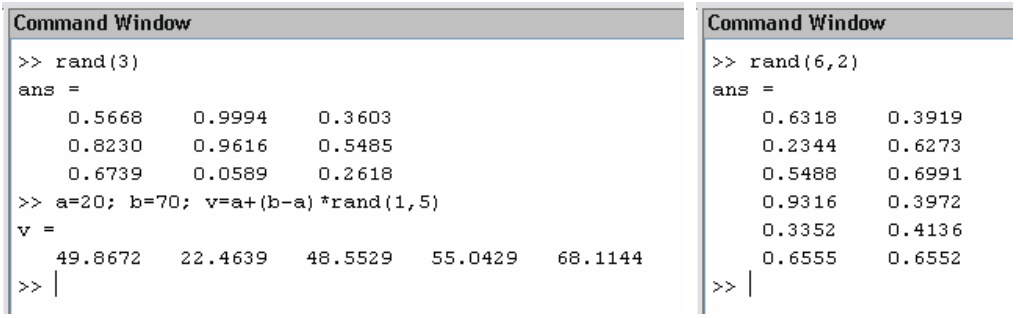


Obr. 3.6: Speciální typy matic – **ones** a **eye**

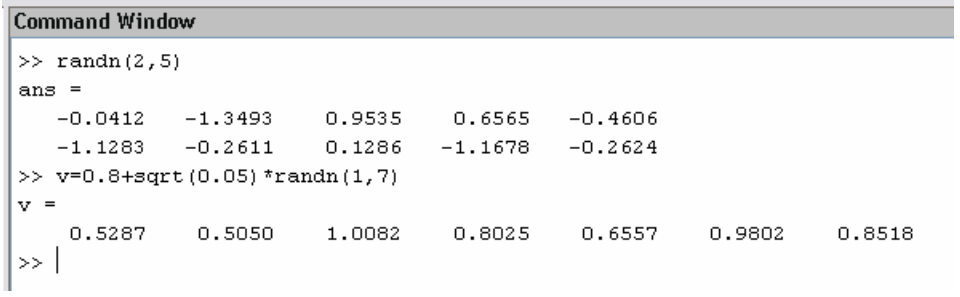


Obr. 3.7: Speciální typy matic – **magic** a **pascal**

Uživatel může také jednoduše definovat vektory či matice náhodných čísel. K tomu slouží příkazy **rand** a **randn** (obr. 3.8 a 3.9). Příkaz **rand** vytvoří matici, jejíž jednotlivé prvky budou pseudonáhodná čísla z intervalu (0, 1) s rovnoměrným rozdělením. Pokud potřebujeme generovat čísla tak, aby ležely ve zvoleném intervalu (a, b), zapíšeme posloupnost příkazů: **a = 5; b = 40; v = a + (b – a) * rand(1,5)**. Příkaz **randn** pak generuje matici definovaného rozměru s jednotlivými prvky tvořenými pseudonáhodnými čísly s normálním rozdělením bez omezení rozsahu. Pro generování čísel s jinou střední hodnotou a rozptylem lze postupovat např. takto: **v = 0.8 + sqrt(0.05) * randn(1,5)**. Střední hodnota je pak 0,8 a rozptyl 0,05.

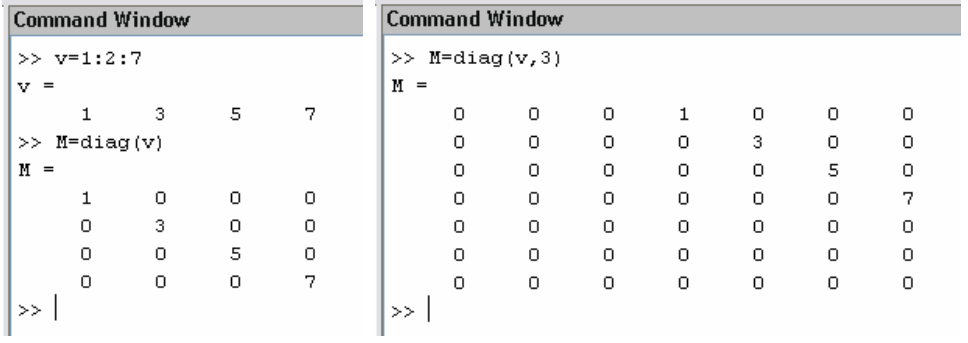


Obr. 3.8: Matice náhodných čísel – **rand**

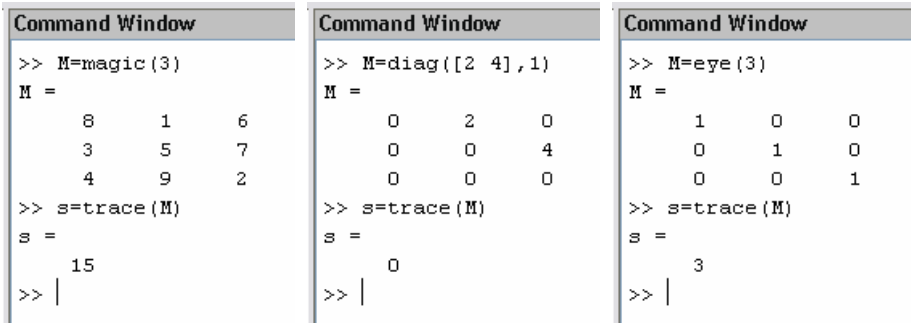


Obr. 3.9: Matice náhodných čísel – **randn**

Velmi užitečný je i příkaz **diag**, pomocí něhož lze obsadit nejen hlavní diagonálu matice vektorem (obr. 3.10). Příkaz **trace** sečte prvky na hlavní diagonále (obr. 3.11).

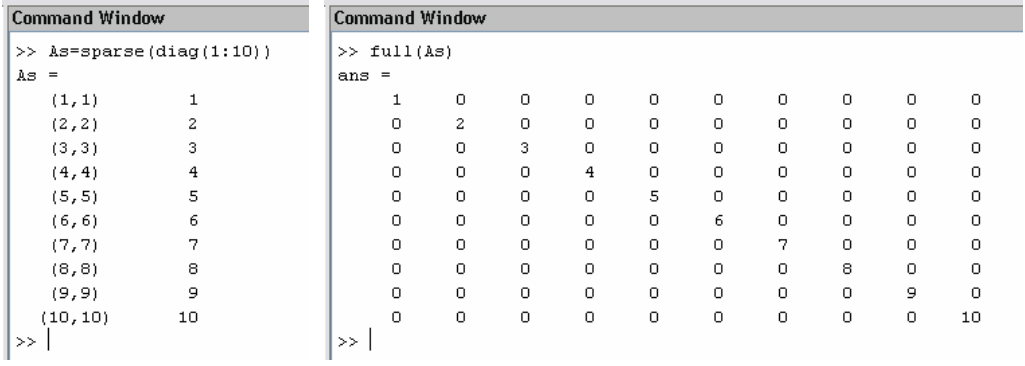


Obr. 3.10: Obsazení diagonály definovaným vektorem

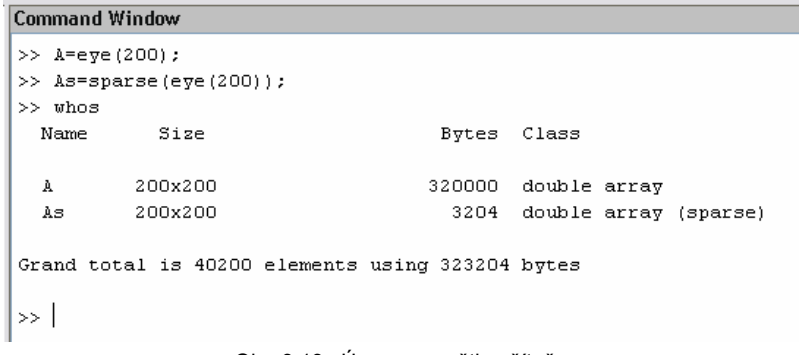


Obr. 3.11: Součet prvků na hlavní diagonále

V MATLABu může uživatel efektivně definovat i tzv. řídké matice. Jsou to velké matice obsahující jen malé množství nenulových prvků. Uložení celé matice zatěžuje paměť počítače. Řešením je uložení pouze nenulových prvků takových matic. Řídké matice jsou ukládány příkazem **sparse**. Na obr. 3.12 je výpis řídké matice vytvořené pomocí tohoto příkazu a také výpis celé matice. Na obr. 3.13 je demonstrováno využití paměti počítače, je-li řídká matice o rozměru 200×200 (v tomto případě jednotková matice) vytvořena běžným způsobem a nebo pomocí příkazu **sparse**. Porovnání je provedeno pomocí výpisu obou uložených proměnných příkazem **whos**.



Obr. 3.12: Uložení řídkých matic



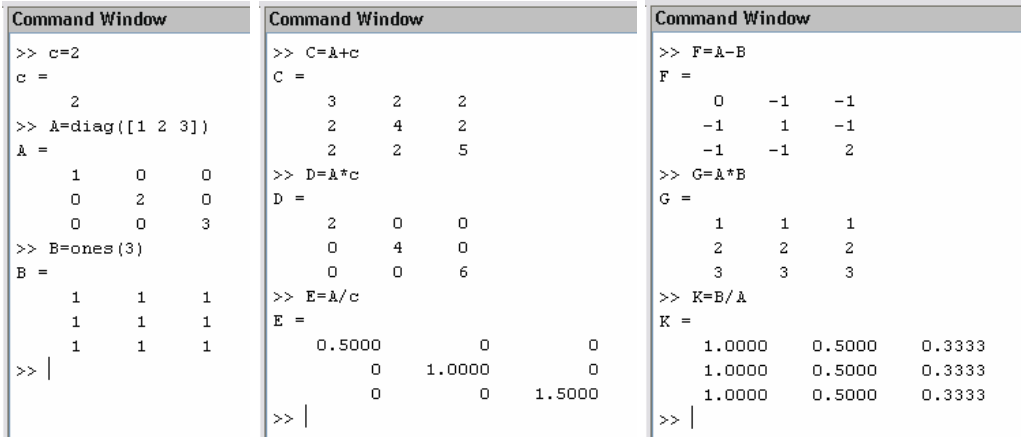
Obr. 3.13: Úspora paměti počítače

3.2 Základní operace s maticemi

Operace s vektory či maticemi se v MATLABu zapisují velice jednoduchým způsobem. Pro operace mezi dvěma nebo více maticemi se používají operátory uvedené v tab. 3.2. Dále lze využít i vnitřní funkce MATLABu pro operace nad jednou maticí. Množství těchto funkcí a operátorů je poměrně velké. Jejich kompletní seznam získáme pomocí **help slash** a **help matfun**. Některé ukázky jednoduchých operací s maticemi jsou na obr. 3.14 a 3.15.

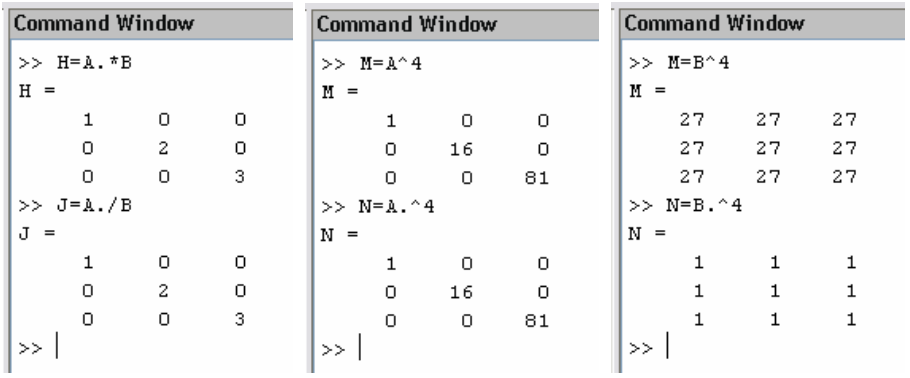
Tab. 3.2: Základní operace s maticemi

Operace	Popis
Sčítání matice a skaláru	**C** = **A** + c, ke každému prvku je přičten skalár c
Násobení matice a skaláru	**D** = **A** * c, každý prvek je vynásoben skalárem c
Dělení matice skalárem	**E** = **A** / c, každý prvek je vydělen skalárem c
Sčítání resp. odčítání matic	**F** = **A** ± **B**, součet resp. rozdíl stejnolehlých prvků
Násobení matic	**G** = **A** * **B**, klasické násobení matic
Prvkové násobení matic	**H** = **A** .* **B**, násobení stejnolehlých prvků
Dělení matic zprava	**I** = **A** / **B**, pravé dělení matic, platí **A** / **B** = **A** $\mathbf{B}^{-1}$
Prvkové dělení zprava	**J** = **A** ./ **B**, podíl stejnolehlých prvků matic **A** a **B**
Dělení matic zleva	**K** = **A** \ **B**, levé dělení matic, platí **A** \ **B** = $\mathbf{A}^{-1}$ **B**
Prvkové dělení zprava	**L** = **A** .\ **B**, podíl stejnolehlých prvků matic **B** a **A**
Mocnění matice	**M** = **A**^n, n-tá mocnina (pouze čtvercové matice)
Prvkové mocnění matice	**N** = **A** .^n, n-tá mocnina každého prvku matice

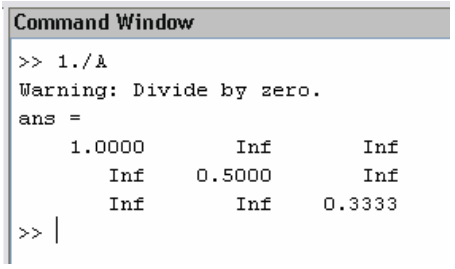


Obr. 3.14: Základní operace s maticemi

Při zápisu vlastního výrazu, je třeba uvažovat rozměr matice. Některé operace není možné provádět vždy, např. výraz 1 / **B** není možné z logických důvodů provést. Je ale možné, provést dělení prvkové, tedy 1 **.** / **B**. Výpočet výrazu 1 **.** / **A**, ale také není zcela v pořádku, dochází při něm k dělení nulou. MATLAB jej ale provede, vypíše varovné hlášení a u prvků, kde došlo ke zmiňovanému dělení nulou, je výsledek **inf** (nekonečno). Situace je na obr. 3.16.

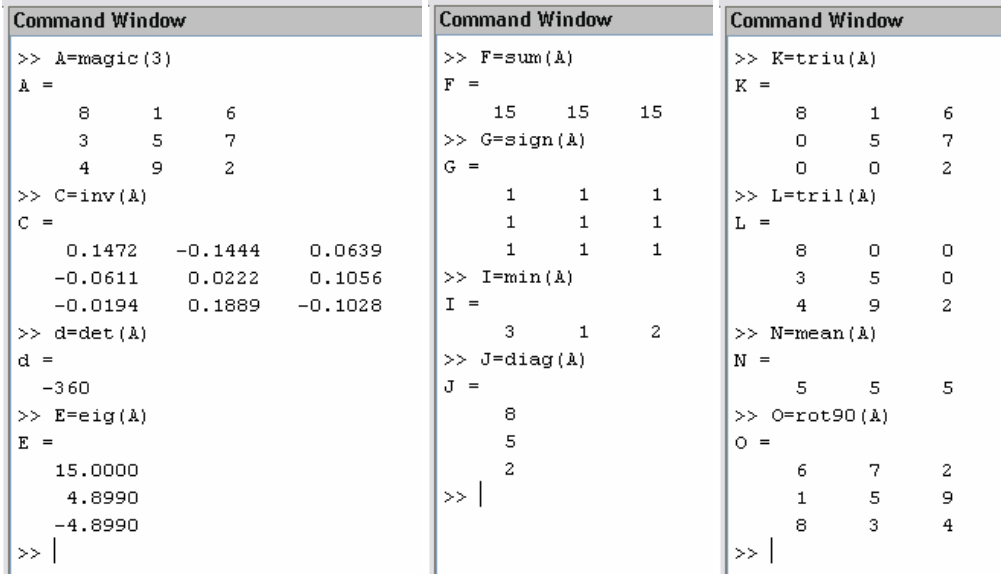


Obr. 3.15: Prvkové operace a mocnění matic



Obr. 3.16: Varovné hlášení (warning) při dělení nulou

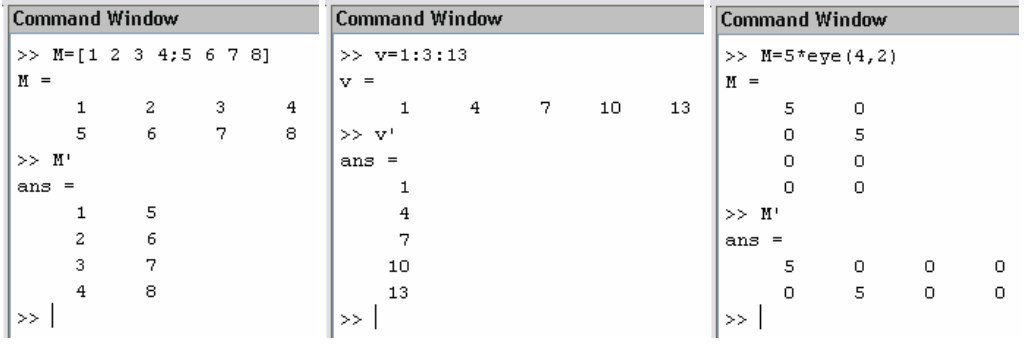
V MATLABu je k dispozici i celá řada tzv. maticových funkcí. Přehled nejpoužívanějších funkcí je v tab. 3.3. Příklady použití některých uvedených funkcí jsou na obr. 3.17 až 3.19.



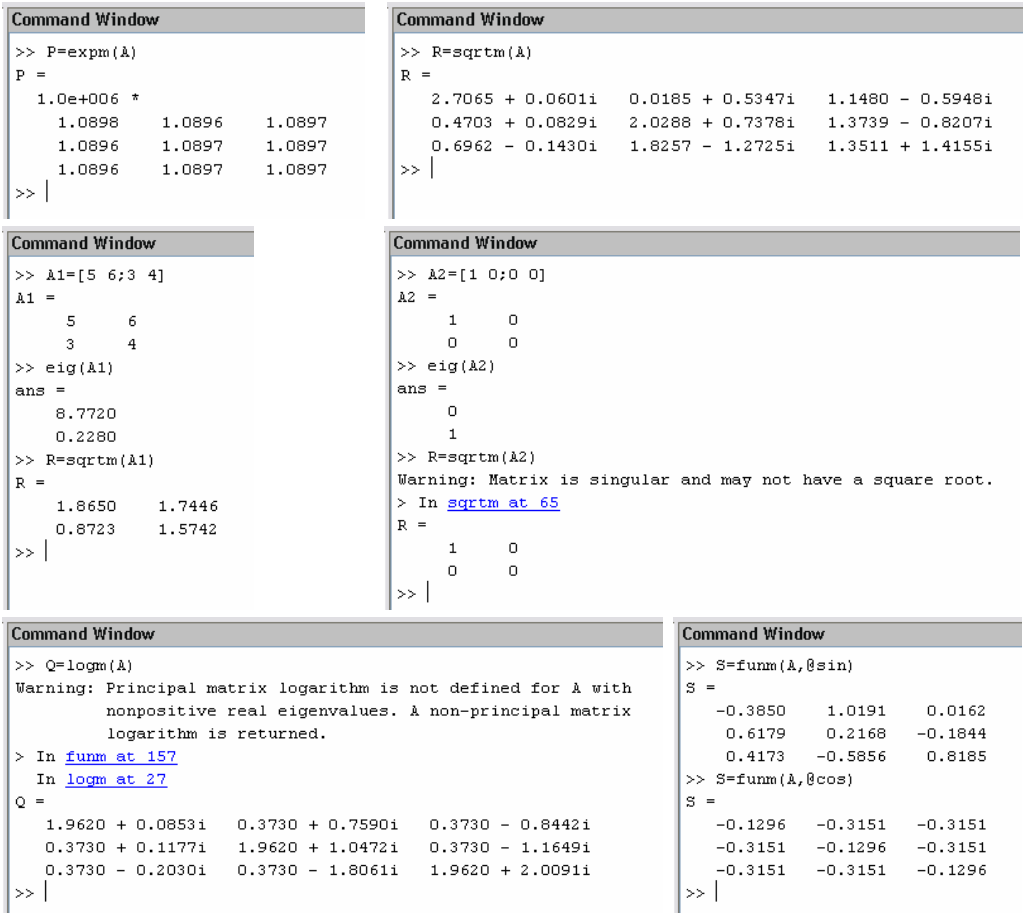
Obr. 3.17: Některé maticové funkce

Tab. 3.3: Maticové funkce

Operace	Popis
B = A′	transpozice vektoru či matice
C = inv(A)	inverze čtvercové matice
d = det(A)	determinant čtvercové matice
E = eig(A)	vrací vektor vlastních čísel čtvercové matice
F = sum(A)	u matic vrací vektor, jehož prvky jsou součtem sloupců matice, u vektoru vrací součet jednotlivých prvků
G = sign(A)	vrací matici stejného řádu, ve které každý prvek je definován: 1 … je-li prvek původní matice kladný, 0 … je-li prvek původní matice nulový, –1 … je-li prvek původní matice záporný.
H = max(A)	u matic vrací vektor maximální hodnot každého sloupce matice, u vektoru vrací přímo největší prvek
I = min(A)	u matic vrací vektor minimálních hodnot každého sloupce matice, u vektoru vrací přímo nemenší prvek
J = diag(A)	vrací vektor prvků, které jsou na hlavní diagonále
K = triu(A)	vrací horní trojúhelníkovou matici z původní matice
L = tril(A)	vrací dolní trojúhelníkovou matici z původní matice
m = isempty(A)	vrací 0, je-li matice neprázdná nebo 1, je-li prázdná
N = mean(A)	u matic vrací vektor, jehož prvky jsou aritmetickým průměrem sloupců matice, u vektorů vrací průměr jejich prvků
O = rot90(A)	rotace matice o 90°
P = expm(A)	exponenciální funkce prvků čtvercové matice
Q = logm(A)	logaritmus prvků čtvercové matice
R = sqrtm(A)	odmocnina prvků čtvercové matice
S = funm(A)	definovaná funkce čtvercové matice (více **doc funm**), např.: **funm(A,@sin)** … sinus, **funm(A,@cos)** … kosinus, **funm(A,@sinh)**, **funm(A,@cosh)** ... hyperbol. sinus a kosinus



Obr. 3.18: Transpozice vektorů a matic



Obr. 3.19: Další maticové funkce

Na obr. 3.19 jsou uvedeny příklady použití speciálních maticových funkcí. Je zřejmé, že tyto funkce nelze aplikovat na jakékoliv matice. MATLAB sice ve všech případech dané funkce provedl, v některých případech došlo ale k vypsání varovného hlášení. Např. v prvním případě je matice **A2** singulární (tj. **det(A2) = 0**). V druhém případě není matice **A** pozitivně definitní, resp. není ani pozitivně semidefintní, jelikož má i záporná vlastní čísla (viz obr. 3.18). Funkce **logm** v tomto případě provede výpočet na základě jiného algoritmu, označeného jako „non-principal“. Více se uživatel opět dozví v rozšířené verzi helpu, zadáním **doc** <*funkce*>.

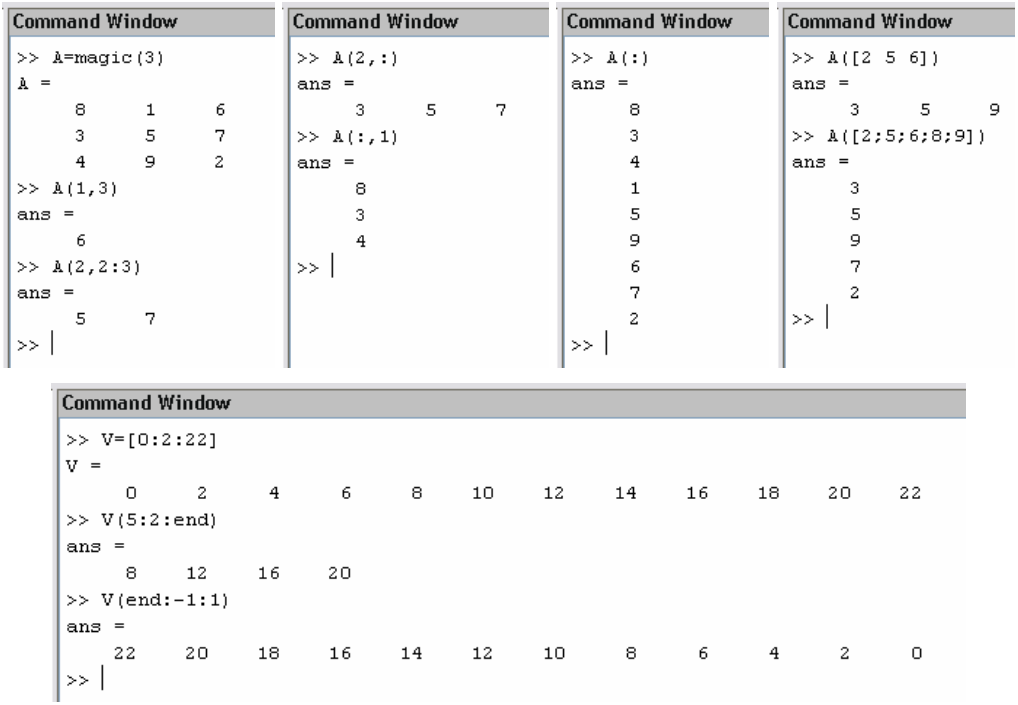
3.3 Indexovaní a třídění vektorů a matic

Důležité je také indexování vektorů a matic resp. přístup k jejich jednotlivým prvkům. MATLAB nabízí několik možností (tab. 3.4). K indexování se využívají kulaté závorky, s výhodou využíváme dvojtečky. Odkaz na příslušný prvek lze provést zadáním jeho pozice ve vektoru nebo řádky a sloupce u matic.

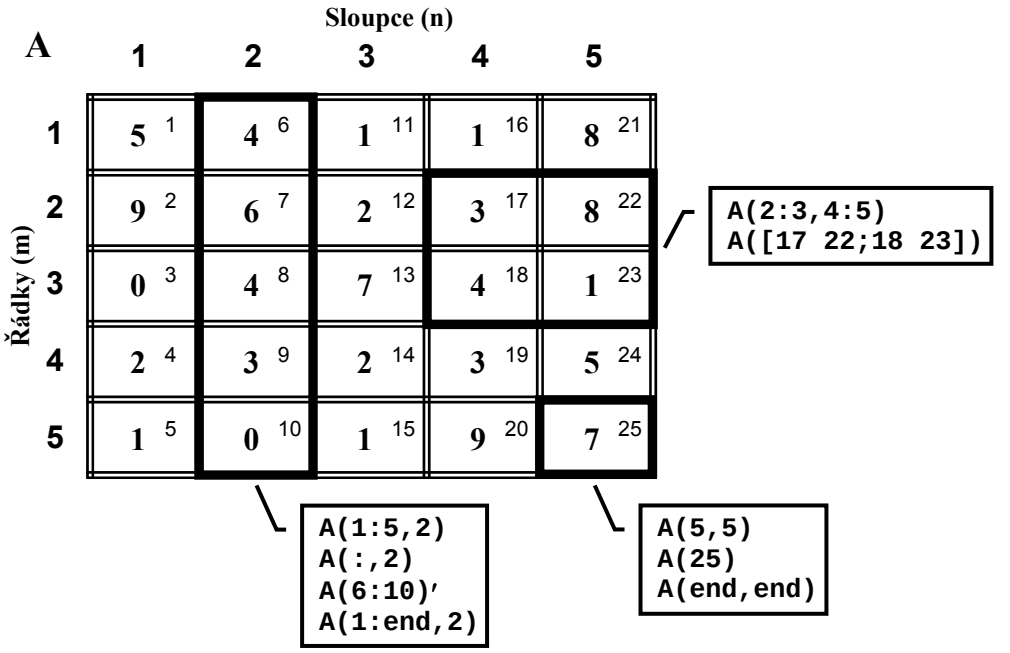
Tab. 3.4: Přístup k prvkům matic

Adresace prvku	Popis
A(r,c)	adresace prvků matice **A** definovaná řádkovým indexem nebo vektorem řádkových indexů a sloupcovým indexem či vektorem sloupcových indexů
A(r,:)	adresace prvků matice **A** definovaná vektorem řádkových indexů **r** a všemi sloupci
A(:,c)	adresace prvků matice **A** definovaná vektorem sloupcových indexů **c** a všemi řádky
A(:)	adresace všech prvků matice **A**, výpis ve formě sloupcového vektoru; pokud se **A(:)** objeví na levé straně přiřazovacího znaménka, znamená to vyplnění pole **A** prvky z pravé strany přiřazovacího znaménka bez změny jeho tvaru
A(I)	adresace prvků matice **A** definovaná jednoduchým indexovým vektorem **I**, jako kdyby **A** byla sloupcovým vektorem **A(:)**

Indexy musí být vždy celočíselné hodnoty a mohou v souladu s tab. 3.4 popisovat např. jeden prvek nebo i souvislý rozsah prvků definovaný pomocí dvojtečky, tj. **min:krok:max**. Příklady použití jsou na obr. 3.20, možné ekvivalentní metody indexace jsou na obr. 3.21.

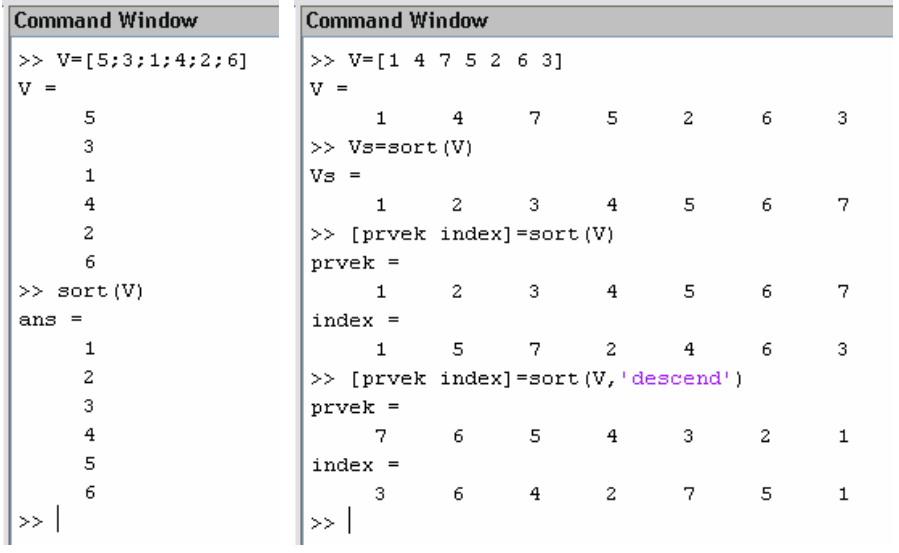


Obr. 3.20: Indexování vektorů a matic

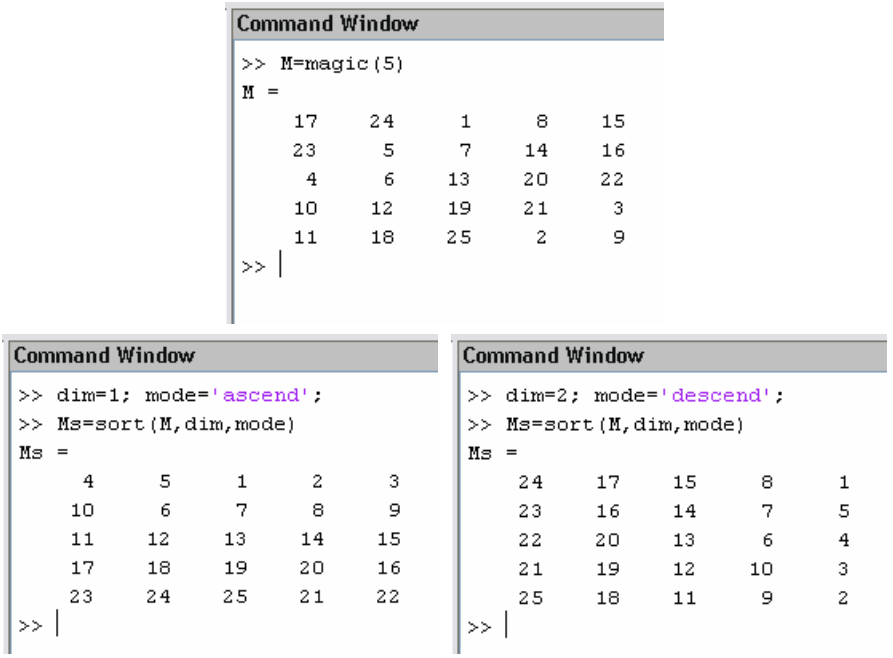


Obr. 3.21: Navzájem ekvivalentní metody indexování

V MATLABu je možné také provádět jednoduché třídění vektorů či matic. K tomuto účelu slouží příkaz **sort**. Existuje několik variant použití tohoto příkazu, ilustrační příklady jsou uvedeny na obr. 3.22 a 3.23.



Obr. 3.22: Třídění vektorů



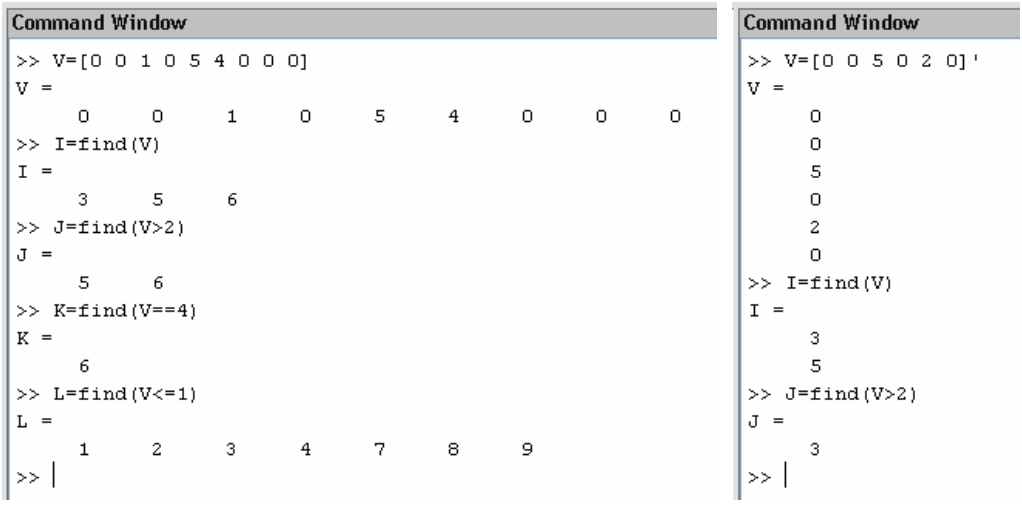
Obr. 3.23: Třídění matic

Uživatel např. může provést vzestupné třídění prvků, které je nastaveno implicitně pomocí parametru *ascend*, příp. i vypsat index jednotlivých prvků, a nebo provést sestupné třídění změnou parametru na *descend*. U matic je třeba také zvolit dimenzi, ve které chceme třídění provádět.

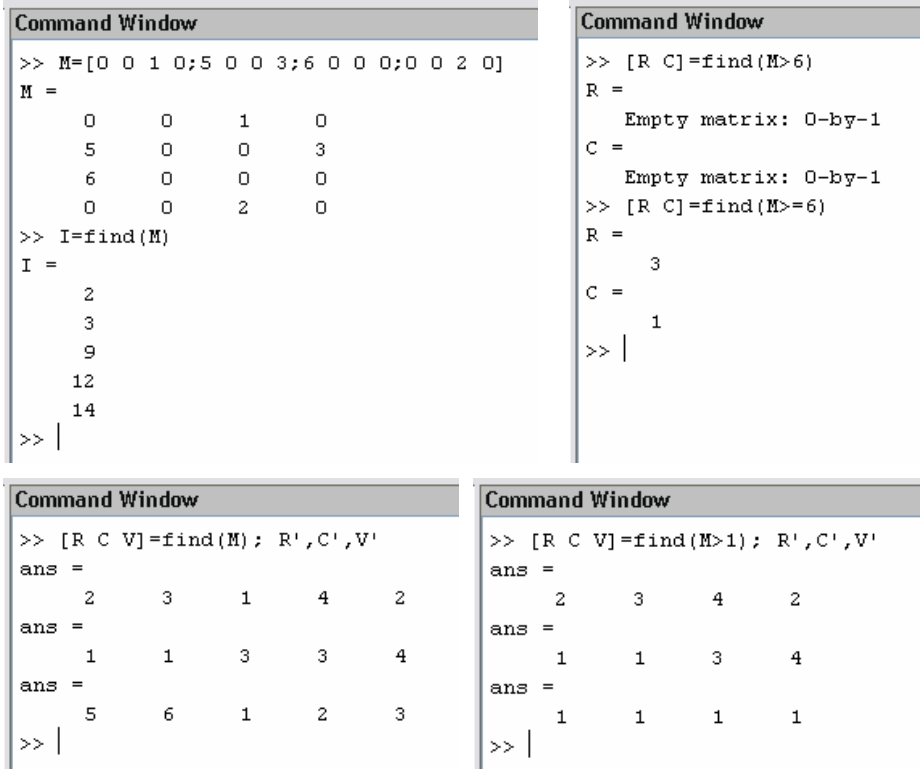
Tab. 3.5: Vyhledávání prvků

Hledání prvku	Popis
I = find(A)	vrací řádkový nebo sloupcový vektor (dle rozměru vektoru či matice **A**) indexů nenulových prvků
J = find(A<p)	vrací vektor prvků matice **A**, které splňují podmínku p
[R C] = find(A)	vrací vektor **R** řádkových a vektor **C** sloupcových indexů prvků matice **A**, které jsou nenulové
[R C V] = find(A)	vrací navíc vektor nenulových prvků **V**
[R C] = find(A>p)	vrací vektor **R** řádkových a vektor **C** sloupcových indexů prvků matice **A**, které splňují podmínku p
[R C V] = find(A>p)	vrací navíc vektor logických hodnot **V**, kde značí: 0 – index je nulový 1 – index není nulový

K vyhledávání jednotlivých prvků či jejich skupiny lze použít funkci **find**, varianty použití jsou v tab. 3.5, ukázkové příklady jsou na obr. 3.24 a 3.25.



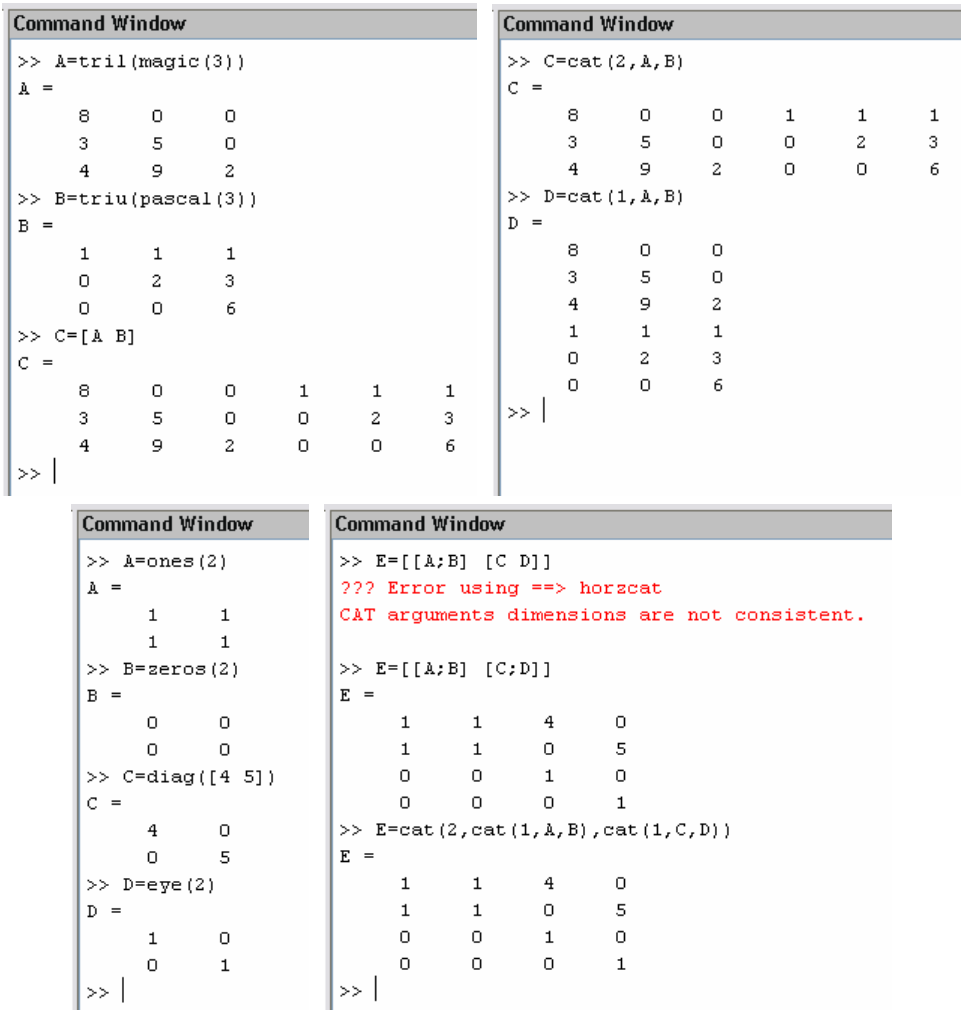
Obr. 3.24: Vyhledávání prvků ve vektoru



Obr. 3.25: Vyhledávání prvků v matici

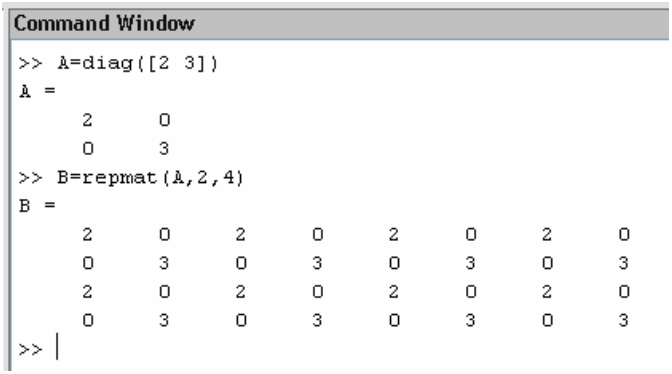
3.4 Slučování a změna tvaru matic

Slučováni matic lze provádět jednoduchým způsobem pomocí hranatých závorek. Jinými slovy, výsledná matice je složena ze submatic. Nutností je samozřejmě respektování rozměrů jednotlivých submatic. Další možností je použití příkazu **cat**. Např. **[A B]** je ekvivalentní zápisu **cat(2,A,B)** a **[A;B]** je pak ekvivalentní **cat(1,A,B)**, viz obr. 3.26.

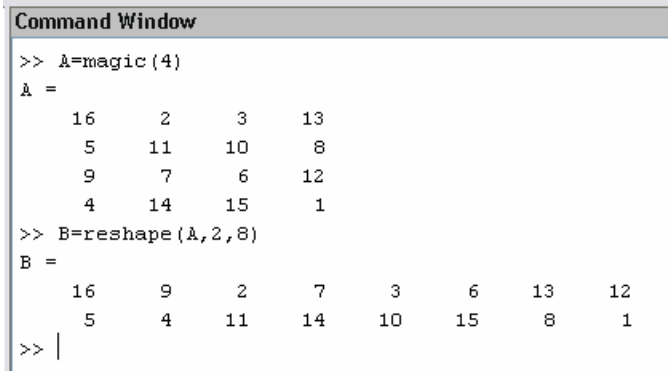


Obr. 3.26: Slučování matic

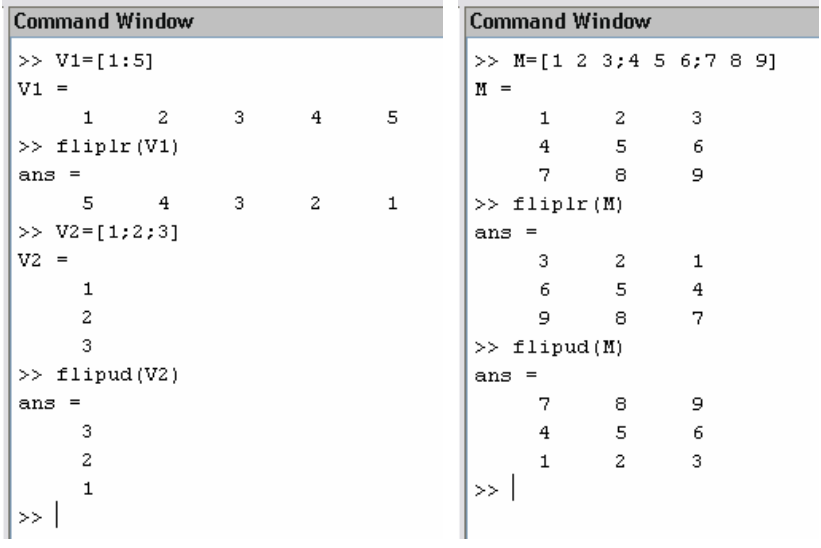
MATLAB umožňuje mimo jiné také vytvoření matice kopírováním submatice do předem zadané struktury pomocí **repmat** (obr. 3.27). Užitečnou funkcí je také změna rozměru matice pomocí **reshape** (obr. 3.28). K přerovnání prvků vektoru nebo matice lze využít příkazy **fliplr** a **flipud**. V prvním případě jsou prvky přerovnány zleva doprava, ve druhém pak odshora dolů (obr. 3.29).



Obr. 3.27: Vytvoření matice pomocí **repmat**



Obr. 3.28: Změna rozměru matice

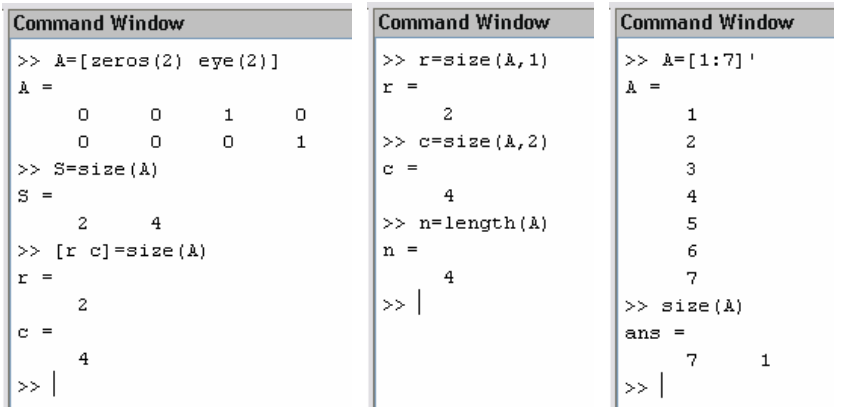


Obr. 3.29: Přerovnávání prvků vektorů a matic

K zjištění rozměru vektoru nebo matice lze standardním způsobem využít příkazy **length** a **size**, podrobnosti viz tab. 3.6, příklady použití na obr. 3.30.

Tab. 3.6: Zjišťování rozměru vektorů a matic

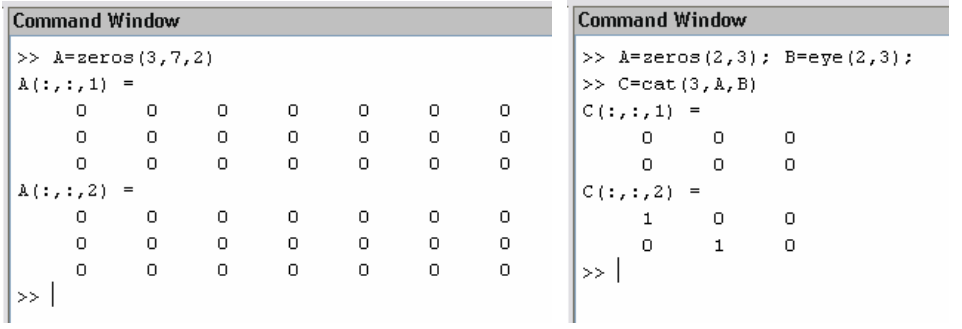
Dotaz na rozměr	Popis
S = size(A)	vrací řádkový vektor **S**, jehož první prvek je počet řádek matice **A** a druhý prvek je počet sloupců **A**
[r,c] = size(A)	vrací dva skaláry r a c, které obsahují počet řádek a počet sloupců matice **A**
r = size(A,1)	vrací skalár r odpovídající počtu řádek matice **A**
c = size(A,2)	vrací skalár c odpovídající počtu sloupců matice **A**
n = length(A)	vrací skalár n odpovídající **max(size(A))**



Obr. 3.30: Rozměr vektoru a matice

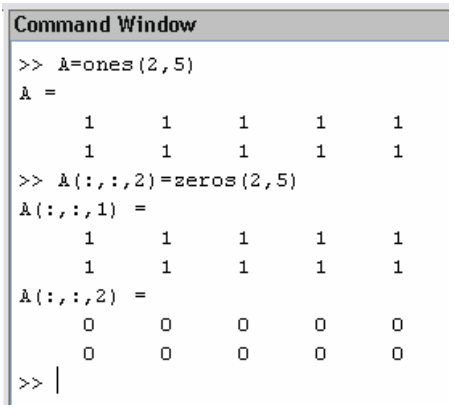
3.5 Vícerozměrná pole

V prostředí MATLAB je také možné jednoduše vytvářet vícerozměrná pole. Počet dimenzí vícerozměrného pole není omezen.



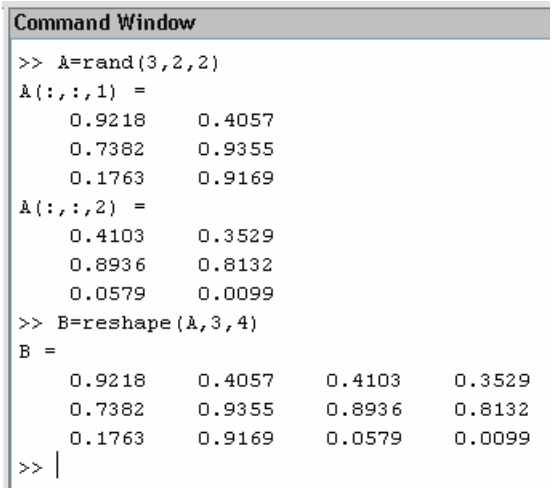
Obr. 3.31: Vytvoření vícerozměrného pole

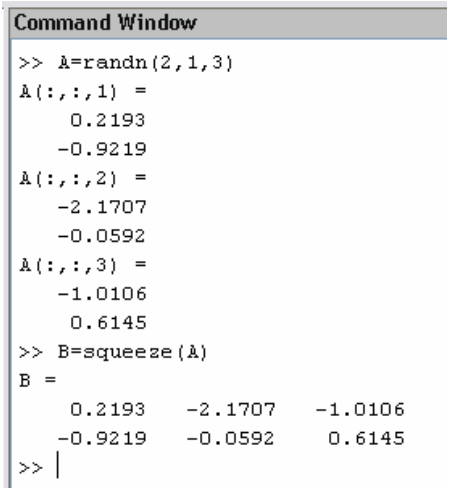
Vícerozměrná pole lze vytvářet pomocí příkazů **zeros**, **ones**, **rand**, **randn**, tedy obdobně jako v kapitole 3.1, nebo pomocí příkazů **cat** a **repmat**. Na obr. 3.31 je ukázka vytvoření takového pole; pro větší představivost o rozměru 3D.



Obr. 3.32: Rozšíření 2D matice na 3D pole

Je také možné rozšířit již vytvořené pole o další rozměr. Na obr. 3.32 je ukázka rozšíření matice (tedy vlastně 2D pole) na třírozměrné (3D) pole. Pomocí příkazu **reshape** lze 3D pole převést na 2D matici, viz obr. 3.33. Další možností je příkaz **squeeze**.





Obr. 3.33: Převedení 3D pole na 2D matici

Mezi další související příkazy patří např. **flipdim** (ekvivalent **flipud** a **fliplr** pro práci s vícerozměrnými poli), **shiftdim**, **sub2ind** a **ind2sub**. Použití těchto příkazů ale již přesahuje rozsah tohoto textu, podrobnější popis a příklady použití nalezne uživatel v nápovědě (**help**, **doc**). Velikost vícerozměrných polí zjistíme opět pomocí příkazu **size** nebo také pomocí příkazu **ndims**, který odpovídá zápisu **length(size(A))** a vypíše pouze počet dimenzí pole.

3.6 Řešení soustav lineárních algebraických rovnic

Detailní rozbor problematiky řešení soustav lineárních algebraických rovnic by přesahoval rozsah tohoto textu. Uživatel vybavený základními znalostmi práce v prostředí MATLAB získanými v předchozích kapitolách, může ale relativně jednoduše soustavy takových rovnic řešit. V prostředí MATLAB, a obecně i v jiných programových prostředcích podobného typu, mluvíme o tzv. numerickém řešení. Koeficienty rovnic jsou k dispozici ve formě vektorů a matic. Aby bylo soustava jednoznačně řešitelná, musí být matice soustavy regulární (tj. mít nenulový determinant). Určité problémy mohou nastat v případě, kdy je matice koeficientů řídká, tj. obsahuje jen malý počet nenulových prvků. Problematika definování řídkých matic a práce s nimi byla podrobněji probírána v odstavci 3.1.

Metody řešení soustavy lineárních algebraických rovnic můžeme rozdělit na metody přímé a iterační. **Přímé metody** jsou takové metody, které dávají v konečném počtu kroků přesné řešení za předpokladu, že výpočet probíhá bez zaokrouhlovacích chyb, tedy zcela přesně. Základní přímou metodou řešení soustav lineárních algebraických rovnic je Gaussova eliminační metoda. V tzv. přímém chodu se matice soustavy převede do horního trojúhelníkového tvaru. Ve zpětném chodu se pak řeší soustava s touto maticí; výsledné řešení dostaneme zpětným dosazováním již vypočtených složek. Řešení přímou metodou tedy obecně dostáváme transformací původní soustavy na soustavu v jednodušším tvaru. Tuto transformaci označujeme jako faktorizaci matice soustavy. Nejznámější je tzv. LU rozklad, Choleského faktorizace a další. Hlavní výhodou přímých metod je jejich konečný charakter, tzn. k přesnému řešení soustavy dojdeme po konečném počtu kroků.

Naopak u **iteračních metod** je výsledkem pouze aproximace řešení, které je dosaženo po určitém počtu iterací. Mezi iterační metody patří např. Jacobiho metoda, Gaussova-Seidelova metoda, metody relaxační, apod.

Dále budeme hledat řešení soustavy lineárních algebraických rovnic ve tvaru $\mathbf{A}\,\mathbf{x} = \mathbf{B}$, kde $\mathbf{x}$ je vektor hledaného řešení, $\mathbf{b}$ je vektor pravých stran a $\mathbf{A}$ je matice soustavy. Budeme uvažovat soustavu n rovnic s n neznámými; matice soustavy $\mathbf{A}$ je čtvercová. Řešení hledáme ve tvaru $\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b}$ nebo také $\mathbf{x} = \mathbf{A}\backslash\,\mathbf{b}$ (dělení zleva), použijeme-li symboliku MATLABu. Na obr. 3.34 je zadána matice $\mathbf{A}$ a vektor pravých stran $\mathbf{b}$ soustavy:

$$\begin{aligned} -x_1 + x_2 + 2x_3 &= 2 \\ 3x_1 - x_2 + x_3 &= 6 \\ -x_1 + 3x_2 + 4x_3 &= 4 \end{aligned}$$

Řešením je vektor $\mathbf{x} = [1\ \ -1\ \ 2]^{T}$.

Řešení je možné provést pomocí příkazu **linsolve**. Jako parametr příkazu je možné upřesnit pomocí logického pole typ matice soustavy a MATLAB pak zvolí nejvhodnější metodu řešení dané soustavy. Zadání provádíme ve tvaru **x = linsolve(A, B, opts)**. Parametry pole **opts** jsou v tab. 3.7. Zadávání provádíme přiřazením pomocí tečky, např. **opts.LT = true**. Je zřejmé, že nejsou možné všechny kombinace. Matice zajisté nemůže být např. horní a zároveň dolní trojúhelníkovou maticí. Seznam možných nastavení získá uživatel zadáním **doc linsolve**. Podrobnější popis přímého řešení soustavy algebraických rovnic pak zadáním **doc mldivide**.

```
Command Window
>> A=[-1 1 2; 3 -1 1;-1 3 4]
A =
    -1     1     2
     3    -1     1
    -1     3     4
>> b=[2;6;4]
b =
     2
     6
     4
>> x=A\b
x =
    1.0000
   -1.0000
    2.0000
>>
```

Obr. 3.34: Přímé řešení soustavy lineárních algebraických rovnic

```
Command Window
>> A = triu(rand(5,3))
A =
    0.4057    0.0579    0.2028
         0    0.3529    0.1987
         0         0    0.6038
         0         0         0
         0         0         0
>> x = [1 1 1 0 0]'
x =
     1
     1
     1
     0
     0
>> b = A'*x
b =
    0.4057
    0.4108
    1.0053
>>
```

```
Command Window
>> x=(A')\b
x =
    1.0000
    1.0000
    1.0000
         0
         0
>> opts.UT=true; opts.TRANSA=true; opts
opts =
        UT: 1
    TRANSA: 1
>> x=linsolve(A,b,opts)
x =
    1.0000
    1.0000
    1.0000
         0
         0
>>
```

Obr. 3.35: Řešení soustavy lineárních algebraických rovnic pomocí **linsolve**

Tab. 3.7: Parametry logického pole příkazu **linsolve**

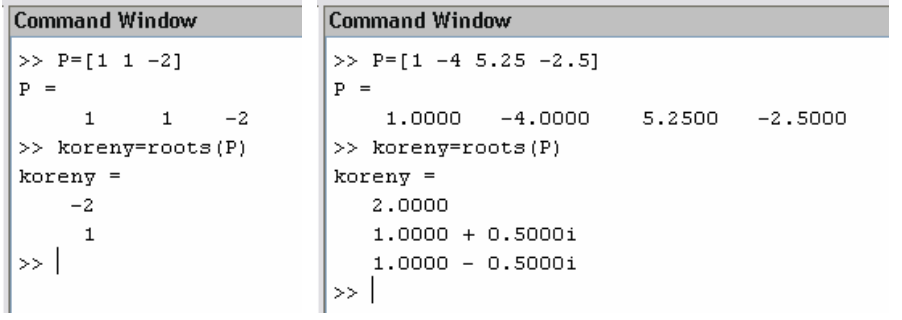
Název položky	Typ nebo vlastnost matice
LT	dolní trojúhelníková matice
UT	horní trojúhelníková matice
UHESS	horní Hessenbergova matice
SYM	reálná symetrická matice nebo komplexní hermitovská matice
POSDEF	pozitivně definitní matice
RECT	obecná obdélníková matice
TRANSA	pro případ, že $\mathbf{A}^T \mathbf{x} = \mathbf{b}$

3.7 Základy práce s mnohočleny

Mnohočleny jsou v MATLABu jednoduše definovány pomocí vektorů, které obsahují jejich koeficienty a to v sestupném pořadí od nejvyšší mocniny nezávisle proměnné k nejnižší. Základní skupinu příkazů, které jsou obsaženy v samotném jádru MATLABu, je možné případně doplnit příkazy z dalších toolboxů. Jde např. o speciální toolbox pro symbolickou matematiku (Symbolic Math Toolbox) a toolbox pro práci s mnohočleny (Polynomial Toolbox). Příkazy těchto toolboxů zde ale nebudeme popisovat.

Jak již bylo uvedeno, mnohočlen vytvoříme zadáním vektoru jeho koeficientů. S takto vytvořeným mnohočlenem již můžeme dále pracovat. Mezi užitečné funkce patří výpočet kořenů, k čemuž slouží příkaz **roots**.

Pomocí příkazu **poly** je možné stanovit charakteristický polynom matice, tedy det($\lambda\mathbf{I} - \mathbf{A}$). V nižších verzích MATLABu bylo také možné pomocí tohoto příkazu sestavovat mnohočlen ze známých kořenů. Zde je třeba upozornit na skutečnost, že pokud má původní mnohočlen u nejvyšší mocniny jiný koeficient než jedna, pak je třeba při aplikaci příkazu **poly** násobit výsledek právě tímto koeficientem. Opětovné určení koeficientů mnohočlenu z kořenů je tedy bohužel nejednoznačnou úlohou. Toto plyne ze skutečnosti, že pokud násobíme mnohočlen libovolným reálným číslem různým od nuly, jeho kořeny se nezmění.



Obr. 3.36: Výpočet kořenů polynomu

Další užitečnou funkcí je pak funkce **polyval**, která vypočte funkční hodnotu pro konkrétní hodnotu nezávisle proměnné. Tuto funkci s výhodou využíváme i při prokládání naměřených hodnot křivkou (regresi). Mnohočleny je také možné derivovat či integrovat pomocí příkazů **polyder** a **polyint**.

Použití všech výše uvedených příkazů je na následujících obrázcích. Na obr. 3.36 je výpočet kořenů polynomu $x^2 + x - 2$ a polynomu $x^3 - 4x^2 + 5{,}25x - 2{,}5$. Na obr. 3.37 je pak také výpočet charakteristického polynomu $\det(\lambda\mathbf{I} - \mathbf{A}) = c_1\lambda^n + c_2\lambda^{n-1} + \ldots + c_n\lambda + c_{n+1}$ čtvercové matice **A** (v tomto případě je $n = 3$) a ukázka dvou možností nalezení vlastních čísel této matice a nalezení vlastních vektorů. Výsledné vlastní vektory jsou ve sloupcích matice, vlastní čísla jsou na diagonále matice. Jednotlivé sloupce obou matic si navzájem odpovídají, tj. např. první sloupec (vlastní vektor) první matice odpovídá vlastnímu číslu v prvním sloupci druhé matice. MATLAB vypočítá vlastní vektory tak, že mají normu jedna, o čemž se snadno přesvědčíme pomocí příkazu **norm**.

```
Command Window
>> A=[2 2 0;-3 -3 5;0 -0.25 2]
A =
    2.0000    2.0000         0
   -3.0000   -3.0000    5.0000
         0   -0.2500    2.0000
>> C=poly(A)
C =
    1.0000   -1.0000   -0.7500   -2.5000
>> lambda=roots(C)
lambda =
   2.0000
  -0.5000 + 1.0000i
  -0.5000 - 1.0000i
>> lambda=eig(A)
lambda =
  -0.5000 + 1.0000i
  -0.5000 - 1.0000i
   2.0000
>>
```

```
Command Window
>> A=[1/2 1;4 7/2]
A =
    0.5000    1.0000
    4.0000    3.5000
>> [vl_vektory vl_cisla]=eig(A)
vl_vektory =
   -0.7071   -0.2425
    0.7071   -0.9701
vl_cisla =
   -0.5000         0
         0    4.5000
>> %vlastni cisla jsou na diagonale
>> norm(vl_vektory(:,1))
ans =
     1
>> norm(vl_vektory(:,2))
ans =
     1
>>
```

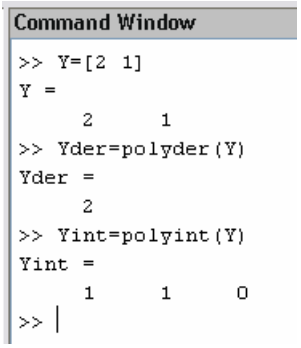
Obr. 3.37: Nalezení charakteristického polynomu, vlastní čísla a vlastní vektory

```
Command Window
>> Y=[2 1]
Y =
     2     1
>> x=3
x =
     3
>> Y_x=polyval(Y,x)
Y_x =
     7
>>
```

```
Command Window
>> Y=[1 0 0]
Y =
     1     0     0
>> X=linspace(0,5,6)
X =
     0     1     2     3     4     5
>> Y_X=polyval(Y,X)
Y_X =
     0     1     4     9    16    25
>>
```

Obr. 3.38: Výpočet funkčních hodnot

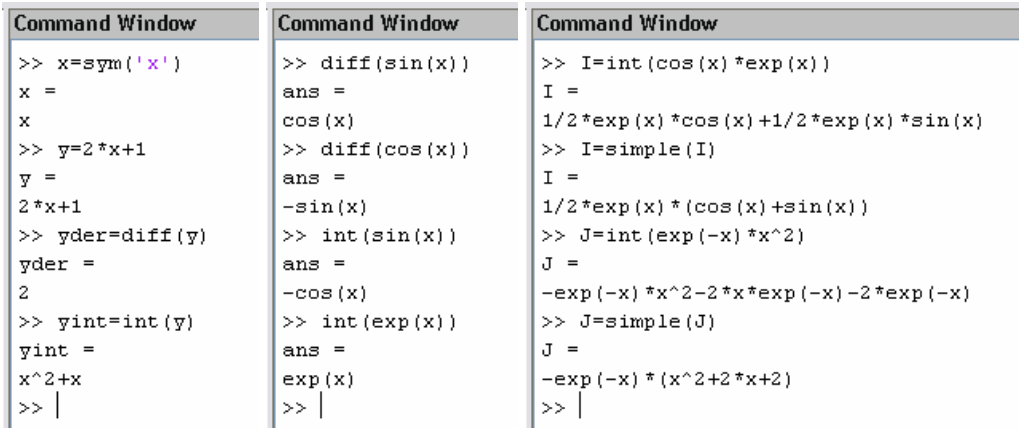
Na obr. 3.38 je ukázka použití příkazu **polyval** pro výpočet funkčních hodnot funkce v daném bodě. Tímto způsobem je vypočítána hodnota funkce $y(x) = 2x + 1$ v bodě $x = 3$ a jsou také vypočítány funkční hodnoty pro $y(x) = x^2$ pro hodnoty $x = 0, 1, \ldots, 5$.



Obr. 3.39: Derivace a integrace polynomu

Pokud derivujeme nebo integrujeme mnohočlen reprezentující např. polynomickou funkci definovanou v MATLABu pomocí vektoru jednotlivých koeficientů, tak výsledkem jsou samozřejmě opět koeficienty výsledných funkcí. Pro větší názornost by bylo možné využít toolbox pro symbolickou matematiku a pomocí příkazu **sym** definovat symbol nezávisle proměnné. Výsledky by pak byly přímo v symbolickém tvaru, který je uživateli bližší a odpovídá matematickému zápisu.

Na obr. 3.39 je derivována a integrována funkce $y = 2x + 1$, která je definována vektorem koeficientů $Y = [2\ \ 1]$. Derivací dostáváme skalární hodnotu 2. Integrací pak vektor $[1\ \ 1\ \ 0]$, který odpovídá zápisu $x^2 + x$.



Obr. 3.40: Derivace a integrace funkcí v symbolickém zápisu

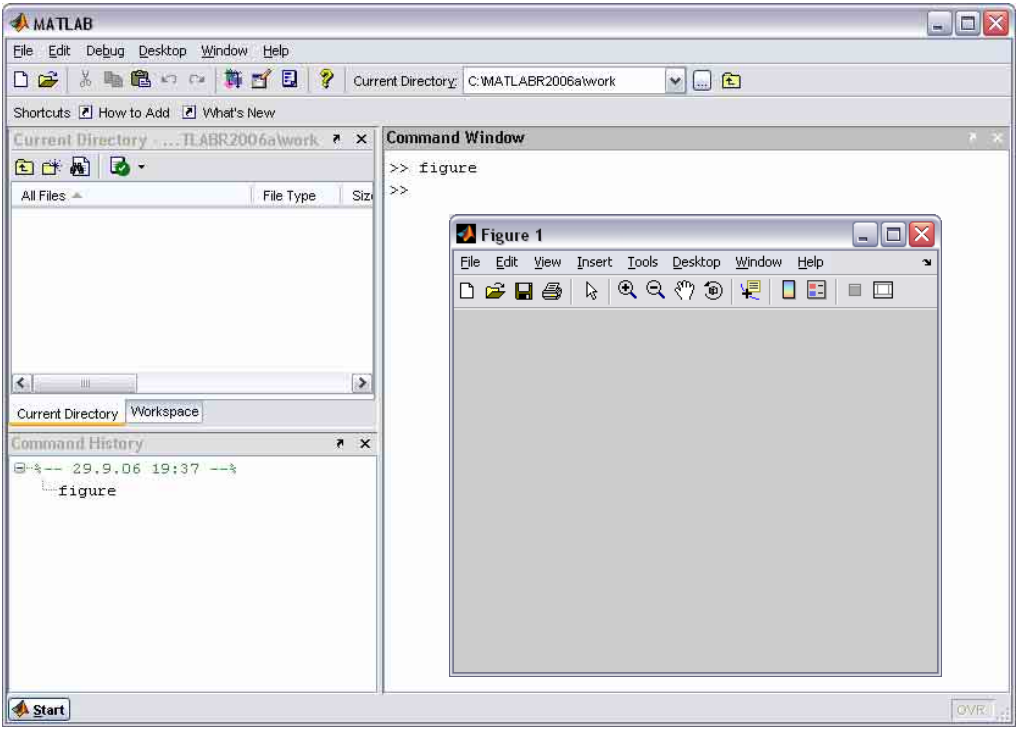
Stejný příklad je řešen i na obr. 3.40, ale již pomocí symbolického zápisu. Toolbox pro symbolickou matematiku není ale přímo součástí jádra MATLABu a proto zde není podrobně probírán. Derivace je v tomto případě vypočítána pomocí příkazu **diff** a integrace pomocí příkazu **int**. Je zřejmé, že tímto způsobem, lze vypočítat derivace resp. integrály i jiných

funkcí. Je možné vypočítat i složitější integrály, které je jinak nutno počítat pomocí metody per partes. Výsledky zjednodušíme použitím příkazu **simple**.

4 Základní použití 2D a 3D grafiky

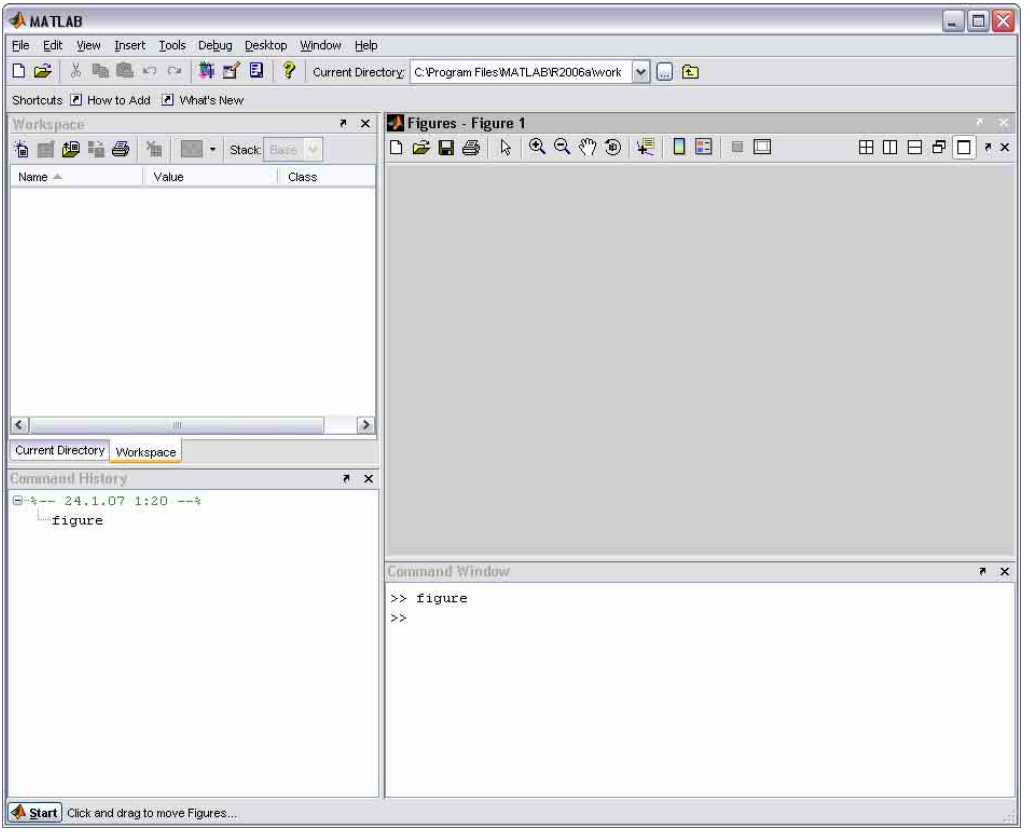
Grafický subsystém prostředí MATLAB umožňuje přehlednou prezentaci výsledků získaných výpočtem a nebo na základě měření. Je možné vykreslit různé druhy grafů: dvourozměrné pro funkce jedné proměnné, třírozměrné pro funkce dvou proměnných, histogramy, koláčové grafy a mnoho dalších. Všem grafickým objektům je možné téměř libovolně měnit vzhled, a to jak již při jejich vytváření, tak i po jejich vykreslení. Vzhled grafických objektů je možno měnit interaktivně, pomocí lišty nástrojů umístěné pod záhlavím obrázku.

Veškerý grafický výstup je v MATLABu realizován v grafickém okně, které se nazývá figure. Těchto oken muže být samozřejmě i několik a každé má své pořadové číslo. Okna lze vytvářet nebo se mezi nimi přepínat pomocí příkazu **figure**. Příkazy pro vykreslování, kterými se budeme v následujícím textu zabývat, vytváří v případě, že uživatel nestanoví jinak, automaticky nové okno. Zapíšeme-li tedy v okně *Command Window* příkaz **figure**, dojde k vytvoření prázdného grafického okna (má šedé pozadí, viz obr. 4.1) s názvem Figure 1. S oknem je možné pracovat standardním způsobem, obdobně jako s jinými okny v systému Windows (lze jej maximalizovat, minimalizovat, posouvat, měnit jeho velikost, apod.).



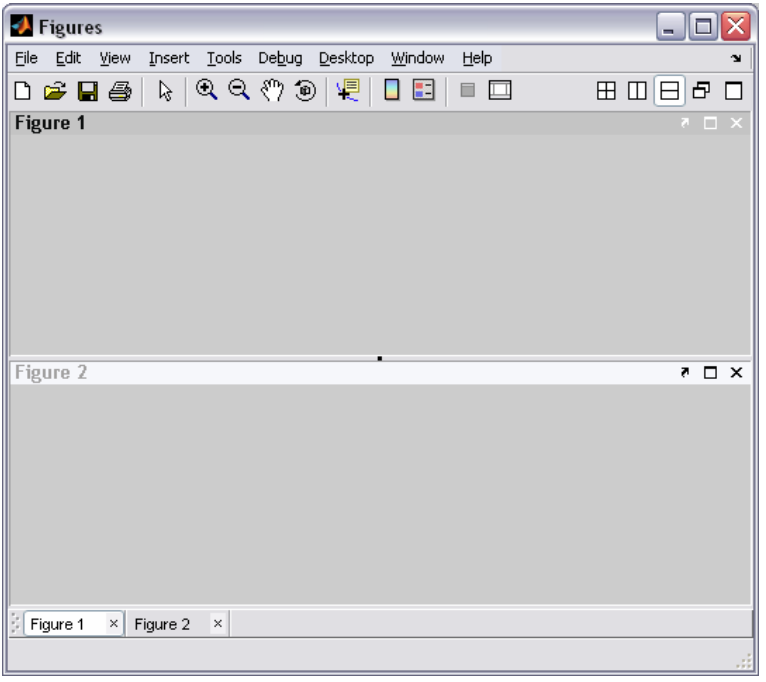
Obr. 4.1: Vytvoření nového grafického okna

Součástí grafického okna je i několik ikon umístěných v toolbaru. Tyto ikony reprezentují některé užitečné a nejvíce používané funkce, které jsou součástí standardních rozbalovacích menu. Pokud najedeme kurzorem myši nad příslušné tlačítko, zobrazí se stručný popis jeho funkce. Jak již bylo řečeno, po zadání příkazu **figure** se vytvoří samostatné okno, které je ale možné také umístit do pracovní plochy (obr. 4.2). K tomuto účelu slouží malé tlačítko se symbolem šipky umístěné v pravém horním rohu okna. Šipka směrem nahoru (dock) znamená zapojit, šipka směrem dolů (undock) pak rozpojit. V tomto okamžiku je možné osamostatnit buď okno Figure 1 (šipka napravo od ikon toolbaru) nebo celý objekt Figures (šipka napravo od záhlaví Figures – Figure 1, viz obr. 4.2 a 4.3).



Obr. 4.2: Zapojení (dock) grafického okna do pracovní plochy

Vytvoříme-li nyní další okno, tak po stisknutí šipky se okno zapojí do objektu Figures. Pomocí tlačítek v pravém horním rohu je pak možné měnit uspořádání jednotlivých oken, viz obr. 4.3. Z důvodu omezeného rozsahu tohoto textu není možné probrat veškeré možnosti interaktivní práce s grafickými objekty. Pro další studium je vhodné použít např. knihu [6]. Zde se omezíme spíše na praktické využití grafického systému MATLABu při prezentaci výsledků. Vhodnější je samozřejmě také přímé nastavení pomocí speciálních příkazů již při samotném vykreslování.



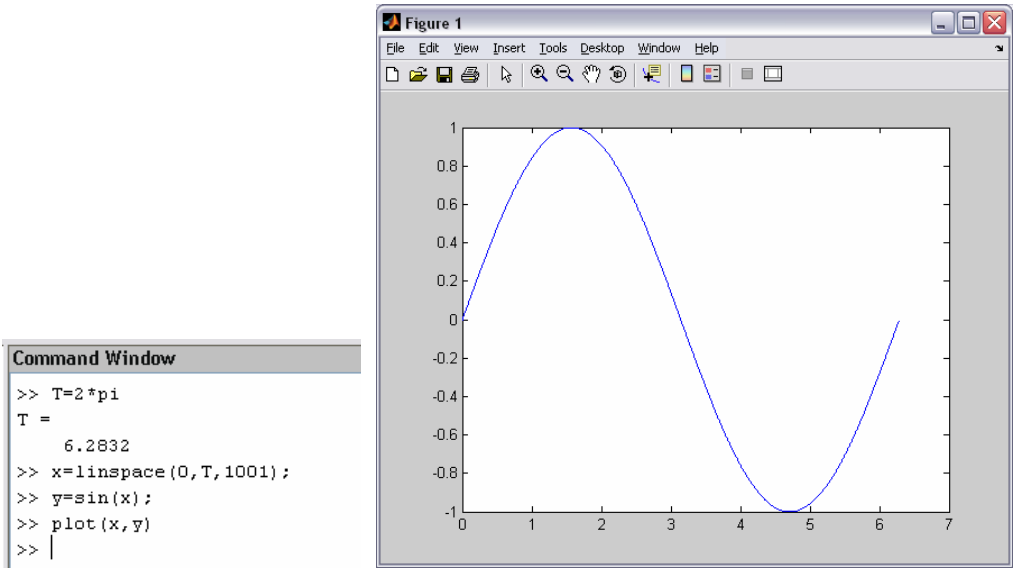
Obr. 4.3: Zapojení grafického okna do objektu Figures

4.1 2D grafika v MATLABu

Základním příkazem pro vykreslování 2D grafů v prostředí MATLABu je **plot**. Obecná syntaxe příkazu **plot** má tvar: **plot**(**x**, **y**, *<barva> <značky> <typ čáry>*). Samozřejmostí je vykreslení několika průběhů do jednoho grafu; příkaz **plot** lze zapsat obdobným způsobem, tedy: **plot**($\mathbf{x_1}$, $\mathbf{y_1}$,*<barva> <značky> <typ čáry>*, $\mathbf{x_2}$, $\mathbf{y_2}$, *<barva> <značky> <typ čáry>*, ...), kde vektory $\mathbf{x_1}$, $\mathbf{y_1}$, $\mathbf{x_2}$, $\mathbf{y_2}$, …, $\mathbf{x_n}$, $\mathbf{y_n}$ přísluší různým průběhům funkcí $\mathbf{y_i} = f(\mathbf{x_i})$ s vlastní definicí značek a čar. Příklad vykreslení průběhu jednoduché funkce (v tomto případě funkce sinus) je uveden na obr. 4.4. Zadání vektorů nezávisle a závisle proměnné je třeba provést způsobem uvedeným v odstavci 3, s výhodou využijeme příkaz **linspace**.

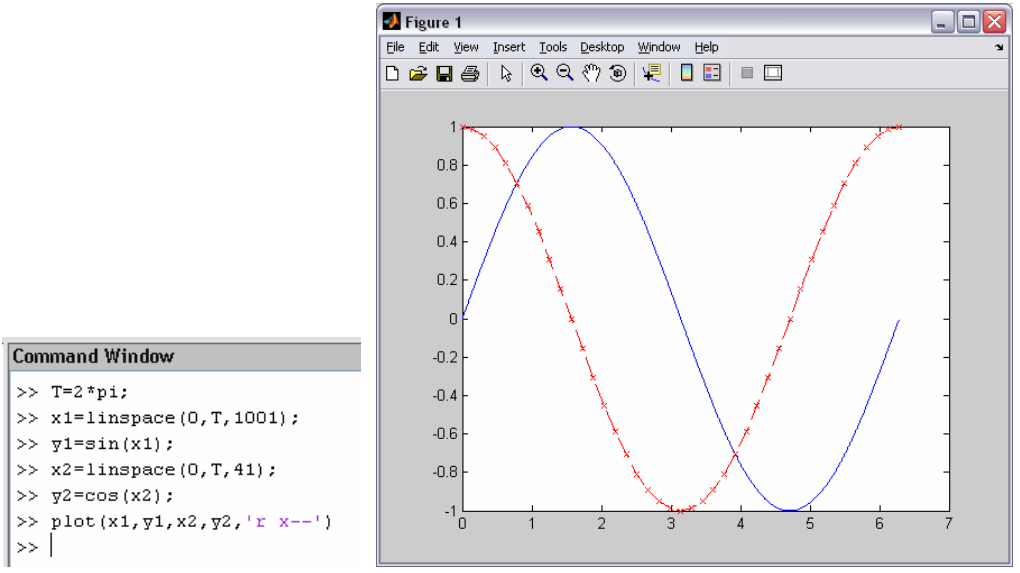
Jak již bylo uvedeno výše, příkaz plot automaticky vytvoří objekt Figure 1 a umístí do něj průběh funkce **y** = sin(**x**). Barva čáry je implicitně nastavena na modrou, šířka čáry na 0,5 a značky jednotlivých bodů nejsou vykreslovány. Graf nemá popisky os, titulek a není ani vykreslena mřížka. Chceme-li, aby barva čáry byla např. červená čárkovaná a jednotlivé body byly označeny křížkem, zapíšeme: **plot(x, y, 'r x--')**, viz také obr. 4.5.

Přehled možných barev, typů čar a značek, které je možné v příkazu **plot** využít, je uveden v tab. 4.1. Šířku čáry lze nastavit pomocí parametru *LineWidth*, velikost značky pak pomocí parametru *MarkerSize*, viz obr. 4.9.




```
Command Window
>> T=2*pi
T =
    6.2832
>> x=linspace(0,T,1001);
>> y=sin(x);
>> plot(x,y)
>>
```

Obr. 4.4: Vykreslení jednoduché funkce



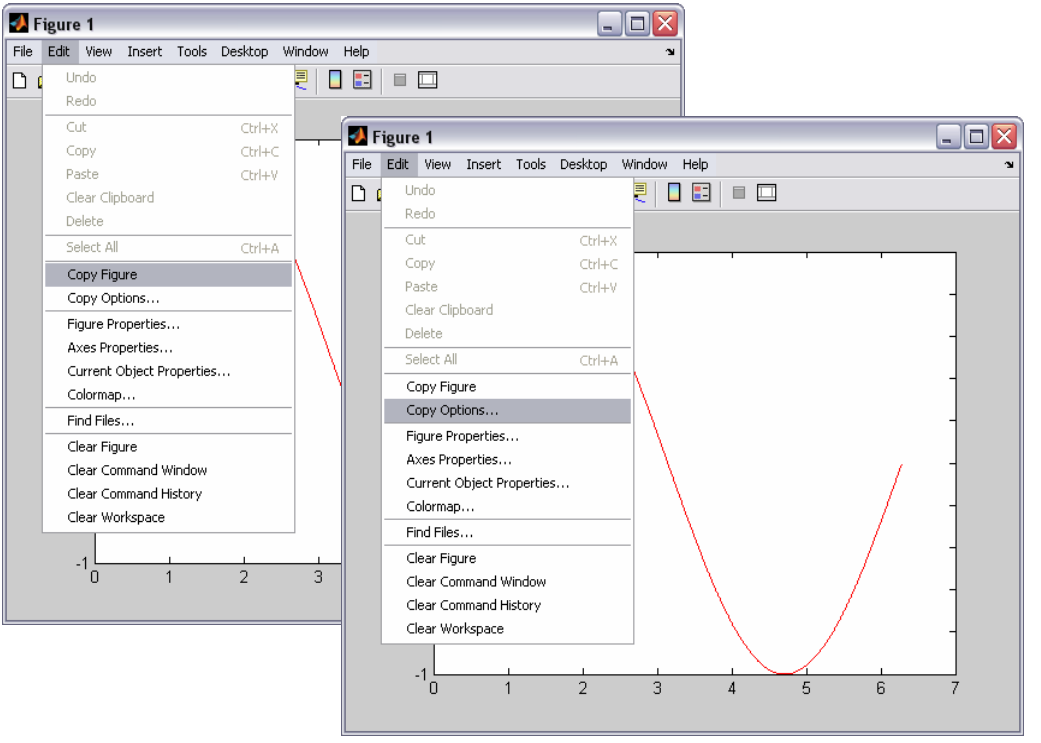

```
Command Window
>> T=2*pi;
>> x1=linspace(0,T,1001);
>> y1=sin(x1);
>> x2=linspace(0,T,41);
>> y2=cos(x2);
>> plot(x1,y1,x2,y2,'r x--')
>>
```

Obr. 4.5: Vykreslení několika funkcí do jednoho grafu, nastavení stylu čar

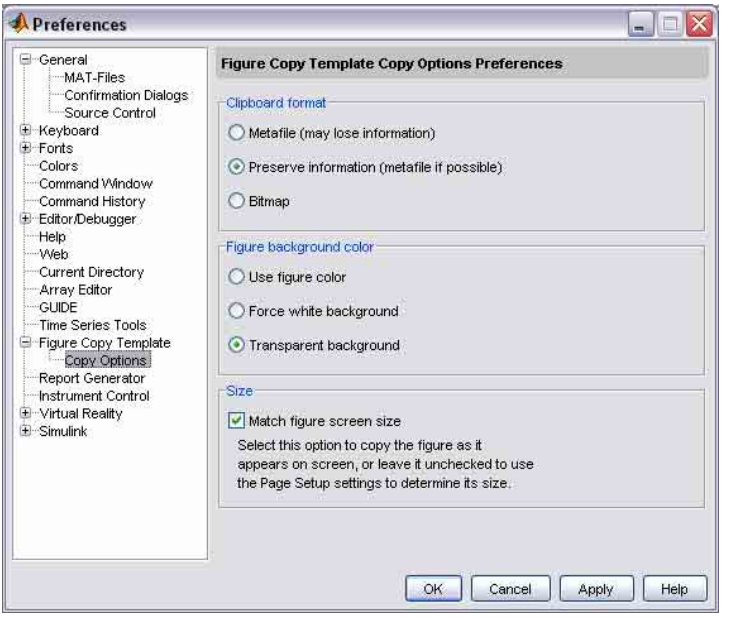
Prostředí MATLABu poskytuje i další možnosti vkládání obrázků do textu, než je prosté vložení celého okna Figure (pomocí Alt + PrintScreen a Ctrl + V), tak jak bylo ukázáno výše. V menu okna Figure vybereme **Edit → Copy Figure**. Je také možné nastavit parametry kopírování v menu **Edit → Copy Options...** (obr. 4.6). Doporučeno je použití formátu metafile (volba Preserve information) a transparentní pozadí (Transparent background) nebo bílé pozadí (Force white background), viz obr. 4.7.

Tab. 4.1: Dostupné barvy a typy čar a typy značek

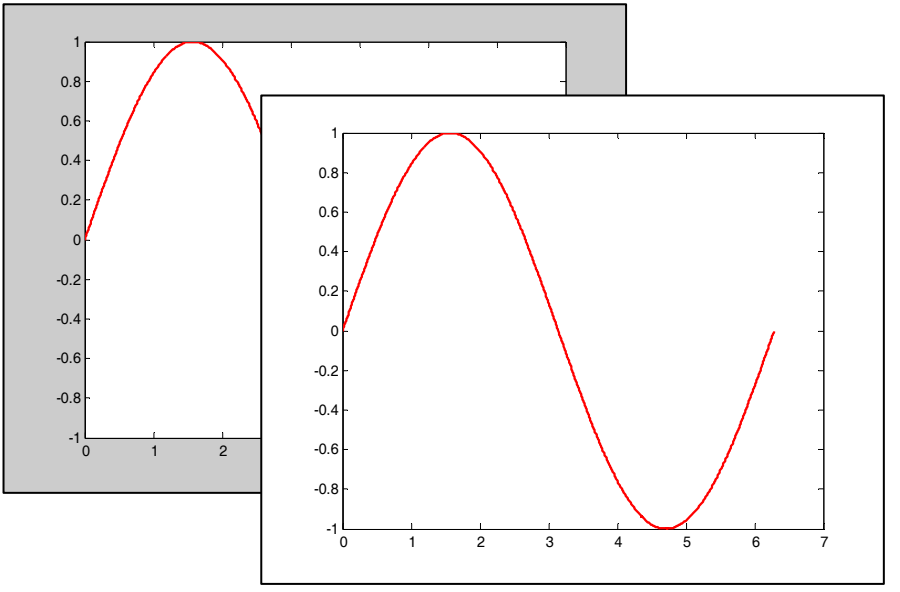
Barva čáry	Typ čáry	Značka bodu
b modrá (blue)	**–** plná (solid)	**.** tečka (point)
g zelená (green)	**:** tečkovaná (dotted)	**o** kroužek (circle)
r červená (red)	**–.** čerchovaná (dash-dot)	**x** křížek (x-mark)
c tyrkysová (cyan)	**– –** čárkovaná (dashed)	**+** křížek (plus)
m purpurová (magenta)	(nic) bez čáry (je-li zadán bod)	***** hvězdička (star)
y žlutá (yellow)		**s** čtverec (square)
k černá (black)		**d** kosočtverec (diamond)
		v trojúhelník (triangle – down)
		^ trojúhelník (triangle – up)
		< trojúhelník (triangle – right)
		> trojúhelník (triangle – left)
		p pětiúhelník (pentagram)
		h šestiúhelník (hexagram)



Obr. 4.6: Kopírování grafu pro vložení do textu



Obr. 4.7: Nastavení parametrů kopírování grafu



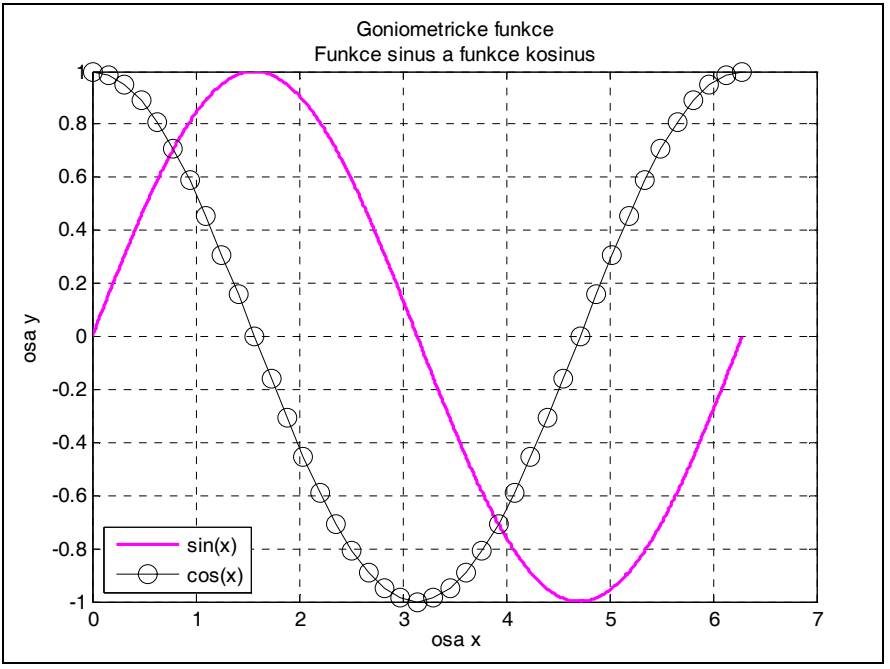
Obr. 4.8: Jiný způsob vložení grafu (Edit → Copy figure a Ctrl + V)

Na obr. 4.8 je ukázka grafu vloženého uvedeným způsobem. Původní nastavení můžeme v části *Figure background color* změnit např. tak, aby zůstaly zachovány původní barvy grafu včetně barvy pozadí (Use figure colors) a nebo použít pozadí bílé (Force white background). Tímto způsobem budou vkládány i některé další grafy v následujícím textu.

4.2 Popis grafu

Popisky os umístíme do grafu pomocí příkazu **xlabel** a **ylabel**, titulek pomocí **title**. Pokud má příslušná popiska obsahovat více řádek, je třeba jednotlivé řádky této popisky zapsat do složených závorek, viz obr. 4.9. Mřížku můžeme zobrazit příkazem **grid**.

```
Command Window
>> x1=linspace(0,2*pi,1001);
>> plot(x1,sin(x1),'m','LineWidth',2)
>> hold on
>> x2=linspace(0,2*pi,41);
>> plot(x2,cos(x2),'k o-','MarkerSize',10)
>> grid on
>> xlabel('osa x'), ylabel('osa y')
>> title({'Goniometricke funkce','Funkce sinus a funkce kosinus'})
>> legend('sin(x)','cos(x)',3)
>>
```



Obr. 4.9: Základní možnosti popisu grafu

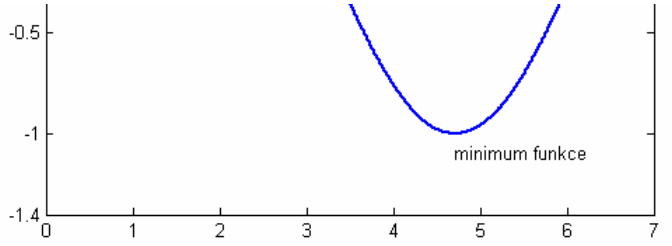
K zobrazení legendy používáme příkaz **legend** (obr. 4.9). V tab. 4.2 jsou uvedeny možnosti umístění legendy v grafu. Zadáme-li **legend boxoff**, tak dojde k odstranění rámečku a výplně plochy legendy. Navrácení původního stavu nastane po zadání **legend boxon**.

Pro zobrazení několika průběhů v jednom grafu je v příkladě uvedeném na obr. 4.9 použit příkaz **hold**. Implicitní nastavení je **hold off**. Vykreslení další funkce do stejného osového systému je možné po zadání příkazu **hold on**.

Tab. 4.2: Umístění legendy v grafu

Číslo pozice	Popis umístění
-1	pravý horní roh, mimo graf
0	automatické umístění
1	pravý horní roh (implicitní nastavení)
2	levý horní roh
3	levý dolní roh
4	pravý dolní roh

Do grafu je také možné pomocí příkazu **text** vkládat jakýkoliv text, který je pak v grafu umístěn na zadanou pozici vztahující se ke konkrétním souřadnicím. Chceme-li např. do grafu umístit text „minimum funkce“ na souřadnice x = 4,7 a y = –1,1, zápis příkazu bude následující: **text(4.7, -1.1, 'minimum funkce')**, viz obr. 4.10.



Obr. 4.10: Umístění textu do grafu

Příkaz **gtext** umožní vložit text na zvolené místo grafu pomocí myši a záměrného kříže, který se při volání této funkce objeví.

Text je také možné umístit do rámečku, jehož vzhled je možno blíže specifikovat. Konkrétně lze nastavit typ a barvu čáry a barvu výplně, viz tab. 4.3. Uvedené parametry lze použít i ve spojení s jinými příkazy; např. parametr *LineWidth* byl již použit (obr. 4.9) jako parametr příkazu **plot**. Na vkládaný text lze samozřejmě aplikovat i všechny níže uváděné formátovací příkazy (tab. 4.4 a 4.5).

Tab. 4.3: Možné parametry vkládaného textu

Parametr	Popis
EdgeColor	barva obrysu rámečku
BackgroundColor	výplň rámečku (implicitně žádná – transparentní)
LineWidth	šířka čáry rámečku (je-li použit parametr *EdgeColor*)
LineStyle	styl čáry rámečku (je-li použit parametr *EdgeColor*)
Margin	zvětší velikost rámečku o zadanou hodnotu v bodech (je-li použit parametr *EdgeColor* nebo *BackgroundColor*)

Mnohdy je také účelné použít v popiscích dolní resp. horní index. K tomu slouží speciální formátovací znaky; pro dolní index používáme podtržítko (_), pro horní index pak stříšku (^). V tab. 4.4 jsou uvedeny další formátovací příkazy, které umožňují měnit typ, velikost a řez písma. Tyto formátovací příkazy mají platnost vždy do konce řetězce nebo se týkají pouze obsahu definovaného uvnitř složených závorek. Příklady použití jsou na obr. 4.11 až 4.13.

Tab. 4.4: Formátovací příkazy

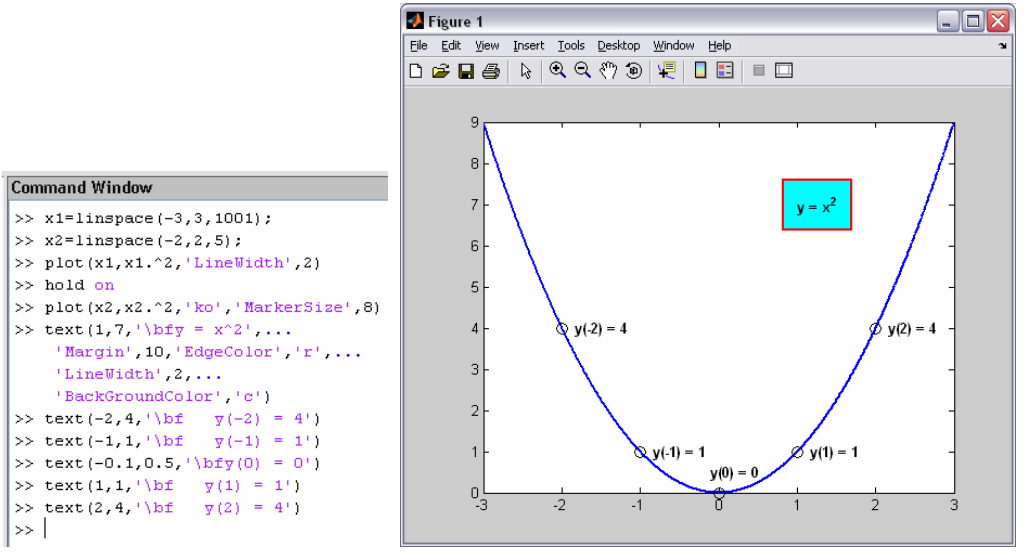
Formátovací příkaz	Popis
\fontname{*fontname*}	bude použito písmo ***fontname***
\fontsize{s}	bude použito písmo o velikosti **s** bodů
\color{*color*}	bude použito písmo barvy ***color***
\bf	tučné písmo
\it	kurzíva
\sl	šikmé písmo
\rm	normální písmo

MATLAB také umožňuje zápis znaků řecké abecedy a dalších znaků často používaných zejména v matematice, jejich přehled je uveden v tab. 4.5.

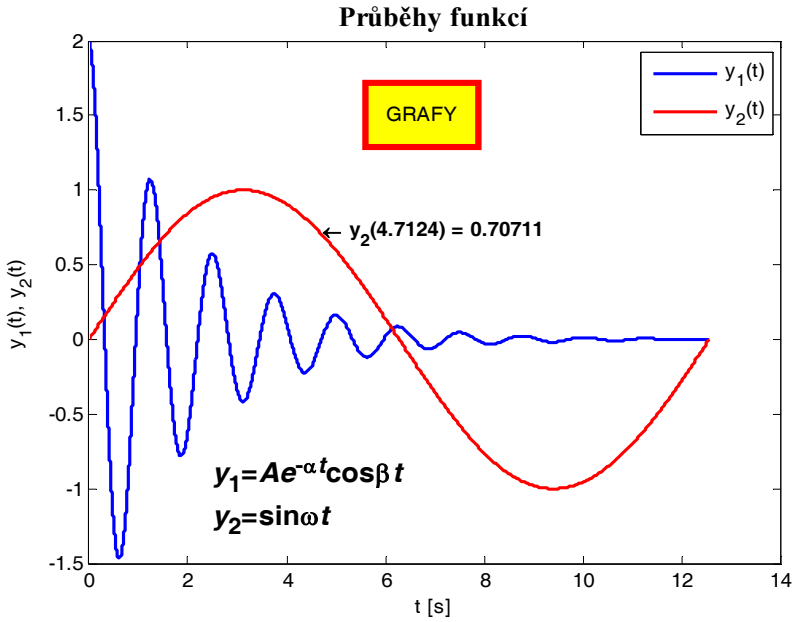
Tab. 4.5: Formátovací příkazy pro speciální znaky v grafech

Znak	Formátovací příkaz	Znak	Formátovací příkaz	Znak	Formátovací příkaz	Znak	Formátovací příkaz
α	\alpha	ς	\varsigma	⟨	\langle	•	\bullet
β	\beta	ϑ	\vartheta	⟩	\rangle	°	\circ
γ	\gamma	Γ	\Gamma	≡	\equiv	ℜ	\Re
δ	\delta	Δ	\Delta	≠	\neq	ℑ	\Im
ε	\epsilon	Θ	\Theta	≈	\approx	ℵ	\aleph
ζ	\zeta	Λ	\Lambda	≅	\cong	℘	\wp
η	\eta	Ξ	\Xi	≤	\leq	∀	\forall
θ	\theta	Π	\Pi	≥	\geq	∃	\exist
ι	\iota	Σ	\Sigma	~	\sim	¬	\neg
κ	\kappa	Υ	\Upsilon	±	\pm	∇	\nabla
λ	\lambda	Φ	\Phi	×	\times	∂	\partial
μ	\mu	Ψ	\Psi	·	\cdot	∫	\int
ν	\nu	Ω	\Omega	÷	\div	√	\surd
ξ	\xi	↔	\leftrightarrow	⊕	\oplus	∝	\propto
ο	\o	←	\leftarrow	⊗	\otimes	∞	\infty
π	\pi	→	\rightarrow	∅	\oslash nebo \0	ϖ	\varpi
ρ	\rho	↑	\uparrow	∩	\cap	∠	\angle
σ	\sigma	↓	\downarrow	∪	\cup	…	\ldots
τ	\tau	⇔	\Leftrightarrow	⊃	\supset	|	\mid
υ	\upsilon	⇐	\Leftarrow	⊂	\subset	⊥	\perp
φ	\phi	⇒	\Rightarrow	⊇	\supseteq	♣	\clubsuit
χ	\chi	⇑	\Uparrow	⊆	\subseteq	♦	\diamondsuit
ψ	\psi	⇓	\Downarrow	∈	\in	♥	\heartsuit
ω	\omega	′	\prime	∋	\ni	♠	\spadesuit

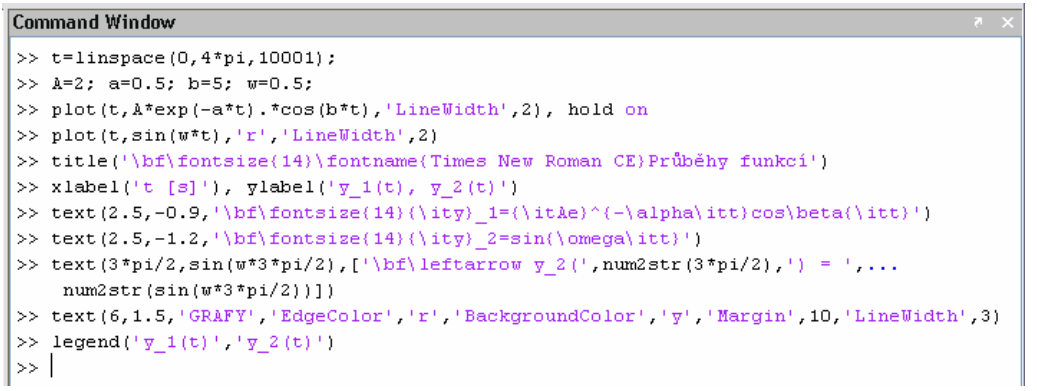
V následujících dvou příkladech (obr. 4.11 až 4.13) je názorně demonstrováno vkládání textu do grafu. V prvním příkladě jsou pomocí textu označeny některé význačné body funkce $\mathbf{y} = \mathbf{x}^2$ a je také vykreslen barevný rámeček s předpisem této funkce. V druhém příkladě jsou ukázány některé další možnosti vytváření složitějších popisek a použití speciálních znaků.



Obr. 4.11: Možnosti vkládání textu do grafu



Obr. 4.12: Použití speciálních znaků v popiscích grafu



Obr. 4.13: Použití speciálních znaků v popiscích grafu (zdrojový kód)

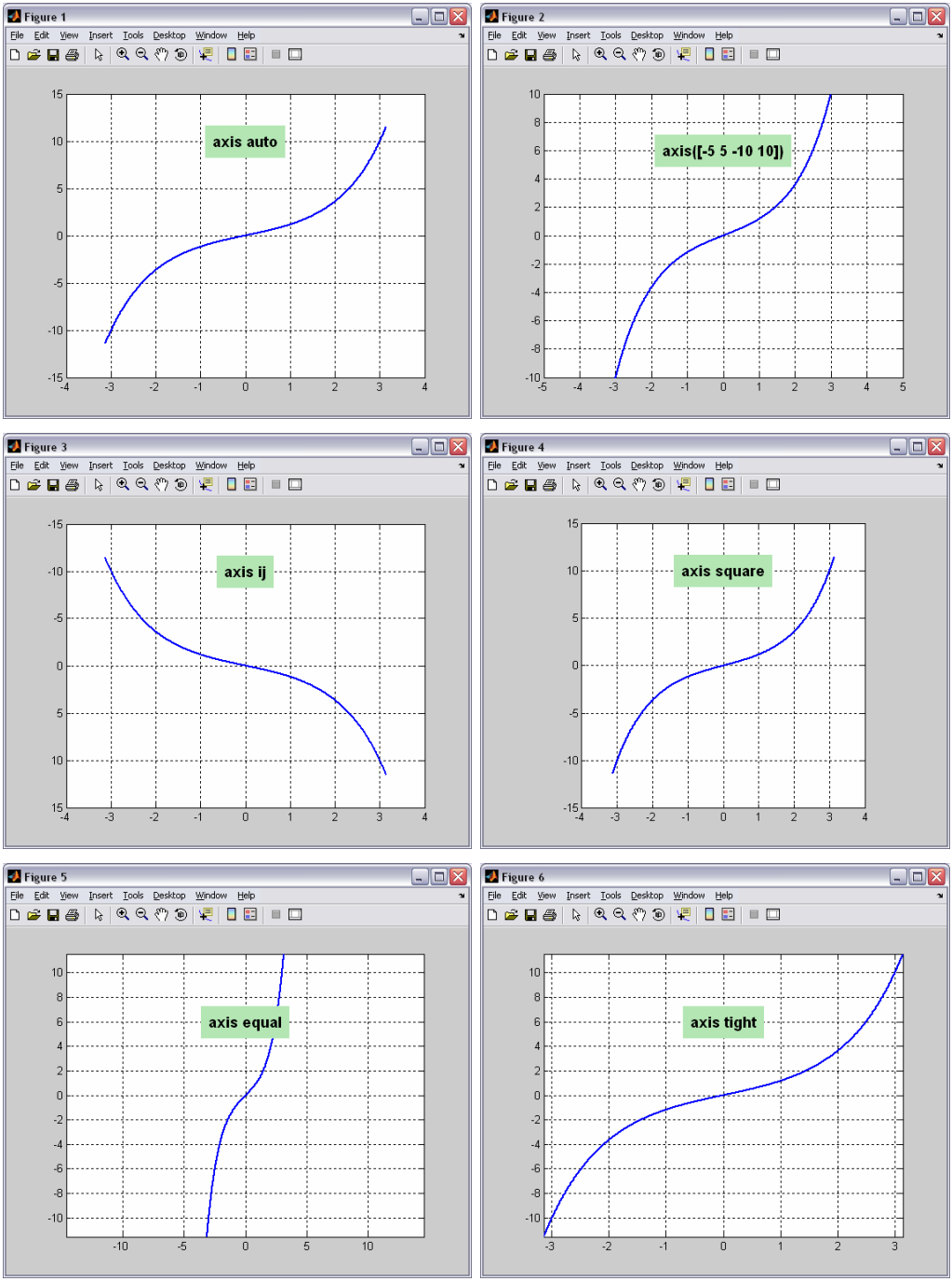
4.3 Ovládání souřadných os

Měřítka na obou osách grafu jsou vždy prostředím MATLAB nastavena tak, aby průběh vykreslované funkce maximálně vyplnil plochu výsledného grafu. Uživatel samozřejmě může pomocí příkazu **axis** toto implicitní nastavení jednoduše změnit, viz tab. 4.6.

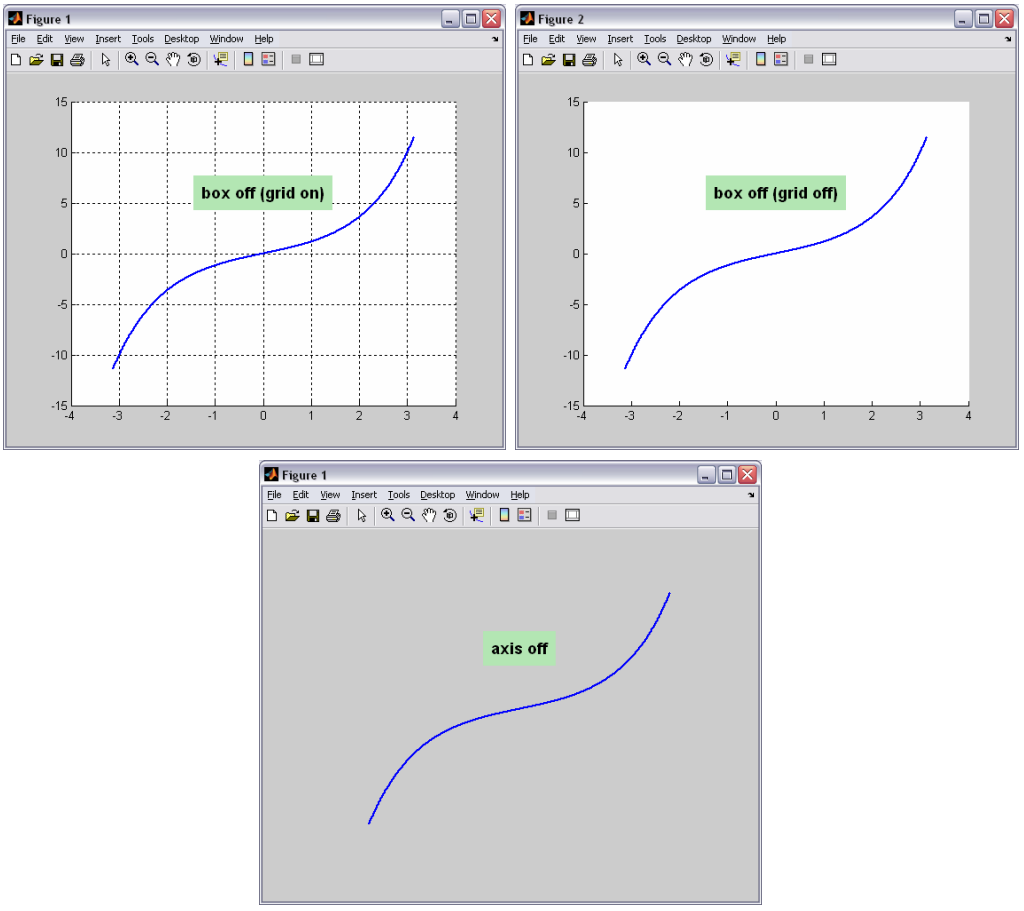
Tab. 4.6: Možnosti ovládání souřadných os

Příkaz	Popis
axis([xmin xmax ymin ymax])	nastaví meze os aktuálního grafu
v = axis	vrátí vektor jednotlivých mezí os aktuálního grafu
axis auto	nastaví implicitní meze os
axis manual	zmrazí meze os
axis xy	použije kartézský souřadný systém, tj. počátek je vlevo dole
axis ij	použije maticový souřadný systém, tj. počátek je vlevo nahoře
axis square	nastaví čtvercové osy
axis equal	nastaví na obou osách stejná měřítka
axis tight	nastaví meze os podle rozsahu dat
axis vis3d	zamezí změně proporcí os při změně pohledu
axis normal	zruší nastavená měřítka a efekty **equal**, **square, tight** a **vis3d**
axis on | **off**	**on** zapne zobrazení os, **off** vypne zobrazení os

Ukázky některých variant použití příkazu **axis** jsou na obr. 4.14 a 4.15. Po zadání příkazu **axis off** dojde nejen k potlačení zobrazení souřadných os, ale nebude zobrazena ani výplň grafu (implicitně bílá). Efekt je podobný jako při použití příkazu **legend boxoff**. Pokud chceme, aby nebyl vykreslován rámeček grafu, zadáme příkaz **box off**. Jeho opětovné zapnutí je možné po zadání **box on**, viz obr. 4.15. V tomto případě je ale nadále vykreslováno pozadí grafu. Výsledný vzhled je také závislý na skutečnosti zda je vykreslena mřížka (**grid on**).



Obr. 4.14: Ovládání souřadných os



Obr. 4.15: Vypnutí zobrazení rámečku grafu a vypnutí zobrazení os

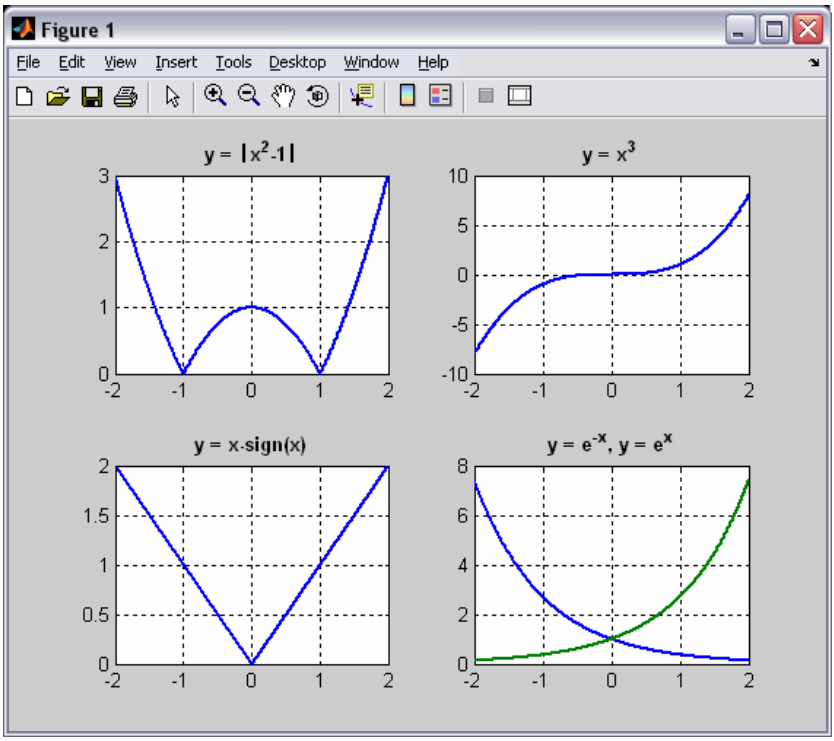
K shora uvedeným vlastnostem a objektům lze také přistupovat interaktivním způsobem přímo z různých menu příslušného grafického okna – Figure. Osy, popisky, texty nebo typy čar lze definovat nebo měnit např. pomocí ikon ve Figure Toolbaru, pomocí voleb v roletovém menu **Insert** a jeho položek **X Label**, **Y Label**, **Title**, **Legend**, **Line**, **Text** nebo **Axes**. V roletovém menu **Edit** jsou dále položky **Figure Properties**, **Axes Properties** a **Current Object Properties**, které umožňují pracovat s objekty ještě podrobnějším způsobem.

Dosud uvedené možnosti práce s grafem jsou určeny především těm uživatelům, kteří nepotřebují ke své práci hlubší znalost grafiky MATLABu, ale potřebují zejména efektivně a přehledně vykreslovat výsledky svých výpočtů či měření. MATLAB však interpretuje všechny tyto výše popisované entity jako grafické objekty, které mají určité vlastnosti a které mají svůj specifický identifikátor – tzv. handle. Mezi jednotlivými objekty je stanovena hierarchie ve smyslu rodiče a potomků. Pro orientaci v tomto hierarchickém stromu a pro práci s vlastnostmi vybraných objektů slouží potom příkazy **get** a **set**. Podrobnější seznámení s *Handle Graphics* by ale značně přesahovalo rozsah tohoto textu. Pro případné další studium je možné použít některou z knih uváděných v seznamu literatury, např. [3, 4, 7].

4.4 Kreslení více grafů do jednoho obrázku

Jak již bylo řečeno, MATLAB umožňuje přehlednou a relativně jednoduchou formou prezentovat výsledky nejrůznějších výpočtů. V mnoha případech může být výhodné vykreslit několik celých grafů do jednoho obrázku. Kreslící plocha může být pomocí příkazu **subplot** rozdělena na několik samostatných částí. Syntaxe tohoto příkazu je následující: **subplot(m,n,p)** nebo případně **subplot(mnp)**. Kreslící plocha je rozdělena na **m** řádků a **n** sloupců. Číslo **p** určuje konkrétní graf, do kterého se má vykreslovat. Jednotlivé grafy jsou kresleny po řádcích od shora dolů. Za příkazem **subplot** následuje příkaz pro vykreslení daného typu grafu (např. příkaz **plot**). Na obr. 4.16 jsou tímto způsobem vykresleny čtyři grafy jednoduchých funkcí.

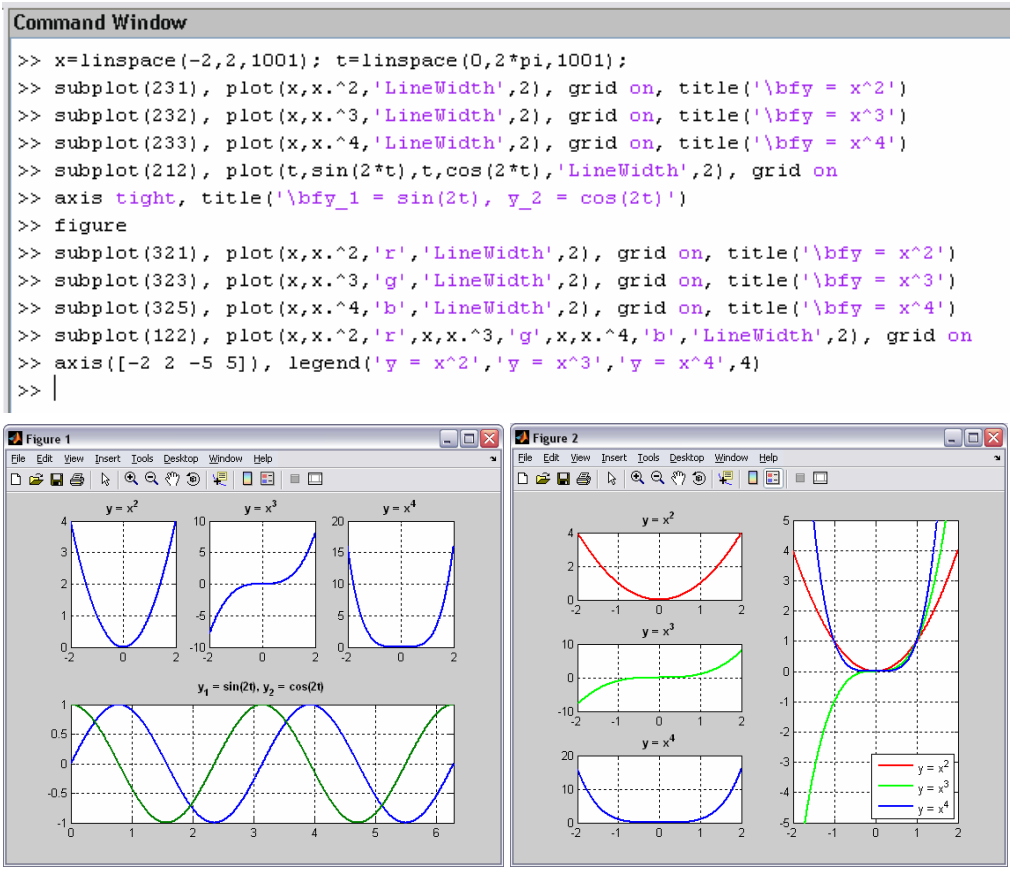
```
Command Window
>> x=linspace(-2,2,1001);
>> subplot(221), plot(x,abs(x.^2-1),'LineWidth',2), grid on
>> title('\bfy = \midx^2-1\mid')
>> subplot(222), plot(x,x.^3,'LineWidth',2), grid on
>> title('\bfy = x^3')
>> subplot(223), plot(x,x.*sign(x),'LineWidth',2), grid on
>> title('\bfy = x\cdotsign(x)')
>> subplot(224), plot(x,exp(-x),x,exp(x),'LineWidth',2), grid on
>> title('\bfy = e^-^x, y = e^x')
>>
```



Obr. 4.16: Kreslení více grafů do jednoho okna

Takto získaný obrázek obsahující několik grafů je možné způsobem uvedeným výše vkládat do textu prostřednictvím menu okna **Edit → Copy Figure**. Možnosti umístění grafů v obrázku jsou rozsáhlé; uživatel prakticky není nijak omezen. V příslušném grafu může být, stejně jako při běžném zobrazení pouze jednoho grafu v grafickém okně, i několik průběhů různých funkcí. Na obr. 4.16 jsou tímto způsobem zobrazeny průběhy funkcí $\mathbf{y} = \mathbf{e}^{-\mathbf{x}}$ a $\mathbf{y} = \mathbf{e}^{\mathbf{x}}$.

Není nutné ani zachovávat stejný počet grafů na jednotlivých řádcích resp. sloupcích, viz obr. 4.17. Můžeme např. umístit na první řádek tři grafy a na řádek druhý umístit graf pouze jeden. Analogicky je také možné do prvního sloupce nad sebe umístit tři grafy, zatímco druhý sloupec je tvořen jedním grafem.



Obr. 4.17: Další varianty umístění více grafů v jednom okně

4.5 Vykreslování matic

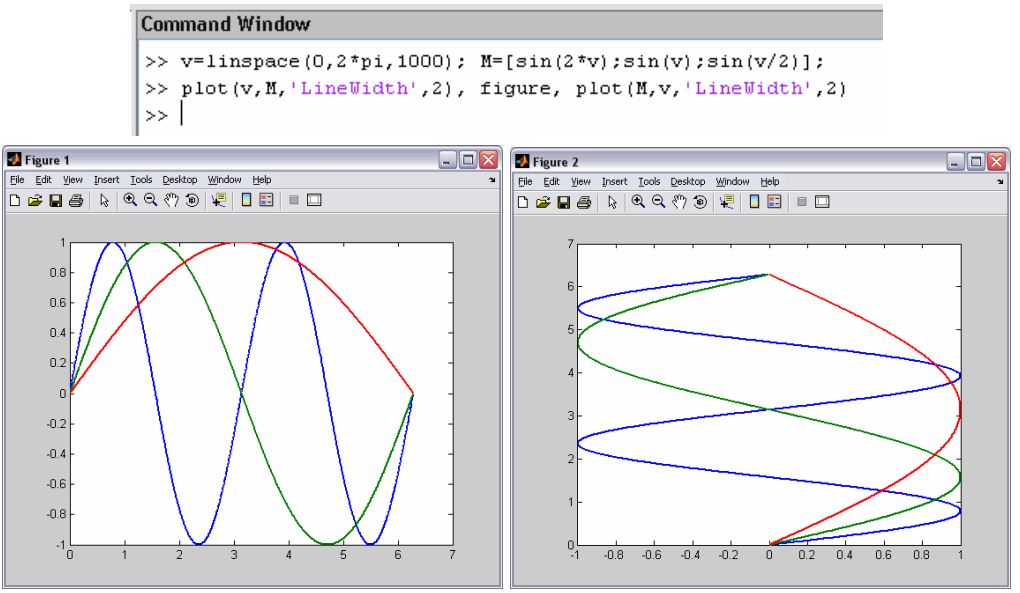
V případě, že jsou argumenty příkazu **plot** vektor **v** a matice **M**, tj. příkaz je zadán ve tvaru **plot(v,M)**, bude postupně vykreslen každý sloupec matice **M** versus vektor **v**. Jednoduchý příklad pro vykreslení průběhů několika goniometrických funkcí uvedeným způsobem je na

obr. 4.18. Zamění-li uživatel navzájem argumenty příkazu, tj. zadá-li **plot(M,v)**, výsledný graf bude otočen o 90° oproti předchozímu případu. Všechny dostupné varianty vykreslování matic jsou popsány v tab. 4.7.

Tab. 4.7: Různé varianty vykreslování matic

Příkaz	Popis
plot(M)	vykreslí pro každý sloupec matice **M** jeden průběh
plot(v,M)	kreslí řádky nebo sloupce matice **M** vhledem k vektoru **v**; o tom, zda budou kresleny řádky či sloupce rozhoduje počet prvků **v**; pokud je matice čtvercová, jsou kresleny vždy sloupce
plot(M,v)	obdoba předchozího případu, graf je ale otočen o 90°
plot(X,Y)	zobrazí sloupce matice **X** vůči sloupcům matice **Y**
plot(X1,Y1,X2,Y2, ...)	zobrazí vždy sloupce matice **X1** vůči sloupcům matice **Y1**, sloupce matice **X2** vůči sloupců matice **Y2**, atd.; v každé dvojici musí být matice stejného typu

Pokud je argumentem příkazu **plot** pouze matice **M** (obecně o rozměru m×n), je pro nezávisle proměnnou automaticky vytvořen sloupcový vektor délky m resp. matice rozměru m×1. V řeči MATLABu by vytvoření tohoto vektoru odpovídal zápis **v = 1:1: m** resp. **v = 1: m** nebo také **v = linspace(1,m,m)**, ve všech případech s následnou transpozicí **v = v′**.

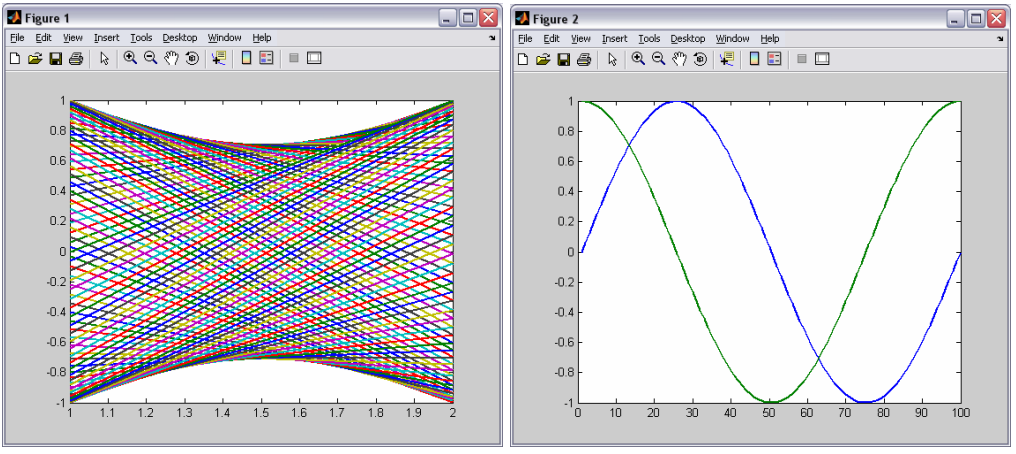


Obr. 4.18: Vykreslování matic

Vytvoříme-li matici **M** obdobně jako v příkladě na obr. 4.19, tak po zadání příkazu **plot(M)** obdržíme velice zajímavý výsledek. Tento výsledek je ale zcela chybný, jelikož cílem bylo vykreslit průběhy dvou goniometrických funkcí – sinu a cosinu. Matice **M** má v tomto případě rozměr 2×100 (zjistíme pomocí příkazu **size(M)**). Dojde tedy k vykreslení sta jednotlivých

křivek – co sloupec matice to samostatná křivka. Pokud ovšem v argumentu příkazu zapíšeme transponovanou matici **M**, tj. zadáme-li **plot(M′)**, tak dojde k vykreslení námi požadovaných průběhů. V příkladě je také ukázána varianta, kdy je vytvořen i vektor pro nezávisle proměnnou způsobem uvedeným výše. Praktická shoda výsledných průběhů je zřejmá; pro kontrolu je opakovaně pomocí příkazu **hold on** kresleno do stejného grafu.

```
Command Window
>> v=linspace(0,2*pi,100); M=[sin(v);cos(v)];
>> plot(M,'LineWidth',2), figure, plot(M','LineWidth',2), hold on
>> [m,n]=size(M')
m =
   100
n =
     2
>> v=1:m; plot(v,M,'LineWidth',2)
>> v=linspace(1,m,m); plot(v,M,'LineWidth',2)
>>
```



Obr. 4.19: Možnosti vykreslování matic

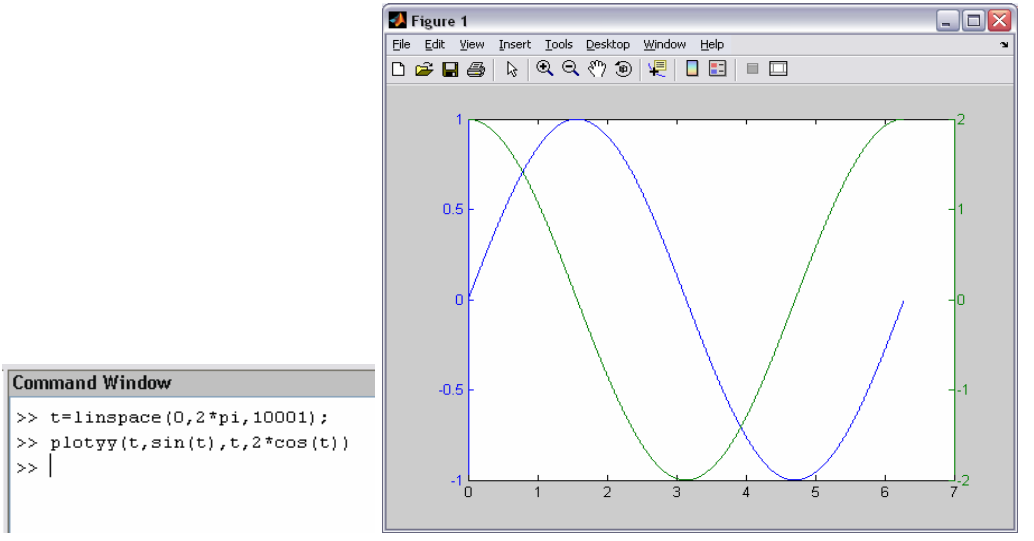
4.6 Speciální typy 2D grafů

MATLAB poskytuje i značné množství specializovaných 2D grafů, jmenovitě např. grafy s dvěmi osami y, s logaritmickými stupnicemi, grafy koláčové, histogramy apod. Kompletní přehled podává tab. 4.8.

Ukázky některých speciálních grafů včetně odpovídajícího zápisu příkazů jsou na obr. 4.20 až 4.34. Chceme-li, aby výsledný graf měl dvě osy závisle proměnné (např. v případě, kdy jsou vykreslované funkce co do rozsahu závisle proměnné značně rozdílné), můžeme použít příkaz **plotyy**. Pokud je ale třeba měnit např. šířku nebo styl čáry, musíme použít prostředků tzv. handle grafiky. Problém nastane i u popisek os závisle proměnných. Standardním způsobem (pomocí příkazu **ylabel**) lze popsat pouze osu na levé straně grafu. Tomuto problému se zde ale nebudeme podrobněji věnovat; další informace uživatel získá zadáním **doc plotyy**.

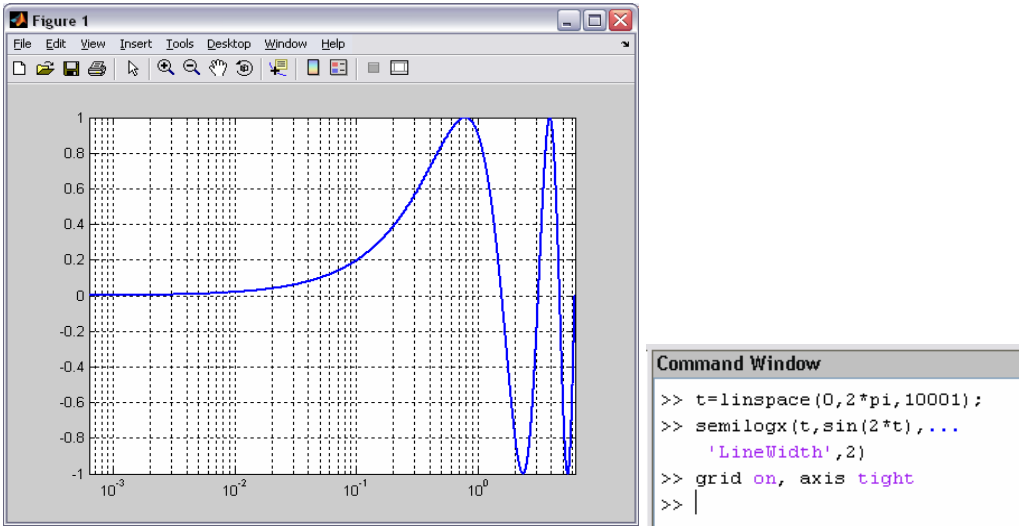
Tab. 4.8: Speciální 2D grafy

Příkaz	Popis
plotyy	graf, který má dvě různé osy závisle proměnné (dvě osy y)
semilogx	graf s logaritmickou stupnicí na ose x a lineární stupnicí na ose y
semilogy	graf s logaritmickou stupnicí na ose y a lineární stupnicí na ose x
loglog	graf s logaritmickou stupnicí na obou osách
stairs	schodový graf
stem	graf diskrétních posloupností (tzv. stopkový graf)
hist	histogram
pie	koláčový graf
fill	vykreslí vyplněný mnohoúhelník
area	vykreslí plošný (vyplněný) graf
comet	průběh funkce je vykreslen pohybujícím se bodem
bar	sloupcový graf
barh	sloupcový horizontální graf
bar3	sloupcový graf, sloupce jsou ve 3D podobě
bar3h	sloupcový horizontální graf, sloupce jsou ve 3D podobě
errorbar	graf chyb
polar	graf v polárních souřadnicích
compass	graf vektorů zobrazovaných ve formě šipek vycházejících z počátku
feather	graf vektorů vycházejících z ekvidistantně rozložených bodů podél horizontální osy
plotmatrix	graf rozptylu
quiver	graf vektorového pole
rose	úhlový histogram
pareto	pareto graf

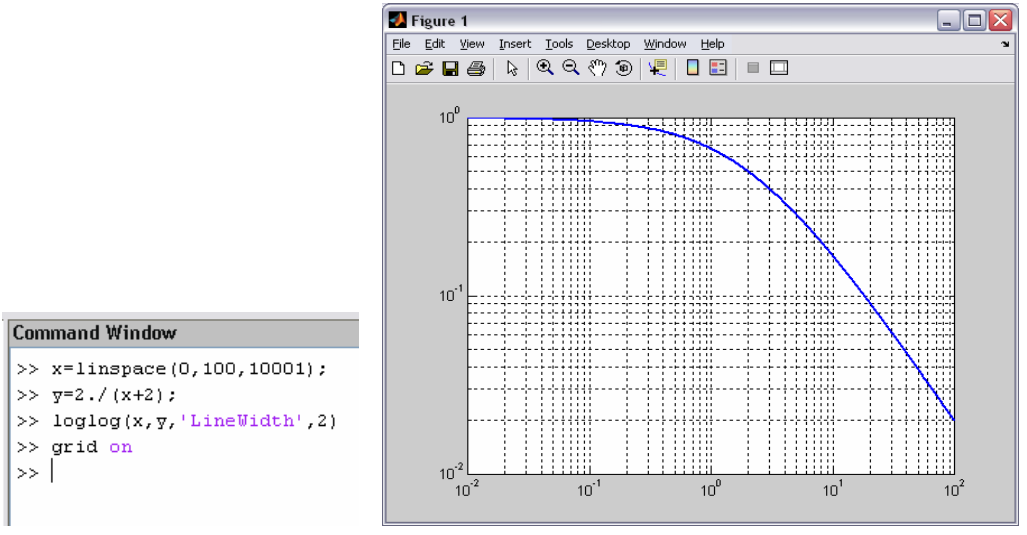


Obr. 4.20: Dvě různé osy závisle proměnné

Často je také vhodné vykreslovat grafy funkcí v logaritmických souřadnicích. MATLAB umožňuje volit logaritmické měřítko pro každou osu zvlášť (**semilogx** a **semilogy**) a nebo pro obě osy současně (**loglog**). Záporné hodnoty jsou při vykreslování ignorovány a MATLAB vypíše varovné hlášení (Warning).

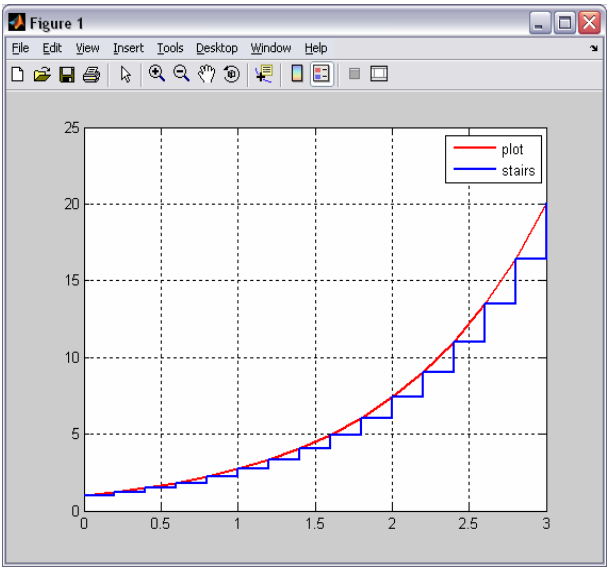


Obr. 4.21: Kreslení v logaritmických souřadnicích – příkaz **semilogx**



Obr. 4.22: Kreslení v logaritmických souřadnicích – příkaz **loglog**

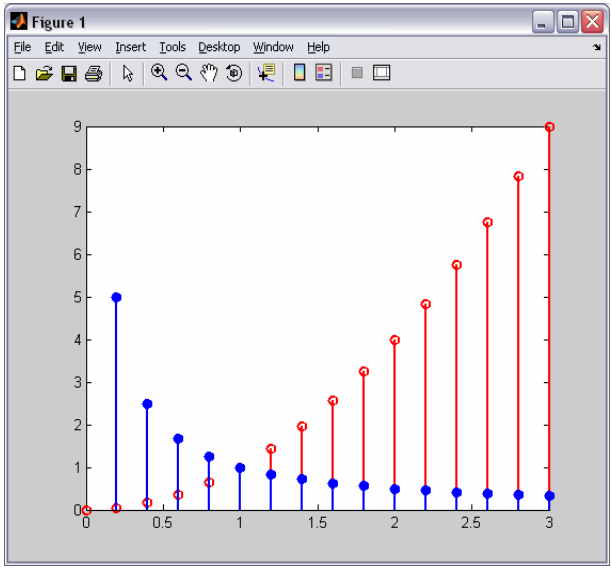
Pomocí příkazu **stairs** lze jednoduše vytvořit schodový graf, který má diskrétní charakter. Hodnoty na ose závisle proměnné se mění pouze v okamžiku, kdy je definována hodnota na ose nezávisle proměnné. Mezi těmito okamžiky je ponechána předchozí hodnota. Na obr. 4.23 je navíc vykreslena standardním způsobem (příkaz **plot**) i spojitá křivka.




```
Command Window
>> dT=0.2;
>> t=0:dT:3;
>> plot(t,exp(t),'r','LineWidth',2)
>> hold on
>> stairs(t,exp(t),'LineWidth',2)
>> grid on
>> legend('plot','stairs')
>>
```

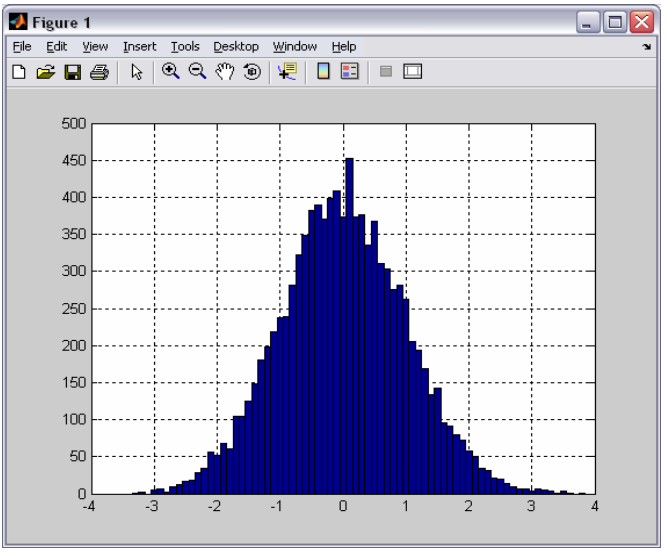
Obr. 4.23: Schodový graf

Pro kreslení diskrétních posloupností (např. vzorků nějakého signálu) je možné použít také příkaz **stem**, který vykreslí tzv. stopkový graf (obr. 4.24). Funkční hodnoty v jednotlivých bodech nejsou navzájem spojeny. Jednotlivé hodnoty jsou ale označeny kroužkem, který je případně možno pomocí parametru příkazu *fill* i vyplnit. MATLAB dokonce při vykreslování funkce **y = 1/t** ohlásí varovnou zprávu – *Warning: Divide by zero*. V čase t = 0 není tato funkce totiž definována, dochází k dělení nulou, nicméně k vykreslení průběhu přesto dojde.




```
Command Window
>> dT=0.2;
>> t=0:dT:3;
>> stem(t,t.^2,'r','LineWidth',2)
>> hold on
>> stem(t,1./t,'fill','LineWidth',2)
Warning: Divide by zero.
>>
```

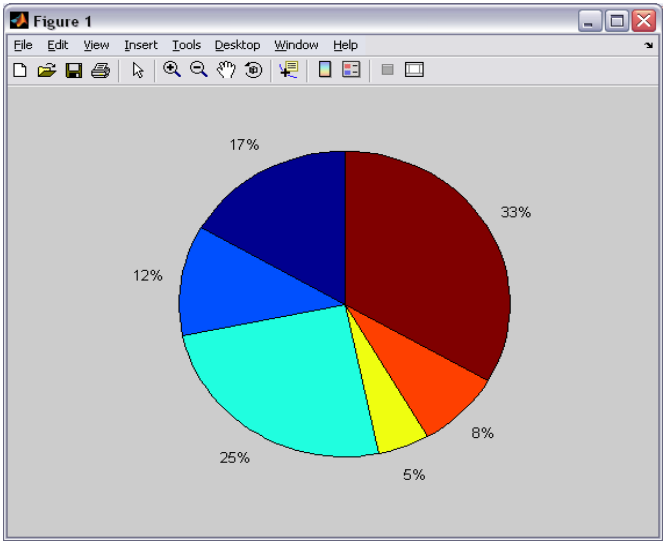
Obr. 4.24: Stopkový graf




```
Command Window
>> v=randn(10000,1);
>> x=-3.9:0.1:3.9;
>> hist(v,x)
>> grid on
>>
```

Obr. 4.25: Histogram

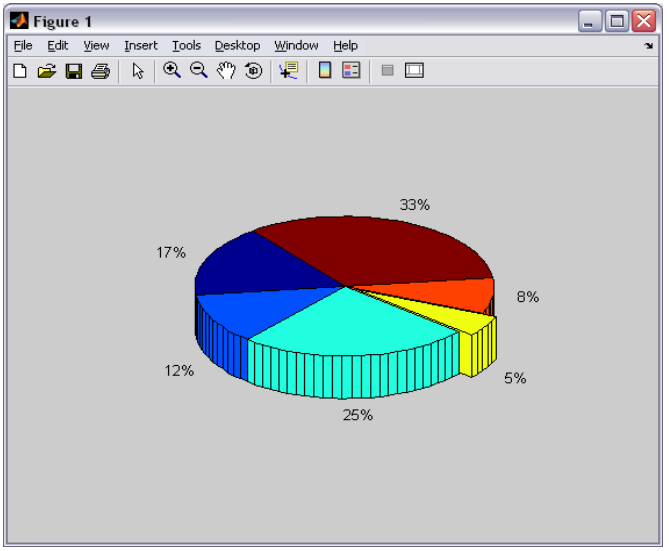
Prostředí MATLAB poskytuje i možnost práce s histogramy. Ty jsou obvykle vhodné pro zobrazování rozložení číselných hodnot, které jsou např. výsledkem realizace náhodného procesu nebo jsou získány měřením, apod. Ve výsledném histogramu jsou četnosti výskytu těchto hodnot reprezentovány sloupci. Příklad použití je na obr. 4.25. V příkladě je vytvořen vektor náhodných čísel **v** s normálním rozložením hustoty pravděpodobnosti. V uvedeném histogramu je znázorněno toto rozložení ve zvoleném intervalu $\mathbf{x} \in \langle -3{,}9;\ 3{,}9 \rangle$. Na ose závisle proměnné je udávána četnost zastoupení hodnot vektoru **x** ve vektoru náhodných čísel **v**. K vlastnímu vykreslení histogramu je použit příkaz **hist**.




```
Command Window
>> v=[1 0.7 1.5 0.3 0.5 2];
>> pie(v)
>>
```

Obr. 4.26: Koláčový graf

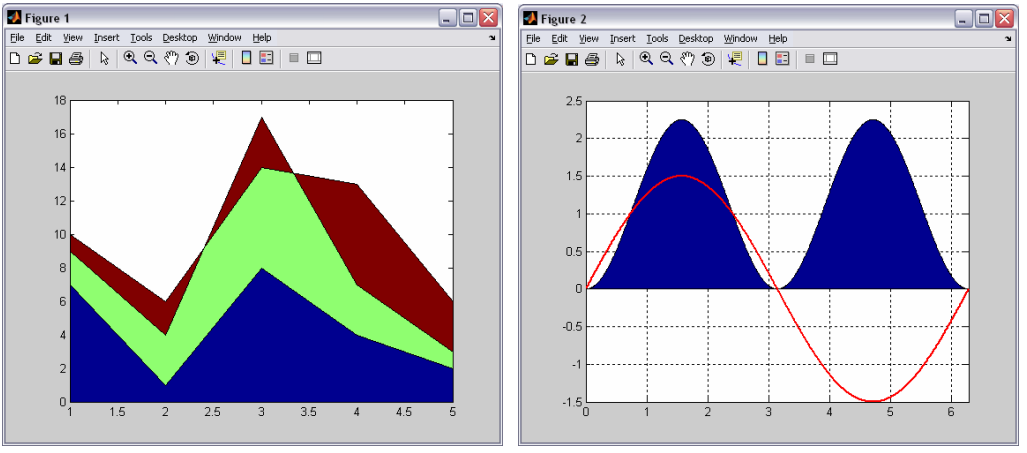
Koláčový graf, často používaný např. při názorném vyjádření podílu jednotlivých částí na celku, lze v MATLABu vykreslit pomocí příkazu **pie**, viz obr. 4.26. Do tohoto typu grafu je možné jednoduše přidat popisky jednotlivých oblastí. Zápis příkazu může být např. následující: **pie([3 4 2], {'rok 2001', 'rok 2002', 'rok 2003'}**. Jednotlivé textové řetězce je třeba umístit




```
Command Window
>> v=[1 0.7 1.5 0.3 0.5 2];
>> z=[0 0 0 1 0 0];
>> pie3(v,z)
>>
```

Obr. 4.27: 3D podoba koláčového grafu

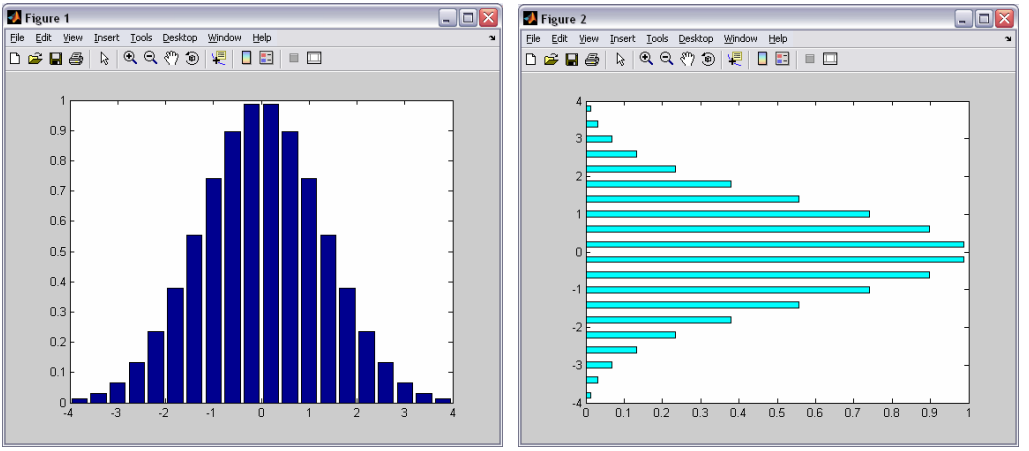
```
Command Window
>> M=[7 2 1;1 3 2;8 9 -3;4 3 6;2 1 3];
>> t=linspace(0,2*pi,1001);
>> area(M), figure, area(t,(1.5*sin(t)).^2), hold on
>> plot(t,1.5*sin(t),'r','LineWidth',2), grid on
>>
```



Obr. 4.28: Plošný graf

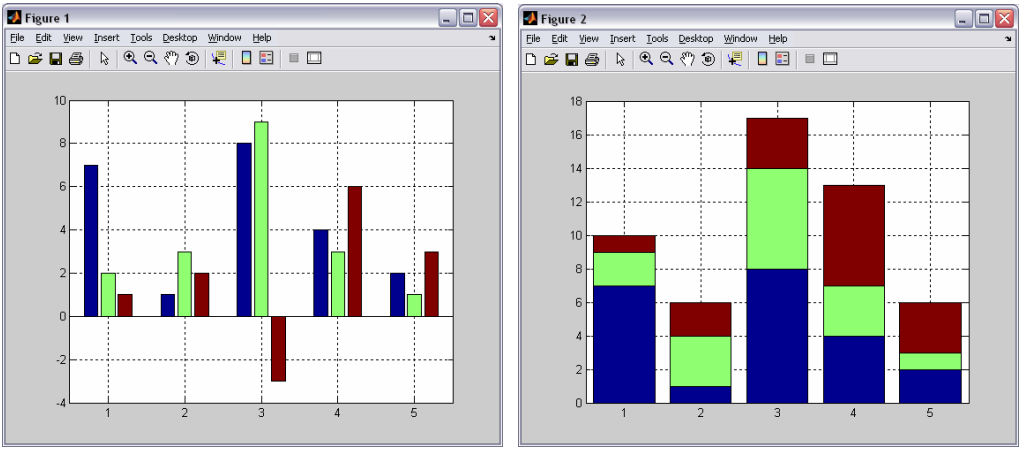
standardním způsobem mezi apostrofy. Jejich počet musí korespondovat s počtem oblastí grafu. Část příkazu pro popisky musí být uzavřena ve složených závorkách. Koláčový graf může mít také 3D podobu; lze jej vykreslit příkazem **pie3**. I v této variantě koláčového grafu, je možné popsat jednotlivé oblasti. Další zajímavou možností je zvýraznění některé oblasti jejím částečným vyčleněním ze základního grafu, viz obr. 4.27.

```
Command Window
>> x=linspace(-3.8,3.8,20); y=exp(-0.3*x.^2);
>> bar(x,y), figure, barh(x,y,0.4,'c')
>>
```



Obr. 4.29: Sloupcový graf

```
Command Window
>> M=[7 2 1;1 3 2;8 9 -3;4 3 6;2 1 3];
>> bar(M), grid on, figure, bar(M,'stack'), grid on
>>
```

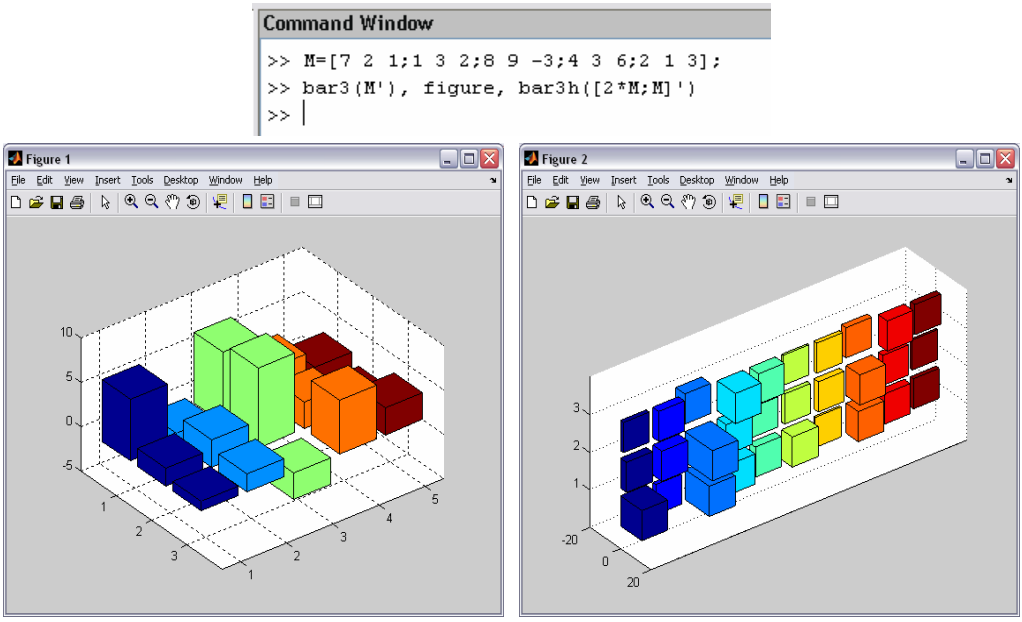


Obr. 4.30: Sloupcový graf – vykreslení matice

Obdobné možnosti jako graf koláčový poskytuje i plošný (vyplněný) graf. Lze jej také využít např. k vyjádření zastoupení jednotlivých složek vzhledem k celku nebo samozřejmě i k jiným účelům, viz obr 4.28. K vykreslení tohoto typu grafu slouží příkaz **area**.

Dalším z mnoha speciálních grafů (tab. 4.8), které MATLAB poskytuje, je graf sloupcový, viz obr. 4.29. Při vykreslování sloupcového grafu je možné měnit barvu a šířku sloupců. Je také možné sloupce vykreslit v horizontálním směru (příkaz **barh**).

Je-li parametrem příkazu **bar** matice, jsou pak sloupce v grafu sdružené do skupin; počet sloupců ve skupině odpovídá počtu sloupců matice (obr. 4.30). Sloupce jsou v grafu navzájem barevně odlišeny a jejich velikost odpovídá hodnotám prvků v řádcích matice. Sloupce je možné také pomocí parametru ***stack*** umístit nad sebe. Velikost každého sloupce pak odpovídá součtu prvků v příslušném řádku matice.

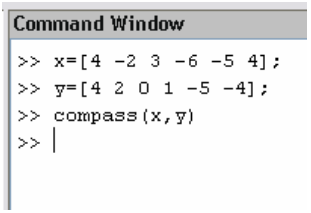


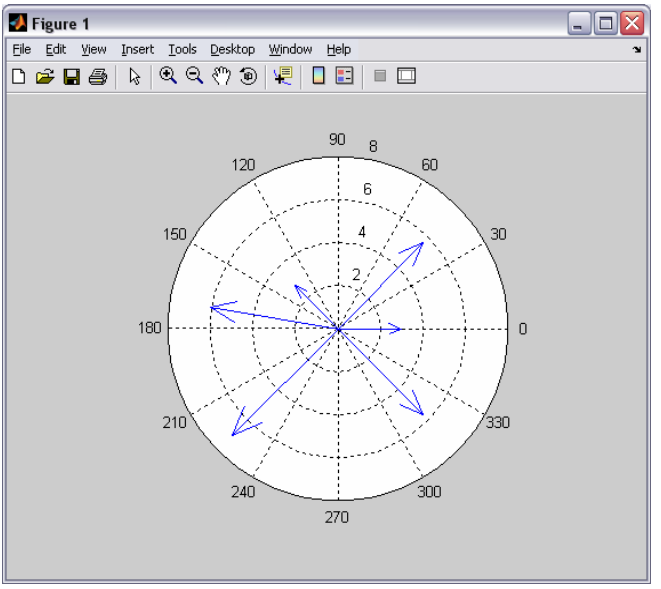
Obr. 4.31: 3D podoba sloupcového grafu

Obdobně jako v předchozím případě, existuje také 3D varianta sloupcového grafu (příkaz **bar3**, obr. 4.31). V argumentu příkazu je samozřejmě možné provádět různé operace, např. vykreslovanou matici transponovat.

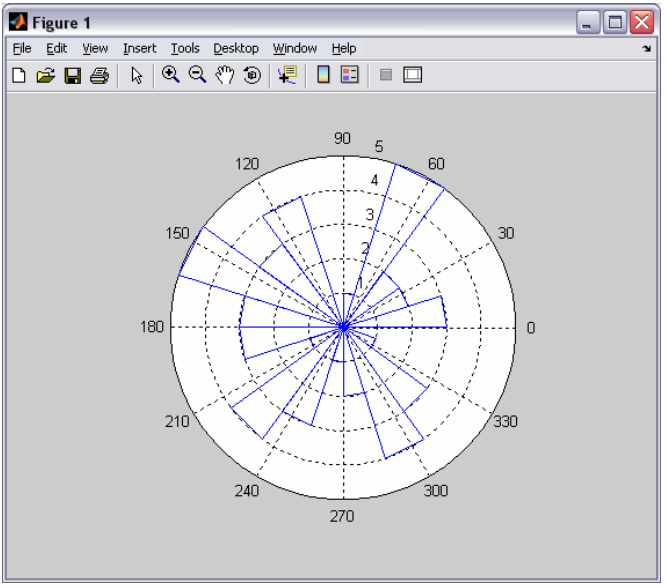
Kompasový graf (obr. 4.32) je možné vykreslit zadáním příkazu **compass**. Parametrem tohoto příkazu může být např. vektor, jehož každý prvek je chápán jako komplexní číslo. V grafu jsou pak jednotlivé prvky (komplexní čísla) znázorněny ve formě šípů. Pokud jsou v příkazu zapsány vektory dva, tedy $\mathbf{x} = [x_1, x_2, ..., x_n]$ a $\mathbf{y} = [y_1, y_2, ..., y_n]$, je poloha koncového bodu každého šípu dána prvky obou těchto vektorů. Příslušný i-tý koncový bod má souřadnice $[x_i, y_i]$.

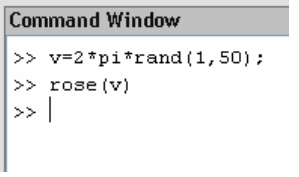
Dalším velice zajímavým grafem je tzv. úhlový histogram (obr. 4.33), který lze vykreslit pomocí příkazu **rose**.





Obr. 4.32: Kompasový graf

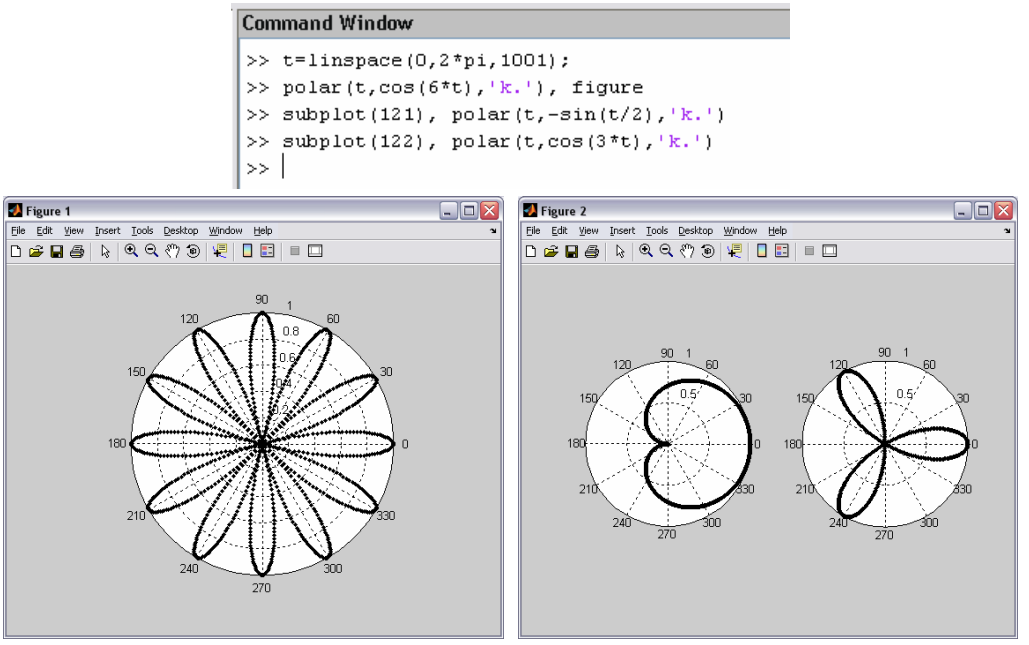




Obr. 4.33: Úhlový histogram

MATLAB umožňuje také pomocí příkazu **polar** efektivně vykreslovat grafy i v polárních souřadnicích. Příkaz má dva parametry. Prvním parametrem je úhel v radiánech, který svírá vektor (někdy označovaný rádius vektor) s kladným směrem osy nezávisle proměnné. Druhým parametrem je délka (modul) tohoto vektoru. Uživatel samozřejmě může také specifikovat styl a barvu kreslených křivek. Stejně jako v ostatních případech je možné i do grafu umístit

nejrůznější popisky. Do obrázku je také možné pomocí příkazu **subplot** umístit několik grafů. Ukázka základního použití příkazu **polar** je uvedena na obr. 4.34. MATLAB poskytuje také dvě funkce, které je lze využít k transformaci souřadnic. Příkaz **cart2pol** slouží k převodu z kartézských souřadnic do polárních, příkaz **pol2cart** k převodu zpětnému.



Obr. 4.34: Graf v polárních souřadnicích

4.7 Základy 3D grafiky v MATLABu

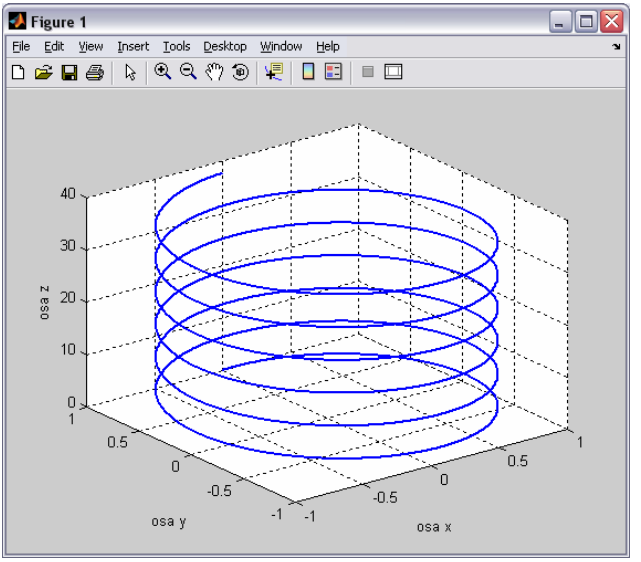
Princip použití třírozměrné 3D grafiky je obdobný jako u 2D grafiky. V platnosti zůstává většina v předchozích kapitolách uvedených poznatků o grafech. Základním příkazem pro vykreslování 3D grafů je příkaz **plot3**. Syntaxe tohoto příkazu je obdobná jako u příkazu **plot**, pouze s rozšířením pro třetí souřadnici: **plot3**(**x**, **y**, **z**, *<barva> <značky> <typ čáry>*). Příklad vykreslení průběhu jednoduché funkce dvou proměnných **z** = f(**x**, **y**) je uveden na obr. 4.35. Zadání vektorů obou nezávisle proměnných a závisle proměnné je třeba provést v souladu s odstavcem 3.

Tab. 4.9: Zobrazování 3D sítí

Příkaz	Popis
mesh	3D síťovaný graf
meshc	3D síťovaný graf s vrstevnicemi
meshz	3D síťovaný graf s nulovou rovinou
waterfall	vodopádový 3D graf

Grafický systém MATLABu také umožňuje vykreslovat různé typy plošných 3D grafů. Pomocí příkazu **mesh** je graf vykreslen pomocí barevné sítě, která vznikne vzájemným propojením definovaných bodů čárami. Barevně jsou podle své velikosti odlišeny hodnoty závisle proměnné. Další možnosti poskytují příkazy **meshc**, **meshz** a **waterfall**, viz tab. 4.9.

```
Command Window
>> t=linspace(0,12*pi,10001);
>> plot3(sin(t),cos(t),t,...
    'LineWidth',2)
>> grid on
>> xlabel('osa x')
>> ylabel('osa y')
>> zlabel('osa z')
>>
```



Obr. 4.35: Spojitý 3D graf

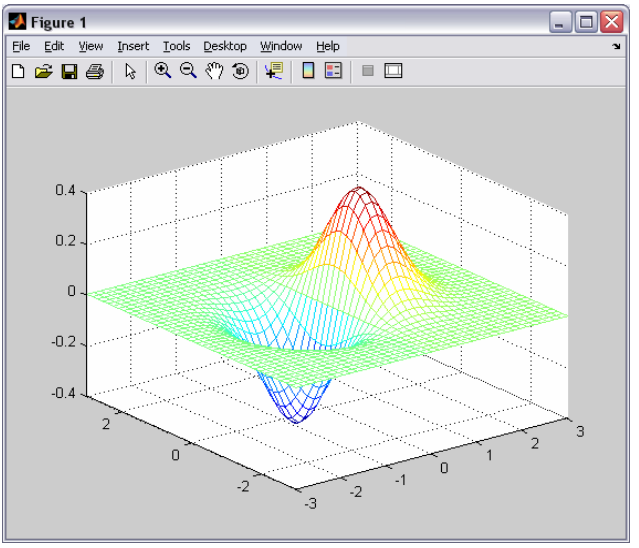
Při kreslení jakýchkoliv 3D grafů je nutné definovat nejprve pomocí příkazu **meshgrid** základní rastr (v podobě matice). V závisle proměnné je pak každému bodu tohoto rastru přiřazena příslušná funkční hodnota. Základní rastr je možné vytvořit např. ze dvou vektorů (obr. 4.36). Další možností je zadání vektoru pouze jednoho (obr. 4.37), rozsah obou nezávisle proměnných je pak totožný. V některých případech, je-li použit speciální příkaz, např. **peaks**, **sphere** či **cylinder**, není nutné rastr příkazem **meshgrid** definovat.

Na obr. 4.36 je pomocí příkazu **mesh** vykreslen průběh funkce $\mathbf{z} = \mathbf{x}\,\mathrm{e}^{(-\mathbf{x}^2-\mathbf{y}^2)}$. Pro definici vektorů obou nezávisle proměnných je použit příkaz **linspace**. Vektory jsou v tomto případě shodné. Hustota základního rastru (vytvořen příkazem **meshgrid**) je dána rozsahem resp. počtem prvků těchto vektorů. V argumentu příkazu **mesh** (platí obecně i pro ostatní příkazy) může být zadána pouze závisle proměnná. Zapíšeme-li tedy **mesh(Z)**, rozsah nezávisle proměnných je pak automaticky nastaven podle rozměru matice **Z**. Nastavení mezí os grafu lze provést pomocí příkazu **axis** (odstavec 4.3, tab. 4.6). Je ale nutné zadat meze i pro třetí osu. Syntaxe příkazu je následující: **axis([xmin xmax ymin ymax zmin zmax])**.

Příkaz **meshc** (použití je uvedeno na obr. 4.37) vykreslí, oproti základnímu příkazu **mesh**, do obrázku navíc ještě i vrstevnice. Barvy jednotlivých vrstevnic korespondují s barevným rozlišením hodnot závisle proměnné.

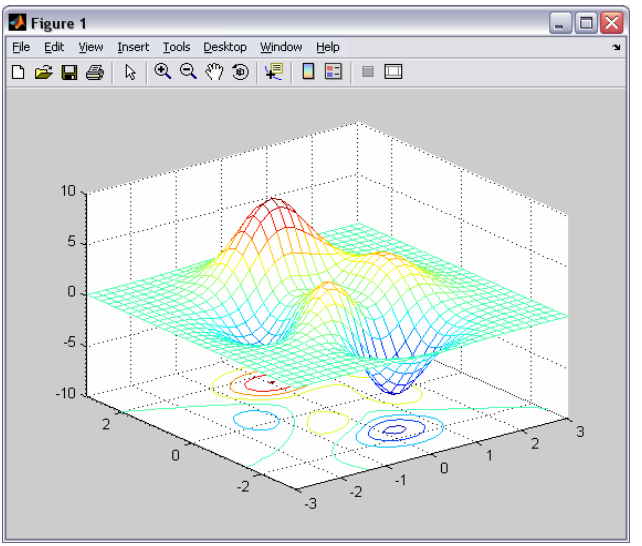
Výše uvedeným způsobem je pomocí příkazu **meshc**, vykreslen průběh v MATLABu předdefinované demonstrační funkce **peaks**. Základní rastr je možné vytvořit zadáním příkazu **peaks(X, Y)** nebo je také možné zadat v argumentu příkazu **peaks** pouze požadovaný rozměr

výsledné matice (obr. 4.38). Zapíšeme-li příkaz **peaks(n)**, bude matice mít rozměr n×n. Pokud zadáme příkaz **peaks** bez argumentu, vytvoří se matice rozměru 49×49.




```
Command Window
>> x=linspace(-3,3,41); y=x;
>> [X,Y]=meshgrid(x,y);
>> Z=X.*exp(-X.^2-Y.^2);
>> mesh(X,Y,Z);
>> axis([-3 3 -3 3 -0.4 0.4])
>>
```

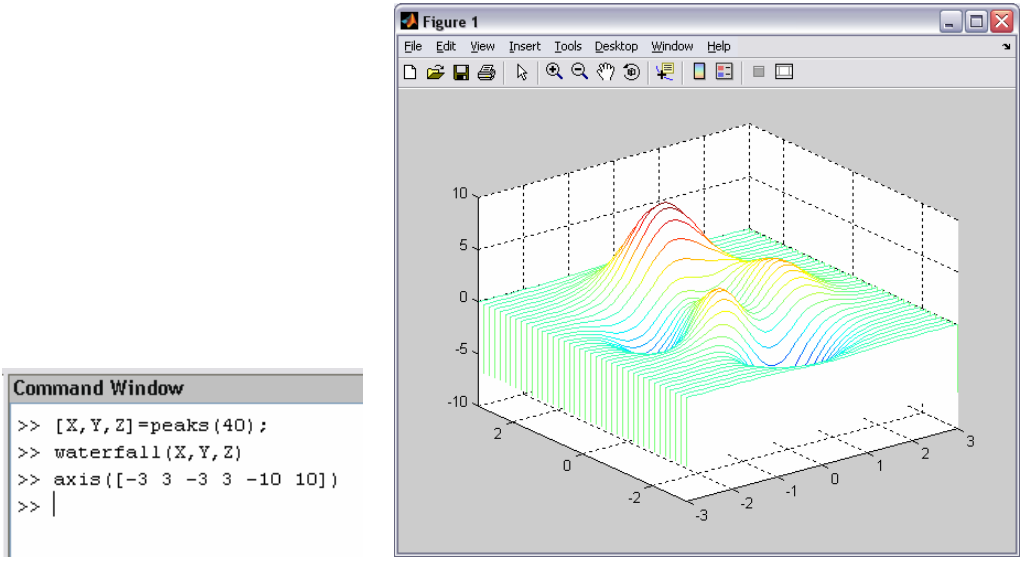
Obr. 4.36: 3D síťovaný graf




```
Command Window
>> [X,Y]=meshgrid(-3:0.2:3);
>> Z=peaks(X,Y);
>> meshc(X,Y,Z);
>> axis([-3 3 -3 3 -10 10])
>>
```

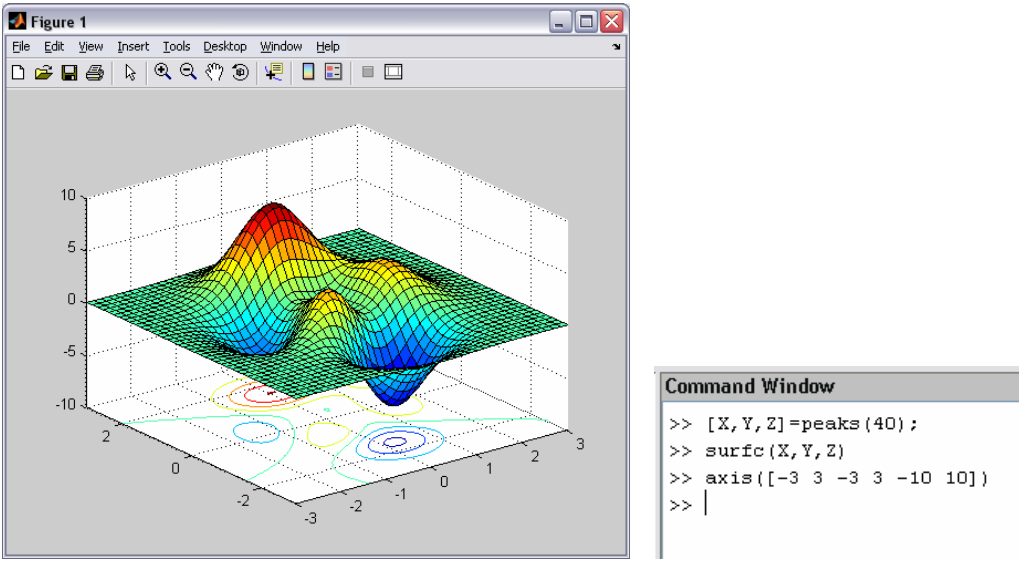
Obr. 4.37: 3D síťovaný graf s vrstevnicemi

Zajímavou variantou 3D síťovaného grafu je vodopádový graf, který je možné vykreslit příkazem **waterfall**, viz obr. 4.38. Syntaxe tohoto příkazu je prakticky shodná jako u příkazu **mesh**. Výsledný průběh je opět tvořen barevnými čárami, ale pouze v jednom směru. Barva těchto čar je, stejně jako ve všech předchozích případech, závislá na velikosti hodnot závisle proměnné.



Obr. 4.38: Vodopádový graf

Obdobou příkazu **mesh** je pro vykreslování celistvých 3D ploch příkaz **surf**. Při vykreslování povrchu jsou zobrazeny nejen propojovací čáry, ale také plochy v příslušné barvě. V tab. 4.10 jsou uvedeny další varianty tohoto příkazu. Příkaz **surfc** je obdobou příkazu **meshc**. Po jeho zadání dojde k vykreslení definované plochy a odpovídajících vrstevnic, viz obr. 4.39.

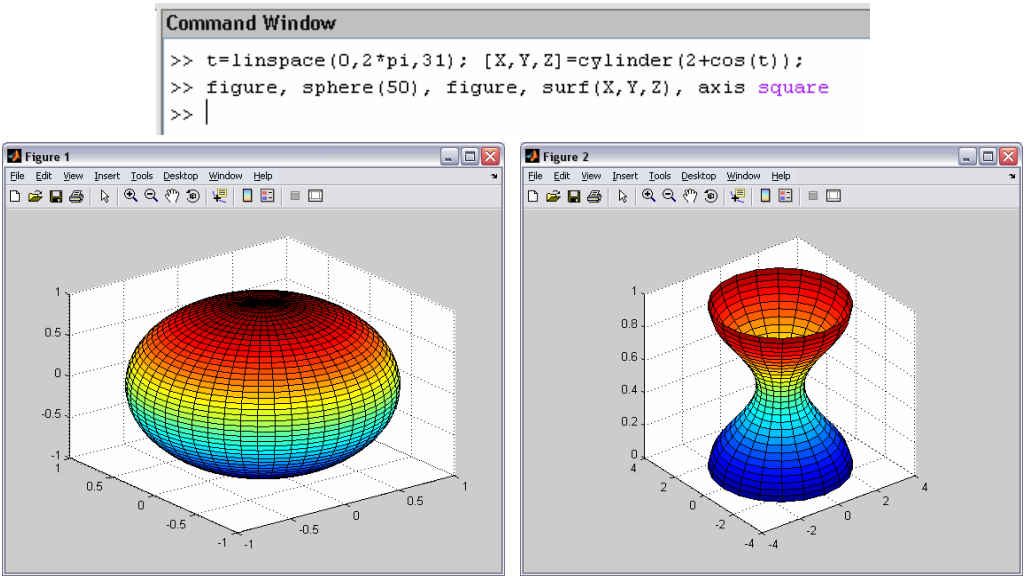


Obr. 4.39: 3D plošný graf

Tab. 4.10: Zobrazování 3D ploch

Příkaz	Popis
surf	stínovaná 3D plocha
surfc	stínovaná 3D plocha s vrstevnicemi
surfl	stínovaná 3D plocha s osvětlením

MATLAB poskytuje i několik předdefinovaných funkcí. Na obr. 4.40 je ukázáno použití příkazu **sphere**, po jehož zadání se vykreslí koule. Příkaz **cylinder** vykresluje povrch válce, který lze jednoduše modifikovat a získat tak velice zajímavé průběhy. Chceme-li na všech osách nastavit stejná měřítka, můžeme použít příkaz **axis square**.

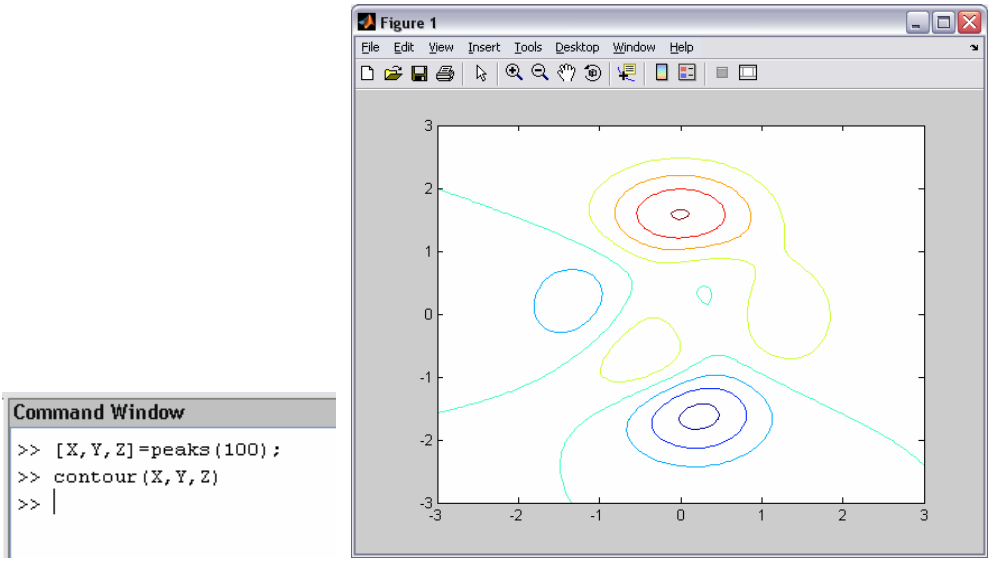


Obr. 4.40: Předdefinované funkce

Uživatel může také velmi efektivním způsobem vykreslovat vrstevnice nějaké 3D funkce. Pro kreslení vrstevnic je třeba nejprve definovat matici dat závisle proměnné. Tuto matici lze vytvořit např. pomocí již uvedeného příkazu **peaks**. K pouhému vykreslení vrstevnic slouží příkaz **contour** (obr. 4.41) resp. příkaz **contour3** pro vrstevnice ve 3D podobě. Jednotlivé křivky (vrstevnice) jsou navzájem barevně odlišeny podle hodnot závisle proměnné, stejně jako v případě **meshc** a **surfc**.

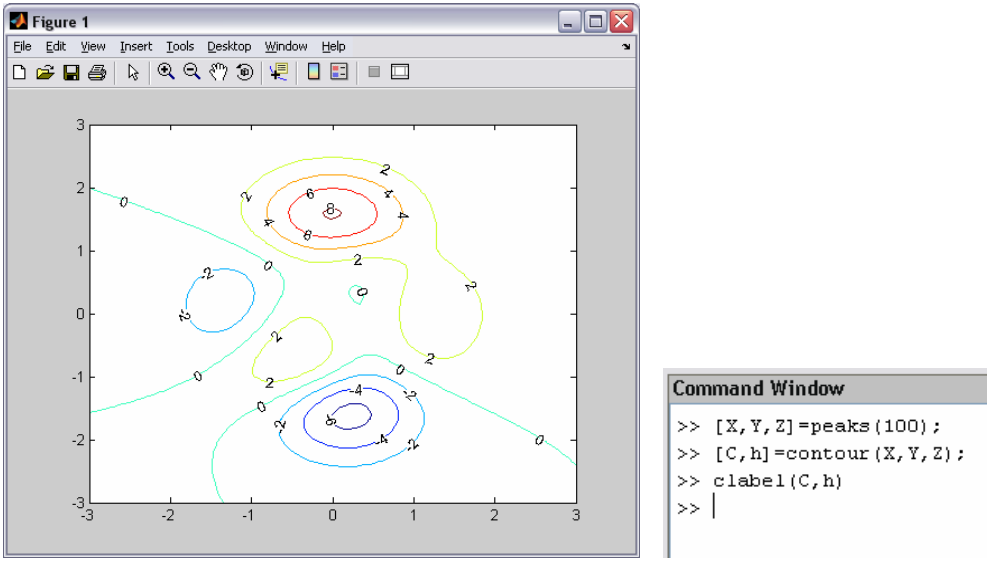
Tab. 4.11: Vykreslování vrstevnic

Příkaz	Popis
contour	vrstevnicový 2D graf
contour3	vrstevnicový 3D graf
contourf	vyplněný vrstevnicový 2D graf
clabel	umístí popisků vrstevnic do grafu

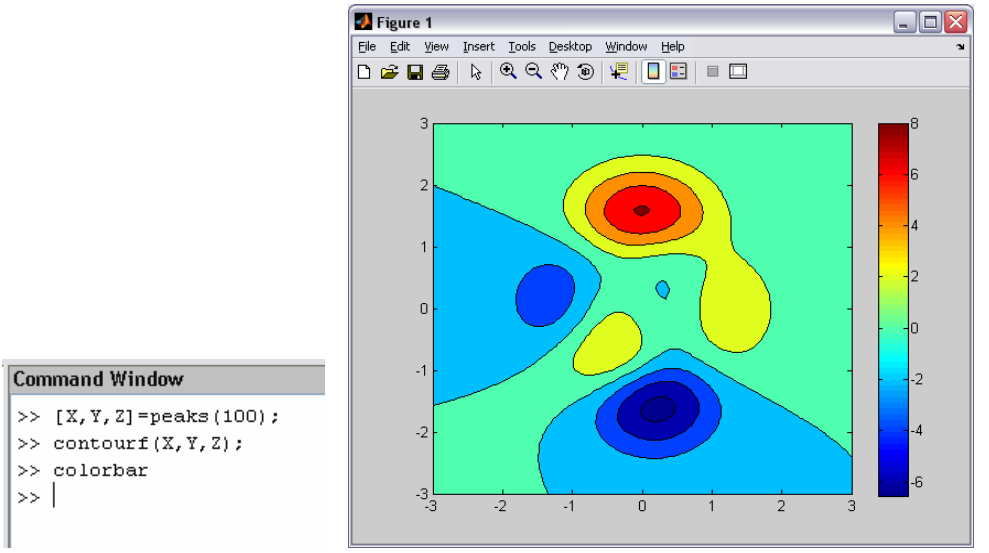


Obr. 4.41: Vrstevnicový graf

Do vrstevnicového grafu je možné přidat čísla reprezentující příslušné hodnoty (hladiny) závisle proměnné, viz obr. 4.42. Pro vkládání popisek je definován příkaz **clabel**. Příkaz **contour** je pak třeba zapsat ve tvaru **[C, h] = contour(X, Y, Z)**, tedy tak, aby navíc vracel i hodnoty pro vkládání popisek. Syntaxe příkazu je **clabel(C, h)**. Další variantu vrstevnicového grafu obdržíme pomocí příkazu **contourf**. V tomto případě jsou navíc plochy mezi vrstevnicemi barevně vyplněné (obr. 4.43). Příkazem **colorbar** lze graf doplnit o tzv. barevnou škálu (pruh vedle osového systému), která odpovídá ose závisle proměnné a obsahuje i konkrétní hodnoty.

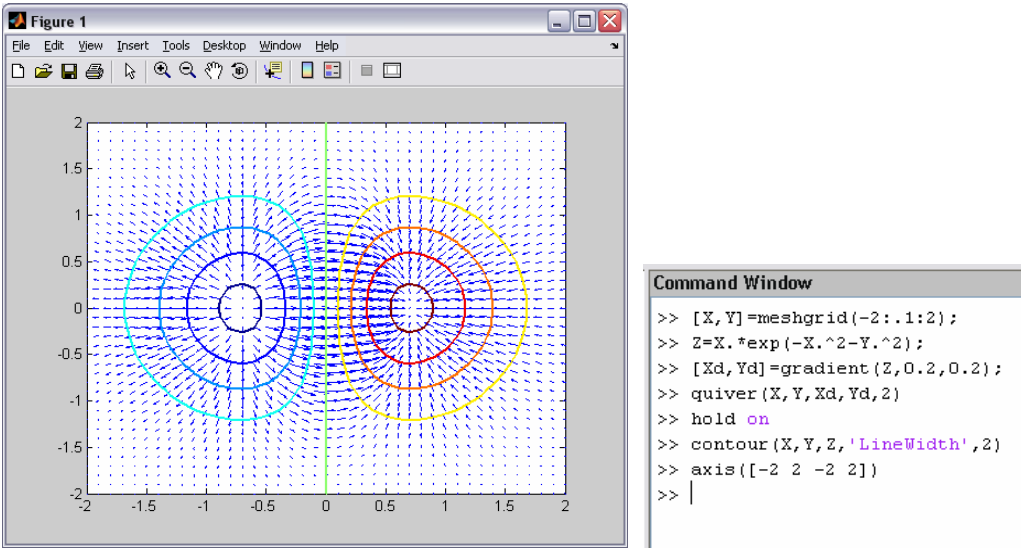


Obr. 4.42: Vrstevnicový graf s popisky vrstevnic



Obr. 4.43: Vyplněný vrstevnicový graf s barevnou škálou

Tabulka 4.12 podává přehled o specializovaných třírozměrných grafech. Příkazy **quiver3**, **fill3**, **stem3** a **comet3** představují 3D varianty typů grafů popisovaných již v odstavci 4.6 (tab. 4.8).



Obr. 4.44: Graf vektorového pole s vrstevnicemi

Zajímavý typ grafu vznikne použitím příkazu **quiver**. Pomocí funkce **gradient** můžeme např. vypočítat gradient ve směru souřadných os x a y funkce dvou proměnných (obr. 4.36). Prostřednictvím příkazu **quiver** pak lze toto pole pro definovanou funkci zobrazit. Výsledný graf je možné navíc pomocí příkazu **contour** doplnit i o vrstevnice, viz obr. 4.44.

Tab. 4.12: Specializované 3D grafy

Příkaz	Popis
quiver3	3D graf vektorového pole
fill3	vykreslí vyplněný 3D mnohoúhelník
stem3	3D graf diskrétních posloupností (stopkový 3D graf)
comet3	vytváří graf pohybujícím se bodem ve 3D
slice	objemová vizualizace pomocí řezů (plátkový 3D graf)
ribbon	proužkový 3D graf

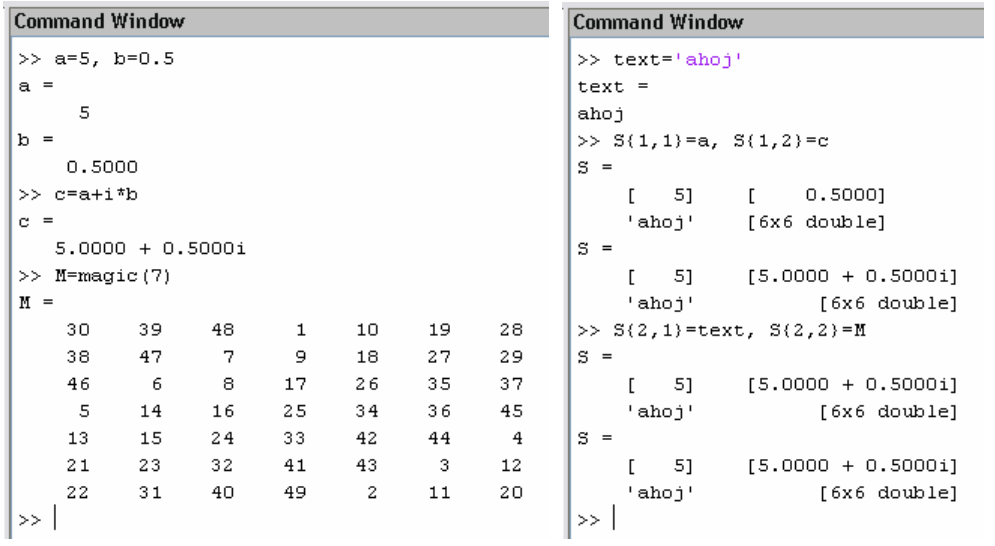
Chceme-li okno s vykresleným grafem zavřít z příkazové řádky, můžeme použít příkaz **close** nebo **close(h)**, kde h je číslo obrázku, které je uvedeno jeho horní liště (např. Figure No. 1) a identifikuje tak každé okno. Pokud máme otevřeno více oken a chceme zavřít všechna najednou, použijeme příkaz **close all**. K vymazání obsahu okna s grafem lze použít příkazy **clf** nebo **clf reset**. Příkaz **clf reset** resetuje všechny vlastnosti okna (figure) kromě zvolených jednotek a jeho polohy. Vysunutí aktuálního okna do popředí na obrazovce zajišťuje příkaz **shg**.

5 Práce se soubory a tvorba skriptů

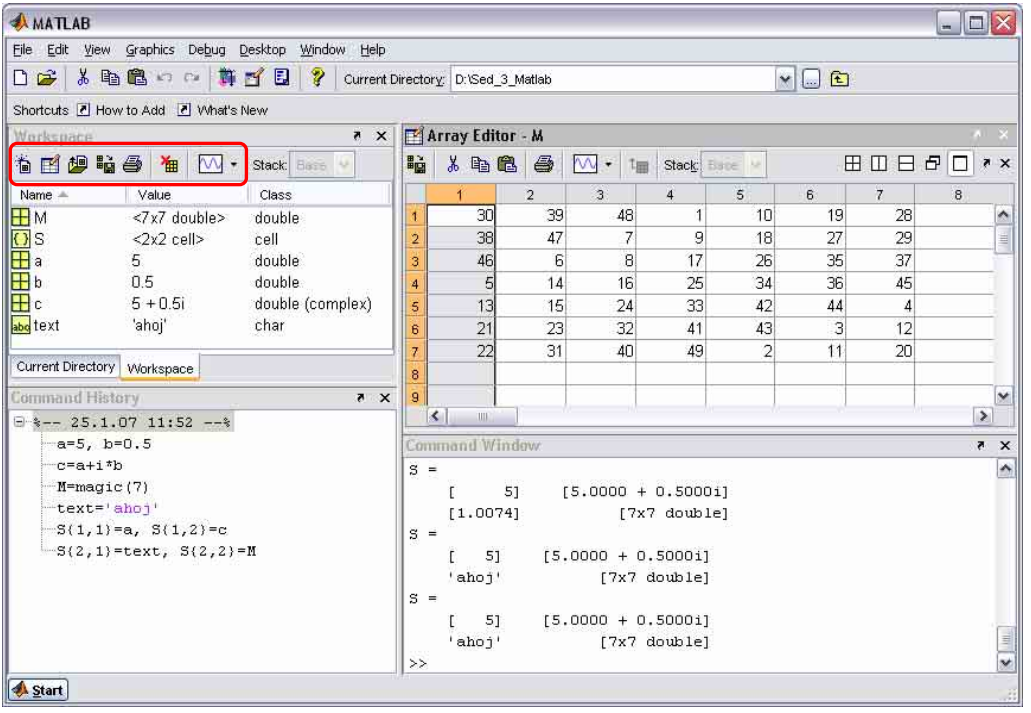
Po ukončení práce s prostředím MATLAB dojde k nevratnému vymazání všech proměnných, které uživatel definoval. Pokud ale tyto proměnné uživatel potřebuje např. při dalším spuštění MATLABu či je chce přenést do jiného počítače, musí tyto proměnné uložit do souboru. Často je také třeba zpracovat data získaná měřením, provést jejich grafické zobrazení, regresi apod. Prostředí MATLAB umožňuje pracovat s nejrůznějšími typy souborů a jak bude ukázáno dále, zpracování naměřených dat je v tomto prostředí velmi jednoduché a elegantní.

Uložení nejrůznějších dat je možné do speciálních souborů MATLABu s příponou MAT. Jedná se o binární soubory a struktura proměnných nehraje žádnou roli. K ukládání slouží příkaz **save** a pokud jej použijeme bez parametrů, dojde k uložení celého prostoru proměnných (*Workspace*) do souboru matlab.mat. Soubor je umístěn v aktuálním adresáři. K uložení celého prostoru proměnných lze také použít položku **Save Workspace as...** z menu **File** okna *Command Window*. Stejným způsobem je možné i uložené proměnné vyvolat. K načtení proměnných ze souboru matlab.mat použijeme příkaz **load** (opět bez parametrů). Další možností je, obdobně jako v předchozím případě, výběr položky **Import Data...** z menu **File**.

Vytvořme nyní několik proměnných a proveďme jejich uložení a opětovné načtení výše uvedeným způsobem. Způsob definování proměnných byl již podrobně popsán v kapitole 2. Na obr. 5.1 je uvedeno vytvoření šesti proměnných – tři skalární proměnné, pole (matice), řetězec a struktura. Tyto proměnné jsou po vytvoření viditelné v okně *Workspace*, viz obr. 5.2. V okně je znázorněna i jejich struktura a typ.

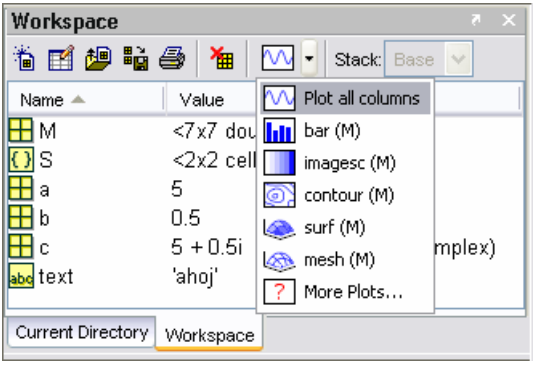


Obr. 5.1: Definice proměnných různých typů

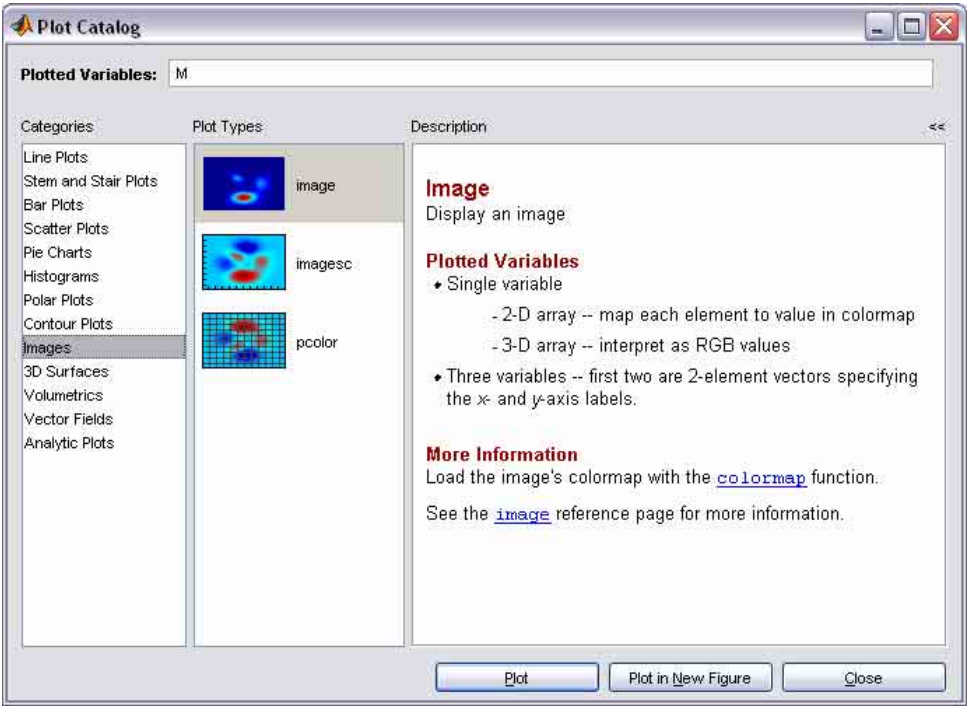


Obr. 5.2: Interaktivní práce s proměnnými, Array Editor

Okno prostoru proměnných *Workspace* (obr. 5.2) obsahuje samozřejmě i několik funkcí pro interaktivní práci s definovanými proměnnými. Proměnné je možné zobrazit v *Array Editoru*. Po dvojitém kliknutí na zvolenou proměnnou, v našem případě matice **M**, se otevře okno editoru a přímo se zapojí (dock) do základního okna MATLABu. V editoru je interaktivním způsobem možné měnit jednotlivé hodnoty a případně je i okamžitě vykreslit, viz obr. 5.3. Zvolíme-li v roletovém menu položku **More Plots...**, dojde k otevření dalšího okna *Plot Catalog* (obr. 5.4), ve kterém lze opět interaktivním způsobem vybrat zvolený typ grafu a okamžitě jej vykreslit. Samozřejmě není možné zvolit jakýkoliv typ grafu, záleží na povaze a struktuře dat.

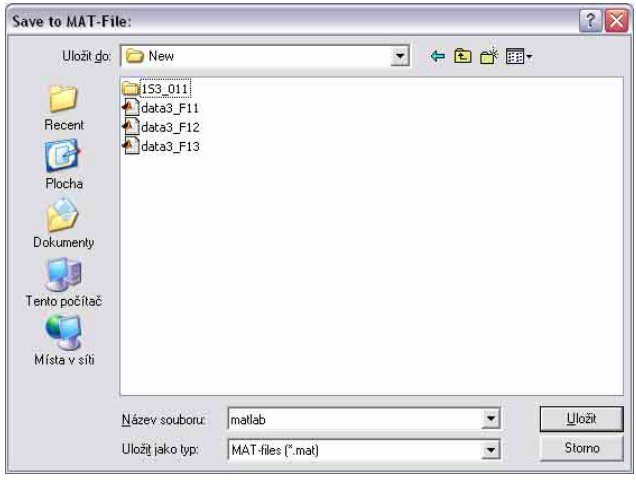


Obr. 5.3: Možnosti okamžitého vykreslení hodnot



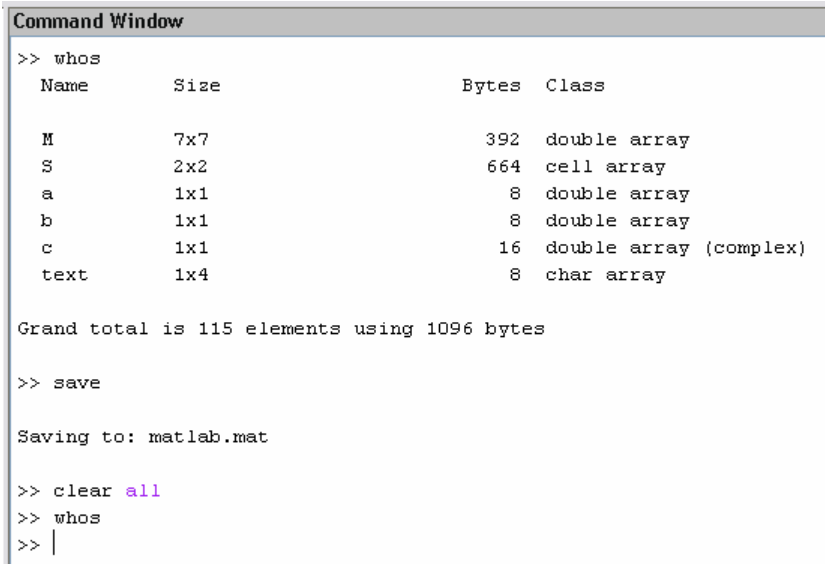
Obr. 5.4: Interaktivní vykreslování dat

Vraťme se ale k možnostem ukládání proměnných do souboru. V okně *Workspace* je také možné prostřednictvím k tomu určené ikony proměnné uložit. Po kliknutí na ikonu lze všechny proměnné uložit do binárního souboru s příponou MAT. V dialogovém okně na obr. 5.5 pouze zvolíme standardním způsobem cílový adresář a název souboru. Ponecháme-li název matlab.mat, budeme moci později proměnné otevřít pomocí příkazu **load**.



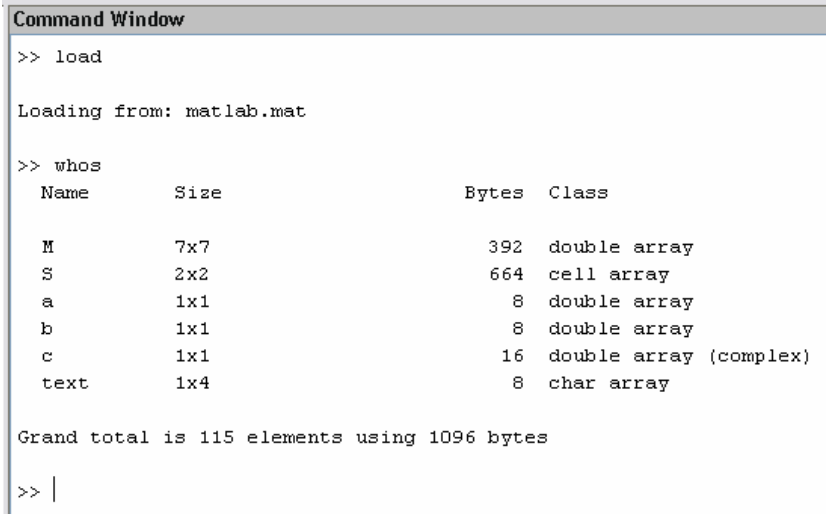
Obr. 5.5: Dialogové okno pro ukládání proměnných

Proměnné vytvořené na obr. 5.1 nyní uložíme příkazem **save** (bez parametrů) do souboru, který bude umístěn v aktuální složce. Následně provedeme vymazání všech definovaných proměnných příkazem **clear all** a pomocí příkazu **whos** se přesvědčíme, zda vymazání skutečně proběhlo. Je zřejmé, že ani v okně *Workspace* nebudou žádné proměnné.



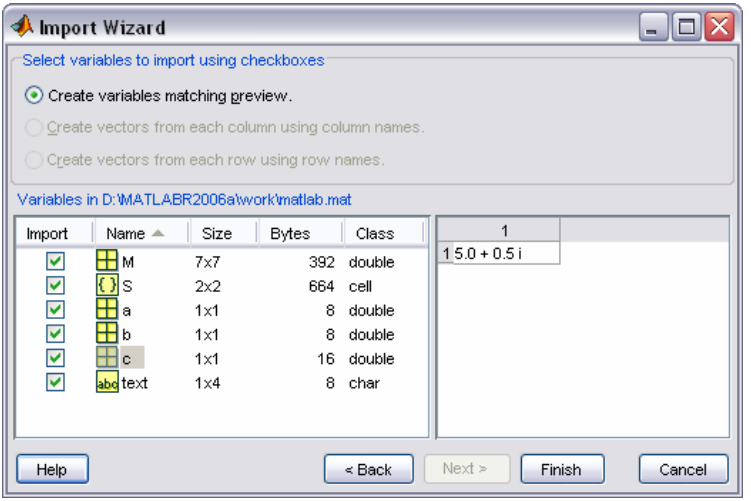
Obr. 5.6: Uložení proměnných do souboru matlab.mat v aktuální složce

Proměnné, které jsme uložili do souboru matlab.mat můžeme načíst příkazem **load** (také bez jakýchkoliv parametrů) a opět se přesvědčit pomocí **whos**, zda jsou proměnné skutečně načtené a odpovídají těm ukládaným, viz obr. 5.7.



Obr. 5.7: Načtení proměnných ze souboru matlab.mat v aktuální složce

Jinou možností je načtení proměnných ze souboru interaktivním způsobem pomocí položky **Import Data...** hlavního menu MATLABu nebo pomocí ikony v okně *Workspace* (obr. 5.2). V tomto případě dojde bezprostředně po výběru příslušného souboru v dialogovém okně *Import Data* (je podobné oknu *Save to MAT-file* na obr. 5.5) k otevření okna průvodce *Import Wizard*, viz obr. 5.8. V průvodci je možné zvolit proměnné, které chceme importovat a k dispozici je i náhled (preview). Implicitně jsou vybrány všechny proměnné. Pokud některou z nich nechceme importovat, zrušíme její výběr v příslušném check boxu v části *Import*.



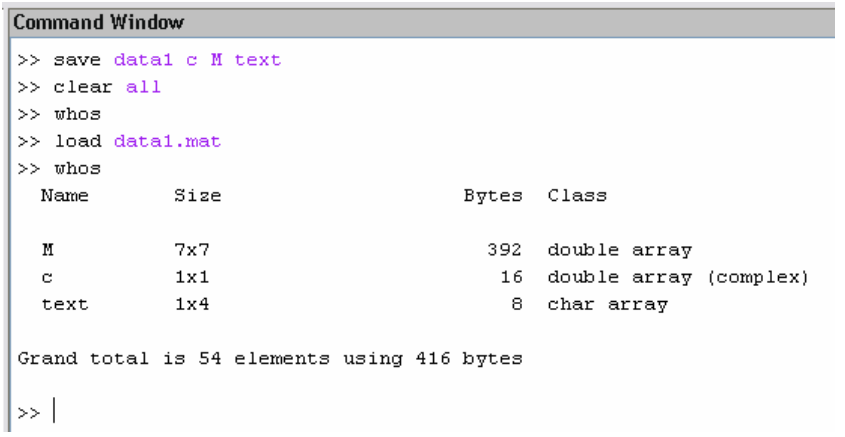
Obr. 5.8: Průvodce importem proměnných

5.1 Základy práce se soubory

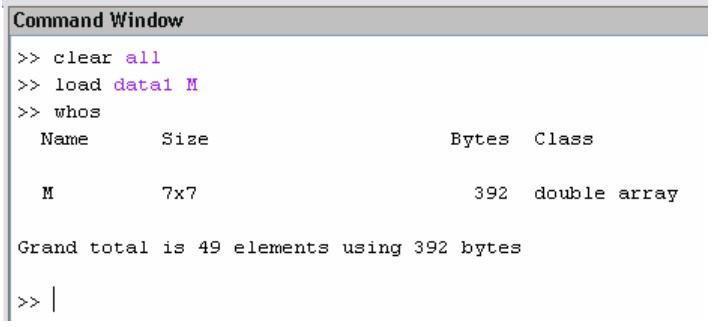
Výše popsaný způsob uložení proměnných do souboru samozřejmě není jedinou možností MATLABu jak zálohovat či načítat data. Není ani účelné vždy ukládat veškeré proměnné, které uživatel v průběhu práce s MATLABem nadefinuje. Příkaz **save** lze také zadat s parametry, které přímo určují název souboru, případně i adresář, do kterého chceme soubor umístit, ale především výčet proměnných pro uložení. Z proměnných, které jsme již vytvořili na obr. 5.1 uložíme zapsáním příkazu **save data1 c M text** do souboru data1.mat pouze proměnné c, M a text, viz obr. 5.9. Pokud nechceme data uložit do aktuálního adresáře, zapíšeme v příkazu i úplnou cestu, např. **save C:\New\data1 c M text**.

Příkaz k načtení dat můžeme zadat pouze s názvem souboru (obr. 5.9, příponu není nutné uvádět), tedy **load data1**. Nebo lze zadat v parametrech příkazu i konkrétní proměnnou, kterou chceme načíst – syntaxe pro načtení např. matice M může být následující **load data1 M**, viz obr. 5.10.

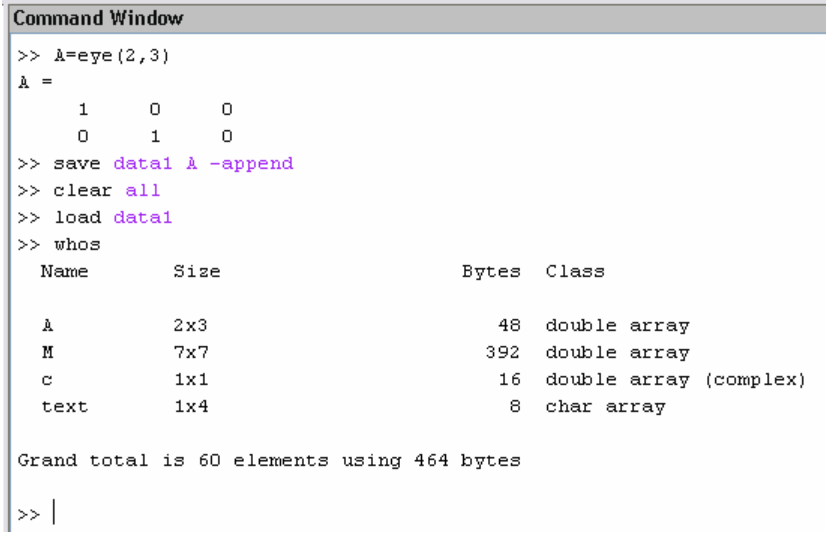
Další proměnné lze také přidávat do souborů, které již existují a to bez rizika přepsání proměnných v souboru původně uložených. Příkaz **save** stačí doplnit o parametr **–append**. Na obr. 5.11 je takto nově vytvořená proměnná A (matice definovaná pomocí **eye**). Tímto způsobem lze připojit proměnné do existujícího souboru i opakovaně.



Obr. 5.9: Uložení vybraných proměnných



Obr. 5.10: Načtení vybraných proměnných



Obr. 5.11: Přidání nové proměnné do existujícího souboru

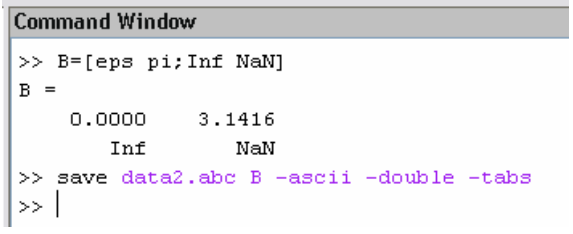
Z důvodu zpětné kompatibility je soubory určené ke čtení v nižších verzích MATLABu (ve verzi 6.5 a nižší) třeba uložit s přepínačem **–v6**. Syntaxe použití přepínače je obdobná jako na obr. 5.11. Možné je i zachování kompatibility s verzemi 4.x, k příkazu přidáme přepínač **–v4**.

MATLAB pracuje i s textovými soubory ve formátu ASCII. Tyto soubory mají většinou příponu TXT, nicméně to není nutnou podmínkou a lze použít i příponu jinou. Pro ukládání textových souborů zapisujeme příkaz **save** obvyklým způsobem, jen jej rozšíříme o přepínač **–ascii**. Tabulka 5.1 uvádí všechny přepínače, které je možné při ukládání textových souborů použít.

Tab. 5.1: Možnosti ukládání textových souborů

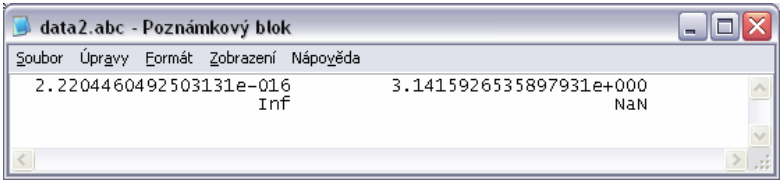
Kombinace přepínačů	Popis
–ascii	ASCII standard s přesností čísla na 8 míst, oddělovač je mezera
–ascii –double	ASCII standard s přesností čísla na 16 míst
–ascii –tabs	ASCII standard, oddělovač je tabelátor
–ascii –double –tabs	ASCII standard s přesností čísla na 16 míst, oddělovač je tabelátor

Matici B definovanou na obr. 5.12 uložíme v textovém ASCII formátu do souboru data2.abc, jako oddělovač zvolíme tabelátor. Syntaxe příkazu je **save data2.abc B –ascii –double –tabs**. Soubor lze otevřít ve standardním textovém editoru, např. v poznámkovém bloku obsaženém v operačním systému Windows, viz obr. 5.13.



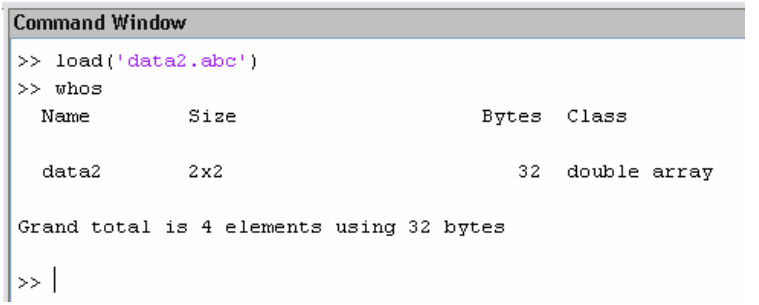
Obr. 5.12: Uložení textového souboru

Zápis příkazu může být také ve tvaru funkce, tedy např. **save('data2.abc','B','–ascii')**. Pokud chceme naopak data z textového souboru načíst, zapíšeme příkaz **load data2.abc** resp. **load('data2.abc')**.



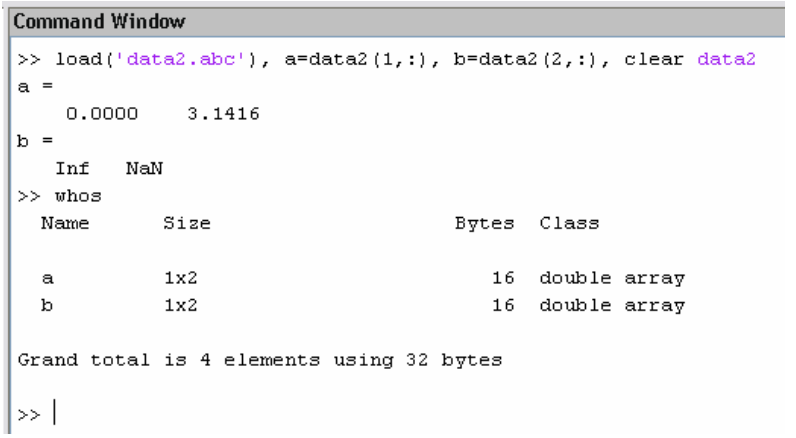
Obr. 5.13: Struktura souboru po otevření v textovém editoru

Po načtení dat není ale v prostoru proměnných původní matice B, ale proměnná s názvem souboru, z kterého data načítáme. V našem případě je to proměnná data2, viz obr. 5.14. Pokud chceme název jiný, musíme provést následně přiřazení jiné proměnné a tu původní pomocí příkazu **clear** příp. vymazat, tak jako na obr. 5.15.



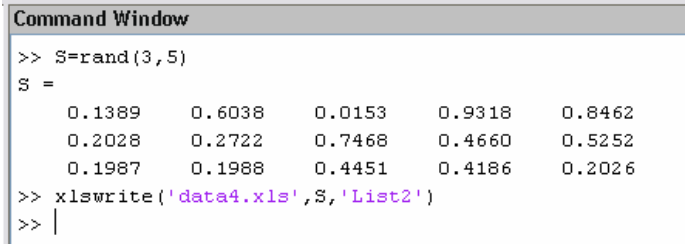
Obr. 5.14: Čtení dat z textového souboru

Uvedený způsob čtení z textového souboru je výhodný zejména při zpracování naměřených dat, která bývají velmi často uložena v textovém formátu. Obvykle je v souboru uložena matice, jejíž jednotlivé sloupce resp. řádky odpovídají např. hodnotám získaným různými měřícími přístroji a po načtení do prostředí MATLABu je možné velmi efektivně tato data zpracovávat a následně i graficky interpretovat. Na obr. 5.15 je načtena datová matice z textového souboru a její řádky jsou následně přiřazeny dvěma proměnným.

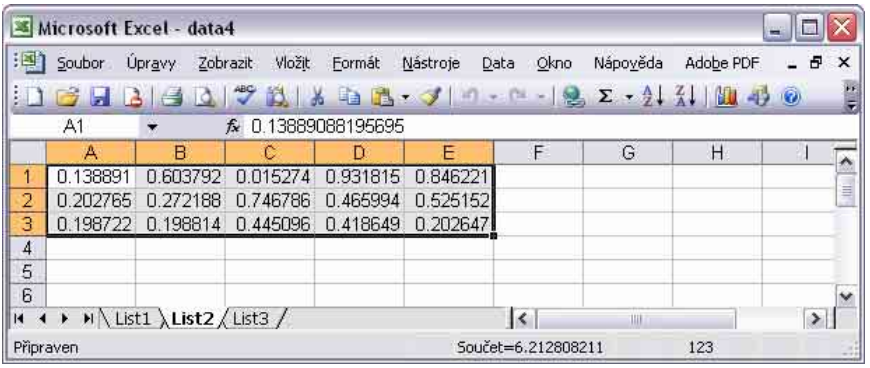


Obr. 5.15: Čtení dat z textového souboru a jejich přiřazení proměnným

Práci značně usnadňuje vytváření skriptů, v rámci nichž lze naměřená data např. filtrovat, převzorkovávat, porovnávat se simulací apod. Základní pravidla pro vytváření skriptů budou uvedena v kapitole 5.2.

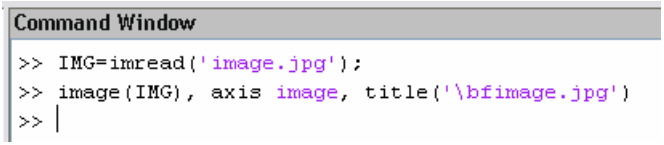


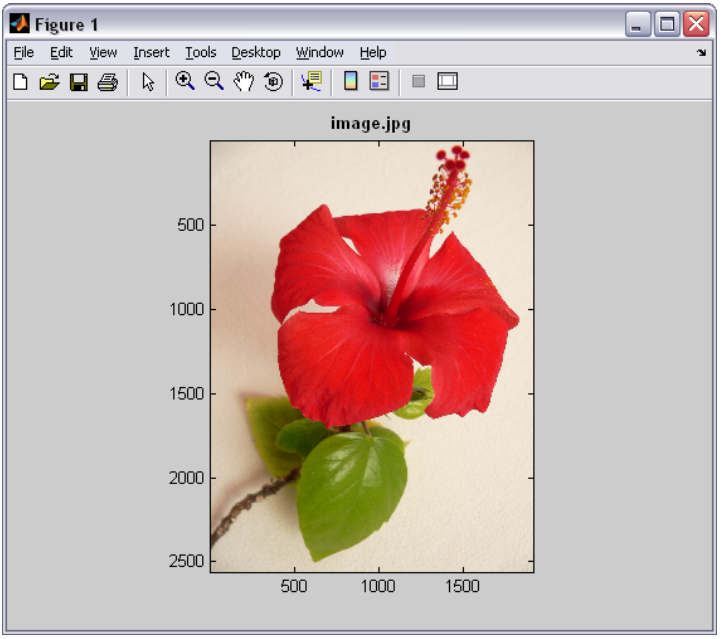
Obr. 5.16: Uložení dat ve formátu XLS



Obr. 5.17: Struktura uložených dat v programu Excel

Data lze také číst a zapisovat ve formátu XLS (soubory tabulkového procesoru Microsoft Excel). Čtení ze souboru zajišťuje příkaz **xlsread**. Syntaxe tohoto příkazu může být např. **M = xlsread('data3.xls')** nebo **N = xlsread('data3.xls', 'list3', 'b2:g10')**, chceme-li ze souboru data3.xls číst data v rozsahu buněk b2 až g10 v listu 3. Zápis do souboru tohoto typu je možný prostřednictvím příkazu **xlswrite**. Na obr. 5.16 je takto zapisována předem vytvořená matice do souboru data4.xls a na obr. 5.17 jsou již uložená data zobrazena přímo v programu Excel.





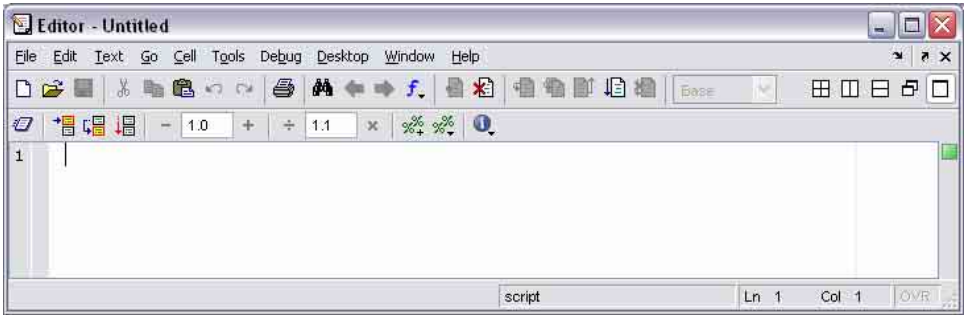
Obr. 5.18: Načtení a zobrazení rastrového obrázku

MATLAB také umožňuje pracovat s rastrovými obrázky. Jejich načtení zajistí příkaz **imread**. Podporované formáty jsou uvedeny v nápovědě (**doc imread**). K následnému vykreslení se pak použije příkaz **image** nebo **imagesc**, viz obr. 5.18. Vykreslení je realizováno v okně *Figure* a uživatel může výsledný obrázek doplnit standardním způsobem (viz kap. 4) např. o titulek. Pomocí příkazu **axis image** zajistíme, aby proporce načteného obrázku byly zachovány. Zápis rastrového obrázku do souboru ve specifikovaném formátu lze provést příkazem **imwrite**. Tato problematika ale již přesahuje rozsah tohoto textu, více se uživatel dozví v nápovědě zadáním **doc imread** a **doc imwrite**.

V MATLABu můžeme číst i zvukové soubory ve formátu WAV (Microsoft Wave), pomocí příkazu **wavread**. Syntaxe příkazu pro načtení souboru s názvem zvuk.wav je následující **[y, fvz] = wavread('zvuk.wav')**. V proměnné **y** je uložen vektor zvukových dat a v proměnné **fvz** příslušná vzorkovací frekvence. Vektor zvukových dat pak lze přehrát (je-li v počítači instalována zvuková karta) zadáním příkazu **sound(y)**. Zadáme-li příkaz ve tvaru **sound(y,fvz)**, dojde k přehrání vektoru **y** „rychlostí" udávanou proměnnou **fvz**. Analogicky lze zvuková data pomocí příkazu **wavwrite** ve formátu WAV i uložit.

5.2 Tvorba skriptů

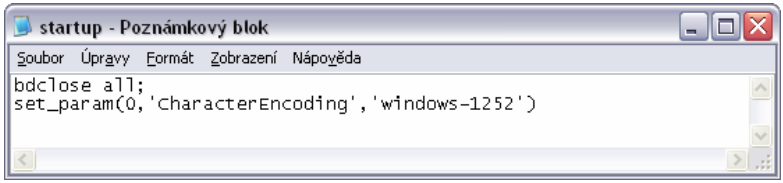
Práci v MATLABu značně usnadňuje vytváření vlastních programů – skriptů. Tato skutečnost dovoluje uživateli při řešení rozsáhlejších problémů zapisovat potřebné příkazy do skriptů, které lze uložit na disk počítače a později kdykoliv vyvolat. Uživatelem vytvořené skripty se ukládají do souborů s příponou *m* a z tohoto důvodu jsou i často označovány jako m-soubory (m-files). Jinými slovy je skript tedy takový textový m-soubor obsahující seznam příkazů MATLABu, který po zavolání v příkazovém řádku jednotlivé příkazy postupně zpracovává. Volání skriptu se realizuje prostým zapsáním jeho jména (bez přípony).



Obr. 5.19: Okno editoru skriptů

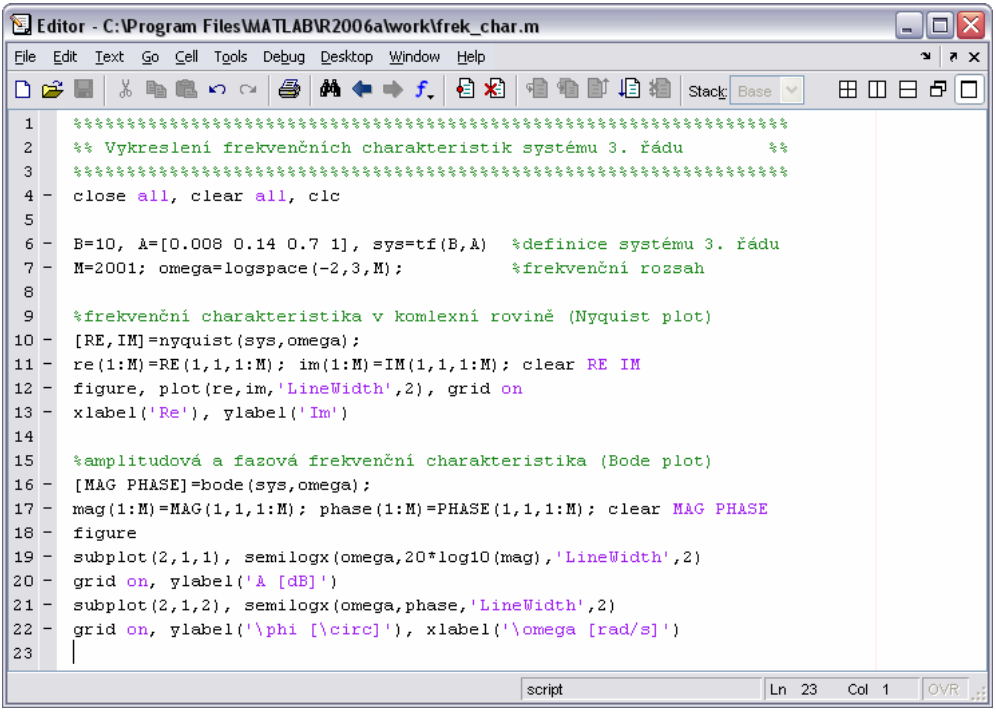
Uživatel skripty vytváří ve speciálním editoru (obr. 5.19), jelikož se ale jedná o textové soubory, je možné je prohlížet nebo i editovat běžným textovým editorem. Vestavěný editor ale poskytuje užitečné funkce a lze z něho i vytvářené skripty přímo spouštět. Editor vyvoláme z nabídky základního okna matlabu **File → New → M-File**, případně pomocí ikony umístěné v toolbaru okna zcela vlevo (symbol prázdného listu). K vyvolání editoru dojde i v případě, že otevíráme již existující m-soubor, ať již z menu **File → Open...** (zvolíme soubor s příponou *m*) nebo pomocí ikony v toolbaru.

Před začátkem vlastního zápisu skriptu je vhodné provést nejprve uložení m-souboru na disk počítače. Uložení se provádí standardním způsobem v menu **File → Save As...**, průběžné ukládání je později možné např. prostřednictvím klávesové zkratky Ctrl + S. Název m-souboru může obsahovat až 31 znaků. Znaky nad tento limit MATLAB ignoruje. Název souboru musí začínat písmenem, za kterým se pak můžou vyskytovat písmena, číslice a podtržítka, tak jako u proměnných.



Obr. 5.20: Skript startup.m otevřený ve standardním textovém editoru

V MATLABu jsou obdobným způsobem, tedy pomocí skriptů, vytvářeny veškeré funkce. K pravidlům pro tvorbu vlastních funkcí majících vstupní a výstupní parametry se vrátíme ještě později. Při spuštění MATLABu se automaticky zavolá skript s názvem startup.m (pokud soubor na disku existuje), v němž může uživatel nastavit požadované parametry, např. změnit používanou znakovou sadu. Popisovaný skript s volbou jiné znakové sady může vypadat jako na obr. 5.20. Běžně se ale tento soubor na disku počítače nenachází a MATLAB je spouštěn s implicitním nastavením. Soubor je třeba tedy po vytvoření umístit do podsložky MATLABu; nezměnil-li uživatel nastavení, tak do C:\Program Files\MATLAB\R2006a\toolbox\local.



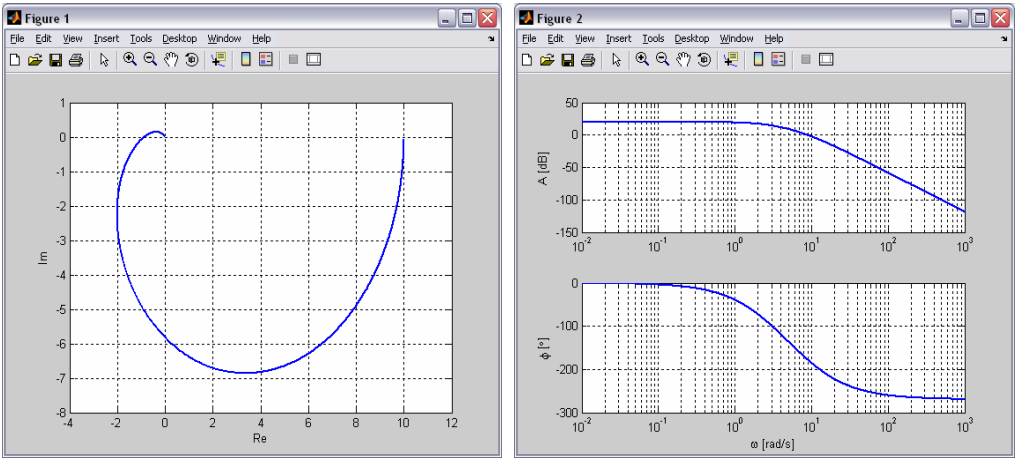
Obr. 5.21: Jednoduchý skript

Jak již bylo řečeno, tak do skriptu lze zapisovat jakékoliv příkazy MATLABu. Skript je ale také možné doplnit o komentáře, které jsou uvozené znakem procenta (**%**). Komentáře mohou být na samostatných řádcích, ale i za běžnými příkazy. Všechny ve skriptu volané funkce pracují s daty uloženými v prostoru proměnných. Nové proměnné je před jejich použitím možné přímo ve skriptu definovat. Na obr. 5.21 je ukázka jednoduchého skriptu, který vypočítá a následně i zobrazí frekvenční charakteristiky systému definovaného obrazovým přenosem. Skript spustíme prostřednictvím menu **Debug**, ve kterém zvolíme položku **Run** případně pomocí funkční klávesy F5. Další možností je spuštění pomocí ikony v toolboxu okna editoru. Odezva MATLABu po vykonání uvedeného skriptu je na obr. 5.22. Vypsány jsou pouze polynomy čitatele a jmenovatele a obrazový přenos systému. U ostatních příkazů byl úmyslně výstup potlačen pomocí středníku, obdobně jako při běžném zápisu v příkazovém řádku okna *Command Window*.

```
Command Window

B =
    10
A =
    0.0080    0.1400    0.7000    1.0000

Transfer function:
                10
---------------------------------
0.008 s^3 + 0.14 s^2 + 0.7 s + 1

>>
```

Obr. 5.22: Odezva MATLABu po vykonání skriptu

Po dokončení skriptu se také zobrazí výsledné průběhy frekvenčních charakteristik. Na obr. 5.23 je uvedena frekvenční charakteristika v komplexní (Gaussově) rovině a také frekvenční charakteristiky (amplitudová a fázová) v logaritmických souřadnicích.



Obr. 5.23: Frekvenční charakteristiky vykreslené pomocí uživatelského skriptu

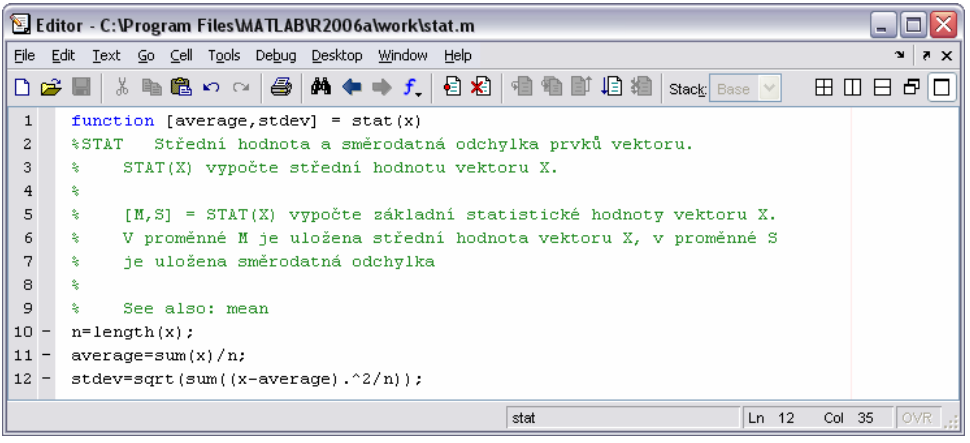
Ne vždy musí uživatelem definovaný skript sloužit pouze k vykonání posloupnosti příkazů. Lze také vytvářet vlastní funkce, které mají vstupní a výstupní parametry a je možné je volat obdobně jako ostatní funkce MATLABu. Aby byla funkce spustitelná, musí splňovat několik kritérií. Název m-souboru, ve kterém bude funkce zapsána a jméno vlastní funkce, které je na prvním řádku souboru má být shodné. Jméno funkce může mít libovolný počet znaků, MATLAB ale bere v úvahu pouze prvních 63 a ostatní ignoruje. Jméno funkce musí začínat písmenem a je třeba používat pouze malá písmena.

Při zápisu funkce je na prvním řádku m-souboru deklarace funkce, která musí obsahovat slovo ***function***. Vstupní a výstupní proměnné v prvním řádku, jsou lokální proměnné funkce. Vstupní proměnné obsahují data předaná funkci a výstupní proměnné obsahují data vrácená funkcí. Následující řádky uvozené procentem (**%**) jsou komentáře sloužící pro nápovědu. Ta se zobrazí standardním způsobem, zapíšeme-li **help <*funkce*>**. První z těchto řádků označovaný H1 většinou obsahuje jméno funkce psané velkými písmeny a stručný popis účelu funkce. Velká písmena jsou použita pouze k zvýraznění, funkci je nutné při jejím volání zapsat malými písmeny, jinak MATLAB ohlásí chybu.

Všechny další řádky funkce tvoří tělo funkce. Tělo obsahuje příkazy MATLABu, které zpracovávají vstupní argumenty a výsledky ukládají do výstupních argumentů funkce. Běh funkce končí poté, co je zpracována poslední řádka m-souboru nebo příkazem **return** kdekoliv v těle funkce.

Funkce může být také přerušena voláním příkazu **error**. Tento příkaz slouží k signalizaci nesprávného použití funkce. Funkce může pomocí příkazu **warning** podat varovné hlášení a poté dále pokračovat. Rozdíl mezi příkazem **warning** a **disp** (také slouží k výpisu hlášení) je v tom, že varovné hlášení lze globálně povolit nebo potlačit zadáním příkazu **warning on** resp. **warning off**. Použití příkazů **error** a **warning** je obdobné, v argumentu příkazu je uveden mezi uvozovkami text, který má být v okně *Command Window* zobrazen.

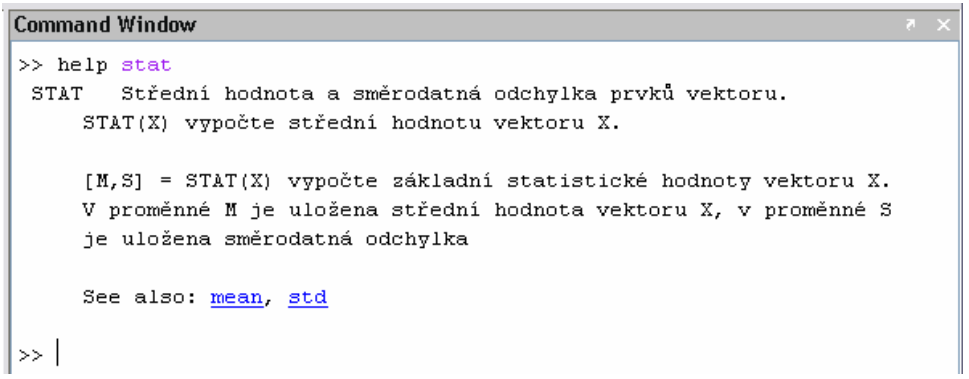
Pokusme se nyní na základě uvedených pravidel vytvořit vlastní funkci např. pro výpočet střední hodnoty prvků vektoru. Funkce by měla navíc signalizovat chybu, pokud vstupním argumentem nebude vektor. Zápis takové funkce, prozatím ale bez signalizace chyby, je na obr. 5.24.



Obr. 5.24: Zápis jednoduché funkce

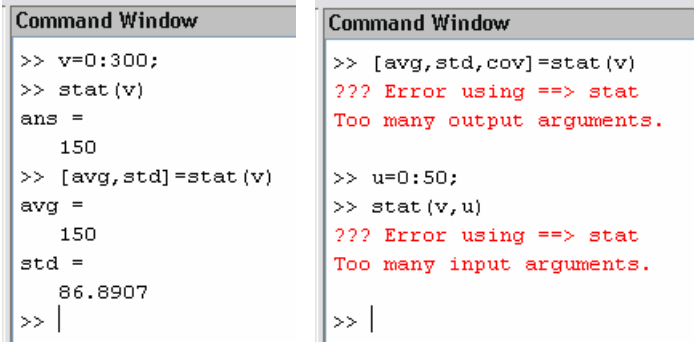
Vytvořená funkce obsahuje i řádky pro nápovědu. Zadáme-li **help stat** (obr. 5.25), dojde k jejímu vypsání. Dokonce je možné v rámci nápovědy definovat odkazy na další podobné funkce. Stačí ve funkci do bloku pro nápovědu přidat heslo „***See also:***“ a za něj doplnit příkazy, na které se chceme v nápovědě odkazovat. Můžeme se odkázat i na námi vytvořenou funkci a nikoliv pouze na standardní funkce MATLABu.

Funkce mohou mít libovolný počet vstupních a výstupních parametrů. Lze je také volat s menším počtem vstupních nebo výstupních parametrů než je specifikováno v řádce deklarace funkce. Nemohou být ale volány s více vstupními nebo výstupními parametry. Počet vstupních resp. výstupních parametrů je možné zjistit příkazy **nargin** a **nargout**.



Obr. 5.25: Nápověda k funkci

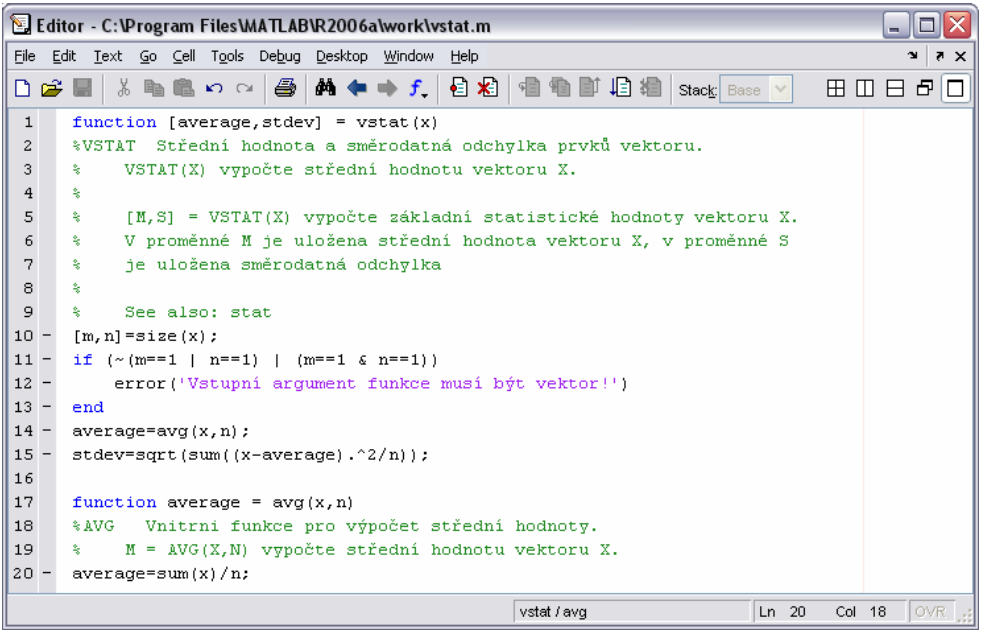
Na obr. 5.26 je odezva MATLABu při zadání pouze jednoho výstupního parametru a také při zadání obou parametrů vytvořené funkce. Pokud zadáme více vstupních či výstupních parametrů, než je ve funkci definováno, dojde k vypsání chyby.



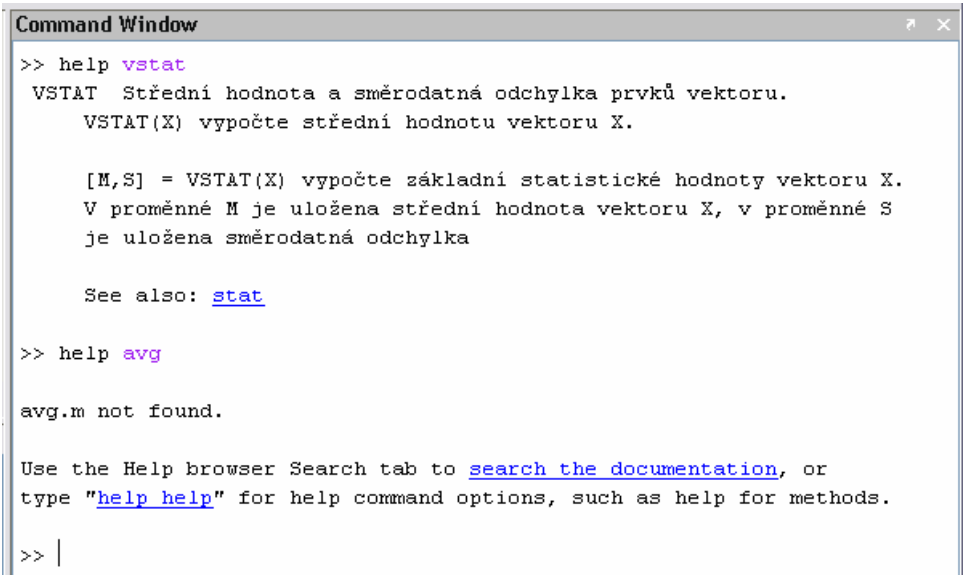
Obr. 5.26: Možné odezvy MATLABu po vykonání funkce **stat**

V jednom m-souboru se může vyskytovat i více funkcí. První funkce je tzv. primární funkce a ostatní jsou podfunkcemi – lokálními funkcemi. Lokální funkce mohou být volány primární funkcí m-souboru nebo jinými lokálními funkcemi ve stejném m-souboru. Volání lokální funkce z jiného m-souboru není možné. Lokální funkce začínají stejným definičním řádkem jako funkce primární a platí pro ně všechna výše uvedená pravidla pro zápis funkce. Nápovědný text pro lokální funkce není přes příkaz **help** dostupný, viz obr. 5.28.

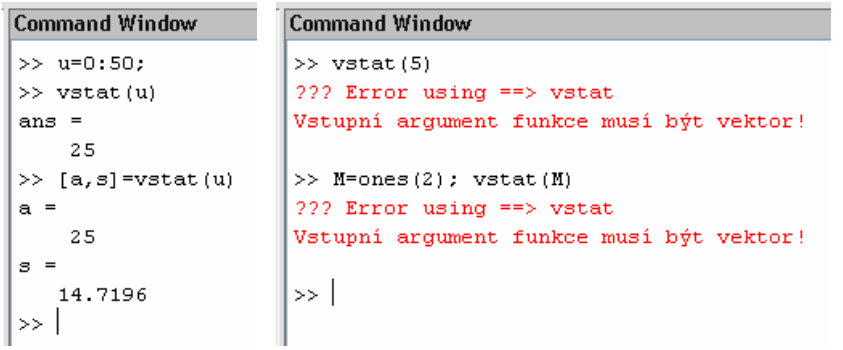
Funkci **stat** dále doplníme o signalizaci chyby, pokud nebude argumentem funkce vektor. Upravenou funkci pojmenujeme **vstat**. V rámci této funkce také vytvoříme lokální funkci s názvem **average**, která vypočte střední hodnotu vektoru. Tato funkce bude volána v rámci funkce **vstat**.



Obr. 5.27: Možnost signalizace chyby, definice lokální funkce

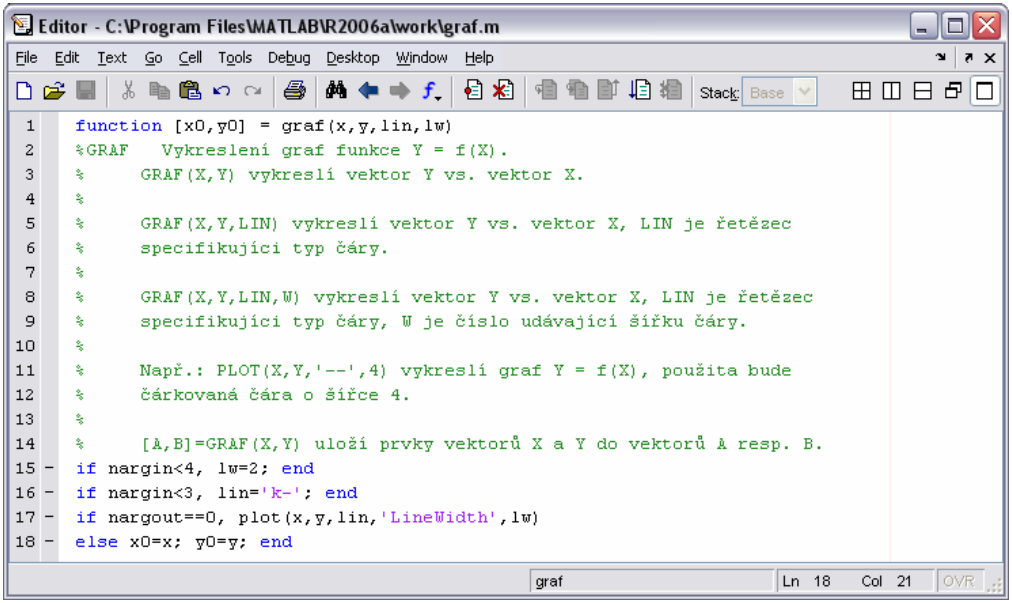


Obr. 5.28: Nápověda k upravené funkci



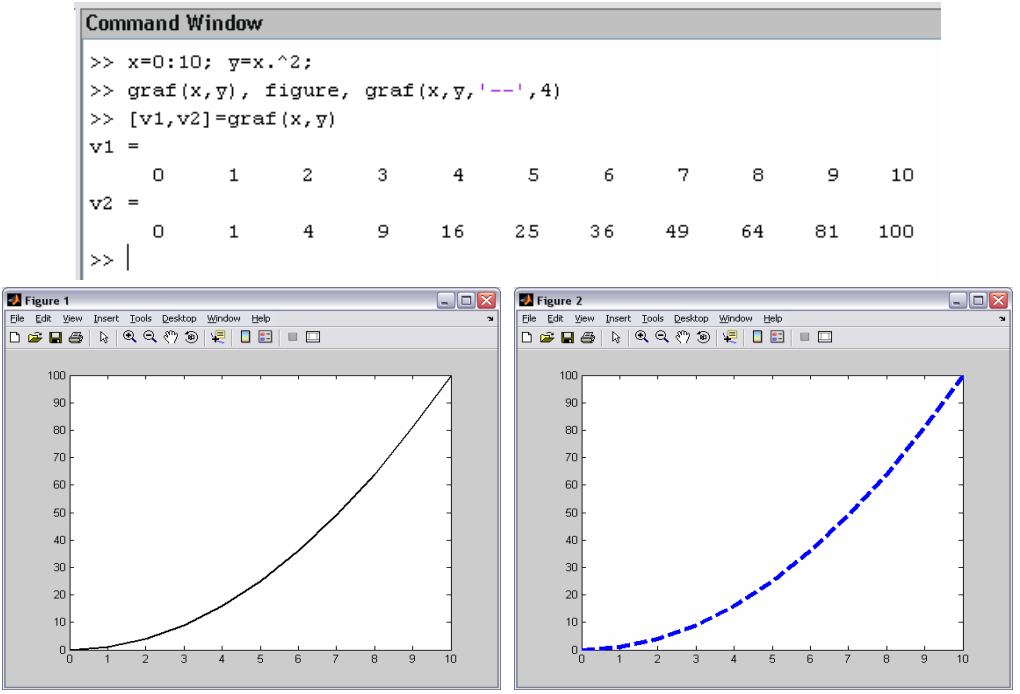
Obr. 5.29: Možné odezvy MATLABu po vykonání funkce **vstat**

Na obr. 5.30 je definice funkce **graf**, která v závislosti na vstupních parametrech vykreslí 2D graf funkce **y** = f(**x**). V deklaraci funkce jsou využity příkazy **nargin** a **nargout**, pomocí nichž jsou parametry, které nejsou zadány přímo uživatelem nastaveny na předem zvolenou hodnotu. Funkce je také doplněna podrobnější nápovědou včetně příkladu použití. Ve funkci je použita řídící struktura **if – else**. Použití této struktury bude podrobněji vysvětleno v kapitole 5.3. Odezva MATLABu po použití funkce graf je na obr. 5.31.

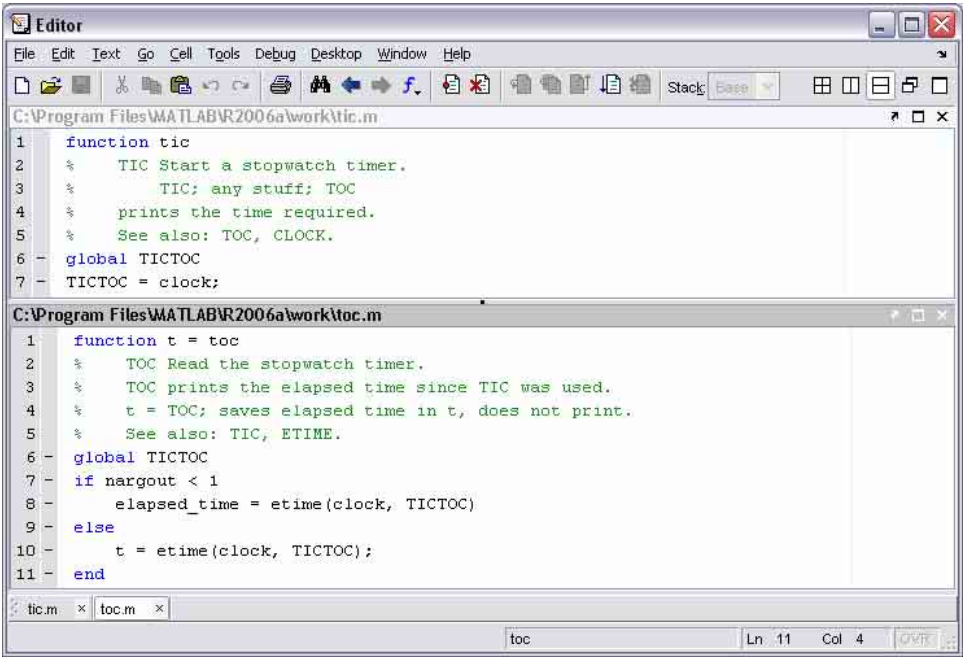


Obr. 5.30: Použití příkazů **nargin** a **nargout** v definici funkce **graf**

Jakákoliv funkce MATLABu definovaná v m-souboru pracuje s proměnnými v přiděleném lokálním prostoru, který je vždy oddělen od prostoru jiné funkce a také od základního prostoru MATLABu. Funkce mohou ale sdílet proměnné s jinými funkcemi, základním pracovním prostorem a rekurzivně volanými funkcemi přes proměnné, které jsou deklarovány jako globální. Je nutné, aby sdílená proměnná byla definována v každém požadovaném pracovním prostoru.



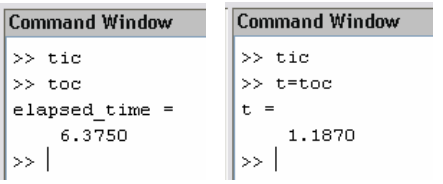
Obr. 5.31: Možné odezvy MATLABu po vykonání funkce **graf**



Obr. 5.32: Použití globálních proměnných

Pokud globální proměnná není ještě definována, MATLAB vytvoří prázdné pole. Pokud již proměnná existuje, je vypsáno varovné hlášení (warning) a proměnná je změněna na globální. Výhodou používání globálních proměnných je lepší využití paměti. Při deklaraci se globální proměnné uvozují slovem **global**. Deklarace se provede obvykle na začátku skriptu. Chceme-li např. vytvořit globální proměnné a, b a M, zapíšeme **global a b M**.

K demonstraci způsobu použití globálních proměnných použijeme dvě funkce, které jsou součástí nápovědy MATLABu (**doc global**). Ve funkci **tic** je definována globální proměnná TICTOC a je jí přiřazena výstupní hodnota funkce **clock** (viz kapitola 2.2). Ve funkci **toc** je v závislosti na tom, je-li použit i výstupní parametr, vypočítán uběhlý čas od spuštění funkce **tic** a nebo je tento čas uložen do proměnné t. Odezva MATLABu po postupném použití těchto dvou funkcí je na obr. 5.33.



Obr. 5.33: Odezvy MATLABu po volání funkcí **tic** a **toc**

Při tvorbě skriptů a jejich následném ladění je užitečné používat příkaz **break**. Skript se postupně vykonává až do místa, kde je zapsán příkaz **break** a veškerý další obsah m-souboru uvedený za tímto příkazem se ignoruje. Pokud dojde např. z důvodu volání chybně napsaného skriptu k nepředpokládané situaci, lze spuštěný skript pomocí Ctrl + C předčasně ukončit.

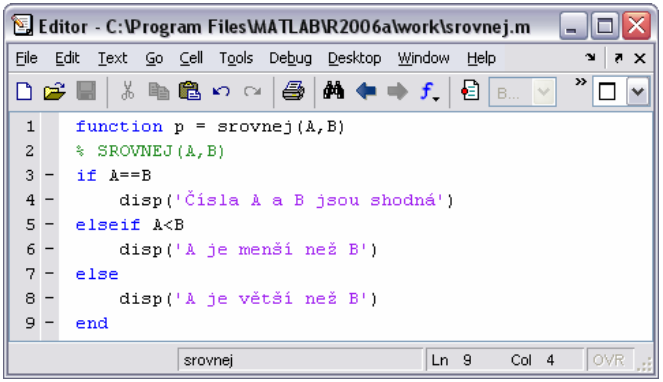
5.3 Příkazy pro řízení běhu programu

Při běhu programu je někdy také třeba využívat za předem specifikovaných podmínek pouze některé jeho části. K tomuto účelu existují v MATLABu funkce **if** a **switch**. K testování vlastností prvků polí a nebo k naplňování proměnných v cyklech slouží funkce **for** a **while**.

Potřebujeme-li se v programu (skriptu) rozhodovat např. podle hodnoty nějakého parametru a v závislosti na tom vykonávat pouze určité bloky programu či program v daný okamžik ukončit, můžeme použít konstrukci **if – else – elseif – end**. Příkaz **if** tedy slouží k větvení programu, jeho obecná syntaxe je na obr. 5.34. Části příkazu v hranatých závorkách nejsou povinné. Příkazy mezi **if** a **end** se vyhodnotí v případě, že hodnota výrazu je *true*.

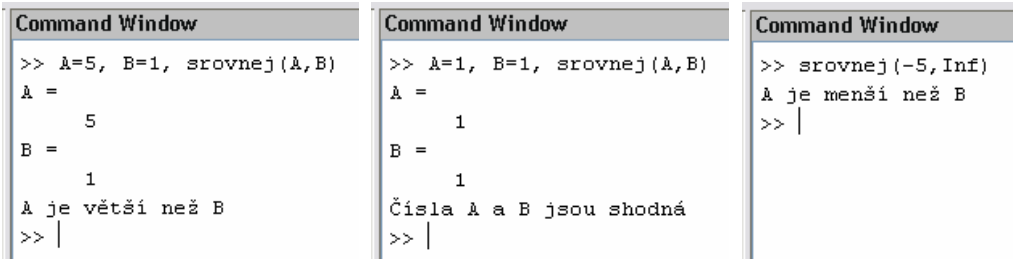
```
if výraz
        ... příkazy ...
[elseif výraz
        ... příkazy ...]
[else
        ... příkazy ...]
end
```

Obr. 5.34: Syntaxe příkazu **if**



Obr. 5.35: Příklad použití příkazu **if**

Jelikož již umíme definovat funkce, přímo se nabízí ukázat použití příkazu **if** na funkci, která porovná dvě čísla a následně zobrazí výsledek, viz obr. 5.35. Možné odezvy MATLABu při zadání různých čísel je na obr. 5.36.

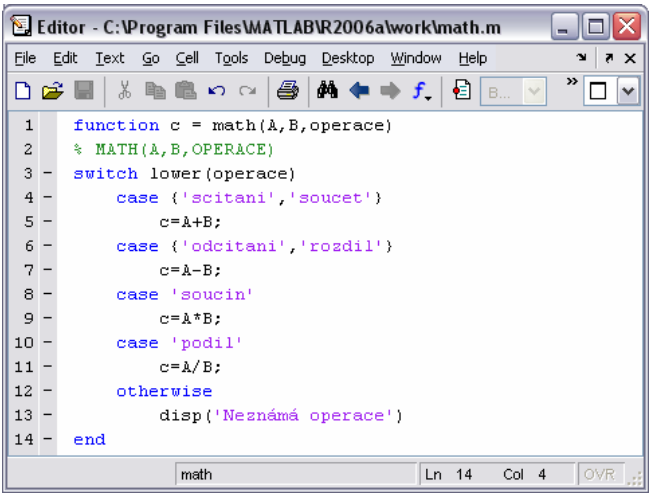


Obr. 5.36: Odezva MATLABu po volání funkce **srovnej**

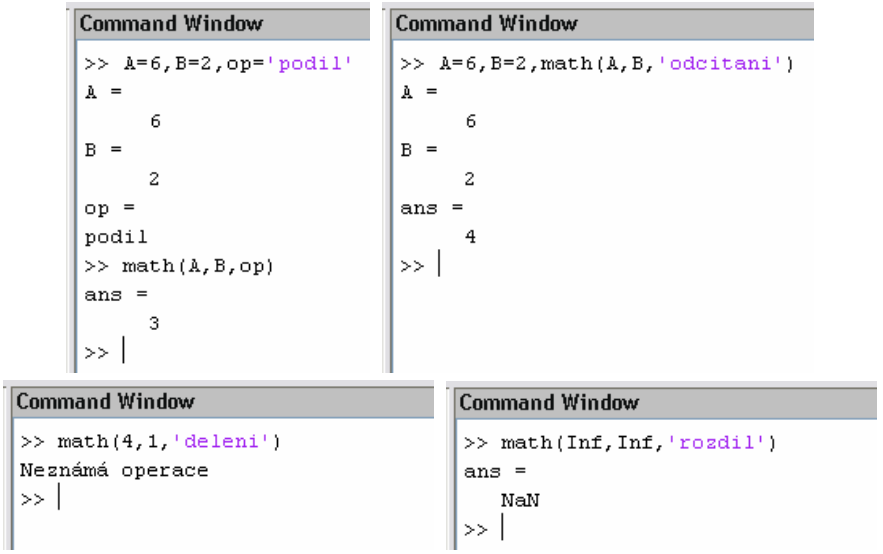
Příkaz **switch** je složitější obdobou příkazu **if** a slouží také k větvení programu. Příkaz používáme v konstrukci **switch – case – otherwise – end**, viz obr. 5.37. Pomocí parametru se rozhodujeme mezi jednotlivými částmi uvozenými slovem **case**. Parametrem může být skalární proměnná nebo řetězec. Na obr. 5.38 je definována funkce **math**, která provede zvolenou operaci se dvěma čísly. Operace se zadává jako řetězec prostřednictvím vstupního parametru funkce. Pokud je zadána nedefinovaná operace, je zobrazeno hlášení. Použití **otherwise** není povinné, stejně jako např. použití **else** u příkazu **if**.

```
switch výraz
      case case_výraz_1
            ... příkazy ...
      case {case_výraz_2, case_výraz_3, ...}
            ... příkazy ...
      [otherwise
            ... příkazy ...]
end
```

Obr. 5.37: Syntaxe příkazu **switch**



Obr. 5.38: Příklad použití příkazu **switch**



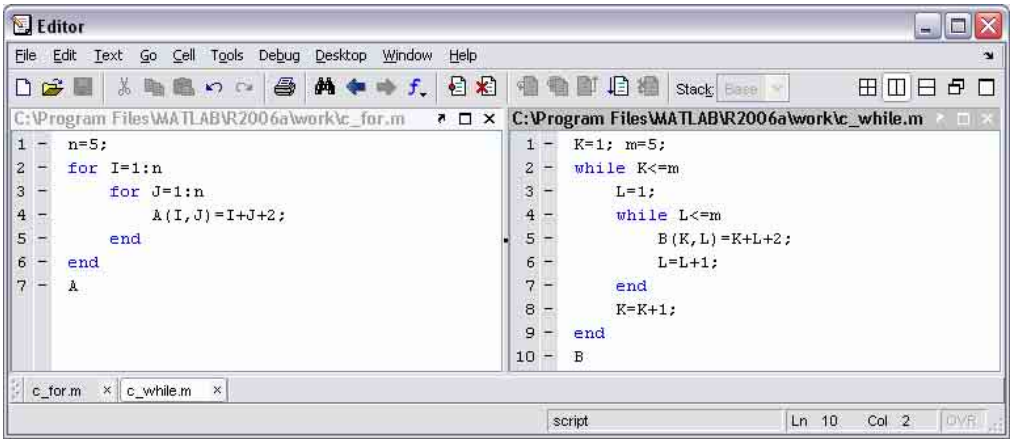
Obr. 5.39: Možné odezvy MATLABu po volání funkce **math**

Cyklus **for**, jehož obecná syntaxe je na obr. 5.40, slouží k provedení předem daného počtu opakovaní skupiny příkazů. Výraz je obvykle ve tvaru **m : n** nebo **m : k : n**. Je tedy zadáván rozsah (v podstatě vektor) a případně i krok.

for ***proměnná*** **=** ***výraz***
... příkazy ...
end

while ***výraz***
... příkazy ...
end

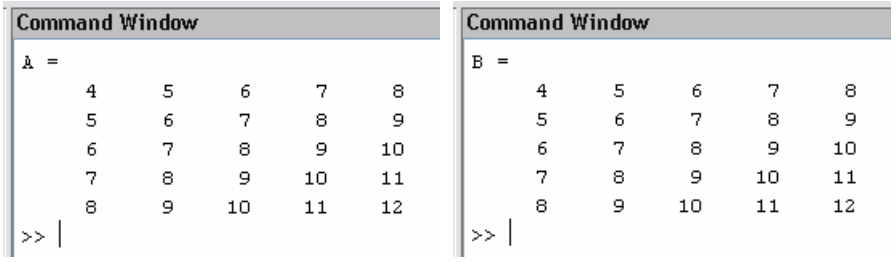
Obr. 5.40: Syntaxe příkazů **for** a **while**



Obr. 5.41: Příklad použití příkazů **for** a **while**

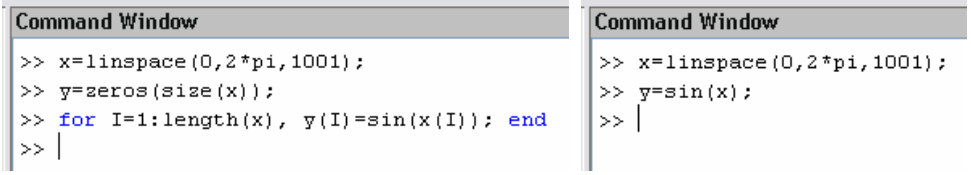
Cyklus **while** umožňuje opakovat příkaz či skupinu příkazů v závislosti na logické podmínce. Syntaxe příkazu je uvedena na obr. 5.40. Příkazy se opakují tak dlouho, dokud jsou všechny prvky ve výrazu nenulové (výraz je matice, může být i vícerozměrná). Výraz je téměř vždy skalárním relačním výrazem, takže nenulové hodnoty odpovídají logické hodnotě *true*. Pokud výraz není skalár, lze jej redukovat příkazem **any** nebo **all**. Více se uživatel dozví v nápovědě, prostřednictvím **doc any** resp. **doc all**.

Na obr. 5.41 je nejprve pomocí cyklu **for** a poté pomocí cyklu **while** vytvářena totožná matice. Výsledná odezva MATLABu je pak na obr. 5.42. Je zřejmé, že zápis pomocí cyklu **for** je jednodušší.



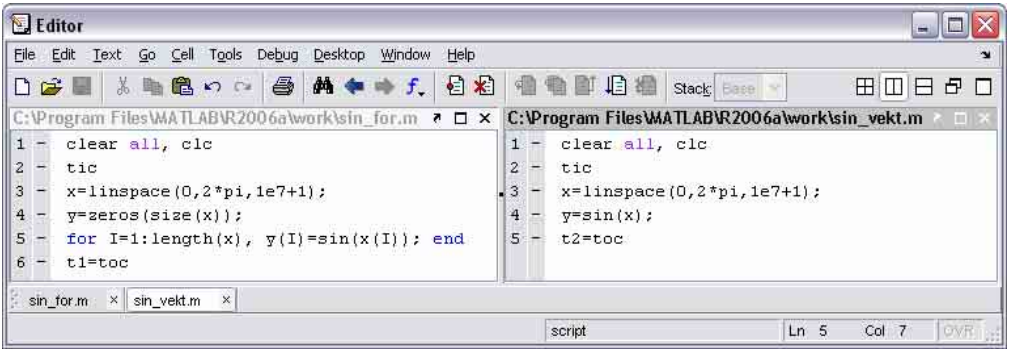
Obr. 5.42: Odezva MATLABu při použití příkazů **for** a **while**

Používání cyklů **for** a **while** v řadě případů značně zpomaluje běh programu. Vhodnější je využívání vlastností MATLABu, zejména tzv. vektorizace. Mimo skutečnost urychlení běhu programu, se i struktura kódu stává jednodušší a přehlednější.



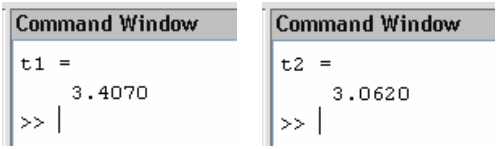
Obr. 5.43: Výpočet funkčních hodnot pomocí cyklu a pomocí vektorizace

Provedeme-li např. výpočet funkčních hodnot funkce $\mathbf{y} = \sin(\mathbf{x})$ obvyklým způsobem a zároveň také i pomocí funkcí MATLABu, zjistíme, že zápis s užitím vektorizace je podstatně jednodušší, viz obr. 5.43. Pro zajímavost můžeme také provést přibližné měření délky výpočtu pomocí výše zmiňovaných funkcí **tic** a **toc** pro obě eventuality. Na obr. 5.44 jsou v okně editoru obě varianty realizovány v podobě skriptu. Hodnoty obdržených časů jsou na obr. 5.45.



Obr. 5.44: Přibližné měření doby výpočtu funkčních hodnot pomocí cyklu a pomocí vektorizace

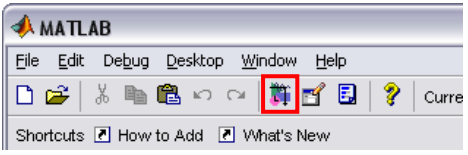
Aby byly patrné rozdíly v časech výpočtu je samozřejmě nutné zvolit velký počet kroků výpočtu. Z tohoto důvodu byl pomocí příkazu **linspace** vygenerován vektor nezávisle proměnné o rozměru 1×10000001 (obsahuje 1e7 + 1 prvků). V těchto bodech byly následně vypočteny funkční hodnoty funkce sinus.



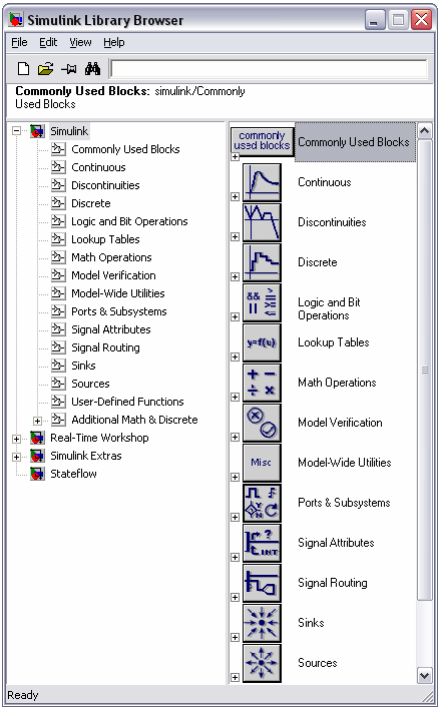
Obr. 5.45: Výsledné časy potřebné pro výpočet

6 Základy práce v nadstavbovém prostředí Simulink

Jak již bylo řečeno v úvodu tohoto textu, nadstavbové prostředí Simulink je určeno především k modelování a simulaci dynamických systémů. Toto prostředí využívá grafických možností operačního systému Windows. Způsob vytváření vlastního modelu je tedy značně odlišný od práce se základním prostředím MATLABu. Modely uživatel definuje jednoduchým způsobem prostřednictvím blokových schémat, která jsou podobná zapojením pro analogový počítač. Pro efektivní práci v Simulinku je nutná základní znalost MATLABu. Nezbytné jsou také alespoň elementární znalosti principů matematického modelování dynamických systémů. V následujících odstavcích se uživatel postupně seznámí se základy prostředí Simulink a jeho možnostmi.



Obr. 6.1: Spuštění prostředí Simulink



Obr. 6.2: Základní okno Simulinku

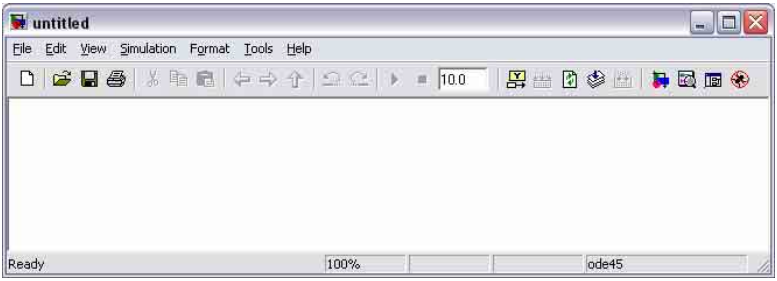
Pokud chceme spustit Simulink, musíme mít již spuštěné základní prostředí MATLAB. Simulink lze spustit přímo pomocí příslušné ikony v toolbaru (viz obr. 6.1) a nebo přímo zadáním příkazu **simulink** v okně *Command Window* MATLABu. Po spuštění prostředí se otevře základní okno *Simulink Library Browser* (obr. 6.2) obsahující standardní knihovny.

Každá z knihoven obsahuje jednotlivé bloky reprezentující příslušné funkce. Tyto bloky lze jednoduchým způsobem přetahovat myší do okna modelu. Nejedná se ovšem o klasické přetažení, neboť blok samozřejmě v knihovně zůstane. Ve skutečnosti se vytvoří pouze vazba na knihovní blok; v okně modelu je pak možno s tímto blokem dále pracovat. Bloky jsou v knihovnách seskupeny podle oblasti jejich použití.

Většina bloků dovoluje zadat hodnoty různých parametrů. Tyto hodnoty lze zadávat přímo jako číselné konstanty a nebo efektivněji – pomocí proměnných. Ty je možné definovat standardním způsobem jako v základním prostředí MATLABu. Proměnné je nutné v tomto případě inicializovat před spuštěním vlastní simulace prostým zadáním

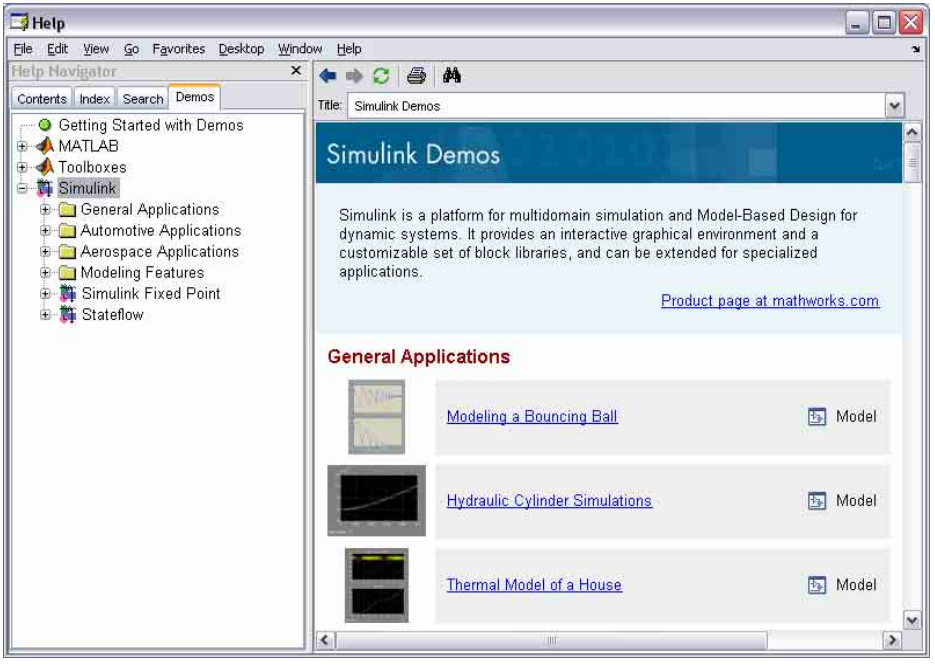
v příkazovém okně MATLABu či pomocí skriptu, jehož součástí může být i případné spuštění simulace a vykreslení výsledků v grafické formě.

Okno nového modelu otevřeme přes základní menu v horní liště okna *Simulink Library Browser*, zvolíme **File → New → Model**. Další možností je použití klávesové zkraty Ctrl + N nebo použití ikony v toolbaru okna. Samozřejmostí je otevření existujícího modelu (Ctrl + O), viz obr. 6.3, úplné zavření okna Simulinku či přístup k vlastnostem *Preferences*.



Obr. 6.3: Okno pro vytváření modelu

Nadstavbové prostředí Simulink samozřejmě poskytuje stejné možnosti práce se systémem nápovědy jako samotný MATLAB. Základní nápovědu můžeme získat zadáním **help simulink** v příkazovém řádku MATLABu. Podrobnější verzi nápovědy zobrazovanou v okně *Help* uživatel vyvolá zadáním příkazu **doc simulink**, viz obr. 6.4. Simulink poskytuje také značné množství poměrně propracovaných dem, např. simulaci skákajícího míčku, základní model tření, nejrůznější modely z oblasti automobilismu, letectví a mnoho dalších.

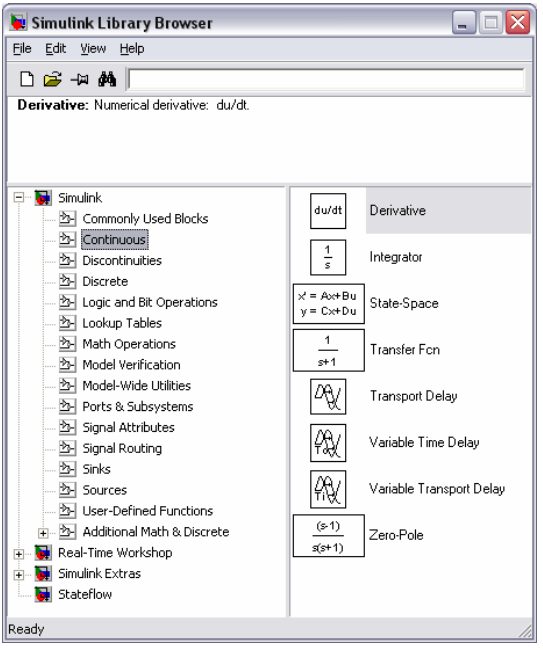


Obr. 6.4: Rozšířený systém nápovědy a přístup k demům

K systému nápovědy lze přistupovat také přímo z hlavního okna Simulinku – v menu okna vybereme **Help → Simulink Help**. Další možností je přístup z okna modelu, zde vybereme v menu **Help → Using Simulink**. Pokud uživatel potřebuje podrobnější popis jednotlivých bloků, přehled datových typů podporovaných v těchto blocích, přehled všech zkratkových kláves, které lze v Simulinku používat nebo výčet možností speciálních funkcí (tzv. S-funkcí), zvolí v menu okna pro vytváření modelu položky **Blocks**, **Block Support Table**, **Shortcuts** resp. **S-Functions**.

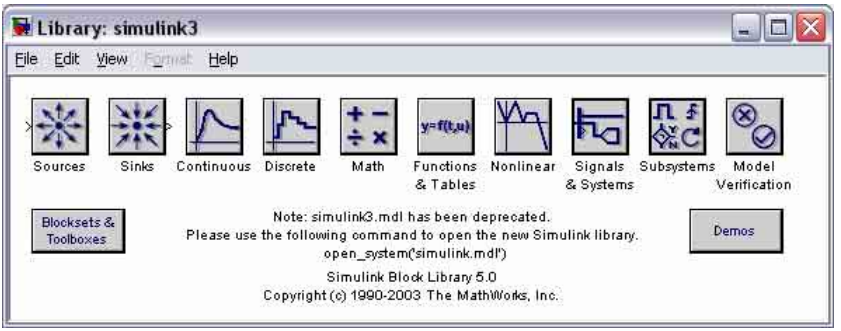
6.1 Standardní knihovny Simulinku

Ke knihovnám Simulinku lze přistupovat, jak již bylo řečeno, prostřednictvím okna *Simulink Library Browser*. Zvolíme-li například knihovnu ***Continuous*** (viz obr. 6.5), tak v pravé části tohoto okna vidíme její obsah. V tomto případě je vybrán blok *Derivative* (numerická derivace) a v horní části okna, bezprostředně pod toolbarem, je uveden jeho podrobnější popis.



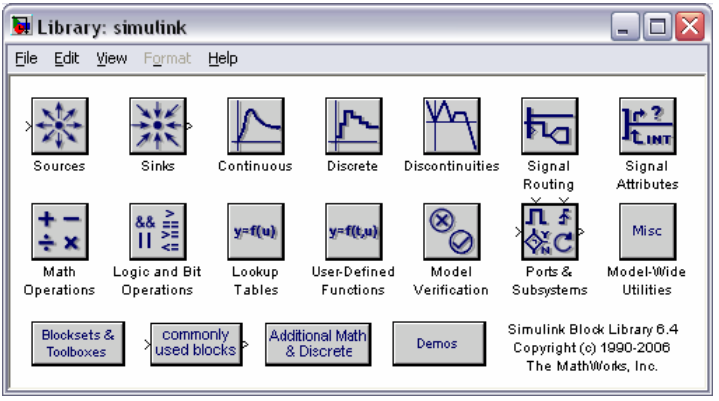
Obr. 6.5: Procházení knihoven Simulinku

Simulink umožňuje ale i jiný způsob přístupu ke knihovnám a jejím blokům. Zadáme-li příkaz **simulink3** v okně *Command Window* MATLABu, dojde k otevření základní knihovny označené ***simulink3*** (okno je označené *Library: simulink3*) a fyzicky uložené v souboru simulink3.mdl, viz obr. 6.6. Zobrazená základní knihovna (Simulink Block Library) je ve verzi 5.0. V textové informaci ve spodní části tohoto okna se uživatel může dočíst, že uvedená knihovna je již nevyhovující a doporučuje se použití příkazu **open_system('simulink.mdl')** k otevření nové základní knihovny Simulinku, viz obr. 6.7. Nová základní knihovna je již ve verzi 6.4 a odpovídá v textu popisované distribuci MATLABu verze R2006a (7.2).



Obr. 6.6: Jiný způsob procházení knihoven Simulinku

Pro efektivní práci se Simulinkem je zřejmě vhodnější použití základní nabídky (okno *Simulink Library Browser*). Nabízený alternativní přístup ke knihovnám je ale přehlednější a dobře poslouží k názornému popisu jednotlivých knihoven Simulinku v dalších odstavcích. Výhodou je ale i skutečnost, že uživatel může mít otevřeno i několik knihoven současně.



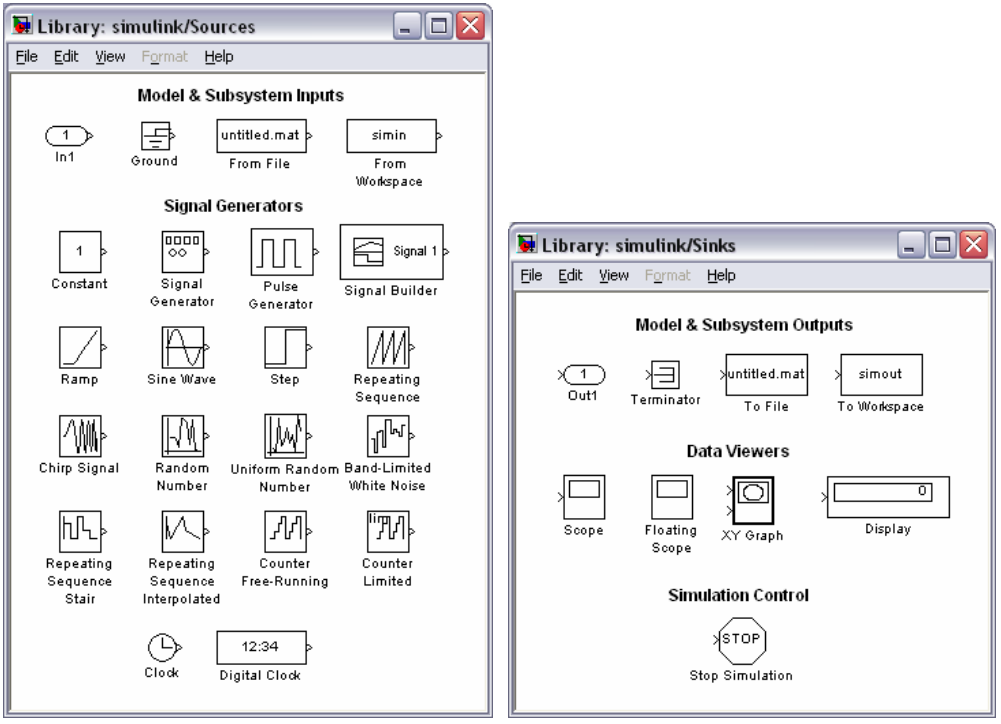
Obr. 6.7: Aktuální základní knihovna Simulinku ve verzi 6.4

Základní knihovna prostředí Simulink obsahuje další knihovny, do kterých jsou jednotlivé bloky seskupeny podle oblasti jejich využití. Knihovny (skupiny bloků) jsou v základním okně (obr. 6.7) reprezentovány ikonami – tlačítky. Po dvojitém kliknutí myší na příslušnou ikonu dojde k otevření zvolené knihovny. Takto byly např. otevřeny knihovny ***Sources*** a ***Sinks*** na obr. 6.8.

Další možností otevření příslušné knihovny (např. knihovny ***Continuous***) je její výběr pomocí myši v levé části základního okna Simulinku. Prostřednictvím pravého tlačítka myši se v tomto případě objeví plovoucí nabídka *Open the Continuous Library* a po jejím vybrání se knihovna ***Continuous*** otevře v samostatném okně.

V knihovně ***Sources*** jsou obsaženy bloky pro vstup dat do subsystémů (*In1*), pro čtení ze souboru na disku (*From File*) resp. z prostoru proměnných (*From Workspace*). Dále také bloky použitelné jako zdroje běžných druhů signálů, např. konstantního signálu (*Constant*), různých harmonických signálů a pulsů (*Signal Generator*, *Sine Wave*, *Chirp Signal*, *Pulse Generator*), zdroje skokových a rampových funkcí a opakovaných sekvencí (*Step*, *Ramp*, *Repeating*

Sequence). Dále také generátory různých typů pseudonáhodných signálů (*Random Number*, *Uniform Random Number*), bílého šumu (*Band-Limited White Noise*), signálů číslicových a generátory času (*Clock*, *Digital Clock*).



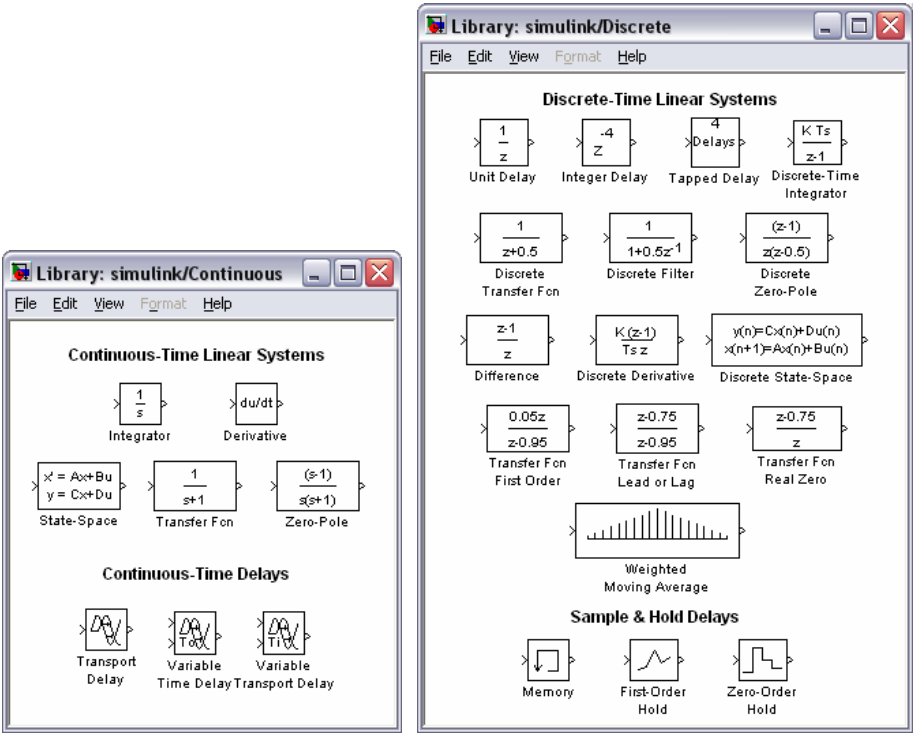
Obr. 6.8: Knihovny ***Sources*** a ***Sinks***

Knihovna ***Sinks*** obsahuje bloky pro výstup dat ze subsystémů (*Out1*), pro zápis do souboru (*To File*) a do prostoru proměnných (*To Workspace*) a bloky pro zobrazení průběhů signálů (*Scope*, *Floating Scope*, *XY Graph*, *Display*). Knihovna také obsahuje blok *Stop Simulation* pro ukončení simulace pokud je vstup do bloku nenulový.

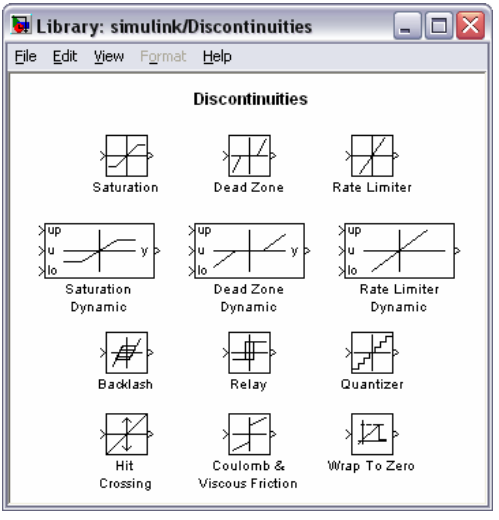
Mezi základní knihovny Simulinku patří knihovny ***Continuous*** a ***Discrete*** sloužící pro spojitý a diskrétní popis dynamických systémů. Nejdůležitější jsou bloky pro numerickou integraci a derivaci. Blok *Integrator* umožňuje zadání počátečních podmínek a limitaci výstupu (tzv. saturaci). Obojí je možno zadat jednak přímo hodnotou parametru, ale také pomocí dalších vstupních signálů. Blok dovoluje i externí reset (nastavení do počátečního stavu) na základě změny na řídicím logickém signálu (na náběžnou, sestupnou nebo obě hrany). Další možností je definování stavového výstupu (state portu), který se využívá právě ve spojitosti s externím resetem integrátoru a poskytuje hodnotu, která by byla na výstupu, pokud by nedošlo k resetu. S uvedenými možnostmi integrátoru se podrobněji seznámíme v odstavci 6.2.

K numerické derivaci slouží blok *Derivative*. Je známou skutečností, že matematicky jasně definovaná derivace je obtížně numericky aproximovatelná. Derivace je v prostoru obrazů (po aplikování Laplaceovy transformace) v tomto případě aproximována obrazovým přenosem $s / ((1 / N)\, s + 1)$, kde $T_v = 1 / N$ představuje tzv. parazitní časovou konstantu. Blok derivace

umožňuje změnu konstanty N, implicitní hodnota je Inf (nekonečno, přesněji aritmetická reprezentace kladného nekonečna podle standardu IEEE). V popisu uvedeném v okně parametrů bloku, ale i v nápovědě k bloku, je uveden chybný přenos ve tvaru s / (N s + 1).



Obr. 6.9: Knihovny ***Continuous*** a ***Discrete***

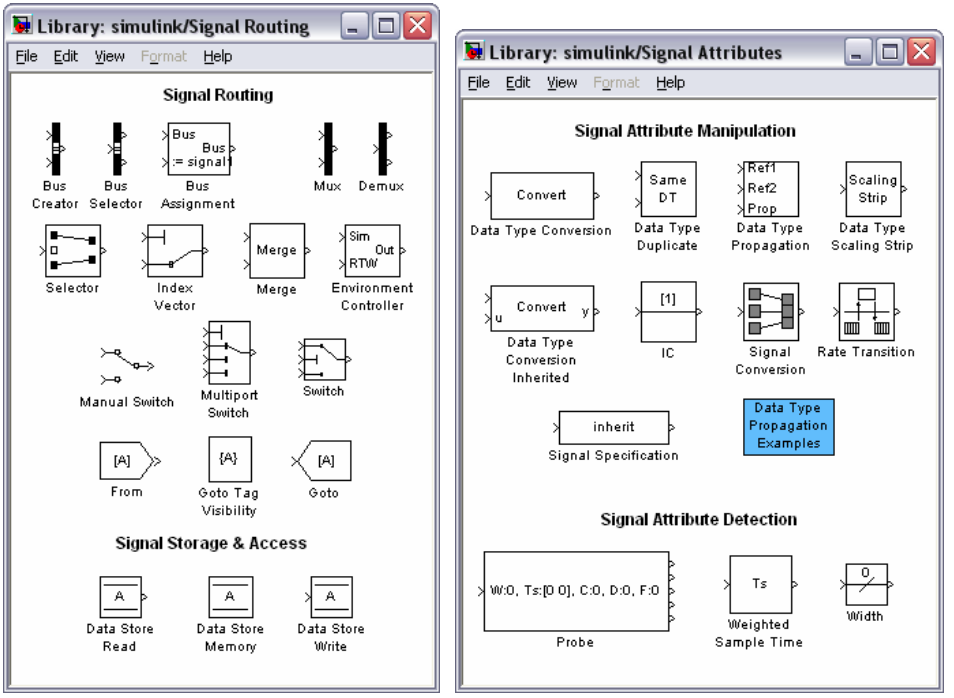


Obr. 6.10: Knihovna ***Discontinuous***

Bloky *State-Space*, *Transfer Fcn* a *Zero Pole* umožňují zadání popisu dynamického systému ve formě stavového popisu, obrazového přenosu či pomocí nul, pólů a zesílení. Zbývající bloky *Transport Delay*, *Variable Time Delay* a *Variable Transport Delay* slouží k modelování dopravního zpoždění (i proměnného).

Knihovna ***Discrete*** obsahuje bloky potřebné pro diskrétní popis systémů. Většina bloků je víceméně ekvivalentní blokům z knihovny ***Continuous***, např. bloky *Discrete-Time Integrator*, *Discrete Transfer Fcn* resp. *Discrete Filter*, *Discrete State-Space*. Knihovna obsahuje také bloky tvarovače nultého řádu (*Zero-Order Hold*) a prvního řádu (*First-Order Hold*). Užitečný je i blok *Memory*, který představuje zpoždění vstupního signálu o jeden simulační krok a lze jej využít např. k odstranění algebraické smyčky.

Knihovna ***Discontinuous*** (obr. 6.10) obsahuje typické nelinearity, jmenovitě saturaci (blok *Saturation*), necitlivost – mrtvé pásmo (*Death Zone*), zubovou vůli (*Backlash*) a statické tření (*Coulomb & Viscous Friction*). Dále blok *Relay*, blok *Rate Limiter* pro omezení rychlosti změny signálu, blok *Hit Crossing* pro detekci průchodu definovanou hodnotou a další.

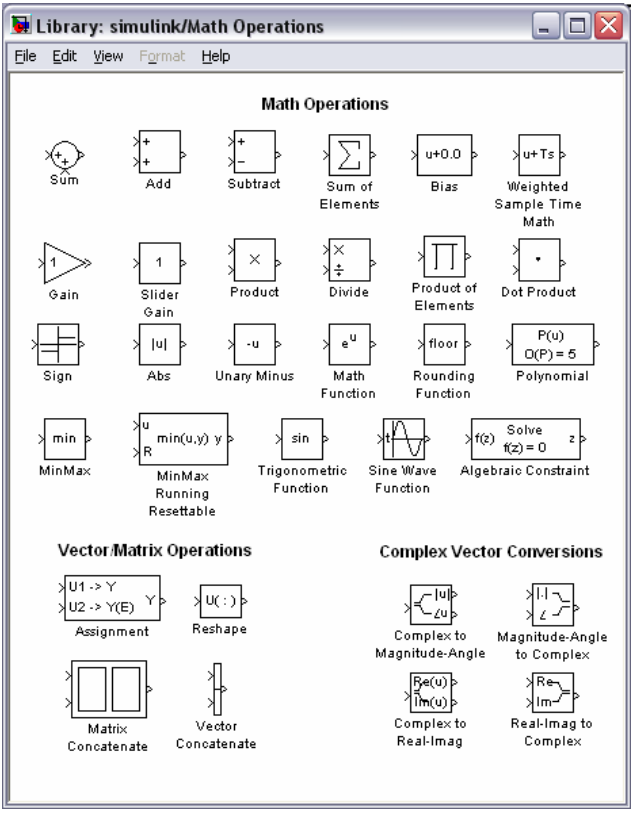


Obr. 6.11: Knihovny ***Signal Routing*** a ***Signal Attributes***

Součástí knihoven ***Signal Routing*** a ***Signal Attributes*** jsou především bloky pro práci se signály. Mezi nejdůležitější bloky patří *Mux* a *Demux*, které umožňují sloučení několika signálů do jednoho a naopak. Pokud uživatel potřebuje např. pomocí daného parametru přepínat mezi několika signály, může využít blok *Multiport Switch*. Výhodné je i použití bloku *Switch*, který umožňuje přepínání mezi dvěma vstupními signály na základě vyhodnocování úrovně referenčního signálu a porovnání s definovanou mezí. Pro rozsáhlá simulační schémata

je vhodné použití bloků *From* a *Goto*, umožňujících přenos signálů mezi těmito bloky bez nutnosti fyzického propojení.

V mnoha případech je důležité použití bloku *Convert*, který zajistí konverzi na požadovaný datový typ (i zcela automaticky dle typu signálu na výstupu bloku). Blok *IC* nastaví počáteční hodnotu signálu na vstupu bloku. Knihovna ***Signal Attributes*** obsahuje samozřejmě i další bloky, např. blok *Probe*, sloužící k zjišťování parametrů signálů (šířka, vzorkování, detekce jestli je signál komplexní) nebo blok *Width*, na jehož výstupu je šířka vstupního signálu (datový typ výstupu lze navíc zvolit).



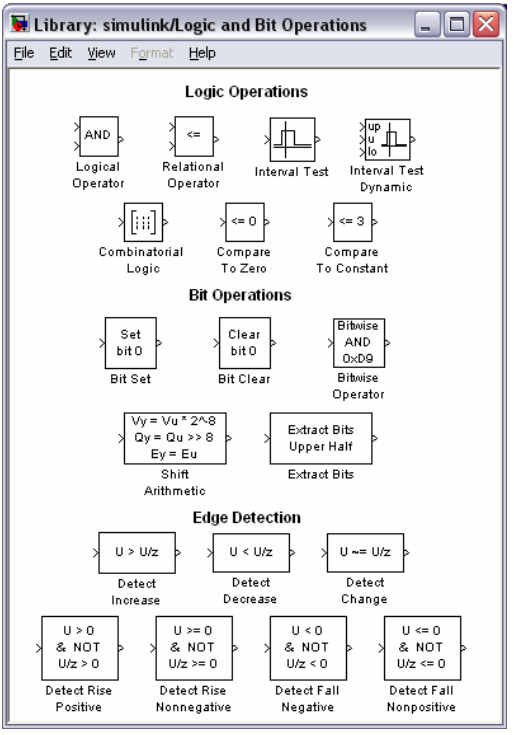
Obr. 6.12: Knihovna ***Math Operations***

Rozsáhlá knihovna ***Math Operations*** (obr. 6.12) obsahuje bloky *Sum*, *Add* a *Subtract* pro sčítání resp. odečítaní signálů. Vzhled a počet vstupů bloku *Sum* lze jednoduše předem nastavit; implicitně je nastaveno sčítání dvou signálů a kruhový tvar bloku. Bloky *Add* a *Subtract* jsou pouze variantou bloku *Sum* a jejich vzhled a počet vstupů lze měnit stejným způsobem.

Mezi další bloky popisované knihovny patří blok pro zesílení signálu *Gain* příp. *Slider Gain* pro zesílení ručně přestavitelné (pomocí posuvníku). K násobení či dělení dvou a více signálů lze využit bloky *Product* (násobení vektorové), *Dot Product* (odpovídá prvkovému násobení, v MATLABu označovanému **.***) a *Divide*.

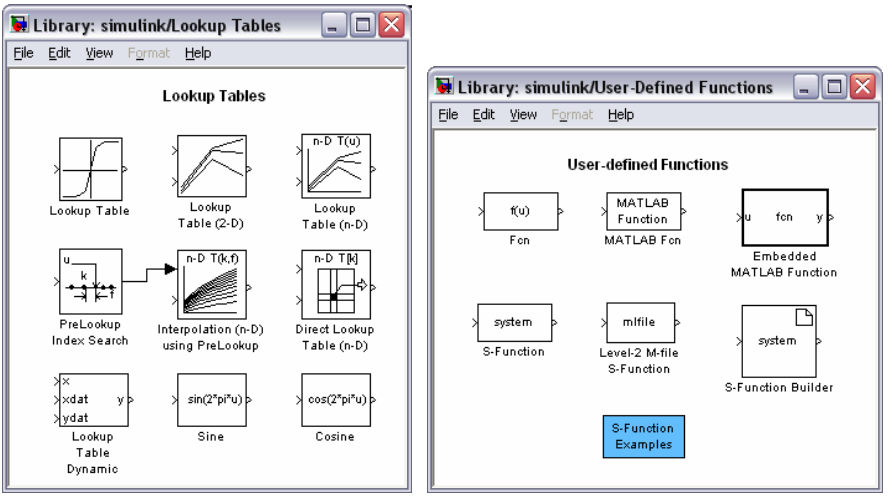
Významnou skupinou jsou bloky reprezentující základní matematické funkce, např. funkci signum (blok *Sign*) a absolutní hodnotu (*Abs*). Knihovna obsahuje dále univerzální blok matematických funkcí a univerzální blok trigonometrických funkcí (*Math Function* resp. *Trigonometric Function*), blok pro zaokrouhlování (*Rounding Function*) a blok pro stanovení minima a maxima (*MinMax*). Další skupinu tvoří bloky pro vektorové a maticové operace a bloky pro nejrůznější konverze komplexních čísel (např. převod na složkový tvar resp. na tvar exponenciální, rozdělení na reálnou a imaginární část).

Knihovna ***Logic and Bit Operations*** (obr. 6.13) zahrnuje bloky logických a bitových operací a bloky pro detekci hran signálů. Mezi základní bloky patří *Logical Operator* pro logické operace (AND, OR, NAND, NOR, XOR, NOT). Uživatel může v okně parametrů bloku změnit i jeho vzhled, tak aby odpovídal zvolené logické operaci (použito je anglosaské značení), a počet vstupů. Blok *Relational Operator* umožňuje navzájem porovnávat dva signály. V bloku je možno zvolit operátory = (= =), ≠ (~ =), <, ≤ (< =), ≥ (> =) a >. Bloky *Compare To Zero* a *Compare To Constant* umožňují porovnávat vstupující signál s nulou resp. s předem definovanou konstantou. Blok *Combinatorial Logic* poskytuje uživateli možnost implementovat do simulačního schématu pravdivostní tabulku.



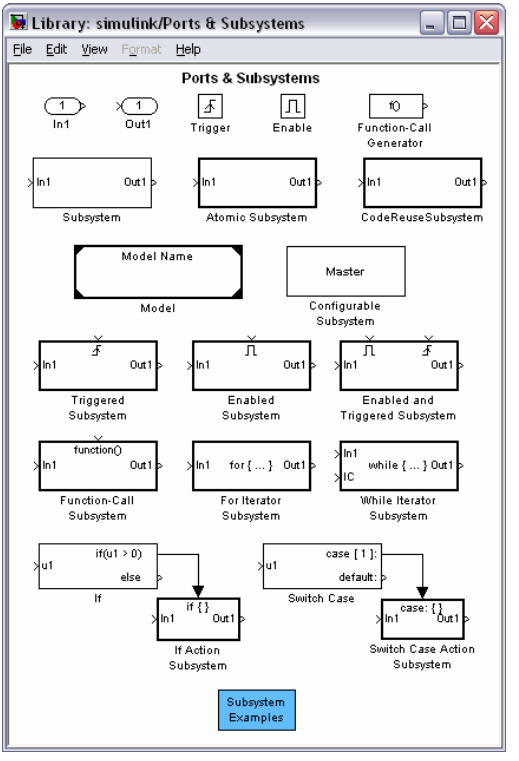
Obr. 6.13: Knihovna ***Logic and Bit Operations***

Součástí knihovny ***Lookup Tables*** (obr. 6.14) jsou bloky pro tabulkovou aproximaci funkcí jedné (*Lookup Table*) či více proměnných (*Lookup Table* (*2-D*), *Lookup Table* (*n-D*)). V okně parametrů bloku se volí rozsah vstupních dat (i včetně kroku), typ a rozsah tabelační funkce (implicitně funkce **tanh**) a metodu aproximace.



Obr. 6.14: Knihovny ***Lookup Tables*** a ***User-Defined Functions***

Velmi významná je knihovna ***User-Defined Functions*** poskytující značné možnosti pro zadávání téměř libovolných funkcí (blok *Fcn*), celých skriptů (*MATLAB Function*) nebo speciálních S-funkcí (*S-Function*).

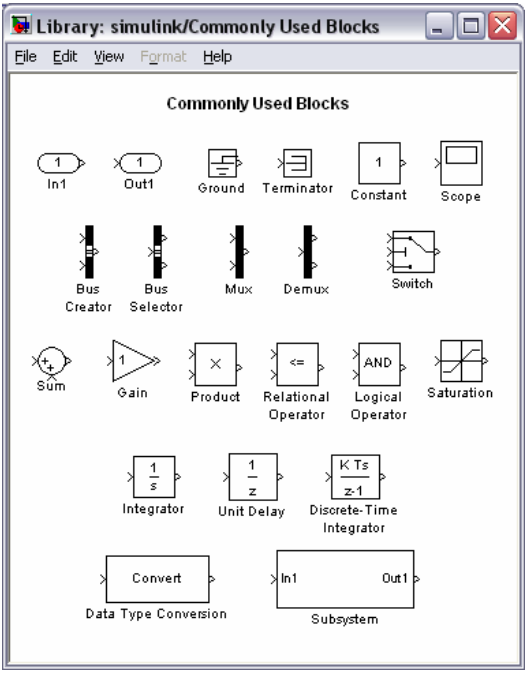


Obr. 6.15: Knihovna ***Ports & Subsystems***

Poslední knihovnou Simulinku, uvažujeme-li pouze základní verzi prostředí bez rozšíření o další toolboxy, je knihovna ***Ports & Subsystems*** (obr. 6.15). Mimo již dříve uvedené bloky *In1* a *Out1* (knihovny ***Sources*** a ***Sinks***), které slouží ke vstupu a k výstupu dat u subsystémů, obsahuje tato knihovna několik bloků pro vytváření vlastních subsystémů (*Subsystem*, *Model* a další).

Blok subsystému představuje ve své podstatě další (vnořené) simulační schéma; propojení s ostatními částmi modelu se realizuje pomocí již zmiňovaných bloků *In1* a *Out1*. Počet vstupů a výstupů bloku lze jednoduše rozšířit přidáním dalších vstupních či výstupních bloků. Blok *Model* umožňuje vnoření celého modelu (nikoliv jen skupiny bloků), který je uložen v souboru.

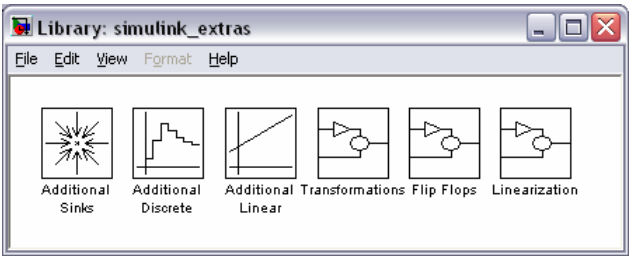
Okno základních knihoven je také doplněno o skupinu běžně užívaných bloků (commonly used blocks), o rychlý přístup k demům (Demos) a o skupiny obsahující knihovny dalších toolboxů a skupin bloků (Blocksets & Toolboxes, Additional Math & Discrete)



Obr. 6.16: Skupina bloků ***Commonly Used Blocks***

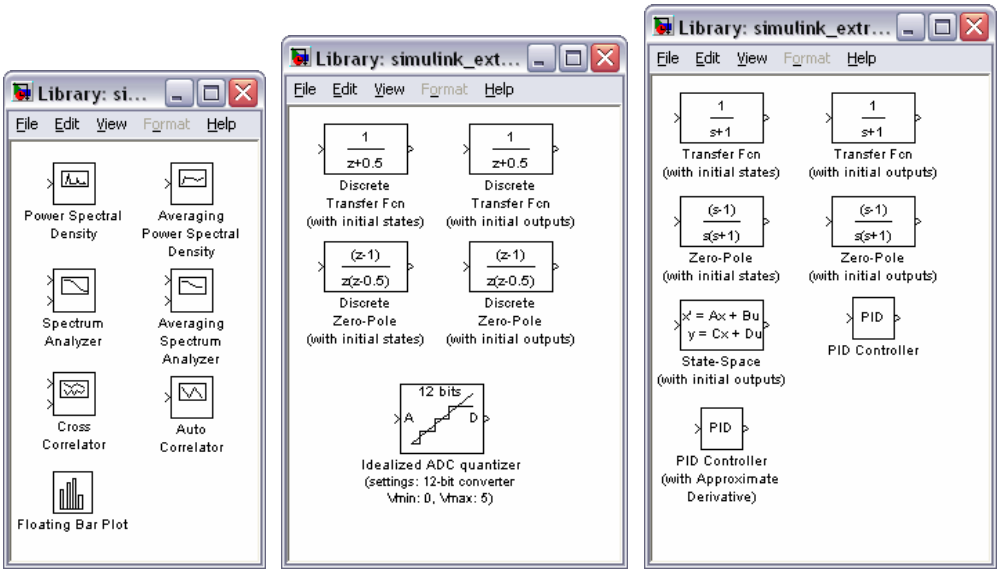
Ze skupiny bloků Blocksets & Toolboxes je v základní verzi Simulinku důležitá hlavně základní knihovna ***Simulink Extras***, která obsahuje další knihovny, viz obr. 6.17. Mimo knihovny ***Additional Sinks***, ***Additional Discrete*** a ***Additional Linear*** (obr. 6.18), které obsahují jakési doplňky k výše popisovaným knihovnám, je její součástí také knihovna ***Flip Flops*** s bloky představujícími základní typy klopných obvodů (obr. 6.19).

Knihovna ***Additional Sinks*** obsahuje některé další bloky pro vyhodnocování průběhů a vlastností signálů. Jsou to např. bloky pro výpočet výkonové spektrální hustoty (*Power Spectral Density*, *Averaging Power Spectral Density*), pro výpočet autokorelační funkce signálu (*Auto Correlator*) či vzájemné korelační funkce dvou signálů (*Cross Correlator*).

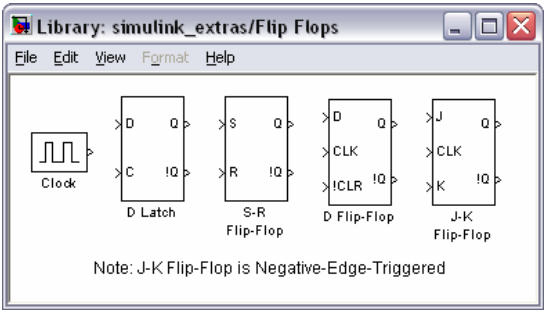


Obr. 6.17: Základní knihovna ***Simulink Extras***

Knihovny ***Additional Discrete*** a ***Additional Linear*** poskytují doplňkové bloky pro spojitý a diskrétní popis dynamických systémů. Tyto bloky umožňují, oproti obdobným blokům z knihoven ***Continuous*** a ***Discrete***, nastavit také počáteční stavy na vstupu resp. na výstupu bloku.



Obr. 6.18: Knihovny ***Additional Sinks***, ***Additional Discrete*** a ***Additional Linear***



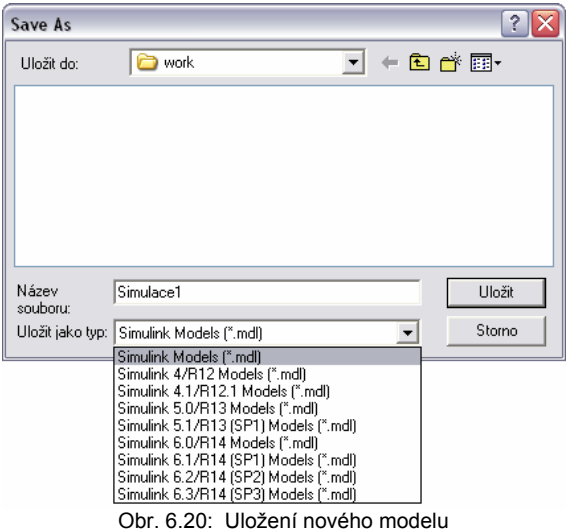
Obr. 6.19: Knihovna ***Flip Flops***

Nejdůležitější z této skupiny jsou ale bloky PID regulátoru. Blok *PID Controller* představuje běžnou spojitou variantu PID regulátoru s přenosem P + I / s + D s (přenos je v tzv. paralelním tvaru), zatímco blok *PID Controller* (*with Approximate Derivative*) představuje PID regulátor s aproximovanou derivací s přenosem P + I / s + (D s) / ((1 / N) s + 1); je tedy možno navíc měnit i konstantu N.

6.2 Vytváření modelu

Prvním předpokladem pro vytváření modelu dynamického systému je, mimo spuštění vlastního prostředí Simulink, otevření prázdného okna modelu. Zopakujme, že nové okno modelu lze otevřít přes základní menu v horní liště okna *Simulink Library Browser*, zde zvolíme **File → New → Model**, další možností je použití klávesové zkraty Ctrl + N nebo také ikony v toolbaru okna (viz obr. 6.2). Velmi vhodné je uložení nového modelu již na počátku jeho vytváření a jeho pozdější průběžné ukládání (standardním způsobem, např. pomocí kombinace kláves Ctrl + S).

Prázdné okno modelu je zpočátku pojmenované *untitled*. Proveďme nyní uložení nového modelu, např. pod názvem *Simulace1*. Dialogové okno pro uložení souboru je na obr. 6.20. Mimo obvyklá omezení operačního systému nesmí název souboru modelu začínat číslicí a obsahovat pomlčku (znak -). Simulink poskytuje i možnost zpětné kompatibility s nižšími verzemi prostředí. Uživatel může prostřednictvím roletového menu **Uložit jako typ** zvolit verzi, kterou potřebuje.



Obr. 6.20: Uložení nového modelu

Aktuální adresář odpovídá nastavení v základním v okně MATLABu, implicitně je nastaven do C:/Program Files/MATLAB/R2006a/Work, lze jej ale samozřejmě změnit. Připomeňme, že v MATLABu je aktuální adresář (*Current Directory*) zobrazen nad hlavním oknem *Command Window*. Na disku počítače je v aktuálním adresáři po uložení umístěn soubor Simulace1.mdl. Nyní již může uživatel začít vytvářet vlastní model dynamického systému.

Pro jednoduchost a názornost bude v dalším textu vytvářen model obecného dynamického systému druhého řádu (resp. systému se zpožděním druhého řádu). Konkrétně se může jednat např. o model RLC obvodu, zjednodušený model sedačky řidiče či pružného závěsu kola automobilu, apod. Popisovaný dynamický systém je popsán diferenciální rovnicí ve tvaru $a_2\, y''(t) + a_1\, y'(t) + a_0\, y(t) = b_0\, u(t)$ s nulovými počátečními podmínkami. Lze jej také popsat obrazovým přenosem ve tvaru $G(s) = b_0 / (a_2\, s^2 + a_1\, s + a_0)$.

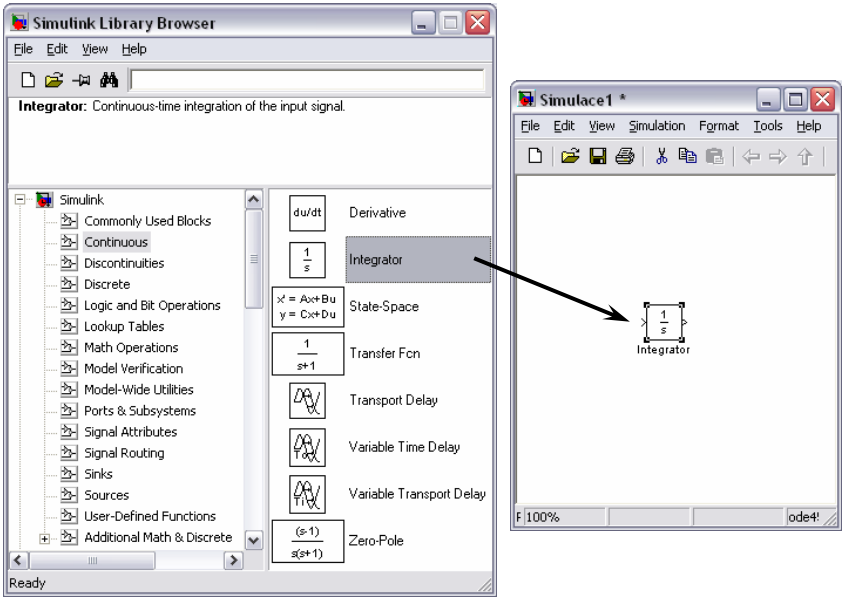
Užijeme-li k řešení uvedené diferenciální rovnice např. metodu postupné integrace, tak po vyjádření nejvyšší derivace obdržíme rovnici

$$y'' = -\frac{a_1}{a_2}\, y' - \frac{a_0}{a_2}\, y + \frac{b_0}{a_2}\, u$$

a po dvojí integraci pak

$$y = -\frac{a_1}{a_2} \int y - \frac{a_0}{a_2} \iint y + \frac{b_0}{a_2} \iint u\ .$$

Pro zjednodušení zápisu jsou jednotlivé integrály zapsány bez příslušných mezí. Např. integrálu $\int_0^t y(\tau)\, d\tau$ odpovídá zápis $\int y$. Výše uvedenou rovnici budeme dále realizovat v Simulinku.

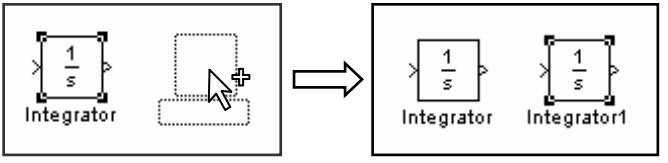


Obr. 6.21: Umístění bloku do okna modelu

Do prázdného okna modelu (*Simulace1*) umístíme postupně několik bloků tak, aby bylo možné realizovat výše uvedenou diferenciální rovnici. Bloky ze zvolené knihovny lze jednoduchým způsobem přetahovat myší do okna modelu. Připomeňme, že se nejedná o klasické přetažení (blok v knihovně samozřejmě zůstane), ale vytvoří se pouze vazba na knihovní blok.

Do okna modelu umístíme nejprve dva integrátory. Blok pro integraci (*Integrator*) je součástí knihovny ***Continuous*** (viz obr. 6.5 a 6.9). Blok do okna modelu umístíme tak, že jej nejprve vybereme v základním okně *Simulink Library Browser*. V levé části okna zvolíme základní knihovnu ***Simulink*** a v ní pak vybereme knihovnu ***Continuous***. V pravé části okna poté v rámci této knihovny myší vybereme blok *Integrator* a přetažením jej umístíme do okna modelu, viz obr. 6.21.

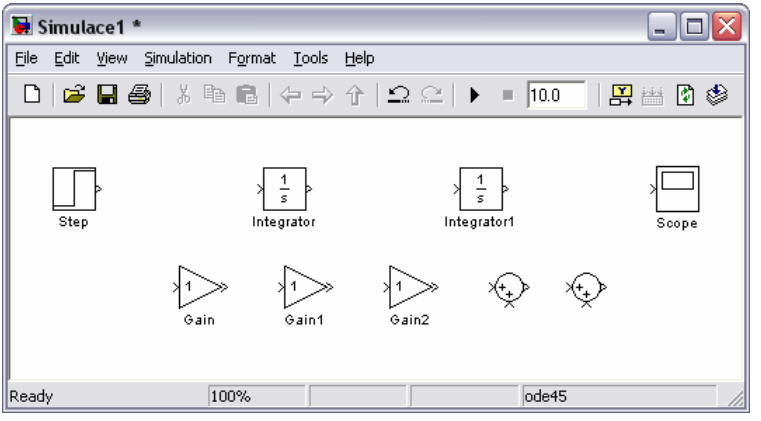
Kromě přesunu z knihovny je možné i bloky v okně modelu kopírovat – standardním způsobem (Ctrl + C a následně Ctrl + V) nebo pomocí myši. Kopírování pomocí myši provedeme tak, že nejprve vybereme kliknutím blok, který chceme kopírovat (po výběru je zřetelně označen), následně stiskneme pravé tlačítko myši (u kurzoru se objeví symbol plus), přesuneme kurzor na požadované místo a uvolníme stisknutí tlačítka. V tento okamžik již máme na ploše druhý blok integrátoru, který je zřetelně označen, viz obr. 6.22. U názvu okna se objevil po vložení nového bloku symbol hvězdičky, který signalizuje, že došlo ke změně v okně modelu a model je vhodné uložit.



Obr. 6.22: Kopírování bloků pomocí myši

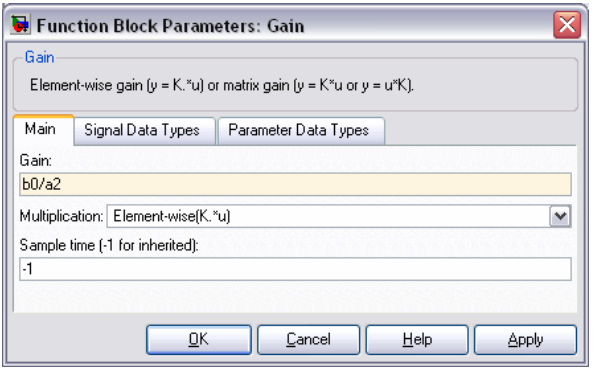
Název nového bloku se nastaví zcela automaticky tak, aby byl v rámci okna modelu jednoznačný. V našem případě má první blok (přetažený z knihovny) název *Integrator*, druhý blok (zkopírovaný) má pak název *Integrator1*.

Obdobným způsobem umístíme do okna modelu z knihovny ***Math Operations*** tři bloky *Gain* a dva bloky *Sum* (případně bloky *Add* mající stejné vlastnosti, ale odlišný vzhled, viz obr. 6.12). Dále umístíme z knihovny ***Sources*** blok *Step* a z knihovny ***Sinks*** blok *Scope*. Bloky (obr. 6.23) je vhodné před jejich vlastním propojením v okně modelu rozmístit, případně některé z nich i otočit tak, aby výsledné zapojení bylo přehledné.



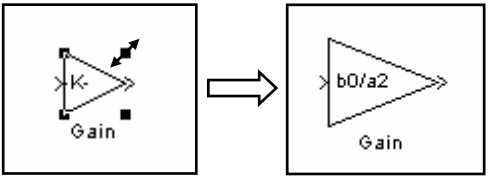
Obr. 6.23: Počáteční rozmístění bloků v okně modelu

Bloky *Integrator*, *Integrator1*, *Step* a *Scope* prozatím ponecháme beze změn. Nejprve se zaměříme na bloky *Gain*, *Gain1* a *Gain2*. Blok *Gain* využijeme k realizaci násobení signálu u(t) konstantou b_0 / a_2, bloky *Gain1* a *Gain2* k vynásobení signálu y(t) konstantou $- a_1 / a_2$ resp. konstantou $- a_0 / a_2$. Bloky *Gain1* a *Gain2* otočíme tak, aby byly ve směru toku signálu. Otočení (rotaci) bloku lze provést pomocí menu **Format → Rotate Block** nebo prostřednictvím zkratkové klávesy Ctrl + R. Rotaci je na uvedené bloky nutné aplikovat dvakrát, neboť po prvním použití je blok orientován svisle (otočení je vždy směrem doprava) a až po druhém použití je blok otočen zrcadlově. Další možností je zrcadlové otočení bloku – opět pomocí menu **Format → Flip Block** resp. prostřednictvím klávesy Ctrl + I.



Obr. 6.24: Zadání parametrů bloku *Gain*

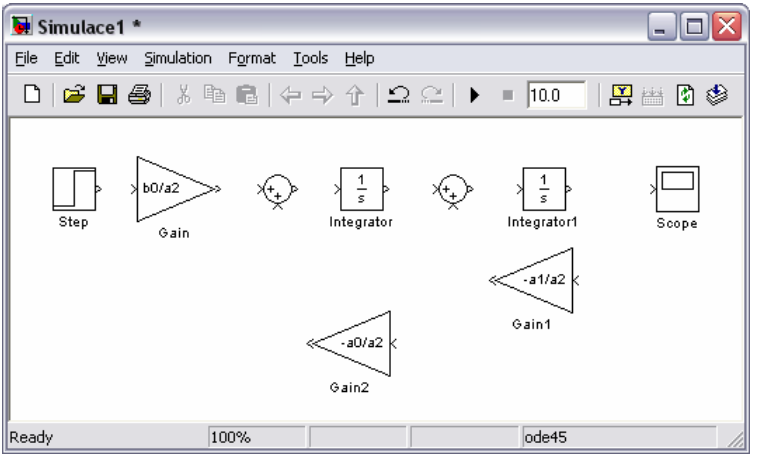
V tomto okamžiku je účelné již zadat konkrétní hodnoty parametrů zesilovačů a případně také změnit jejich velikost. Implicitně je ve všech blocích zesilovačů nastavena hodnota 1, viz obr. 6.23. Dvojitým kliknutím na bloku *Gain* otevřeme dialogové okno parametrů bloku (obr. 6.24) a v záložce **Main** zadáme v položce **Gain** konstantu b_0 / a_2. Bezprostředně po zadání se změní podbarvení položky, což signalizuje, že došlo ke změně parametru. U ostatních parametrů ponecháme implicitní hodnoty. Poznamenejme pouze, že položka **Multiplikation** umožňuje pomocí roletového menu změnit implicitně nastavené prvkové násobení (**Element--wise K.*u**) na násobení maticové a to i včetně pořadí (**Matrix K*u** a **Matrix u*K**). Položka **Sample time** umožňuje volbu vzorkování resp. kroku výpočtu; obvykle se doporučuje ponechat implicitní hodnotu **–1** a nastavení kroku výpočtu řešit centrálně v parametrech simulace. Obdobným způsobem definujeme i parametry zesilovačů *Gain1* a *Gain2*.



Obr. 6.25: Změna velikosti bloku

Po zadání konstanty b_0 / a_2 do bloku *Gain* se změní vzhled tohoto bloku (obr. 6.25) a na místo původního symbolu **1** je uvnitř bloku zobrazen symbol **-K-**. Pro zlepšení přehlednosti, zejména u rozsáhlých simulačních schémat, je vhodné zvětšit velikost bloku tak, aby byla viditelná zadaná hodnota zesílení. Zvětšení bloku lze provést uchopením vybraného bloku za

rohovou značku (změní se kurzor myši) a tažením tak dlouho, dokud není zadaná hodnota viditelná, viz obr. 6.25.



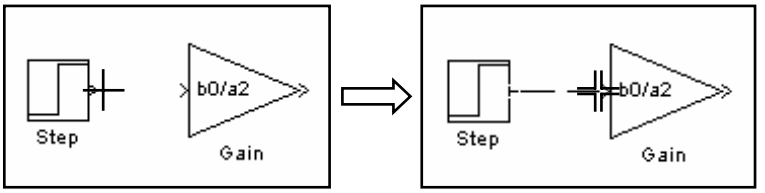
Obr. 6.26: Optimální rozmístění bloků před vlastním zapojením

Situace po provedení shora uvedených změn je na obr. 6.26. Nyní již můžeme postupně provést vlastní propojení bloků modelu. Naprostá většina bloků Simulinku má alespoň jeden port sloužící k propojení s ostatními bloky. Např. blok *Gain*, má jeden vstupní port a jeden výstupní port. Na obr. 6.28 je ukázán postup vzájemného propojení bloku *Step* (má pouze výstupní port) s blokem *Gain*.

Obr. 6.27: Tip pro automatické propojování bloků

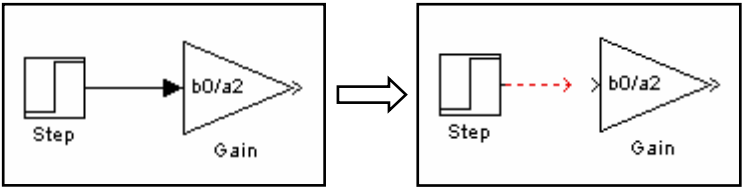
Při prvním pokusu o propojení bloků nás Simulink upozorní na možnost rychlého automatického propojení bloků – dojde k zobrazení okna *Automatic Block Connection Tip* (obr. 6.27). Aby nedocházelo nadále k zobrazování tohoto okna, může uživatel zatrhnout volbu **Do not show this message again**. Rychlé propojení dvou bloků lze provést tak, že nejprve levým tlačítkem myši zvolíme první blok a současně stiskneme a držíme klávesu Ctrl. Následně myší zvolíme druhý blok a uvolníme klávesu Ctrl – bloky jsou navzájem propojeny.

Výhodné je propojování bloků směrem od výstupu prvního bloku ke vstupu bloku druhého. Nejdříve umístíme kurzor myši na výstupní port bloku *Step* (kurzor se změní na kříž), následně stiskneme levé tlačítko a myší táhneme až ke vstupnímu portu bloku *Gain*. Tlačítko myši uvolníme až v okamžiku, kdy se kurzor změní na dvojitý kříž, viz obr. 6.28.



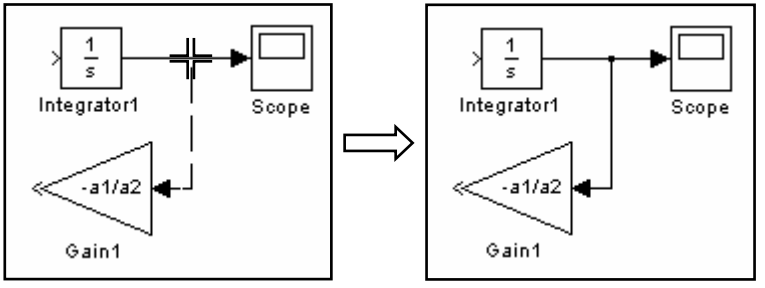
Obr. 6.28: Propojování bloků

Jedině tak zajistíme správné spojení bloků. Pokud tlačítko uvolníme dříve, nedojde ke spojení a vlastní spoj je zobrazen červenou čárkovanou čarou, viz obr. 6.29. Nedokončený spoj je ovšem možné opravit uchopením jeho konce a dotažením až k bloku.



Obr. 6.29: Správné a chybné spojení bloků

Pokud potřebujeme signál větvit, je vhodnější postupovat v propojování od cílového bloku směrem ke spoji, na který chceme navázat. Zalomení spoje je zcela automatické. Nejprve opět umístíme kurzor myši na vstupní port bloku *Gain1* (pozor, blok je již otočen a vstupní port je na pravé straně) a stejným způsobem jako na obr. 6.28 myší táhneme směrem doprava a poté směrem nahoru. Tlačítko myši uvolníme až v okamžiku změny kurzoru na dvojitý kříž, viz obr. 6.30.

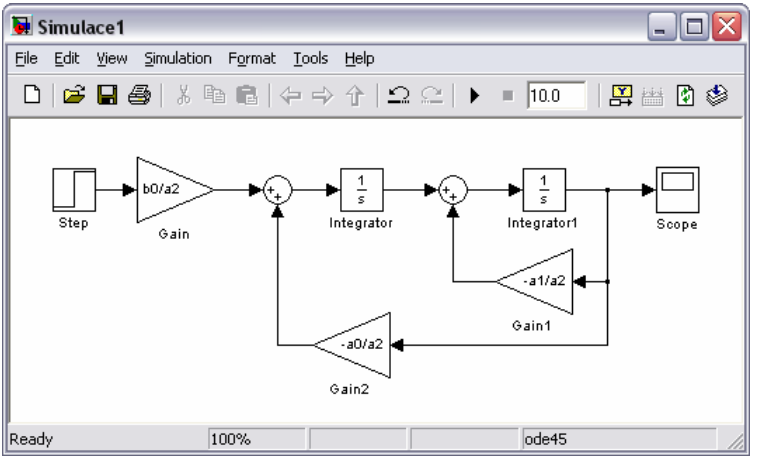


Obr. 6.30: Větvení signálů

Zbývá již jen dokončit vzájemné propojení ostatních bloků. Výsledný model dynamického systému je na obr. 6.31. Závěrem ještě uveďme, že výstup jednoho bloku může být připojen na libovolný počet vstupů jiných bloků či toho samého bloku. V případě potřeby většího počtu pravoúhlých zalomení je možné tažený spoj ukončit bez připojení ke konkrétnímu bloku (tento spoj je zobrazen červenou čárkovanou čarou, tak jako na obr. 6.29) a z tohoto místa táhnout další spoj.

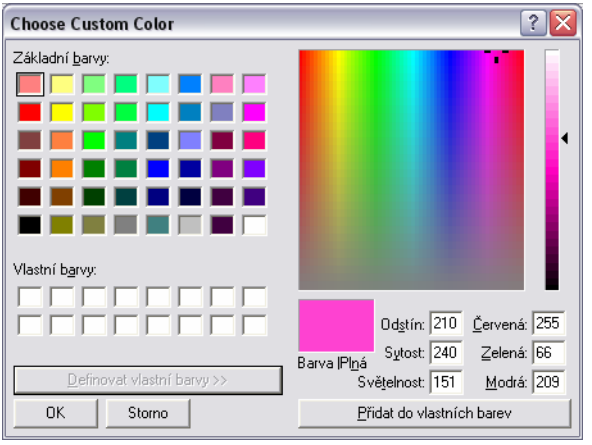
Simulink také umožňuje jednoduše měnit vzhled bloků. Po kliknutí pravým tlačítkem myši nad zvoleným blokem dojde k zobrazení plovoucího menu. Prostřednictvím tohoto menu je možné, mimo základních možností pro kopírování a mazání bloku a přístupu k oknu parametrů

bloku, také nastavit font popisky bloku, umístění této popisky, barvu pozadí a obrysu bloku, případně zobrazit kontextovou nápovědu.



Obr. 6.31: Výsledný model dynamického systému

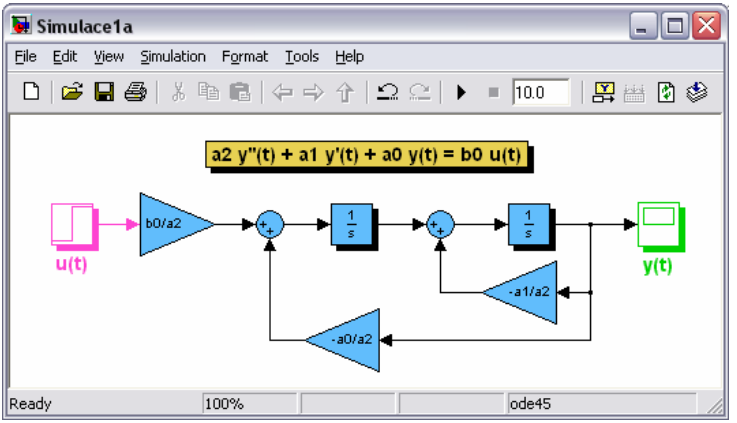
Font popisky a její umístění lze nastavit, zvolíme-li v plovoucím menu **Format** jeho položky **Font...** resp. **Flip Name** či položku **Hide Name** pro úplné odstranění popisky. Další možností je zvolení položky **Show Drop Shadow**, která zvýrazní vybraný blok doplněním o stín. Barvu obrysu bloku můžeme změnit, zvolíme-li v položce **Foreground Color** některou z nabízených barev, případně zvolíme **Custom** a vybereme barvu z rozsáhlé nabídky. Požadovanou barvu si může uživatel pomocí barevné palety i sám namíchat. Zvolí-li v dolní části okna položku Definovat vlastní barvy, dojde k rozšíření okna *Choose Custom Color* o barevnou paletu, viz obr. 6.32.



Obr. 6.32: Definování vlastní barvy

Stejným způsobem je možné změnit i výplň bloku – prostřednictvím položky **Background Color**. K dispozici máme stejné možnosti volby barev, včetně možnosti definování požadované barvy uživatelem.

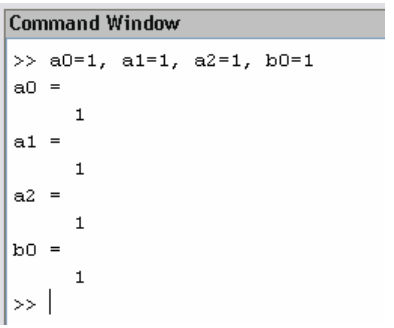
Dvojitým kliknutím na popisku vybraného bloku přejdeme do režimu editace (uvnitř popisky se objeví svislý kurzor) a standardním způsobem lze text popisky přepsat. Není ale možné označit dva bloky stejným textem. V libovolném místě okna modelu je také možné umístit komentář. Dvojklikem na zvoleném místě se uživatel dostane do režimu editace a může zapsat požadovaný text. U textu komentáře lze obvyklým způsobem měnit velikost, řez a barvu písma, barvu pozadí apod. V nabídce okna modelu **Format** volíme položky **Font...**, **Text Alignment** (**Left**, **Center** nebo **Right**), **Show/Hide Drop Shadow**, **Foreground Color** a **Background Color**. Můžeme změnit i pozadí celého okna – k tomu slouží položka **Screen Color**. Vzhled modelu (obr. 6.31) můžeme pomocí popisovaných úprav změnit např. do podoby na obr. 6.33.



Obr. 6.33: Upravený vzhled modelu

6.3 Spuštění simulace a zobrazení výsledků

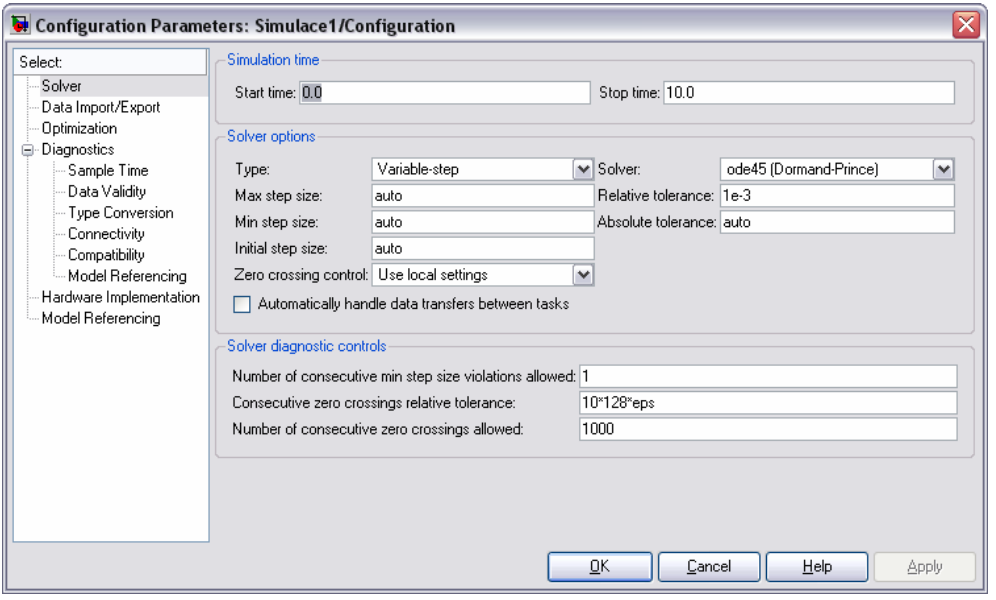
Model dynamického systému vytvořený v předchozím odstavci je již ve stavu, kdy je možné přejít k vlastní simulaci. Je ale třeba ještě definovat koeficienty a_0, a_1, a_2 a b_0, které byly zadány v zesilovačích pomocí proměnných a0, a1, a2 a b0. Nejjednodušší je zadat tyto hodnoty přímo v příkazovém okně MATLABu. Pro jednoduchost byly všechny koeficienty zvoleny jednotkové, viz obr. 6.34.



Obr. 6.34: Definování proměnných

Před spuštěním simulace je třeba nastavit základní parametry simulace. Po otevření nabídky **Simulation** z okna modelu a výběru položky **Configuration Parameters...** (nebo po stisku Ctrl + E) se otevře okno parametrů simulace (obr. 6.35) se záložkami (jsou v levé části okna v nabídce *Select*) **Solver**, **Data Import/Export**, **Optimization**, **Diagnostics**, **Hardware Implementation** a **Model Referencing**. Pro základní práci s prostředím Simulink budou postačovat nastavení pouze v záložce **Solver** příp. **Data Import/Export**.

V záložce **Solver** je nejdůležitější nastavit v části *Simulation time* počátek simulace **Start time**, zde zpravidla ponecháme nulu, a zejména pak konec resp. délku simulace **Stop Time**. Pro simulaci v textu popisovaného systému bude postačovat čas 15 s. Nastavení numerické metody výpočtu, kroku výpočtu a tolerancí je možné v části okna *Solver options*. Zde obvykle ponecháme parametr týkající se velikosti kroku (**Variable-step**, druhou variantou je **Fixed Step**) a pokud chceme zvýšit přesnost výpočtu, změníme pouze hodnoty **Max step size**, **Min step size** a **Initial step size**.



Obr. 6.35: Okno parametrů simulace – nastavení řešitele

Implicitní nastavení u všech těchto parametrů je **auto**. Parametr **Max step size** je Simulinkem volen automaticky podle zvolené doby simulace $dt_{max} = (t_{stop} - t_{start}) / 50$. V našem případě, kdy jsme zvolili čas simulace 15 s, Simulink nastaví automaticky podle uvedeného vzorce tuto hodnotu na 0,3 s (v příkazovém okně MATLABu je dokonce zobrazeno i hlášení – Warning). Pro zvýšení přesnosti nastavíme **Initial step size** a **Min step size** na hodnotu 0,001 a hodnotu **Max step size** na desetinásobek, tedy na 0,01. Tolerance (**Relative tolerance** a **Absolute tolerance**) ponecháme beze změny.

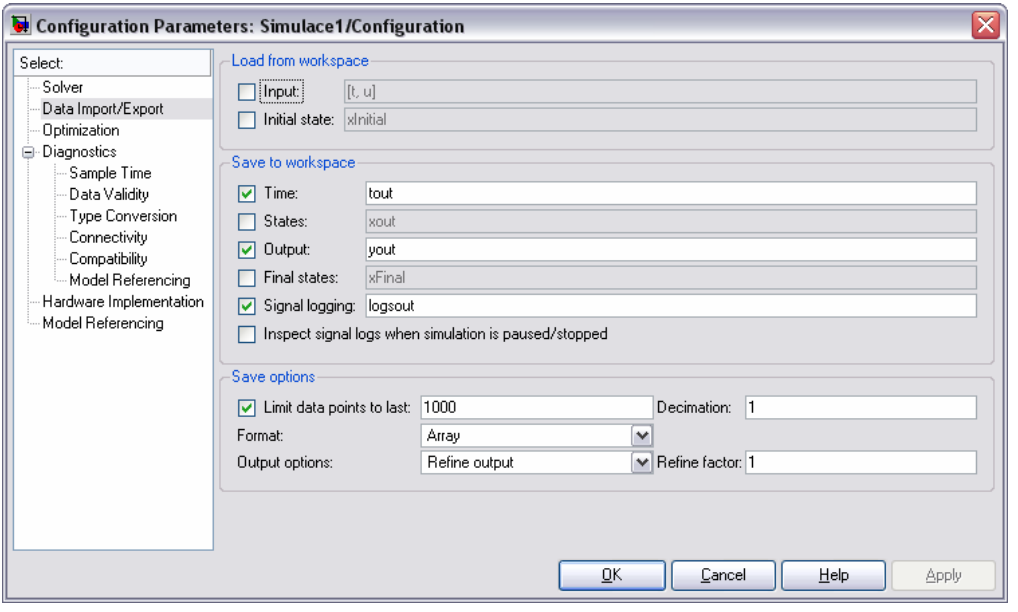
Parametr **Solver** umožňuje zvolit numerickou metodu výpočtu. Implicitně je nastavena metoda **ode45 (Dormand-Prince)** a není třeba ji většinou měnit. Přehled všech možných metod výpočtu podává tab. 6.1. Často užívaný pojem *stiff* označuje tzv. tuhé (těžko řešitelné) systémy. Jedná se např. o elektrické obvody s velmi malými a současně i velkými časovými

konstantami, které jsou obtížně numericky řešitelné (doba řešení, stabilita, konvergence). Zvolená numerická metoda je zobrazena také ve status baru okna modelu, viz obr. 6.31 a 6.33.

Tab. 6.1: Přehled numerických metod

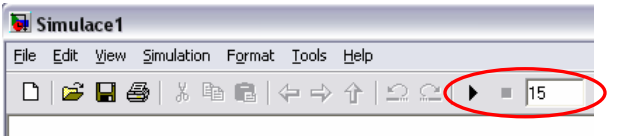
Řešitel (solver)	Třída problémů	Metoda
ode45 (Dormand-Prince)	non-stiff	explicitní Runge-Kutta (implementace Dormand-Prince)
ode23 (Bogacki-Shampine)	non-stiff	explicitní Runge-Kutta (implementace Bogacki-Shampine)
ode113 (Adams)	non-stiff	Adams-Bashforth-Moultonova
ode15s (stiff/NDF)	stiff	NDFs (BDFs)
ode23s (stiff/Mod. Rosenbrock)	stiff	Rosenbrockova
ode23t (Mod. stiff/trapezoidal)	moderately stiff	lichoběžníkové pravidlo
ode23th (stiff/TR–BDF2)	stiff	implicitní Runge-Kutta (implementace TR–BDF2)

Na obr. 6.36 je v okně parametrů simulace zobrazena záložka **Data Import/Export**. Zde lze nastavit vstup (část okna *Load from workspace*) či výstup (*Save from workspace*) dat z prostoru proměnných. Je možné ukládat do proměnných čas výpočtu, stavy a výstupy modelu, atd. Pokud se rozhodne uživatel využít této možnosti, nastaví v části *Save Options* položku **Format** na **Array** a případně změní či zcela zruší limitaci délky výstupních dat (**Limit data points to last**).



Obr. 6.36: Okno parametrů simulace – vstup a výstup dat z modelu

Nyní je již možné spustit simulaci vytvořeného modelu. První možností je spuštění simulace z nabídky okna modelu **Simulation → Start** (nebo Ctrl + T). Rychlejší je ale spuštění prostřednictvím ikony ► v toolbaru okna, viz obr. 6.37. Napravo od této ikony je také možné rychle zadávat délku simulace; je tam uvedena hodnota zadaná prostřednictvím okna parametrů simulace. Pokud dojde k neočekávaným problémům v průběhu simulace nebo pokud je zvolen příliš dlouhý čas výpočtu (resp. příliš malý krok), lze simulaci zastavit pomocí ikony ■.



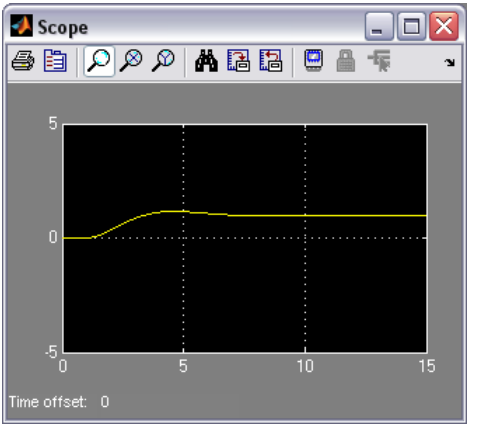
Obr. 6.37: Rychlé ovládání běhu simulace

Průběh výpočtu je signalizován ve status baru okna modelu, zcela vlevo dole je zobrazován stav *Running* – běží. Přibližně uprostřed je pak aktuální čas, konkrétně např. na obr. 6.38 T = 7,423 s. Po ukončení simulace se stav změní zpět na *Ready*, viz obr. 6.39.

Obr. 6.38: Signalizace průběhu simulace

Obr. 6.39: Signalizace stavu bez probíhající simulace

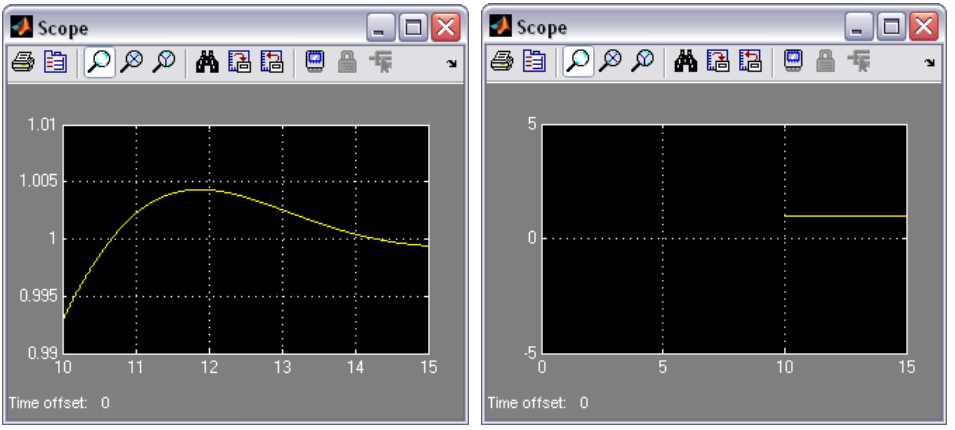
Po ukončení výpočtu můžeme dvojitým kliknutím na ikonu zobrazovače (blok *Scope*) otevřít okno s grafickým záznamem řešení, viz obr. 6.40. Zobrazovací okno má implicitně nastavené určité měřítko a pokud jsou obdržené výsledky výrazně jiné, nemusíme nic vidět. V tomto případě je nutné manuálně změnit měřítko, např. pomocí ikony se symbolem dalekohledu nastavíme měřítko tak, aby se zobrazilo vše.



Obr. 6.40: Zobrazení řešení

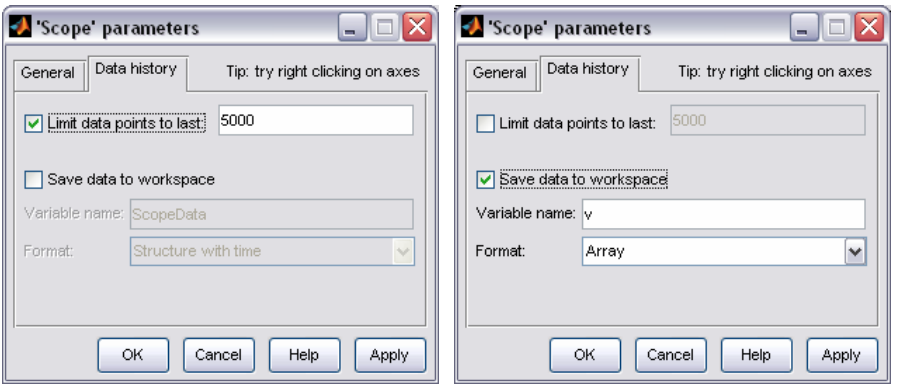
Pokud je čas simulace příliš dlouhý resp. krok výpočtu malý, dojde k vykreslení pouze posledních 5000 vzorků. Ponecháme-li délku simulace 15 s a zvolíme-li maximální krok

simulace např. 0,001 a minimální krok simulace 0,0001, dojde k vykreslení pouze části průběhu. Konkrétně pořadnic v čase od 10 do 15 s – ostatní data jsou ztracena, viz obr. 6.41.



Obr. 6.41: Ztráta dat při limitaci počtu vzorků na posledních 5000

Uvedený problém lze vyřešit zrušením implicitně nastavené limitace prostřednictvím okna parametrů bloku, které uživatel vyvolá pomocí ikony (obr. 6.41) nacházející se v toolbaru nalevo od ikony tisku. V okně *'Scope' parameters* (obr. 6.42) zvolíme záložku **Data history** a zrušíme uvedenou limitaci odznačením parametru **Limit data points to last**.

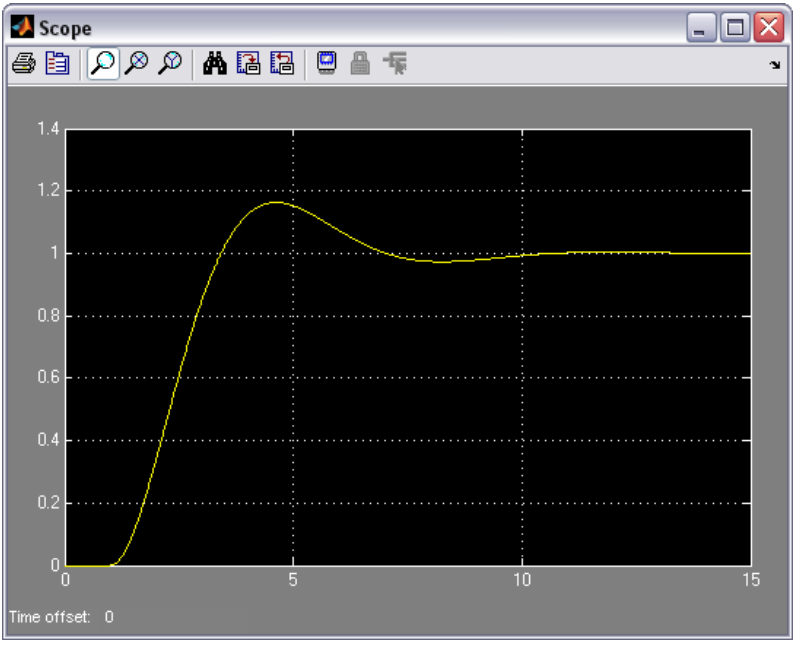


Obr. 6.42: Nastavení parametrů zobrazovače

Okno zobrazovače je vhodné také obvyklým způsobem zvětšit (nebo i maximalizovat). Po nastavení měřítek může být výsledek simulace zobrazen např. jako na obr. 6.43. Pokud ale chceme výsledky simulace prezentovat např. v nějakém textu, ve většině případů v tištěné formě, není možné použít přímo grafického výstupu zobrazovače. Převážně z důvodu nemožnosti nastavení změny barvy pozadí, barvy a šířky čar apod., je výstup z bloku *Scope* pro tyto účely přinejmenším nevhodný.

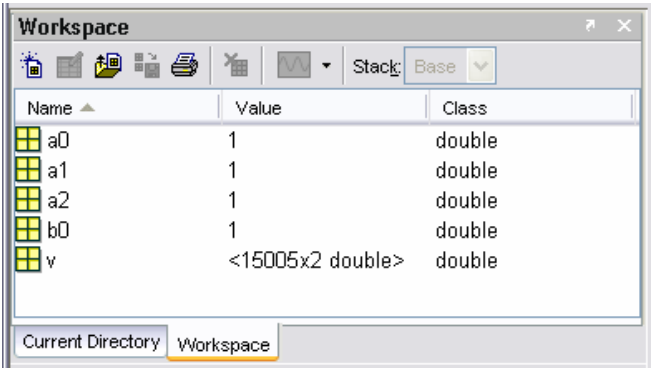
Velmi užitečné je tedy uložení dat získaných simulací do prostoru proměnných. V okně parametrů zobrazovače (obr. 6.42), opět v záložce **Data History**, označíme položku **Save data to workspace**. Zvolíme název proměnné **Variable name**, v našem případě **v**, a formát ukládaných

dat **Format** změníme na **Array** namísto implicitně nastaveného typu **Structure with Time**. Po proběhnutí simulace již můžeme uložená data vykreslit v prostředí MATLABu zcela běžným způsobem (kapitola 4). V prostoru proměnných se objeví proměnná **v** rozměru 15005×2, viz obr. 6.44.

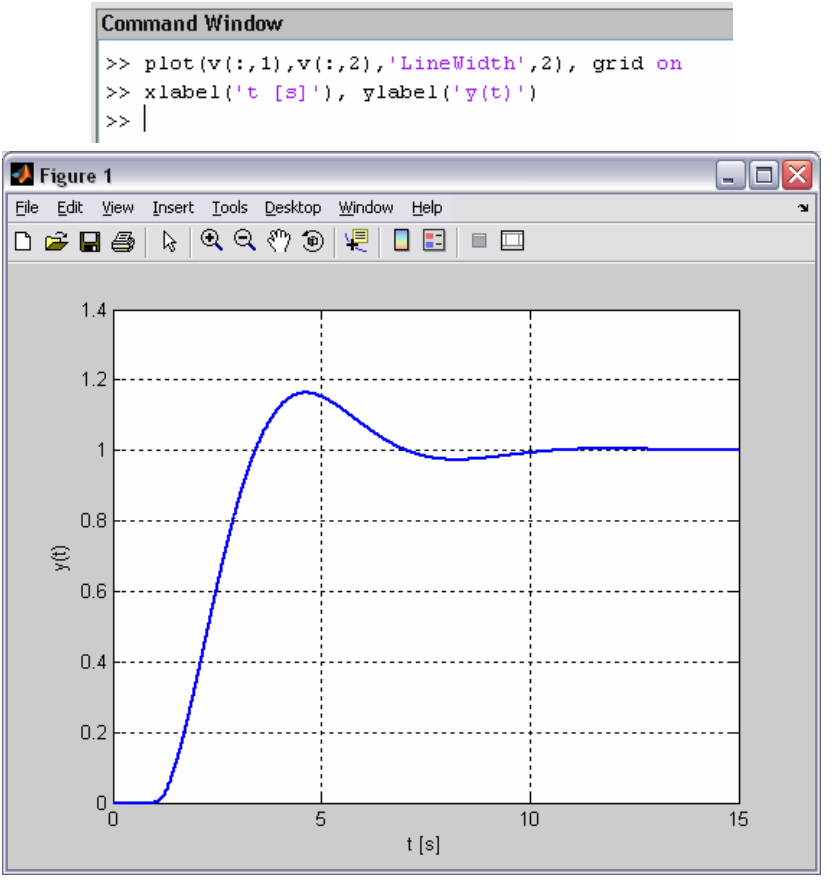


Obr. 6.43: Zobrazení řešení při vhodném nastavení měřítek

Vykreslení dat provedeme pomocí standardní funkce **plot**. S ohledem na strukturu dat musí být syntaxe příkazu **plot(v(:,1),v(:,2))**. První řádek matice v obsahuje vektor času, druhý pak vektor pořadnic simulované přechodové odezvy. Výsledný průběh je na obr. 6.45. Pokud zadáme příkaz pouze ve tvaru **plot(v)**, dojde k vykreslení obou vektorů v závislosti na kroku výpočtu – průběh je chybný.



Obr. 6.44: Uložení dat v prostoru proměnných

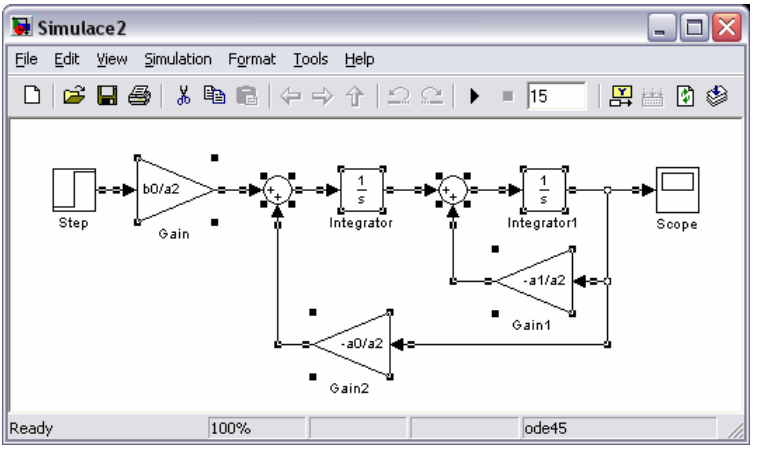


Obr. 6.45: Vykreslení řešení pomocí MATLABu

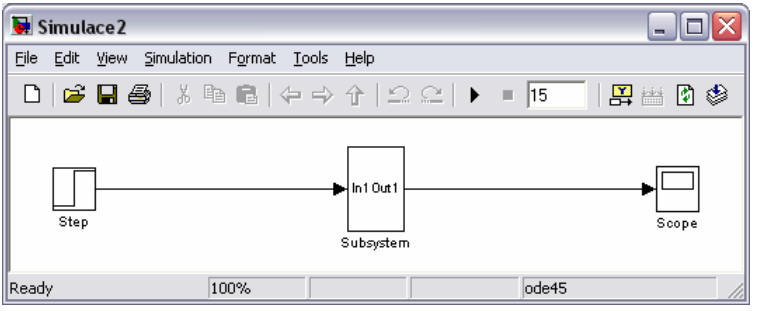
6.4 Vytváření subsystémů a knihoven

Někdy je užitečné zahrnout část simulačního schématu do jednoho bloku – subsystému. Obzvláště, je-li simulační schéma již značně rozsáhlé a ponechání celého modelu v jednom okně by bylo na úkor přehlednosti. Zajímavé možnosti Simulink poskytuje také při vytváření vlastních (uživatelských) knihoven. Můžeme např. vytvořit knihovnu, která bude obsahovat nejrůznější varianty regulátorů.

Pokusme se nyní vytvořit vlastní subsystém. Budeme vycházet z modelu na obr. 6.31 a nejprve jej uložíme pod názvem *Simulace2*. Nejjednodušší způsob jak zahrnout část schéma do bloku subsystému je výběr této části (viz obr. 6.46) s následnou volbou položky **Create Subsystem** z menu **Edit** okna modelu (nebo Ctrl + G). Výsledný vzhled schéma modelu je na obr. 6.47. Nově vzniklý blok má název *Subsystem* a jeho ikona obsahuje i popis vstupů a výstupů (v tomto případě In1 a Out1).

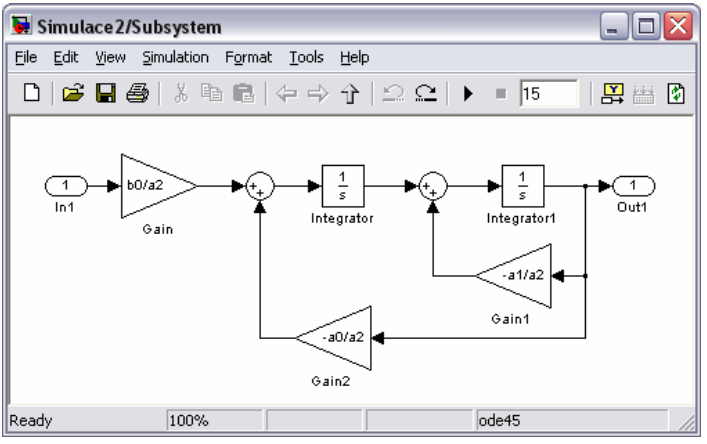


Obr. 6.46: Výběr části modelu pro zahrnutí do subsystému



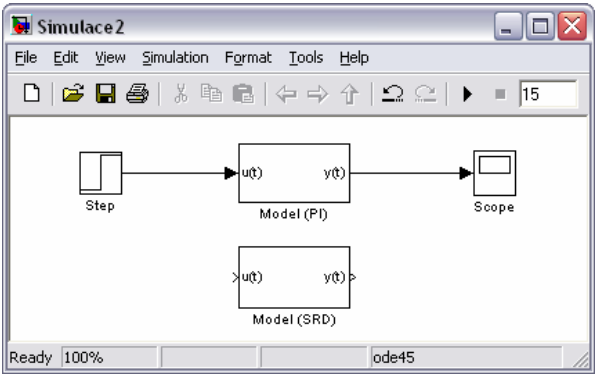
Obr. 6.47: Vytvoření subsystému

Dvojklikem na bloku otevřeme okno subsystému, které je pojmenované *Simulace2/Subsystem*, viz obr. 6.48. Toto okno poskytuje stejné možnosti jako základní okno modelu, např. přístup k parametrům simulace, spuštění simulace apod.



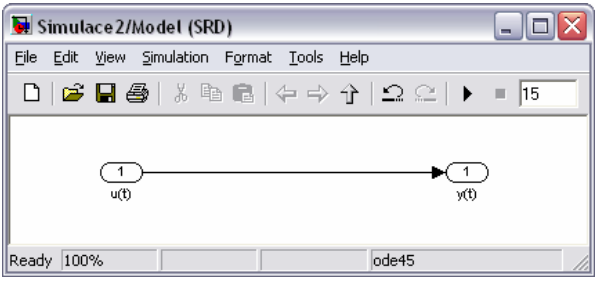
Obr. 6.48: Okno subsystému

Pro přehlednost je velmi vhodné přejmenovat vlastní blok subsystému, ale i všechny jeho vstupy a výstupy. Subsystém v našem případě pojmenujeme jako **Model (PI)**, tato změna se projeví i v názvu okna subsystému, vstup přejmenujeme z **In1** na **u(t)** a výstup z **Out(1)** na **y(t)**. Přejmenování vstupů a výstupů je nutné provést přímo z okna subsystému, základní okno modelu tuto možnost neposkytuje.



Obr. 6.49: Doplnění modelu o další subsystém

Druhou možností jak vytvořit subsystém, je vložení příslušného bloku do okna modelu. K tomuto účelu slouží blok *Subsystem* z knihovny ***Ports & Subsystems*** (viz obr. 6.15). Nově vložený blok má název opět název *Subsystem* (první subsystém byl již přejmenován a proto není v názvu číslice jak bychom očekávali). Přejmenujme jej tedy na **Model (SRD)** a jeho porty na **u(t)** resp. **y(t)**, obdobně jako v předchozím případě. Vzhled modelu by měl být podobný jako na obr. 6.49.



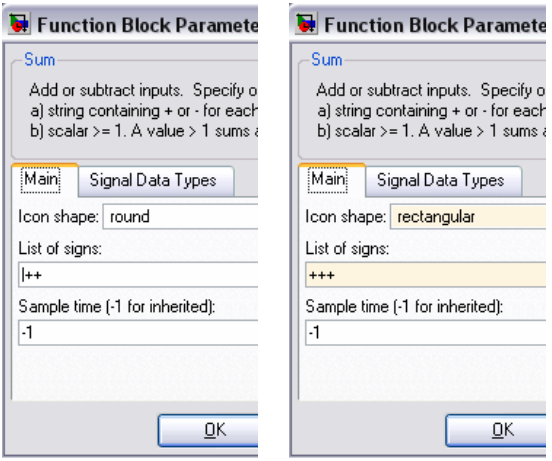
Obr. 6.50: Okno nového subsystému

Otevřeme-li okno subsystému **Model (SRD)** (obr. 6.50), zjistíme, že obsahuje pouze navzájem propojené vstupní a výstupní porty. Pokusme se nyní v rámci tohoto subsystému realizovat model shora popisovaného dynamického systému na základě jeho diferenciální rovnice metodou snižování řádu derivace. Připomeňme, že dynamický systém je druhého řádu a je popsán diferenciální rovnicí ve tvaru $a_2\, y''(t) + a_1\, y'(t) + a_0\, y(t) = b_0\, u(t)$.

Z uvedené diferenciální rovnice stačí jednoduše vyjádřit nejvyšší derivaci

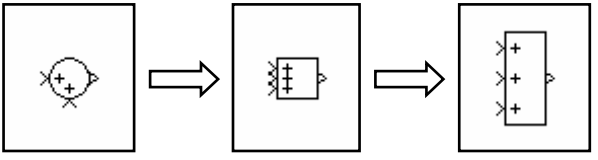
$$y'' = -\frac{a_1}{a_2}\, y' - \frac{a_0}{a_2}\, y + \frac{b_0}{a_2}\, u$$

a provést vlastní zapojení. Pro zjednodušení práce můžeme většinu bloků zkopírovat ze subsystému **Model (PI)**. Využijeme oba integrátory a všechny zesilovače včetně zadaných hodnot zesílení. K vlastnímu zapojení bude postačovat pouze jeden blok *Sum*, je ale třeba upravit počet jeho vstupů. V okně parametrů bloku (obr. 6.51) změníme v záložce **Main** typ ikony **Icon Shape** z původního **round** na **rectangular**. Dále změníme počet a typ vstupů definovaný v položce **List of signs**. Původní nastavení vstupů |++ nahradíme +++ a blok dále také zvětšíme, viz obr. 6.52.



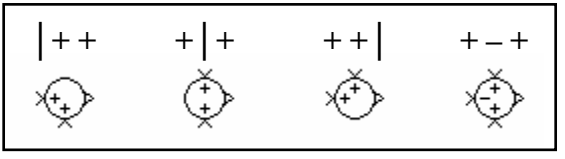
Obr. 6.51: Nastavení tvaru a počtu vstupů bloku *Sum*

Symbol | představuje neobsazený port a má smysl jej používat pouze u kruhového tvaru bloku *Sum*. Pokud potřebujeme, aby některý z portů měl záporné znaménko, použijeme místo symbolu plus (+) symbol mínus (–).



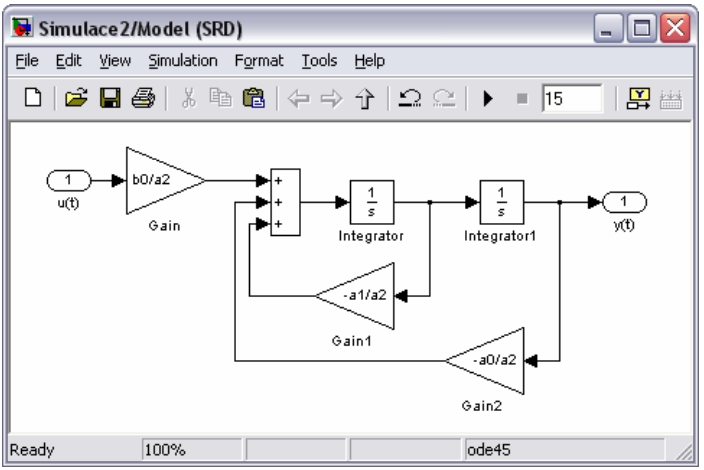
Obr. 6.52: Změna tvaru a počtu vstupů bloku *Sum*

Na obr. 6.53 je uvedeno několik možností konfigurace portů bloku *Sum* kruhového tvaru. Pokud uživatel potřebuje více jak tři vstupy, je vhodnější použití obdélníkového tvaru.



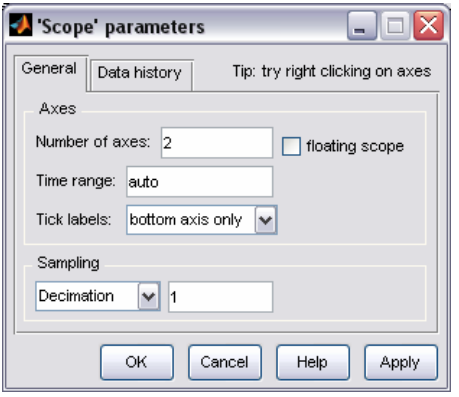
Obr. 6.53: Několik možností konfigurace portů bloku *Sum*

Výsledný model realizovaný metodou snižování řádu derivace (SŘD), tedy subsystém s názvem **Model (SRD)**, je na obr. 6.54.



Obr. 6.54: Model realizovaný metodou SŘD

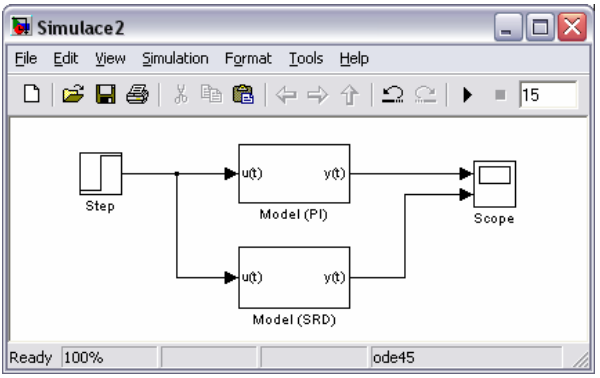
Zbývá pouze dokončit zapojení v hlavním okně modelu. Vstup subsystému **Model (SRD)** připojíme na signál generovaný blokem *Step*. Uživatele by jistě zajímalo, zda jsou výsledky simulace získané řešením na základě dvou různých modelů totožné. Z tohoto důvodu není příliš vhodné vložení dalšího bloku zobrazovače. Lepším řešením je změna konfigurace zobrazovače tak, aby bylo možné sledovat oba průběhy pokud možno v jednom okně.



Obr. 6.55: Nastavení zobrazení více průběhů v jednom zobrazovači

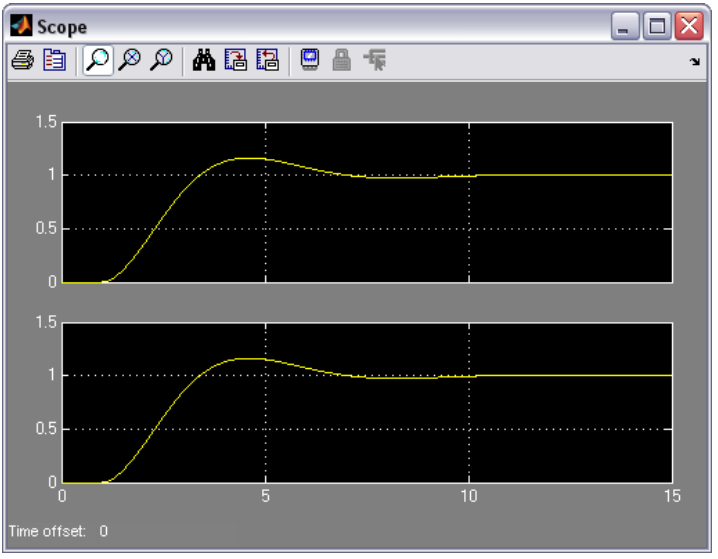
V okně parametrů bloku *Scope* nastavíme v záložce **General** v položce **Number of axes** hodnotu **2**. Po potvrzení má blok *Scope* již dva vstupy a umožňuje samostatné zobrazení dvou průběhů. Výsledné schéma obsahující dva subsystémy (různé modely stejného dynamického systému) je na obr. 6.56.

Shoda výsledků obou modelů (subsystémy **Model (PI)** a **Model (SRD)**) je patrná z obr. 6.57. V následující kapitole budou ukázány i další možnosti zobrazení více průběhů v jednom zobrazovači.



Obr. 6.56: Výsledné simulační schéma

Velmi užitečnou funkcí Simulinku je tzv. maskování subsystému. Ne vždy je totiž vhodné, aby po otevření subsystému bylo viditelné v samostatném okně jeho vnitřní schéma, tak jak bylo ukázáno výše. Výhodnější může být naopak zobrazení okna, ve kterém uživatel pouze zadá příslušné parametry.

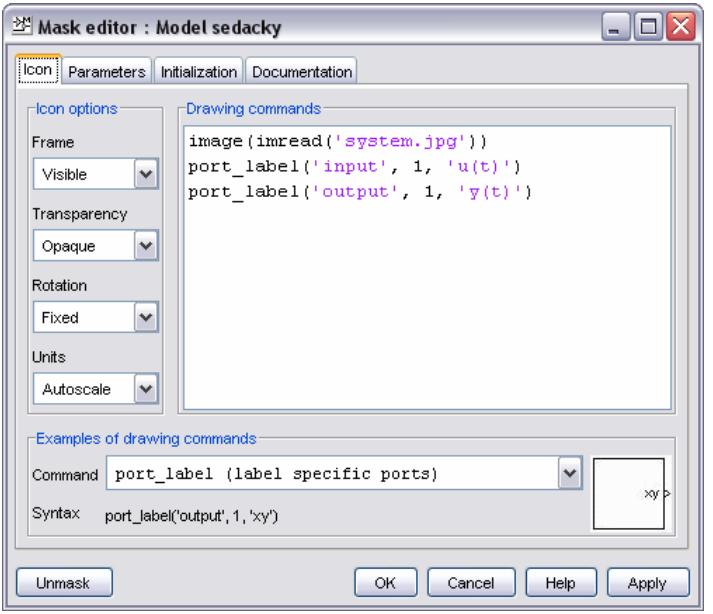


Obr. 6.57: Zobrazení více průběhů v jednom okně

Okno maskování subsystému (obr. 6.58), vyvolané z hlavní nabídky okna modelu volbou **Edit → Mask Subsystem...**, případně volbou z plovoucího menu **Mask Subsystem...** po vybrání subsystému nebo pomocí kombinace kláves Ctrl + M, dovoluje zadat ikonu subsystému, názvy a případné implicitní hodnoty proměnných použitých v parametrech bloků subsystému a popis či nápovědu.

Vytvořený model obecného dynamického systému druhého řádu může po definování příslušných parametrů představovat např. zjednodušený model sedačky řidiče. Pomocí funkcí maskování upravíme např. subsystém **Model (SRD)**.

Diferenciální rovnice nejjednoduššího modelu sedačky má tvar $m\,y'' + c\,y' + k\,y = F\,u$. Tuto diferenciální rovnici obdržíme aplikací d'Alembertova principu rovnováhy sil působících na hmotu m. Výsledná síla je rovna součtu síly setrvačného odporu sedačky, síly vyvozené pružinou a síly vyvozené tlumičem. V pracovním rozsahu předpokládáme lineární chování pružiny i tlumiče. Je zřejmé, že s ohledem na diferenciální rovnici dynamického systému druhého řádu v obecném tvaru, můžeme formálně ztotožnit $m = a_2$, $c = a_1$, $k = a_0$ a $F = b_0$.



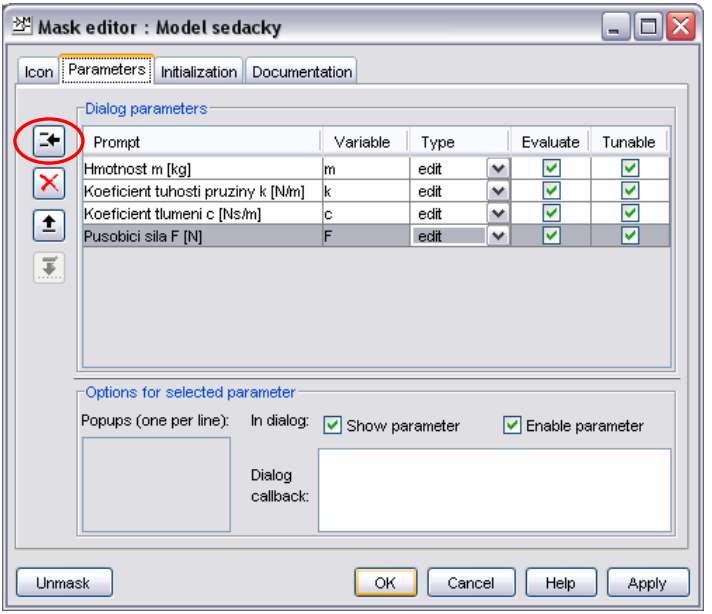
Obr. 6.58: Úprava ikony subsystému

V souvislosti s výše uvedeným, změníme před vlastním maskováním nejprve definici konstant v jednotlivých zesilovačích tak, aby nebyly v rozporu s definovanými proměnnými a0, a1, a2 a b0, které jsou využity v subsystému **Model (PI)**. Touto záměnou umožníme přímé zadávání parametrů modelu prostřednictvím okna parametrů subsystému. Subsystém se nadále bude chovat obdobně jako standardní blok – po dvojitém kliknutí na bloku se namísto vnitřního schématu otevře právě okno pro zadávání parametrů.

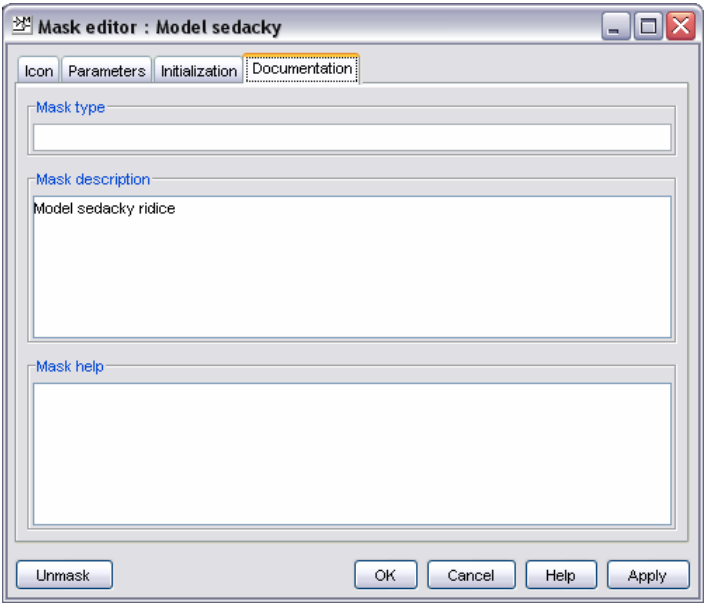
V bloku *Gain* bude nyní konstanta F / m, v bloku *Gain1* konstanta c / m a v bloku *Gain2* konstanta k / m. Subsystém **Model (SRD)** také přejmenujeme na **Model sedacky**. V tuto chvíli již můžeme vyvolat editor masky (např. pomocí Ctrl + M). Editor umožňuje podstatně změnit vzhled ikony. Pomocí speciálních příkazů lze v části okna *Drawing Commands* záložky **Icon** (viz obr. 6.58) do ikony umístit obrázek (soubor musí být v aktuálním adresáři) a současně zachovat i označení jednotlivých portů. Příklady použití jsou uvedeny ve spodní části okna *Examples of drawing commands*.

V záložce **Parameters** okna editoru (obr. 6.59) můžeme definovat jednotlivé parametry, které budou moci být zadávány uživatelem prostřednictvím speciálního okna. Parametry přidáváme pomocí ikony se symbolem šipky. V části **Prompt** zadáme požadovaný název položky a v části **Variable** název jí odpovídající proměnné.

Jakmile je subsystém zamaskován, není již možný obvyklý přístup k jeho vnitřnímu schématu. Úprava takto zamaskovaného systému je možná např. volbou z plovoucího menu **Look Under Mask** nebo prostřednictvím stejně pojmenované položky menu **Edit** okna modelu, příp. pomocí kombinace kláves Ctrl + U.



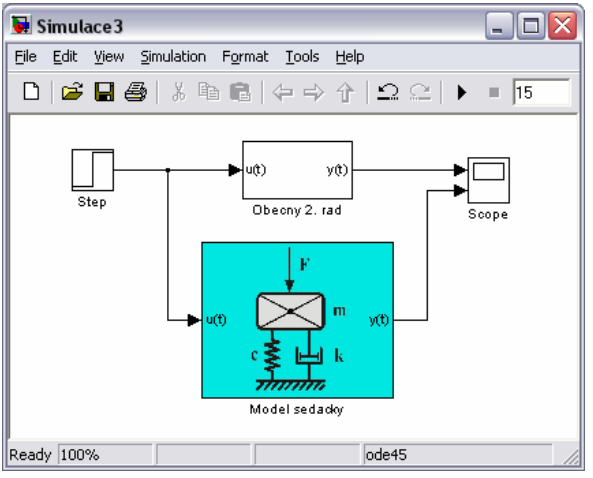
Obr. 6.59: Definování parametrů



Obr. 6.60: Zadání popisu a nápovědy

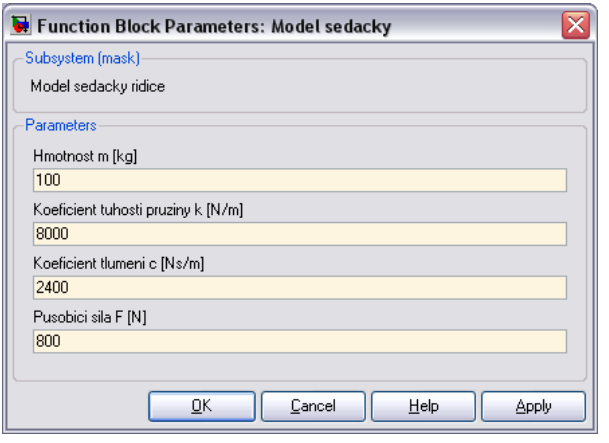
Úprava masky je možná prostřednictvím položky **Edit Mask** (Ctrl + M). Úplné zrušení vytvořené masky je možné provést v editoru masky (obr. 6.58, 6.59 a 6.60). K tomuto účelu je v levém dolním rohu okna k dispozici tlačítko **Unmask**.

V záložce **Initialization** okna editoru masky lze v části *Initialization Command* zadat případné implicitní hodnoty proměnných použitých v parametrech bloku maskovaného subsystému. Je možné zapsat i jiné standardní příkazy MATLABu, které se provedou po zadání hodnot. Záložka **Documentation** umožňuje zadaní nejrůznějších komentářů. Na obr. 6.60 je takto v části *Mask description* zadán text „Model sedacky ridice“, který se pak zobrazuje v okně pro zadávání parametrů (obr. 6.62).



Obr. 6.61: Vzhled maskovaného subsystému

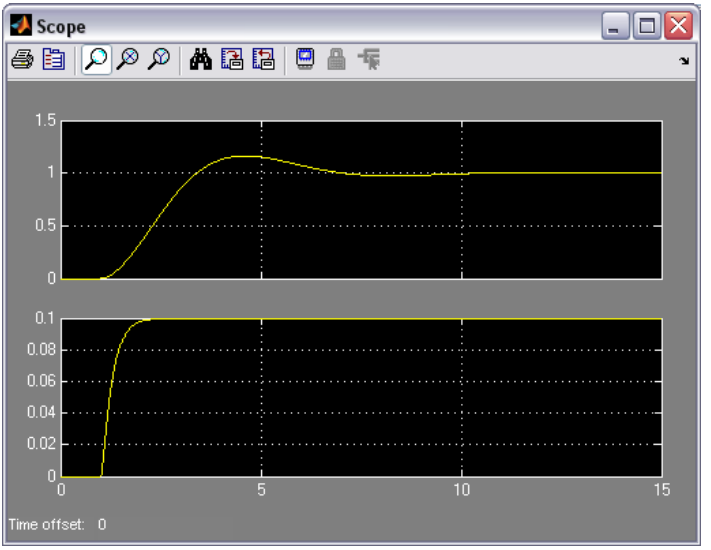
Výsledný vzhled modelu po provedených úpravách je na obr. 6.61. Ikona subsystému **Model sedacky** obsahuje obrázek mechanického modelu sedačky s pružinou a tlumičem. Uvedený obrázek se načítá pomocí funkce **imread** (obr. 6.58) ze souboru system.jpg. Porty ikony jsou standardně označeny a bylo možné je propojit se zbytkem modelu.



Obr. 5.62: Okno parametrů maskovaného subsystému

Po dvojitém kliknutí na ikoně subsystému **Model sedacky** se otevře dialogové okno parametrů (obr. 6.62). V tuto chvíli nezbývá než zadat konkrétní hodnoty parametrů a následně spustit simulaci. Povšimněme si, že jednotlivé parametry (část okna *Parameters*) mají názvy, které jsme dříve definovali (obr. 6.59). Konkrétně zadáme hodnoty m = 100 kg, k = 8000 N/m, c = 2400 Ns/m a F = 800 N.

Po dokončení simulace a otevření okna zobrazovače obdržíme výsledky viz obr. 6.63. Průběh výchylky sedačky řidiče (jedná se o dolní průběh) je aperiodický, tedy bez překmitu. Pokud budeme prostřednictvím okna parametrů (obr. 6.62) měnit např. hodnotu koeficientu tlumení, získáme výsledky jiné. Systém může být kmitavý nebo naopak více tlumený. I když lze očekávat, že po dosednutí řidiče (síla F) dojde k průhybu sedačky směrem dolů, výsledek simulace je opačný – výchylka je kladná. Toto je způsobeno pouze zvolenou orientací osy.

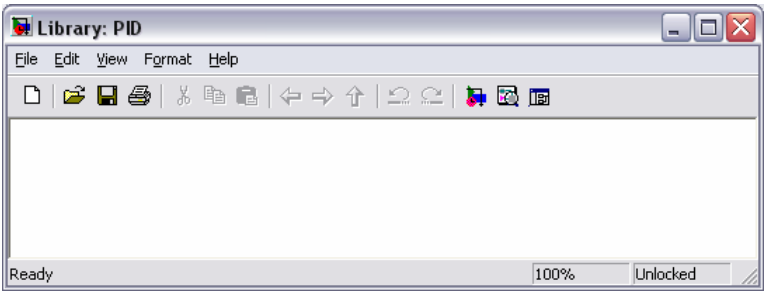


Obr. 6.63: Zobrazení řešení

Popisovaný přístup je výhodný hlavně pro demonstraci vlivu změn parametrů na dynamiku systému, zejména v případě, máme-li poměrně složité simulační schéma a změnou parametrů přímo v blocích nebo prostřednictvím příkazového okna MATLABu by se ztrácela přehlednost. Přístup je také vhodný pro uživatele, který nepotřebuje znát přesně vnitřní strukturu modelu.

Simulink umožňuje také vytváření uživatelských knihoven. Definování knihovny (skupiny bloků) má význam pouze v případě, že chceme tyto bloky opakovaně používat v různých modelech. Uložení bloků v knihovně dovoluje jejich jednoduché vkládání do nově vytvářených schémat. Výhodou je také automatická aktualizace bloků použitých v modelu, pokud dojde k jejich změně v rámci knihovny. Vytvořme nyní knihovnu PID, která bude obsahovat tři varianty spojitého PID regulátoru. Klasický ideální PID regulátor, PID s aproximovanou derivací a PID s vážením žádané hodnoty.

Novou knihovnu vytvoříme volbou **File → New → Library** z nabídky okna *Simulink Library Browser*. Knihovnu běžným způsobem uložíme pod názvem PID. Okno knihovny je velmi podobné klasickému oknu modelu, viz obr. 6.64.

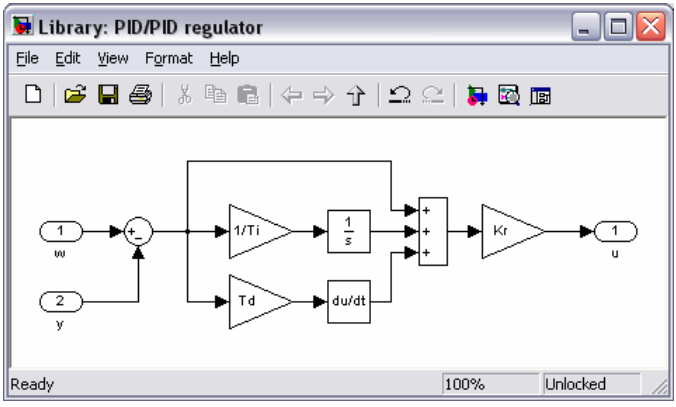


Obr. 6.64: Okno nové knihovny

Pokud otevřeme již existující knihovnu, je chráněna proti nechtěným změnám. Ve status baru (v pravém dolním rohu) je indikován stav *Locked*. Před provedením jakýchkoliv změn je nutné tuto ochranu pomocí volby **Edit → Unlock Library** zrušit – je indikován stav *Unlocked* (obr. 6.64). Uzavřením okna knihovny se ochrana opět automaticky nastaví.

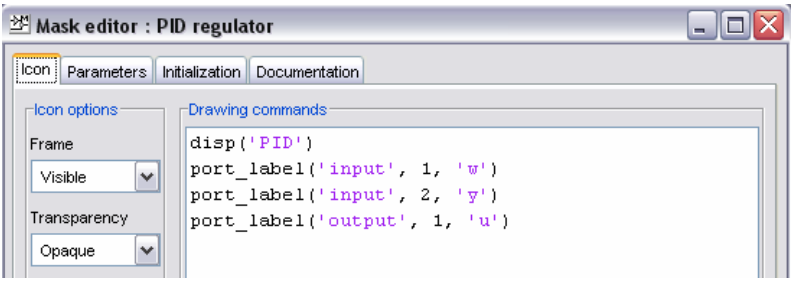
Vytvořme nyní postupně jednotlivé bloky nové knihovny PID. Nejprve sestavíme model klasického PID regulátoru. Oproti bloku *PID Controller* (viz obr. 6.18), který je standardně k dispozici v Simulinku, použijeme odlišnou implementaci PID algoritmu. Přenos nově realizovaného regulátoru v paralelním tvaru je roven $G_R(s) = K_R\,(1 + 1 / (T_i\,s) + T_d\,s)$. Je zřejmé, že není možné zadat nulovou integrační časovou konstantu T_i, jelikož by docházelo k dělení nulou. Z přenosu je také patrná interakce konstant. Vzájemný vztah mezi konstantami obou variant regulátoru je $P = K_R$, $I = K_R / T_i$ a $D = K_R\,T_d$.

Do okna knihovny vložíme blok *Subsystem* z knihovny ***Ports & Subsystems*** a po jeho otevření realizujeme podle uvedeného přenosu zapojení PID regulátoru. Blok přejmenujeme na **PID regulator**. Ve výsledném zapojení na obr. 6.65 je vyřešen i výpočet regulační odchylky e = w – y. Blok nově vytvořeného regulátoru má tedy dva vstupy a jeden výstup.

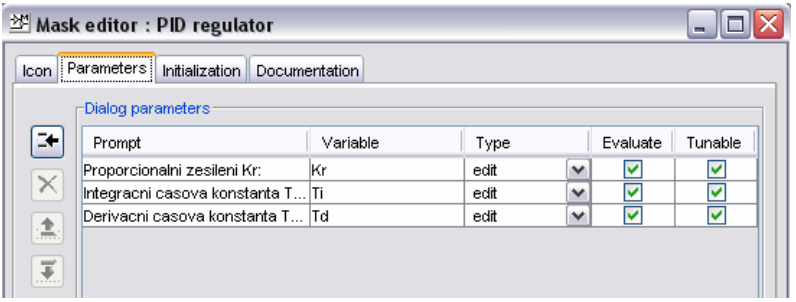


Obr. 6.65: Realizace PID regulátoru

Již dříve uvedeným způsobem subsystém zamaskujeme a definujeme některé parametry masky. Do středu ikony umístíme příkazem **disp('PID')** text PID a popíšeme i všechny porty, viz obr. 6.66. V záložce **Parameters** okna editoru (obr. 6.67) definujeme parametry, které budeme chtít zadávat.



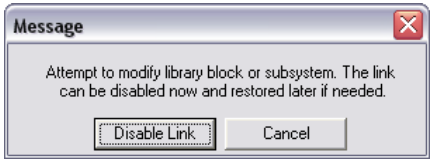
Obr. 6.66: Nastavení ikony



Obr. 6.67: Definování parametrů

Vytvořený blok nyní zkopírujeme, aby jsme jej mohli modifikovat a vytvořit tak druhou variantu PID regulátoru. V případě spojitého PID regulátoru s aproximovanou derivací bude přenos tohoto regulátoru ve tvaru $G_R(s) = K_R\,(1 + 1 / (T_i\,s) + (T_d\,s) / (T_v\,s + 1))$. Zapojení je obdobné jako u předchozí varianty, pouze je nutné změnit nastavení parametrů derivace.

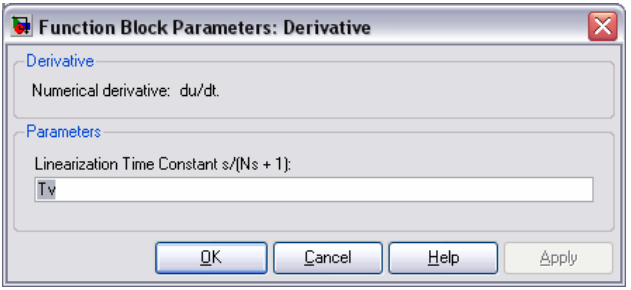
Využijeme tedy zapojení na obr. 6.65 a zkopírujeme blok **PID regulator**. Knihovnu je nejprve nutné pomocí volby **Edit → Unlock Library** odemknout. Vlastní kopírování můžeme provést pomocí myši (viz obr. 6.22). Před zkopírováním bloku (v okamžiku uvolnění pravého tlačítka myši) se ale navíc objeví plovoucí nabídka obsahující položky **Copy Here**, **Copy and Break Link** a **Cancel**. To je způsobeno propojením (linkem) s původním knihovním blokem. Zvolíme-li **Copy Here**, bude toto propojení zachováno a po otevření zkopírovaného bloku prostřednictvím **Edit → Look Under Mask** (Ctrl + U) a pokusu o jakoukoliv změnu se objeví zpráva na obr. 6.68.



Obr. 6.68: Hlášení o existujícím linku na knihovní blok

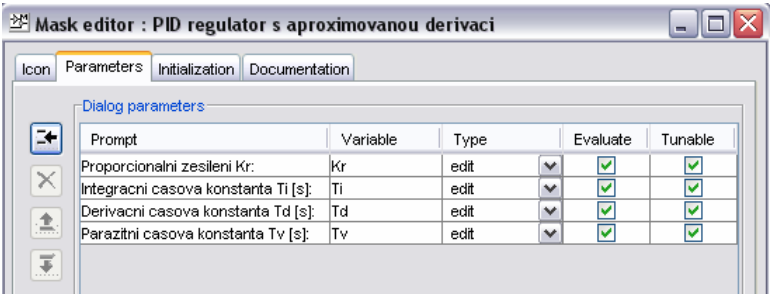
Stiskneme-li tlačítko **Disable link**, propojení bude vypnuto a změny lze již provést. Pomocí **Edit → Link Options** nabídky (nebo pomocí plovoucího menu vyvolaného myší či Ctrl + L) lze volbou **Go To Library Block** přejít ke knihovnímu bloku, volbou **Break Link** přerušit propojení nebo jej volbou **Restore Link** naopak obnovit. V našem případě, kdy chceme vytvářet

nový knihovní blok, propojení samozřejmě zrušíme. Další možností je toto propojení s původním blokem přerušit již při kopírování – zvolíme možnost **Copy and Break Link**.



Obr. 6.69: Změna parametrů bloku *Derivative*

Pro realizaci varianty regulátoru s aproximovanou derivací stačí změnit v bloku Derivative implicitní hodnotu **Inf** na proměnnou **Tv** (obr. 6.69). Dále je také třeba nový blok přejmenovat na **PID regulator s aproximovanou derivaci** a upravit jeho masku tak, aby bylo možné zadávat i parazitní časovou konstantu derivace T_v, viz obr. 6.70. Vzhled ikony můžeme ponechat.



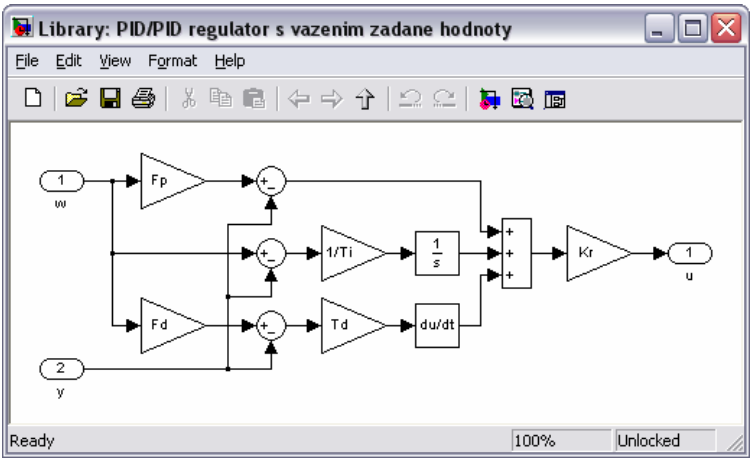
Obr. 6.70: Přidání nového parametru

Další variantou regulátoru, kterou budeme realizovat je PID s vážením žádané hodnoty. Struktura tohoto regulátoru je již poněkud složitější a umožňuje odvozovat proporcionální a derivační složku od regulované veličiny (nikoliv od regulační odchylky jako u klasického PID regulátoru). Akční veličina u(t) je vytvářena následujícím způsobem

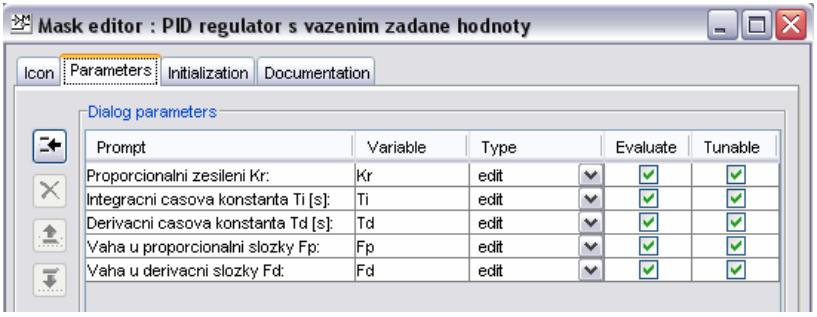
$$u(t) = K_R\left[F_p w(t) - y(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{d}{dt}(F_d w(t) - y(t))\right].$$

Je-li regulátor nastaven především s ohledem na dobré vyregulování poruchy a odezva na skokovou změnu žádané hodnoty je proto příliš kmitavá, uvedená modifikace způsobí zpomalení náběhu na žádanou hodnotu a zmenšení překmitů. Schéma zapojení modifikovaného regulátoru je na obr. 6.71.

Opět je třeba pozměnit masku nově vytvořeného bloku s názvem **PID regulator s vazenim zadane hodnoty**, viz obr. 6.72. Uveďme ještě, že běžně bývají koeficienty F_p a F_d voleny rovny buď jedné či nule, obecně lze ovšem uvažovat celý interval $\langle 0; 1\rangle$.

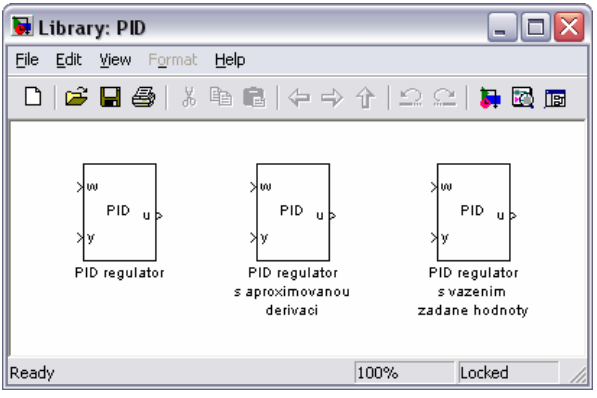


Obr. 6.71: Realizace PID regulátoru s vážením žádané hodnoty

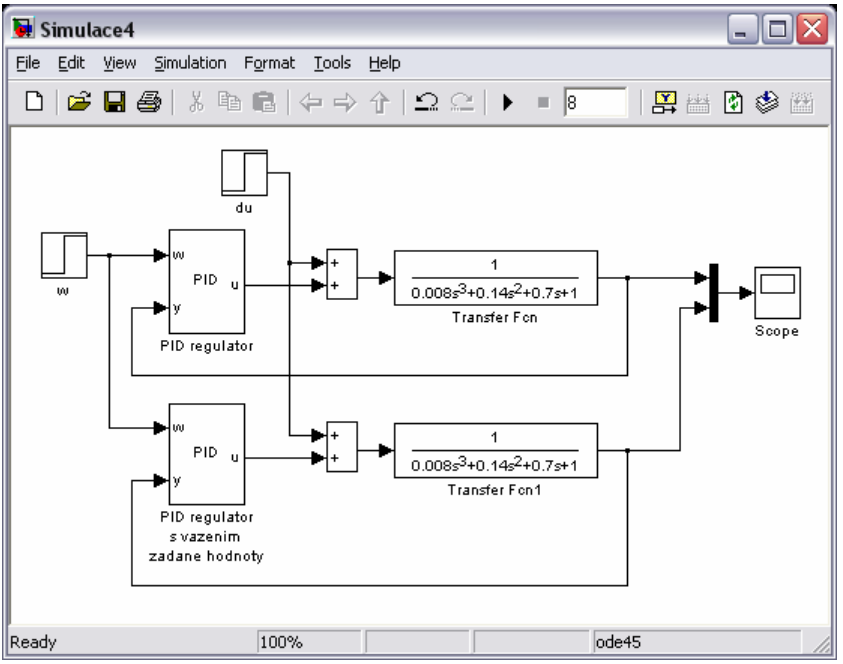


Obr. 6.72: Definice parametrů

Nová knihovna PID je na obr. 6.73 a můžeme ji již používat při vytváření nejrůznějších modelů. Vyzkoušejme tedy použití nových bloků a sestavme schéma uzavřeného regulačního obvodu.

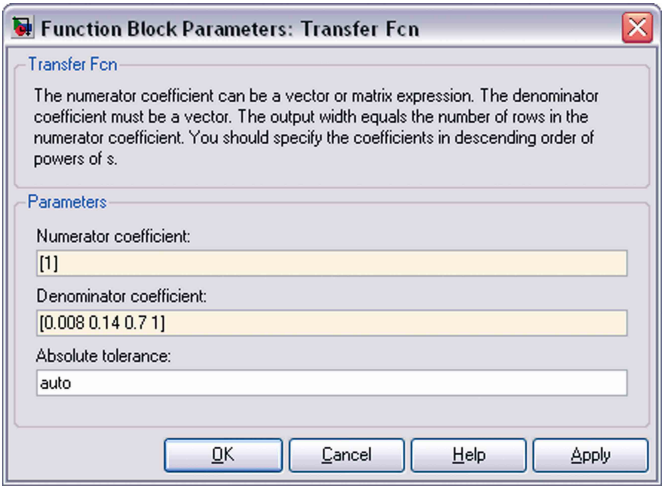


Obr. 6.73: Uživatelská knihovna PID



Obr. 6.74: Ověření dvou různých regulátorů

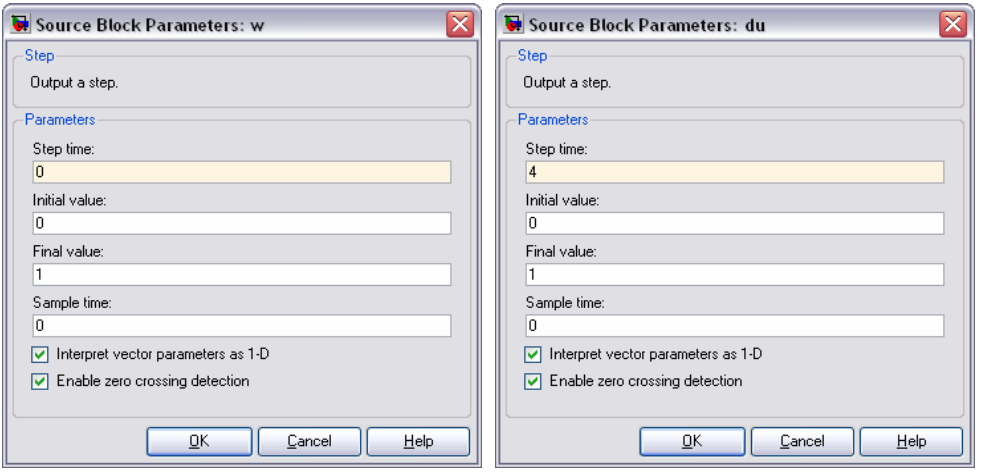
Zajímavé jistě bude porovnání klasického a modifikovaného PID regulátoru. Na základě shora probraných témat lze poměrně snadno sestavit simulační schéma. Přenos regulované soustavy může např. být $G_S(s) = 1 / (0{,}008\ s^3 + 0{,}14\ s^2 + 0{,}7\ s + 1)$. K definování přenosu v Simulinku slouží blok *Transfer Fcn* z knihovny ***Continuous***. Do okna nového modelu (uložíme jej pod názvem *Simulace4*) postupně vložíme všechny potřebné bloky, včetně bloků regulátorů z naší nové knihovny PID, viz obr. 6.74.



Obr. 6.75: Zadání koeficientů obrazového přenosu

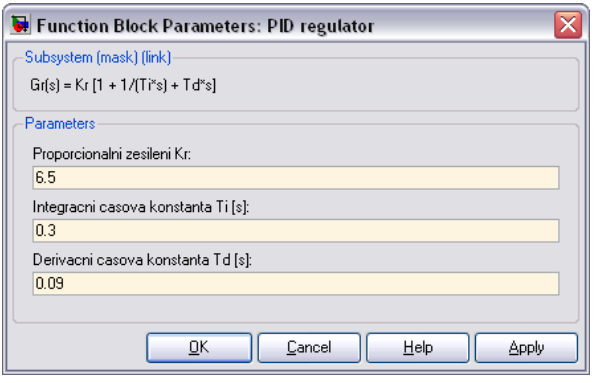
Zadání koeficientů obrazového přenosu provedeme prostřednictvím okna parametrů bloku *Transfer Fcn*. Jejich zadání odpovídá zápisu v příkazovém řádku MATLABu. Důležité je dodržení zápisu vektorů čitatele v položce **Numerator coefficient** a jmenovatele v položce **Denominator coefficient**. Koeficienty zadáváme vždy sestupně od koeficientu u nejvyšší mocniny operátoru **s**, viz obr. 6.75. Možné je samozřejmě i použití proměnných. Ikona bloku se po zadání změní a obsahuje přímo definovaný obrazový přenos.

Testovací skokové signály definujeme pomocí bloků *Step* z knihovny ***Sources***. Blok pro zadávání žádané hodnoty w(t) přejmenujeme na **w** a v okně parametrů bloku změníme čas skoku **Step time** z původní hodnoty 1 s na 0. Blok pro zadávání poruchy na akční veličině $d_u(t)$ obdobně také přejmenujeme na **du**, čas skoku změníme na 4 s a ostatní parametry ponecháme, viz obr. 6.76. V parametrech simulace nastavíme čas simulace na 8 s.



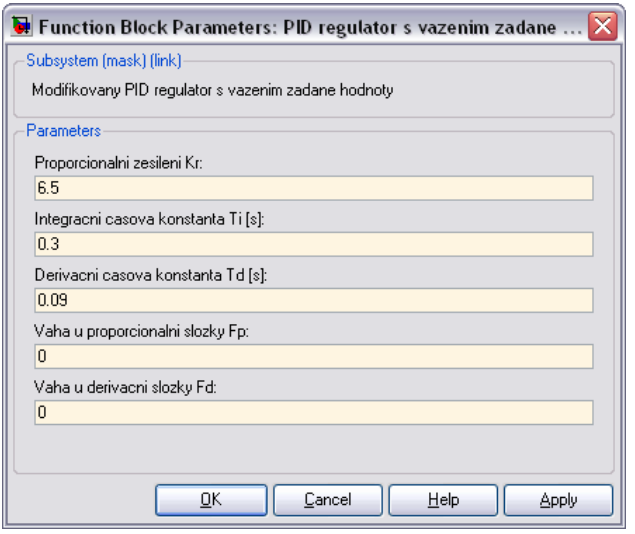
Obr. 6.76: Definování řídicí veličiny a poruchy

Dále je nutné definovat také vlastní hodnoty konstant obou variant PID regulátoru. Můžeme na základě průběhů odezvy uzavřeného regulačního obvodu provést např. ruční seřízení regulátoru nebo využít některou z mnoha metod. V našem případě byly konstanty regulátoru navrženy na základě metody kritického zesílení Zieglera-Nicholse.



Obr. 6.77: Koeficienty klasického PID

Pravidla pro nalezení vhodných konstant jsou u metody Zieglera-Nicholse optimalizována z hlediska dobrého potlačení poruch a pro sledování skokových změn žádané hodnoty jsou nevhodná a prakticky nepoužitelná. Do obou regulátorů nastavíme shodné hodnoty konstant – $K_R = 6{,}5$, $T_i = 0{,}3$ s a $T_d = 0{,}09$ s. V regulátoru s vážením žádané hodnoty zvolíme váhy u proporcionální a derivační složky $F_p = F_d = 0$ a tím zajistíme jejich odvození od regulované veličiny y(t) (nikoliv od regulační odchylky e(t) jako u integrační složky).

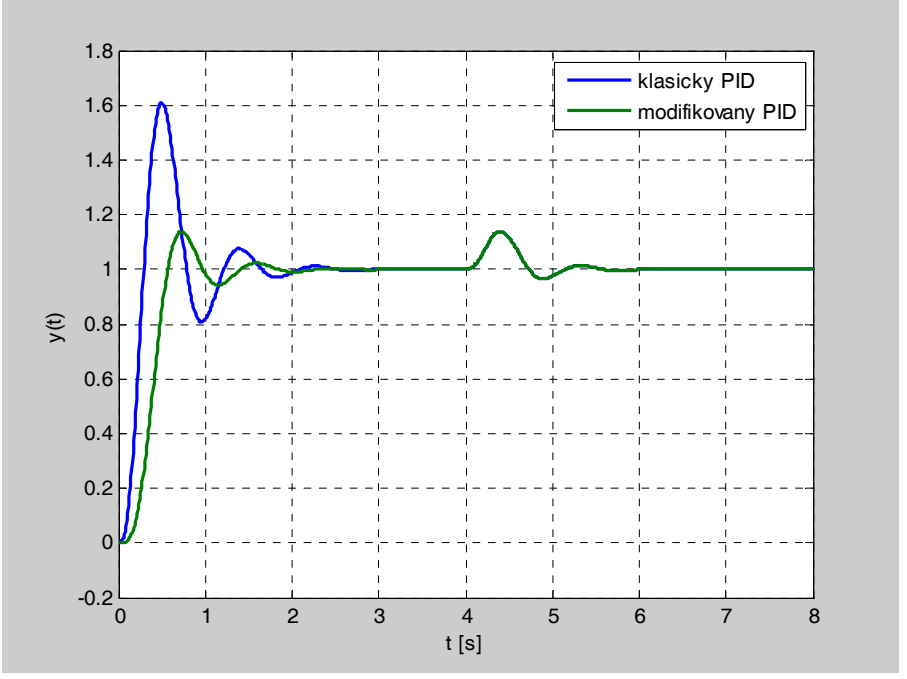


Obr. 5.78: Koeficienty PID s vážením žádané hodnoty

V okně parametrů bloku *Scope* zajistíme již dříve popisovaným způsobem (obr. 6.42) uložení vektoru řešení do prostoru proměnných. Jelikož nás zajímá vzájemné porovnání obou regulačních pochodů, tedy porovnání chování klasického PID s regulátorem s vážením žádané hodnoty, sloučíme obě regulované veličiny pomocí bloku *Mux* (knihovna ***Signal Routing***) do jednoho signálu a tento přivedeme na vstup zobrazovače. K podrobnějšímu použití bloku *Mux* se ještě vrátíme v následující kapitole. Výsledky zobrazíme pomocí standardního příkazu MATLABu **plot**. Jednoduchým způsobem můžeme také popsat osy a pro přehlednost zobrazit legendu, viz obr. 6.79.

Zobrazení obou odezev uzavřeného obvodu v jednom grafu je velmi efektivní. Jak vidíme, tak shora uvedená hypotéza se potvrdila, neboť modifikovaný PID regulátor podstatně lépe vyreguluje změnu žádané hodnoty při zachování dobrého potlačení poruchy. Výsledky lze samozřejmě zobrazit i v bloku *Scope*, ten je ale spíše vhodný k rychlému zobrazení řešení a jeho výstup nelze prakticky v přijatelné formě přenést např. do textového editoru.

```
Command Window
>> plot(v(:,1),[v(:,2) v(:,3)],'LineWidth',2), grid on
>> xlabel('t [s]'), ylabel('y(t)')
>> legend('klasicky PID','modifikovany PID')
>>
```



Obr. 6.79: Zobrazení výsledků simulace

6.5 Práce se signály ve složitých modelech

V této části textu bude podrobněji probírána práce se signály. Konkrétně např. sdružování několika signálů do jednoho a naopak pomocí bloků *Mux* a *Demux* a přenos signálů v rámci modelu bez nutnosti fyzického propojení prostřednictvím bloků *From* a *Goto*. Uvedené bloky jsou součástí knihovny ***Signal Routing***. Různé varianty řešení budou demonstrovány na složitějším modelovém příkladě simulace pohybu elektronu v elektromagnetickém poli. Jedná se o pohyb ve třech souřadných osách, sestavení simulačního schéma bude s ohledem na tuto skutečnost značně složitější. V dalším textu budou vektory značeny tučně.

Úkolem je zobrazit pohyb elektronu, který s určitou počáteční rychlostí $\mathbf{v_0}$ vletí do elektromagnetického pole s konstantní magnetickou indukcí **B**. Vzhledem k značně vysokým silám v tomto poli a nízké hmotnosti elektronu je možné zanedbat působící gravitační sílu. Zadejme nyní konkrétní hodnoty při kterých bude pohyb elektronu simulován. Elektron nese elementární náboj $q = -1{,}6 \cdot 10^{-19}$ C a má hmotnost $m = 9 \cdot 10^{-31}$ kg. Jeho počáteční rychlost je

rovna $\mathbf{v_0} = [v_{0x}\ \ v_{0y}\ \ v_{0z}]^T = [0\ \ 7\ \ 1]^T \cdot 10^7$ m /s. Indukce $\mathbf{B} = [B_x\ \ B_y\ \ B_z]^T = [0\ \ 0\ \ 10^{-2}]^T$ T a intenzita elektrického pole je nulová, tedy $\mathbf{E} = [E_x\ E_y\ E_z]^T = [0\ \ 0\ \ 0]^T$ V/m. Na nabitou částici v elektromagnetickém poli působí Lorentzova síla o velikosti $\mathbf{F} = q\,(\mathbf{E} + \mathbf{v} \times \mathbf{B})$ a výsledná pohybová rovnice tedy je $m\,\mathbf{a} = q\,(\mathbf{E} + \mathbf{v} \times \mathbf{B})$.

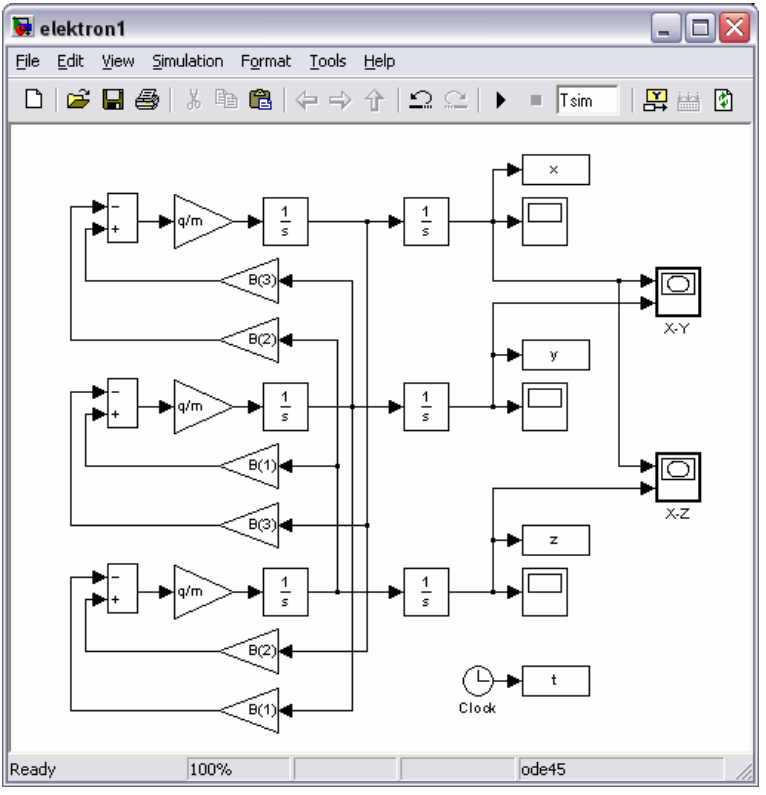
Jelikož se elektron pohybuje v prostoru, je nutné rozepsat uvedenou pohybovou rovnici na jednotlivé složky. Vzniknou tak tři rovnice, které jsou navzájem provázané. Zavedeme-li současně značení $a_x = \ddot{x}$, $a_y = \ddot{y}$, $a_z = \ddot{z}$, $v_x = \dot{x}$, $v_y = \dot{y}$, $v_z = \dot{z}$ a vyjádříme-li vektorový součin po jednotlivých složkách, lze psát

$$\ddot{x} = \frac{q}{m}(E_x + \dot{y}\,B_z - \dot{z}\,B_y)\,,$$

$$\ddot{y} = \frac{q}{m}(E_y + \dot{z}\,B_x - \dot{x}\,B_z)\,,$$

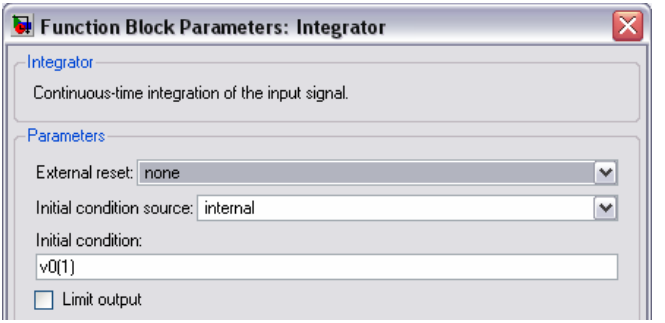
$$\ddot{z} = \frac{q}{m}(E_z + \dot{x}\,B_y - \dot{y}\,B_z)\,.$$

Podle uvedených rovnic sestavíme simulační schéma na obr. 6.80. Model je vybuzen počáteční rychlostí $\mathbf{v_0}$, jejíž jednotlivé složky je nutné zadat jako počáteční podmínky do integrátorů.



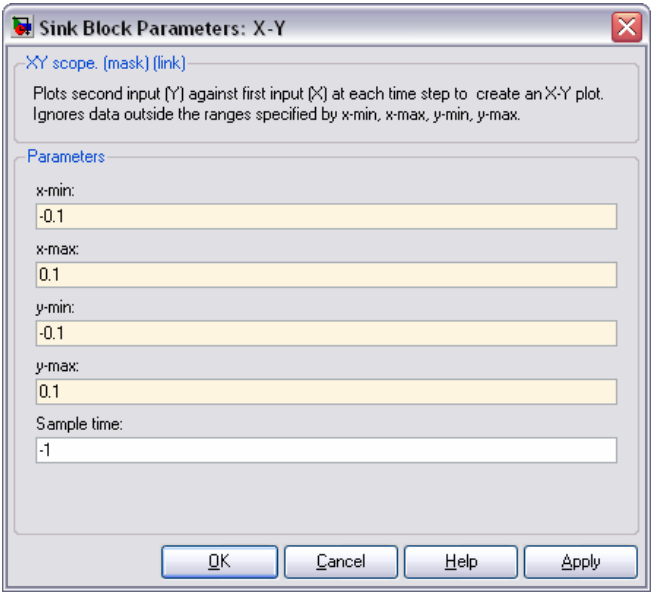
Obr. 6.80: Model pohybu elektronu v elektromagnetickém poli

Např. v části schématu realizující první rovnici (je zcela nahoře) zadáme do levého integrátoru jako počáteční podmínku složku rychlosti v_{0x}, tedy první prvek vektoru $\mathbf{v_0}$, viz obr. 6.81. Ke sledování trajektorie pohybu v jednotlivých souřadnicích použijeme bloky *Scope*. Složky pohybu x, y a z ale také pomocí bloků *To Workspace* (knihovna ***Sinks***) uložíme do prostoru proměnných. Jako formát dat nastavíme v parametrech bloků formát **Array**.



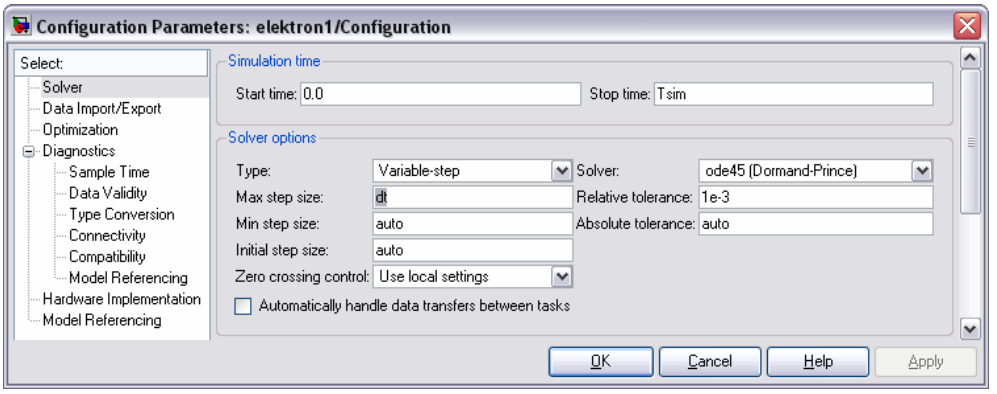
Obr. 6.81: Zadání počáteční podmínky v integrátoru

Můžeme také použít blok *XY Graph* (knihovna ***Sinks***), který je schopen zobrazit průběh dvou signálů v rovině. Zobrazíme průběh trajektorie v rovině X-Y a v rovině X-Z. Nejdříve je ale třeba nastavit měřítka na obou osách. Na obr. 6.82 je uvedeno nastavení měřítek bloku pro zobrazování roviny X-Y.



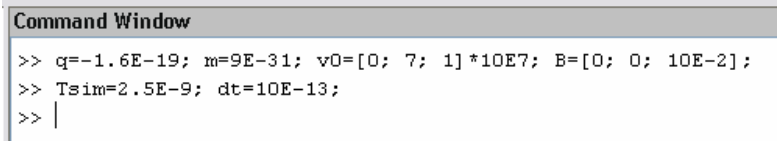
Obr. 6.82: Volba měřítek XY zobrazovače

Délku simulace a krok výpočtu zadáme v okně parametrů simulace (obr. 6.83) pomocí proměnných. Délku simulace **Stop time** pomocí proměnné Tsim a hodnotu **Max step size** pomocí proměnné dt. Ostatní parametry ponecháme beze změn.



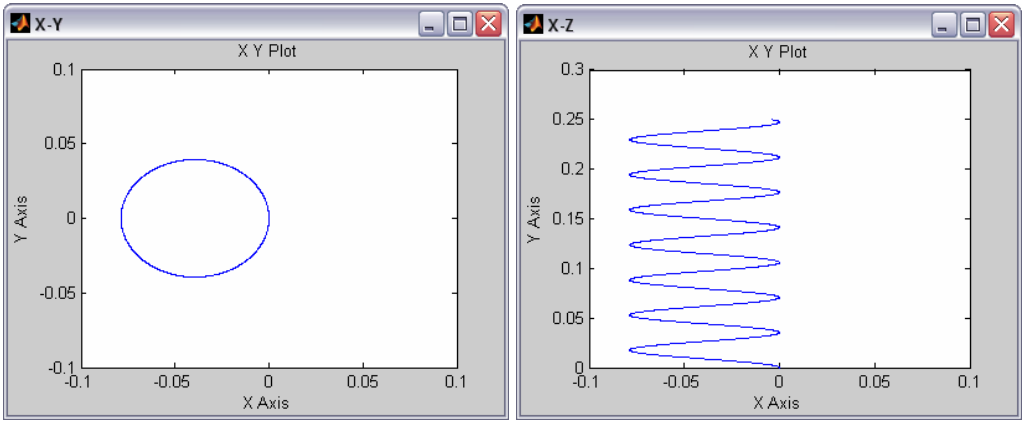
Obr. 5.83: Nastavení parametrů simulace

Před spuštěním simulace je ještě nutné definovat její délku a krok výpočtu. Jelikož se jedná o velmi rychlý děj, zvolíme čas simulace $T_{sim} = 2{,}5 \cdot 10^{-9}$ s. Zvolíme také dostatečně jemný krok výpočtu dt = 10^{-13} s tak, aby výpočet byl dostatečně přesný. Na obr. 6.84 je definice všech potřebných veličin v příkazovém řádku MATLABu.



Obr. 6.84: Definice proměnných

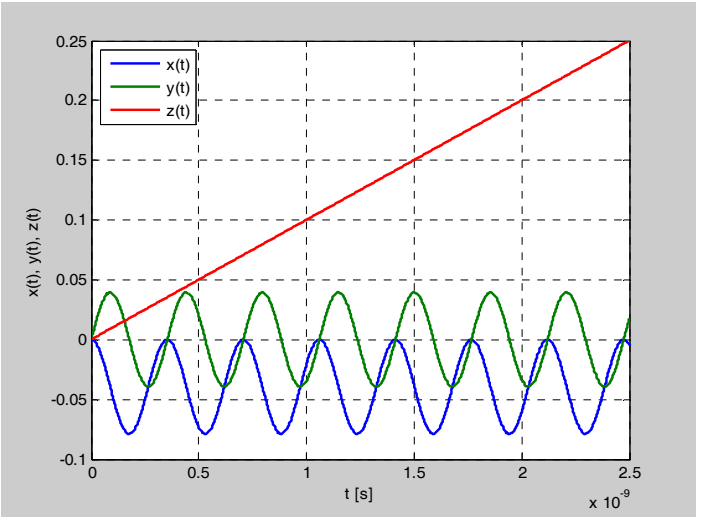
Ihned po spuštění simulace dojde k otevření oken XY zobrazovačů (obr. 6.85) a v průběhu simulace můžeme sledovat vývoj trajektorie v rovinách X-Y a Y-Z.



Obr. 6.85: Zobrazení trajektorie v rovinách X-Y a Y-Z

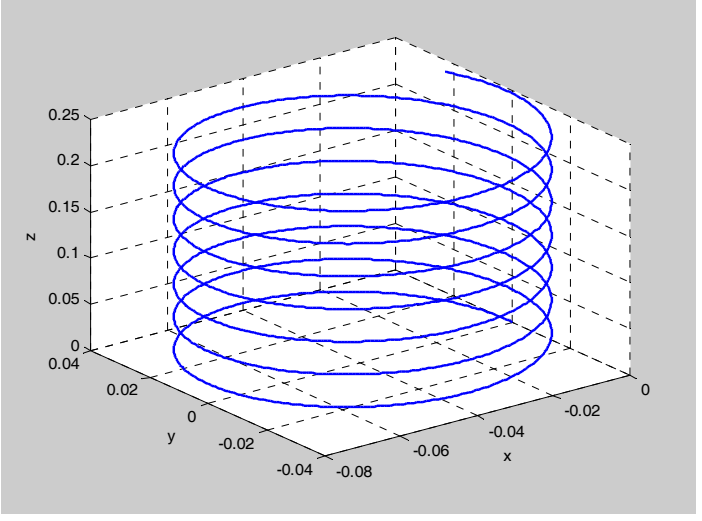
Jelikož jednotlivé složky pohybu a čas (je generován blokem *Clock*) v průběhu simulace ukládáme prostřednictvím bloků *To Workspace* do prostoru proměnných, můžeme je vykreslit pomocí příkazu **plot**. Výslednou trajektorii ve 3D souřadnicích pak pomocí příkazu **plot3**.

```
Command Window
>> plot(t,[x y z],'LineWidth',2), grid on
>> xlabel('t [s]'), ylabel('x(t), y(t), z(t)')
>> legend('x(t)','y(t)','z(t)',2)
>>
```



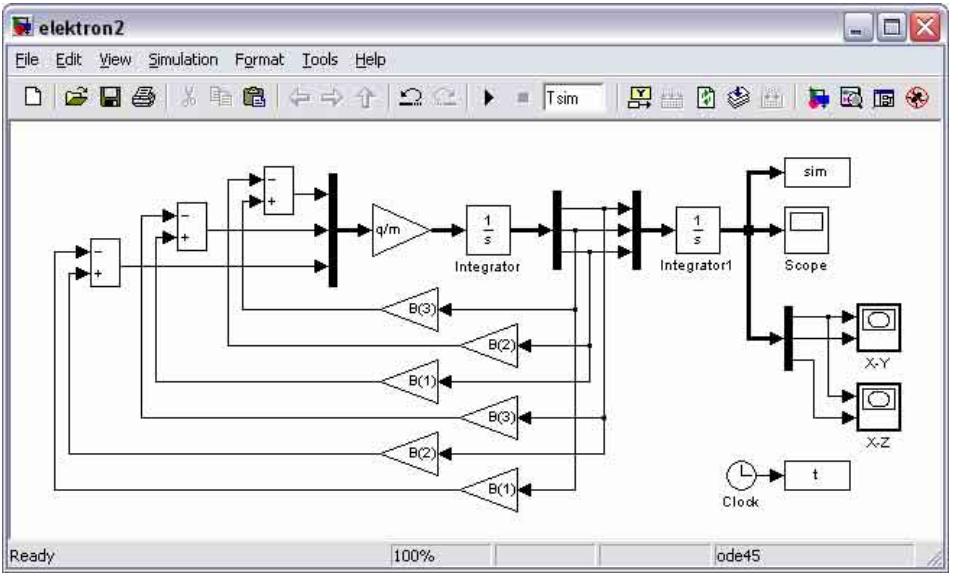
Obr. 6.86: Vykreslení jednotlivých složek pohybu

```
Command Window
>> plot3(x,y,z,'LineWidth',2), grid on
>> xlabel('x'), ylabel('y'), zlabel('z')
>>
```



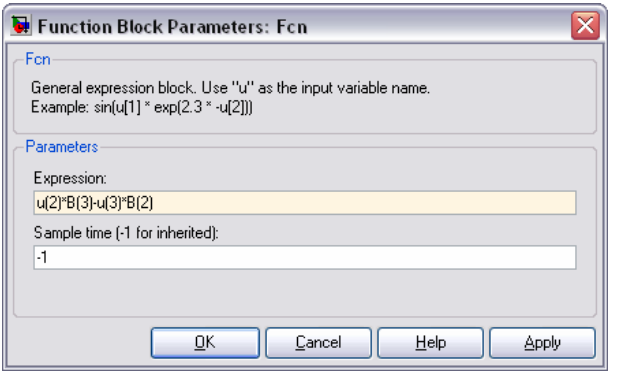
Obr. 6.87: Trajektorie pohybu elektronu

Pozorný čtenář si jistě všiml, že ve schématu na obr. 6.80 jsou některé bloky obsaženy vícekrát. Pomocí bloků *Mux* (multiplexor) a *Demux* (demultiplexor) lze signály jednoduše, tam kde je to třeba, sloučit resp. opět rozdělit. U obou těchto bloků můžeme podle potřeby měnit počet jejich vstupů resp. výstupů. Upravíme-li takto původní schéma, zredukuje se počet integrátorů z původních šesti na pouhé dva. Buzení systému počátečními podmínkami zajistíme zadáním jejich vektoru (počáteční rychlost $\mathbf{v_0}$) do prvního integrátoru ve směru toku sdruženého signálu.



Obr. 6.88: Upravený model pohybu elektronu

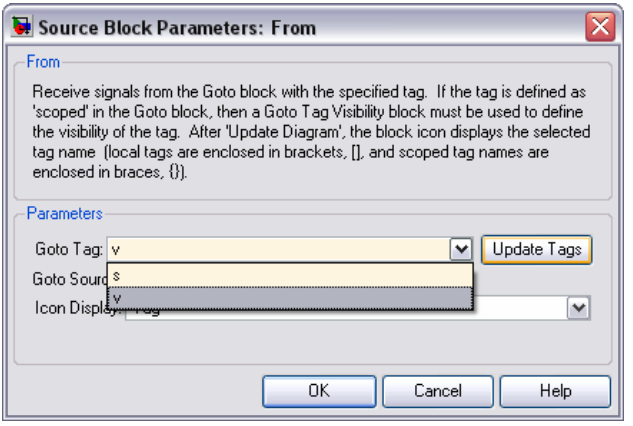
Sdružené (vektorové či neskalární) spoje můžeme volbou z menu okna modelu **Format → Port/Signal Displays → Wide Nonscalar Lines** zvýraznit. Ačkoli je výsledné simulační schéma kompaktnější, lze jej přesto ještě zjednodušit. Velmi výhodné je použití bloku *Fcn* z knihovny ***User-Defined Functions***. Pokud je na vstupu tohoto bloku vektorový signál, je možné se na jeho jednotlivé složky odkazovat pomocí kulatých závorek.



Obr. 6.89: Zápis výrazu v bloku *Fcn*

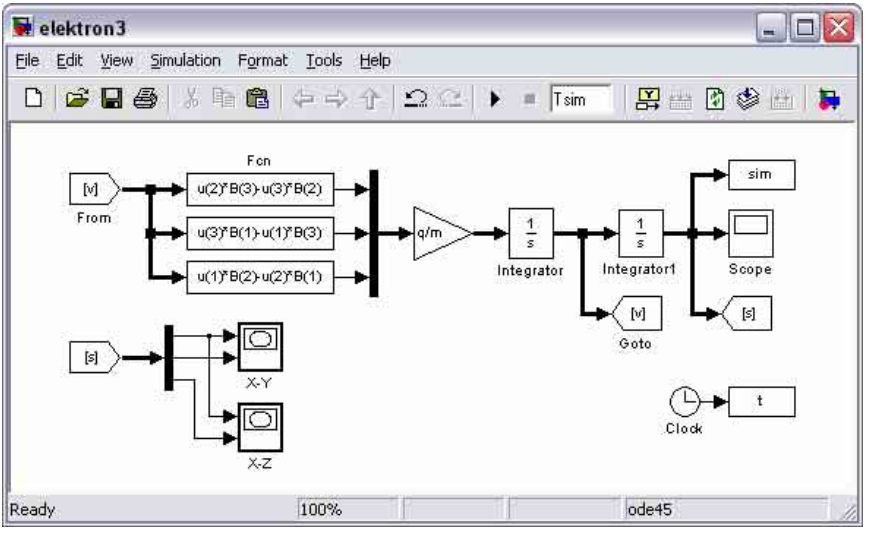
Procházející signál je označen **u**. Chceme-li se odkázat např. na druhou složku vektorového signálu, zapíšeme ji ve výrazu **Expression** jako **u(2)**, viz obr. 6.89. Tímto způsobem je možné ve schématu na obr. 6.88 nahradit skupinu zesilovačů a součtových členů.

Další možnosti poskytují i bloky *From* a *Goto*. Vektor okamžité rychlosti elektronu (za prvním integrátorem) propojíme s blokem *Goto*. V okně parametrů bloku zadáme název signálu v položce **Goto Tag** – v našem případě **v**. Umístěním bloku **From** kdekoliv ve schématu máme tento signál k dispozici, aniž by muselo dojít k fyzickému propojení. V okně parametrů bloku **From** nejprve klikneme na tlačítko **Update Tags** a z roletového menu položky **Goto Tag** zvolíme požadovaný signál, viz obr. 6.90. Výstup bloku můžeme následně propojit s dalšími bloky ve schématu.



Obr. 6.90: Volba signálu v bloku *From*

Upravený model pohybu elektronu v elektromagnetickém poli je uveden na obr. 6.91.

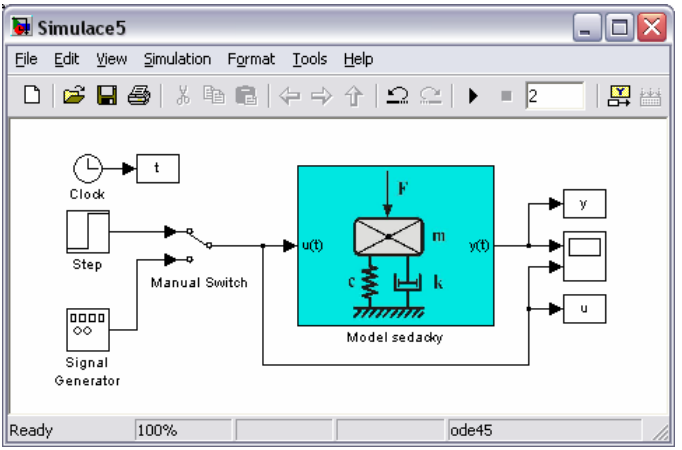


Obr. 6.91: Výsledný zjednodušený model pohybu elektronu

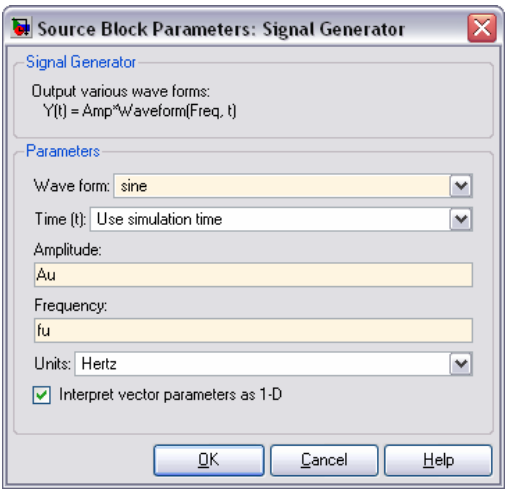
Ve výše uvedeném textu byly téměř výhradně používány pro buzení modelů bloky *Step*, představující jednotkový (tzv. Heavisideův) skok. Simulink ale poskytuje značné množství nejrůznějších generátorů funkcí, které jsou součástí knihovny ***Sources***, viz kapitola 6.1.

Vraťme se nyní k již dříve vytvořenému modelu sedačky (obr. 6.61). Model budeme budit i nadále jednotkovým skokem, do schématu ale navíc doplníme blok *Signal Generator*. Volbu příslušného budicího signálu provedeme pomocí bloku *Manual Switch*, který je součástí knihovny ***Signal Routing***. Nastavení parametrů provedeme pomocí proměnných, které budeme definovat ve skriptu. Pomocí tohoto skriptu spustíme i vlastní simulaci.

Upravený model sedačky je na obr. 6.92. V bloku *Step* zadáme do **Step Time** proměnnou ts a do položky **Final Value** proměnnou us. V bloku *Signal Generator* ponecháme implicitně nastavenou funkci sinus a pouze zadáme velikost amplitudy a frekvenci v položce **Amplitude** resp. **Frequency** – pomocí proměnných Au a fu, viz obr. 6.93.

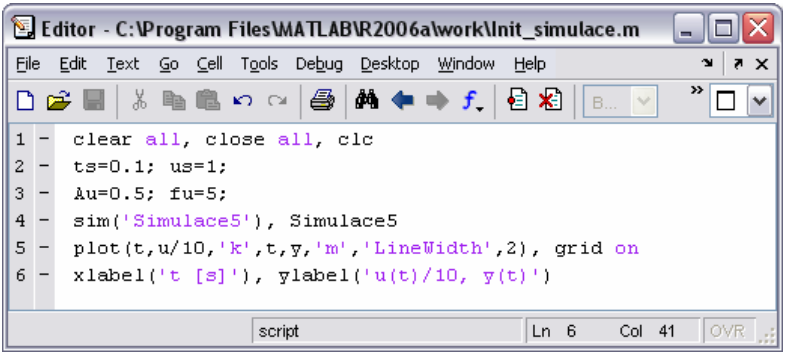


Obr. 6.92: Model s různými variantami generace budicího signálu

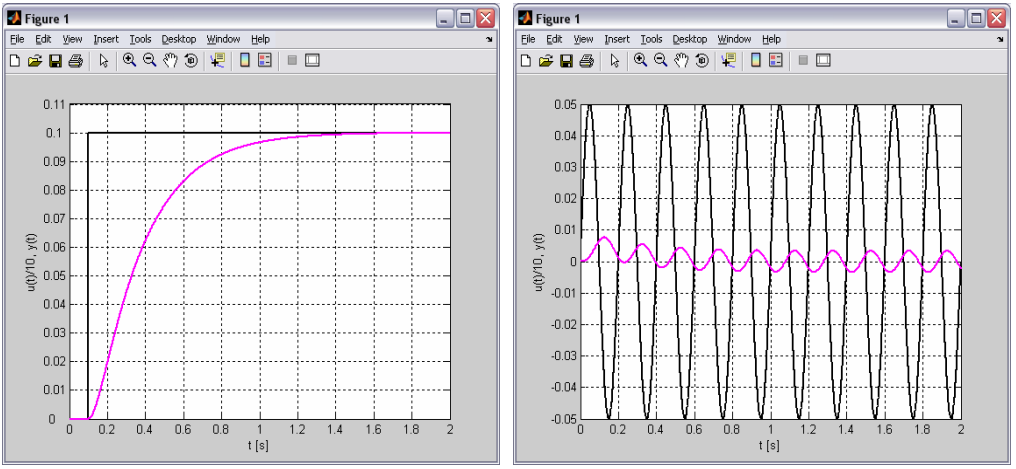


Obr. 6.93: Nastavení parametrů bloku *Signal Generator*

Hodnoty všech výše uvedených proměnných zadáme ve skriptu (obr. 6.94). Simulaci lze následně spustit pomocí příkazu **sim**. Vhodné je také otevření okna modelu zapsáním jména souboru (příponu není nutné uvádět), v našem případě *Simulace5*. Pokud je okno již otevřeno, dojde k jeho vysunutí do popředí. Jelikož v modelu na obr. 6.92 ukládáme veličiny t, u(t) a y(t), můžeme i jednoduchým způsobem zobrazit příslušné průběhy, viz obr. 6.95.



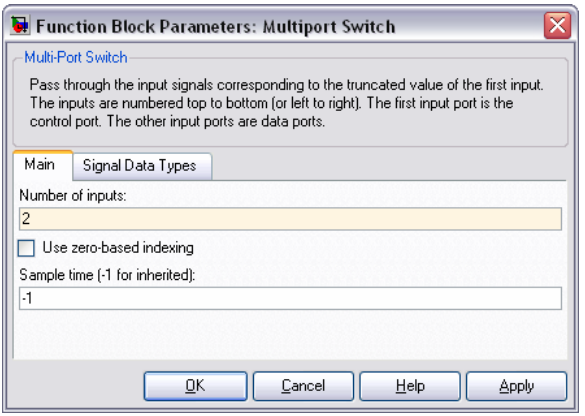
Obr. 6.94: Definice proměnných, spuštění simulace a zobrazení výsledků pomocí skriptu



Obr. 6.95: Grafické zobrazení výsledků simulace při různých typech buzení

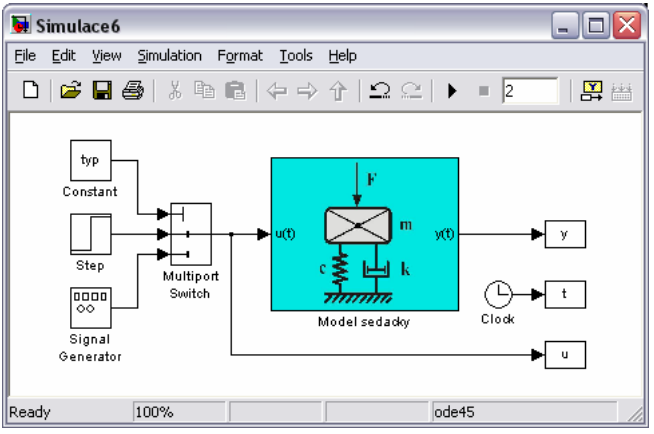
Pokud chceme změnit typ budící funkce, musíme pomocí myši přepnout přepínač (blok *Manual Switch*). Výhodnější je ale použití bloku *Multiport Switch*, který umožňuje přepnutí pomocí daného parametru. Parametr zadáme do bloku *Multiport Switch* prostřednictvím bloku *Constant*, v jehož okně parametrů zapíšeme v položce **Constant Value** proměnnou s názvem typ.

Požadovaný signál budeme volit pomocí proměnné typ ve skriptu. Pokud chceme systém budit jednotkovým skokem, tak proměnné přiřadíme hodnotu 1 (bude zvolen první vstupující signál do bloku). Chceme-li systém budit harmonickým signálem, přiřadíme proměnné typ hodnotu 2. V okně parametrů bloku *Multiport Switch* (obr. 6.96) lze případně měnit i počet jeho vstupů. Implicitně má blok vstupy tři.

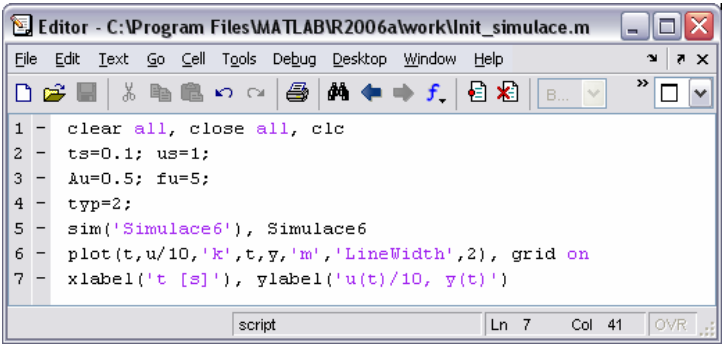


Obr. 6.96: Okno parametrů bloku *Multiport Switch*

Modifikované simulační schéma je na obr. 6.97. Model byl uložen pod názvem *Simulace6*, je tedy nutné změnit jeho název i ve skriptu. Skript ještě také doplníme o proměnnou typ; ostatní části můžeme ponechat, viz obr. 6.98.



Obr. 6.97: Upravený model s různými variantami budicích signálů



Obr. 6.98: Modifikovaný skript

Literatura

[1] Dušek, F.: *Matlab a Simulink: řešené příklady.* 1. vyd. Pardubice: Univerzita Pardubice, 2000. ISBN 80-7194-273-1.

[2] Dušek, F.: *Úvod do používání Matlab.* Pardubice: Univerzita Pardubice, 1997.

[3] Heringová, B., Hora, P.: *MATLAB díl I.: práce s programem.* [Studijní materiál online.] Plzeň, 1995. [cit. leden 2007.] Dostupné na URL: www.cdm.cas.cz/czech/hora/vyuka/mvs/tutorial.pdf

[4] Heringová, B., Hora, P.: *MATLAB díl II.: popis funkcí.* [Studijní materiál online.] Plzeň, 1995. [cit. leden 2007.] Dostupné na URL: www.cdm.cas.cz/czech/hora/vyuka/mvs/reference.pdf

[5] The MathWorks Inc.: *Using MATLAB.* [Uživatelská příručka online.] Natick (USA), 2006. [cit. leden 2007.] Dostupné na URL: http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html

[6] Zaplatílek, K., Doňar, B.: *Matlab pro začátečníky.* 2. vyd. Praha: BEN – technická literatura, 2005. ISBN 80-7300-175-6.

[7] Humusoft spol. s r.o.: *MATLAB ver. 7.x: úvod do práce s programem*. [Soubor Adobe Acrobat.] Praha, 2006. [cit. leden 2007.]

Modelování fyzikálních a chemických procesů

COMSOL Multiphysics

COMSOL Multiphysics usnadňuje pochopení fyzikálních a chemických procesů díky názorné grafice, snadné změně vstupních parametrů, možnosti simulace dané úlohy...

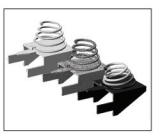
- Modelování fyzikálních a chemických procesů
- Simulace více fyzikálních dějů v jedné úloze
- Metoda konečných prvků v prostředí **MATLAB**
- Přenosy tepla, difuze, elektrostatika
- Elektromagnetismus, akustika, proudění
- Pružnost a pevnost, polovodiče, geofyzika
- Diagnostika a vibrace, aplikovaná matematika
- Otevřený, snadno ovladatelný interaktivní systém
- Rozsáhlá knihovna vzorových modelů a příkladů

Specializované profesních moduly

Chemical Engineering module
Earth Science Module
RF Module
AC/DC Module
Acoustics Module
Heat Transfer Module
MEMS Module
Structural Mechanics Module
CAD Import Modulle
COMSOL Script
COMSOL Reaction Engineering Lab

COMSOL Multiphysics je určen k modelování a simulaci fyzikálních úloh metodou konečných prvků. Základem řešení je 2D nebo 3D geometrie prostředí, součásti nebo sestavy, na kterou působí například zatížení, zdroj tepla, elektromagnetické pole, proud vzduchu nebo jiné tekutiny, chemické působení, atd. Úkolem programu je vypočítat, jak se analyzovaná součást nebo prostředí pod těmito vlivy mění a to nejenom navenek, ale i uvnitř. Názorné zobrazení výsledků zajišťuje účinná grafika a animace procesu, což přispívá k hlubšímu pochopení fyzikální podstaty dějů probíhajících všude kolem nás. **COMSOL Multiphysics může využívat své vlastní programovací prostředí nebo prostředí systému MATLAB.**




Výhradní distributor pro ČR a SR

www.humusoft.cz

HUMUSOFT s.r.o.
Pobřežní 20
186 00 Praha 8
Česká republika

info@humusoft.cz
tel.: +420 284 011 730
fax: +420 284 011 740

Vydavatel: SOŠ a SOU, Lanškroun
Kollárova 445, 563 01 Lanškroun
tel.: 465 521 081, telefax: 465 521 083
e-mail: info@spslan.cz, www.spslan.cz

Podpora: Tento dokument byl vytvořen za finanční pomoci Evropské unie, Evropského sociálního fondu a Ministerstva školství, mládeže a tělovýchovy. Za obsah tohoto dokumentu je výhradně odpovědná SOŠ a SOU Lanškroun a nelze jej v žádném případě považovat za názor Evropské unie.

Autor: Ing. Libor Kupka

Náklad: 100 výtisků
Tisk: JS PRINT CZ s.r.o., Lanškroun
Vyšlo: únor 2007

ISBN 978-80-239-8871-0