Cvičení v MATLABu.

Sylabus

Ing. Jiří Blahuta

Slezská univerzita v Opavě
Filozoficko-přírodovědecká fakulta v Opavě
Ústav informatiky

Poslední aktualizace: 27.7. 2012

Anotace:
Tento sylabus, vznikl jako inovace předmětu UI/N, má za cíl ukázat čtenáři jak si osvojit základní znalosti a praktické dovednosti v programovém prostředí MATLAB, jak vytvářet základní syntaxe, příkazy, skripty, funkce a jak pracovat v GUI prostředí. Rovněž dává možnost seznámit se s logikou na praktických příkladech zpracování dat z různých oblastí vědy. Je zmíněn také pohled na další rozšíření v podobě Toolboxes, možnosti kompilace programů a práce s externími daty a vytvoření spustitelné .exe aplikace ve standardním prostředí MS Windows.
Pozn: kapitoly, ve kterých je uvedeno něco, co není přímou součástí MATLABu, jsou vyznačeny hvězdičkou „*“.

Cvičení v MATLABu.

Sylabus

Ing. Jiří Blahuta

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě
Bezručovo nám. 13, Opava

Tato inovace předmětu *UI/N* je spolufinancována Evropským sociálním fondem a Státním rozpočtem ČR, projekt č. CZ.1.07/2.2.00/28.0014, „Interdisciplinární vzdělávání v ICT s jazykovou kompetencí“.

Obsah

1 Co je MATLAB

Systém MATLAB, který vyvíjí již více než 15 let společnost MathWorks, Inc., je patrně nejznámějším matematickým nástrojem v profesionální sféře. MATLAB se dá charakterizovat jako výkonný matematický programovací jazyk, zároveň jako vynikají vývojové prostředí. Jaké jsou hlavní rysy tohoto systému?

- výkonný matematický nástroj se stovkami integrovanými funkcemi
- možnost tvorby skriptů a programů
- vytváření GUI pro aplikace
- modularity díky Toolboxes – otevřená architektura
- propojení s jinými matematickými nástroji (např. Maple)
- vykreslování grafů a dalších objektů, vizualizace dat

MATLAB se řadí mezi vývojová prostředí se svým programovacím jazykem a spolupracuje s externími kompilery jazyků C a C++. Primárně je to vynikající matematický nástroj, ale díky přidaným modulům má právě širokou škálovatelnost pro širokou oblast akademických a vědeckých projektů. Toolboxes, rozšiřující moduly, jsou hlavní předností. Díky těmto modulům tak dostaneme stovky nových funkcí, mezi které patří:

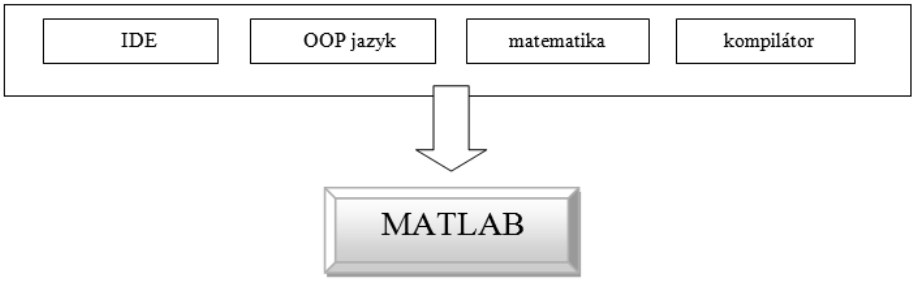
- zpracování obrazu a videa
- symbolický integrální a diferenciální počet
- tvorba zkompilovaných EXE aplikací
- modelování umělých neuronových sítí
- finanční a statistická matematika
- a stovky dalších

MATLAB se dá chápat dvojím způsobem – jako jazyk a jako IDE. Toto chápání jej spojuje do vysoce efektivního matematického nástroje a mnohem více díky volitelným Toolbox modulům. Tyto moduly[1] jsou v češtině popsány na webu společnosti Humusoft, jež je oficiálním distributorem MATLABu v ČR. MATLAB nabízí přehledné graficky orientované prostředí s dvojím režimem práce:

- režim k přímým výpočtům (Command Window)
- M-file editor – tvorba M-skriptů, které lze uložit a kdykoli spustit

Obecně můžeme na MATLAB nahlížet dle schématu:

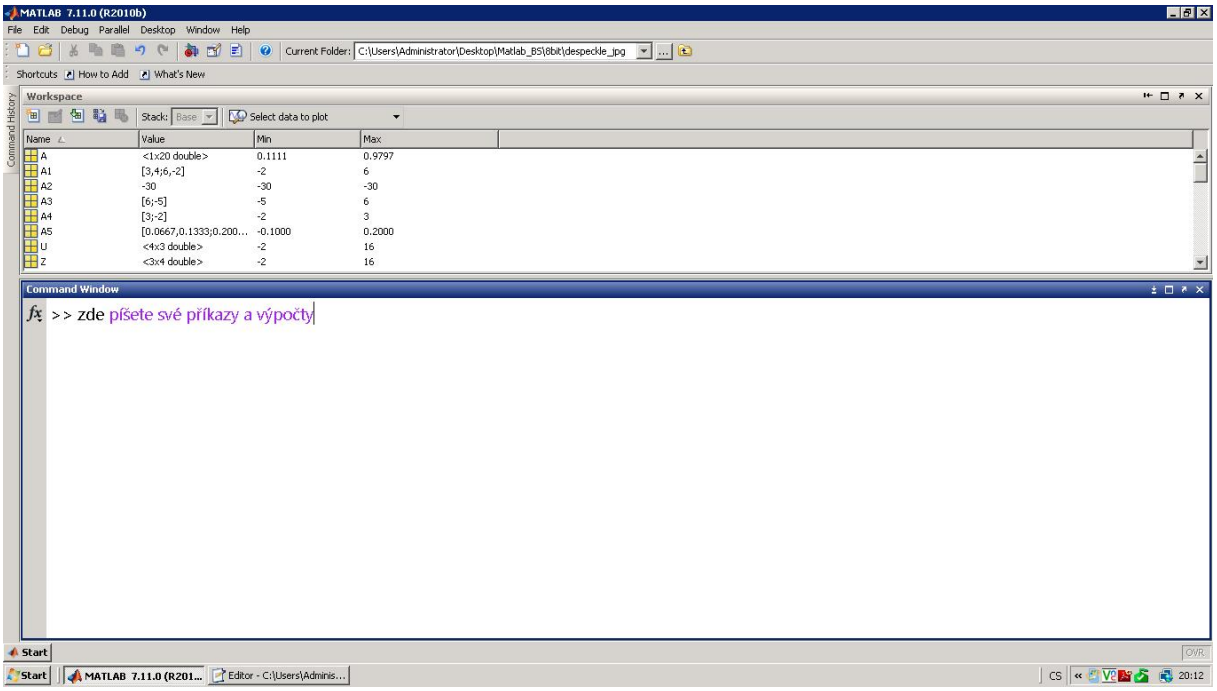
[1] http://www.humusoft.cz/produkty/matlab/aknihovny/



Obrázek 1.2: Pohled na MATLAB jako celek

MATLAB existuje ve verzích pro Windows a ve verzích pro Unixové platformy.

Po spuštění MATLABu uvidíme prázdné okno v něm systémový prompt >> a MATLAB tak očekává vaše vstupy. Vidíme také podokno Workspace s načtenými proměnnými (použité v dalším textu).



Obrázek 1.3: Základní prostředí s příkazovým řádkem a Workspace podoknem

Shrnutí: MATLAB je modulární systém. Jako základ je chápán jako vynikající matematický software s možností programování aplikací pomocí M skriptů. Přidané moduly je nutno zakoupit zvlášť a po instalaci se integrují do MATLABu.

1.1 Další prvky prostředí

MATLAB umožní zobrazení dalších dílčích podoken, mezi důležité patří:

- *Command History* – **historie napsaných a vyhodnocených příkazů v příkazovém řádku**
- *Workspace* – **správa deklarovaných a uložených proměnných, viz. kap. 3.2.1**

- *Current Directory* – **obsah aktuální pracovního adresáře – má význam, pokud manipulujeme s externími soubory a ukládáme např. Figure okna či jiné výstupy, viz. kap. 8**

Zastavme se krátce u Current Directory. Zajímavostí je, že pro práci s tímto adresářem fungují vesměs stejné příkazy, jako v běžném Příkazovém řádku DOS, tedy *cd* pro změnu adresáře, *mkdir* pro vytvoření nového, *rmdir* pro přejmenování, *delete* smazání, *dir* pro výpis obsahu, atd. Tyto příkazy samozřejmě píšeme běžně do Command Window v MATLABu.

Shrnutí: Základním prostředím MATLABu je příkazový řádek, tedy hlavní okno, ve kterém píšeme příkazy a funkce. Rozložení je modifikovatelné a může zobrazovat několik podoken. Další možností je použití tzv. Editoru ke psaní skriptů.

1.2 Kde rychle získat nápovědu

Hlavní nápovědu získáme příkazem Help z hlavní nabídky. Tato nápověda je snad nejpodrobnější příručkou pro celý MATLAB, která existuje. Je také plně k dispozici na webu produktu.

Někdy však potřebujeme nápovědu ke konkrétnímu příkazu či funkci. Tu vyvoláme jednoduchým napsáním příkazu *help xxx*, kde *xxx* je název funkce. Například rychlou nápovědu pro použití funkce *corr* (výpočet korelačního koeficientu vektorů) získáme zapsáním *help corr*. Obdržíme popis funkce, její parametry a syntaxi s příkladem.

Další možností nápovědy je použití demo videí, které ukazují v praktických úlohách použití MATLABu. Je doprovázeno i výpisem celého postupu. Příkaz *demo* odkáže na stránku nápovědy s těmito ukázkami.

Další zdroje informací jsou uvedeny ve webových referencích, z literatury pro začátečníky doporučuji knihu (1).

Shrnutí: Nejrozsáhlejší nápověda k MATLABu je samotný Help, který obsahuje podrobný výklad ke každé funkci včetně příkladů. Nápovědu ke konkrétní funkci můžeme rychle vyvolat napsáním help xxx.

1.3 Typy asociovaných souborů

MATLAB využívá některé vlastní přípony souborů, se kterými se při práci můžeme setkat. Jsou to zejména:

- MAT – soubory vytvořené v základním příkazovém řádku
- M – programové skripty vytvořené v editoru (viz kapitola 6)
- MEX – zkompilované moduly napsané v jiném programovacím jazyce, např. C
- FIG – soubory oken Figure pro GUI a vykreslování grafických objektů (viz kapitola 5)

To jsou nejčastější přípony, se kterými se setkáme. Moduly (Toolbox) jsou vlastně knihovny s M a MEX soubory, které obsahují také kompletní dokumentaci. Funkce jsou M soubory uložené

v adresáři, kde se nachází MATLAB. To je také důvodem toho, že pokud se zobrazí k dané funkci chyba, uvidíme často i přímo soubor dané funkce, kde je definovaná funkcionalita.

Shrnutí: MATLAB si asociuje několik vlastních typů souborů. Nejdůležitější jsou MAT a M soubory, obsahující napsané výpočty a skripty.

2 Jak MATLAB pracuje

Syntaxe výpočtů a jazyka pro programování aplikací je poměrně jednoduchá a spousta funkcí již existuje interně. Než přistoupíme k samotné syntaxi, musíme si povědět o vnitřní logice MATLABu.

2.1 Vše je matice

Název MATLAB (Matrix Laboratory) dává tušit, že MATLAB bude souviset s maticemi. MATLAB je na maticích založen. Vše v MATLABu se chová jako matice. Jakákoli proměnná, číslo, vektor – vše je maticí. Ano, i číslo je prezentováno jako matice 1×1, vektor pak m×1. To je základním rysem celého MATLABu, tím nejdůležitějším.

2.2 Základní syntaxe

V systému MATLAB lze libovolně používat proměnné či počítat přímo. Do spuštěného rozhraní napišme `1+3` a posléze `a=1+3`. Rozdíl je v tom, že v prvním případě MATLAB výraz hned vyhodnotí, tedy vypíše 4. V druhém vypíše také 4, ale zároveň uloží hodnotu do proměnné *a* (typ 1x1). Pokud za výraz dáme středník, nevypíše se výsledek, ale jen uloží do proměnné. Tedy:

==Př. 1==

- `1+3` – vypíše ihned výsledek a nic neuloží, resp. hodnoty bude uložena v dočasné proměnné *ans*, dokud nebude přepsána jiným výsledkem
- `a=1+3` – vypíše výsledek a uloží jej do proměnné *a*
- `a=1+3;` - nevypíše výsledek, ale vnitřně jej uloží do proměnné *a*

K deklaraci proměnné *a* jsme použili přiřazovací operátor =, neplést s ==, což je rovnost, bude ještě zmíněno dále v textu. To je základní syntaxe, kterou musíme striktně odlišovat. Podívejme se dále. Napišme následující jednoduchý skript:

==Př. 2==

```
a=5+3;
b=a-4;
c=a/b
```

Ano, MATLAB vypíše na konci hodnotu *c*=2. Napsali jsme první jednoduchý skript, který lze uložit. Pokud v budoucnu přepíšeme proměnné *a*, *b*, dostaneme jiné *c*. Nyní se podívejme na zápis matic a vektorů.

Shrnutí: Opakování je matka moudrosti a tedy – vše je reprezentováno jako matice s určitým datovým typem. Taktéž je velmi důležité odlišovat zápisy syntaxe se středníkem a bez něj. Nedeklarujeme-li proměnnou, příkaz (výpočet) se ihned provede a uloží do dočasné proměnné ans.

2.3 Matice a vektory a operaci s nimi

Matice, základ MATLABu. Vektor, 1-D případ matice. Matice je definována jako 2D pole rozměru *m*×*n* nad množinou čísel (reálných, komplexních, atd.). Zabrousíme nyní trochu do statistiky a ukážeme si v praxi, jak jednoduše využít zabudovaných funkcí MATLABu.

Představme si vektor jako statistický znak. V něm vypočteme aritmetický průměr, směrodatnou odchylku, minimum a maximum. Pamatujme si: to co je za znakem *%*, je poznámka, podobně jako např. *//* v Javě či C#.

==Př. 3==

```
>> A=[3, 1, 7, 11, -4]; % vektor mx1
>> std(A)

ans =

    5.7271

>> mean(A)

ans =

    3.6000

>> max(A)

ans =

    11

>> min(A)

ans =

    -4
```

Obecný zápis matice je v hranatých závorkách a každý řádek je oddělen středníkem. Počet prvku musí být v každém řádku matice stejný. Vytvořme matici *U* rozměru 4×3 a k ní transponovanou matici *Z* (tedy 3×4):

==Př. 4==

```
   U = [16 3 2 7; 5 10 11 -2; 9 6 7 12]; % vytvořili jsme matici U
4x3
   Z = transpose(U)
```

```
Z =

16  5  9
3  10  6
2  11  7
7  -2  12
```

MATLAB zná i mnohé předdefinované typy matic, například zeros vytvoří nulovou matici, kde rozměr je zapsán vektorem:

==Př. 5==

```
zeros(12,4) %vygeneruje nulovou matici 12x4 se samými nulami
```

Vektor můžeme taktéž vyjádřit zápisem, kdy se vytvoří vektor zadaný počáteční hodnotou, krokem a koncovou hodnotou, tedy takto:

==Př. 6==

```
u = [0:5:50]
u =
    0  5  10  15  20  25  30  35  40  45  50
```

To se hodí pro tvorbu různých řad a posloupností číselných hodnot s daným pravidlem. A opět můžeme využít operací z **Př. 3** a mnohé další, například transpozici vektoru u na sloupcový vektor w:

```
w=u´
% nebo zápisem
w=transpose(u)
```

Shrnutí: Maticové a vektorové operace jsou pilířem MATLABu. Nabízí mnoho funkcí a matice jako prvek se dá použít kdekoli jinde.

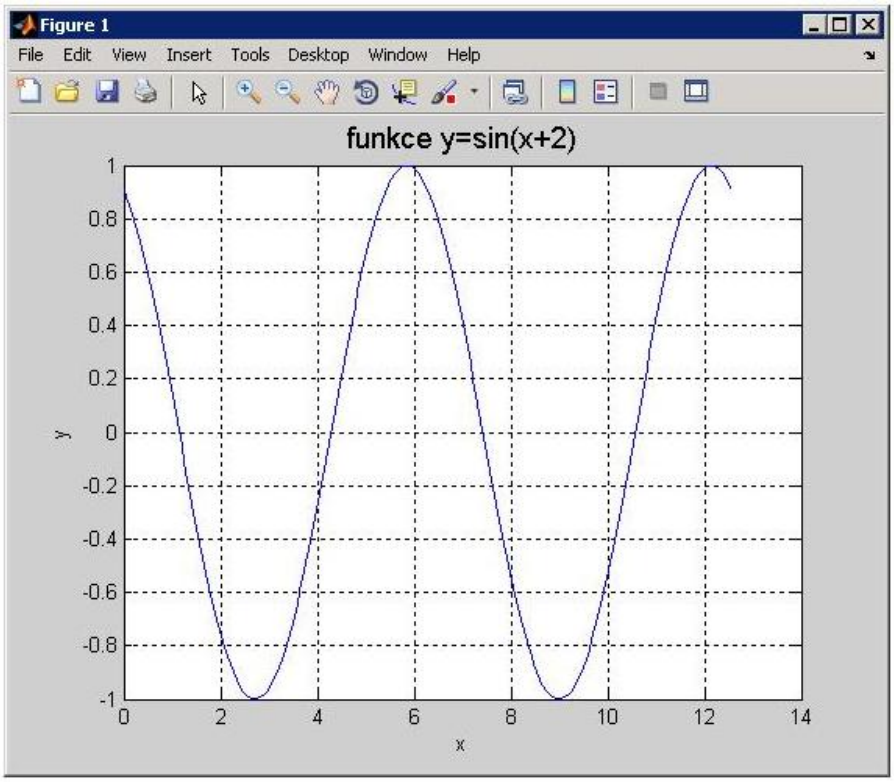
2.4 Kreslíme grafy funkcí

O něco složitější bude vykreslení jednoduché funkce *y=sin(x+2)*. Nelze totiž napsat jen tuto funkci, musíme definovat rozsah osy *x* a dělení. Proto, že opět toto číslo je matice. Na závěr graf vykreslíme do tzv. Figure, což je okno pro kreslení grafických objektů. Pojďme na zmíněnou funkci.

==Př. 7==

```
x = 0:pi/50:4*pi; % definujeme rozsah a dělení x osy, tedy <0; 4pi> s dělením pi/50
y = sin(x+2); % definujeme funkci
```

```
plot(x,y) %vykreslíme funkci do Figure okna

% parametry vykreslení – nadpis, popis os a zobrazení mřížky (grid)
title('funkce y=sin(x+2)','fontsize',14), xlabel('x'),ylabel('y')
```

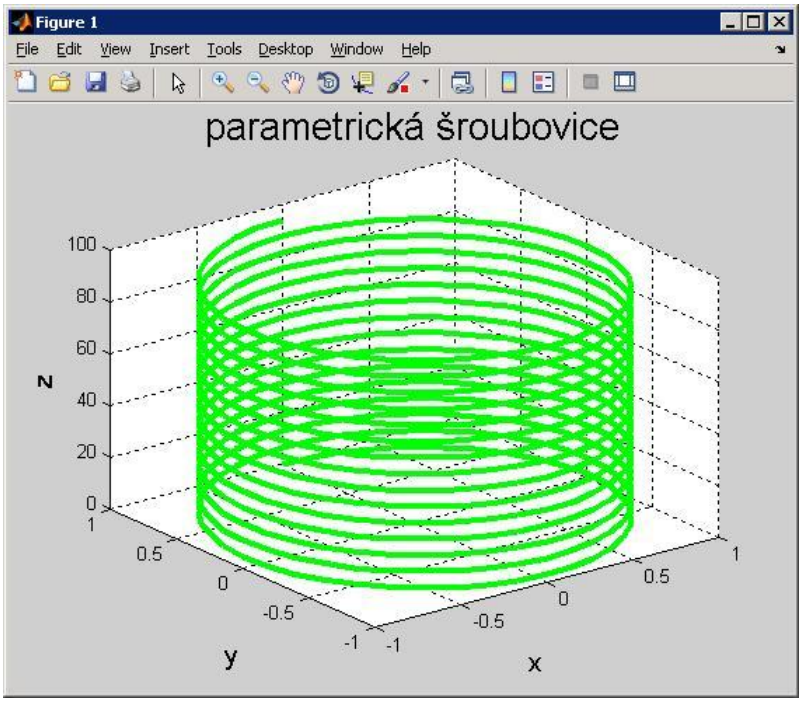


Obrázek 2.1: Vykreslená funkce f(x)=sin(x+2) v okně Figure

Analogicky takto můžeme kreslit 3D grafy funkcí více proměnných. Ukažme si příklad vykreslení parametricky zapsaného grafu šroubovice.

==Př. 8==

```
t=0:pi/20:30*pi;
plot3(sin(t),cos(t),t,'LineWidth',3,'Color','green'),
title('parametrická šroubovice','fontsize',20),
xlabel('x','fontsize',14), ylabel('y','fontsize',14),
zlabel('z','fontsize',14)
grid on
```



Obrázek 2.2: Parametrická šroubovice jako ukázka 3D grafiky

Na tomto příkladě vidíme užití opět stejných možností vykreslení, navíc však je definována tloušťka čáry a její barva, parametry *LineWidth* a *Color*. Napsáním *hold on* je možnost kreslit více grafů do jednoho Figure okna. Všimněme si, že jakékoli popisky, názvy, atd., které nejsou proměnnými, se musí uzavřít do *'nějaký text'* uvozovek.

Shrnutí: Figure okno je oknem pro vykreslování 2D i 3D grafiky. Pro vykreslení funkcí a jiných grafů je nutno definovat rozsah os. Vlastnosti Figure okna jsou velmi modifikovatelné a vykreslenou grafiku můžeme doplnit názvem, popisky os, stupnicí, atd. Jakýkoli textový řetězec musí být odlišen uzavřením do jednoduchých uvozovek.

2.5 Programovací struktury

V MATLABu lze používat i klasické programovací struktury, jako IF-THEN podmínky, cykly FOR, WHILE a mnohé další. Ukažme si použití podmínky IF-THEN-ELSE na jednoduchém příkladu porovnání čísla s výpisem hlášky do okna typu Message Box. Využijeme k tomu generování náhodných čísel pomocí funkce *randi*.

==Př. 9==

```
   p = randi([-10 10],1,1); % generování 1 náhodného čísla
z intervalu <-10 ; 10>
   p
   if p>0
   str = 'Zadane cislo je kladne'; %co se vypíše do okna
   msgbox(str,'Porovnani cisla','warn') %titulek okna a typ warn
(upozornění)
```

```
    else %tedy pokud p<0
    str = 'Zadane cislo je zaporne';
    msgbox(str,'Porovnani cisla','warn')
    end
```

Podrobně bude programovacím konstrukcím věnována samostatná kapitola dále v textu s příklady. Tento příklad by se mohl uložit jako M-soubor, prozatím jsme jej použili přímo.

Shrnutí: MATLAB poskytuje možnost programování s běžnými programovacími strukturami, jako jsou podmínky či cykly.

3 Operátory, výrazy a proměnné

Základní logika již byla nastíněna v úvodu, tedy že vše se chová jako matice a jak deklarovat proměnné. Podívejme se nyní blíže na operátory, vestavěné funkce a související příklady.

3.1 Operátory a funkce

MATLAB disponuje stovkami vestavěných funkcí. Co se operátorů týče, můžeme využít základních i pokročilých operátorů – aritmetických, algebraických, logických, relačních, atd. MATLAB dodržuje všeobecně uznávanou prioritu operací, takže nastíníme rozdíl:

```
a=(50+20)/10
a=50+20/10
```

V prvním příkladě dostaneme výsledek *a*=7, v druhém pak *a*=52. Nejvyšší prioritu mají závorky, nejnižší pak součet a rozdíl. Dále je možné využít množství vestavěných funkcí, např. výpočet odmocniny, goniometrické funkce, statistické funkce (ukázáno na příkladu s vektorem), maticové operace, zaokrouhlovací funkce, logické funkce a mnohé další. Ukažme si příklad:

==Př. 10==

```
cislo1=sqrt(45) % do promenne cislo1 vypočteme odmocninu z čísla
45
cislo2=floor(cislo1) % vypočtené číslo zaokrouhlí funkce floor na
nejbližší nižší celé
cislo3=ceil(cislo1)% vypočtené číslo zaokrouhlí funkce ceil na
nejbližší vyšší celé
```

Dostaneme tedy: *cislo1*=6,7082039…, *cislo2*=6 a *cislo3*=7.

Důležitá poznámka – operátor součinu * je nutno uvádět, jinak nebude výraz vyhodnocen. Například funkci *y=2x+sqrt(x)* je nutno zapsat jako y=2*x+sqrt(x). U relačního operátoru rovnosti je třeba psát takto *A==2*, např. v IF podmínkách.

Dále máme operátory pro maticové dělení obdélníkových matic – levé a pravé dělení operátory \ a /. Umocnění značí ^. Unární - je shodné s binárním, záleží na kontextu.

*Shrnutí: Operátory mají v MATLABu exaktně danou prioritu, která je matematicky globálně zavedena a dodržuje ji. Takže pozor na rozdíly při psaní výrazů v závorkách. A především si neplést relační operátor rovnosti == za přiřazovací příkaz = u deklarace proměnné. Taktéž vynechání symbolu * pro součin má fatální následek – výraz se nevyhodnotí, tedy je nutné jej používat. Funkci 2x nutno zapsat jako 2*x.*

3.1.1 Relační operátory

MATLAB užívá standardní sadu relačních operátorů, tedy <, >, <=, >= a ==. Relační operátory užijeme zejména při programování podmínek s IF-THEN pravidly či jednoduše pro výpočty obecně. V dřívějších verzích MATLABu existovaly také tzv. IS funkce, které vracely logickou boolean hodnotu 0 nebo 1, zda objekt splnil danou otázku. Např. *isempty* zjišťovalo, zda je daný objekt (vektor, matice) prázdný. Pozor – nezaměnit přiřazovací operátor = za relační operátor ==.

3.2 Proměnné

Jak již bylo předvedeno v úvodu, jakákoli hodnota se dá uložit do proměnné. Každou proměnnou deklarujeme svým názvem. Obecně počet proměnných může být libovolných, limitace je de facto jen pamětí RAM. Pozor na 3 zásadní skutečnosti:

- ***názvy proměnných jsou case-sensitive, tedy rozlišují se malá a velká písmena***
- ***název nemůže být vnitřní funkcí (nemůžeme tedy deklarovat např. cos)***
- ***MATLAB nijak neupozorňuje na přepis proměnné novou!!!***

Zejména třetí bod může mít fatální následky. Vytvoříme-li proměnnou *v*, počítáme s ní a poté vytvoříme novou proměnnou *v*, bude automaticky přepsána novou hodnotou! Proto je nutné mít v proměnných systém, zejména u rozsáhlejších projektů. Můžeme například pojmenovávat proměnné podle funkce a hodnoty. Maximální délka názvu proměnné je 31 znaků. Rozdíl mezi zápisem se středníkem a bez, je ukázán v **Př. 1**.

3.2.1 Správa proměnných

Proměnné můžeme jednoduše spravovat v okně Workspace. Všechny proměnné se ukládají do paměti, zobrazit si je můžeme v okně Workspace, kde uvidíme jejich název, hodnotu a datový typ. Výpis všech aktuálně deklarovaných uložených proměnných ve Workspace obdržíme napsáním příkazu *who*. V tomto okně můžeme proměnné zobrazovat, mazat či je všechny uložit do souboru.

Shrnutí: Proměnné jsou deklarovány přiřazením názvu. Pozor na case-sensitive názvů a zejména na přepis proměnných! Zdůrazněno v předchozím textu. Je vhodné zvláště u náročnějších projektů dodržovat určitý systém pojmenování proměnných. Délka proměnné je omezena na 31 alfanumerických znaků. Výpis aktuálních proměnných dostaneme příkazem who či v okně Workspace.

3.3 Datové typy proměnných

Každá uložená proměnná má svůj název a datový typ. Jak jsme řekli, vše je maticí, datový typ se tedy odvíjí od rozměrů matice a hodnot v ní. Datových typů existuje několik, nejčastější jsou int8, uint8, int16, uint16, int32 a uint32 dle typu matice. Například binární matice je „jednodušší“, obsahuje pouze binární hodnoty, naopak komplexní matice (viz 3.5) bude reprezentována složitějším datovým typem. Při zpracování obrazu v rámci Image Processing Toolbox (viz kap. 10) se setkáme s různými datovými typy matic dle reprezentace obrazu (RGB, 8-bit, binární).

Dále je možno definovat pole (cell array) a další struktury. Nejjednodušším datovým typem je binární logická hodnota, nejsložitějším pak matice např. RGB obrazu či komplexní matice, kde každá hodnota v matici je 2-D číslem.

Shrnutí: Datové typy jsou vlastně datové typy matice. Typ udává strukturu matice.

3.4 Globální proměnné

Smysl globálně deklarovaných proměnných má význam u rozsáhlejších aplikací, tedy M-skriptů obsahujících mnoho deklarovaných funkcí. Globální proměnná je deklarována pro více funkcí a její hodnota se nemění od první deklarace. Globální proměnnou deklarujeme takto:

```
global a b c
```

Tedy máme 3 proměnné, které budou použity na různých místech programu, avšak budou nabývat stále neměnné hodnoty. Více o M-skriptech a programování v kapitolách 6 a 7, dále v kap. 11 s praktickým příkladem.

Shrnutí: Globální proměnné užíváme pro zavedení proměnné a její užití v dalších funkcích mimo aktuální. Využití u rozsáhlejších programů či tam, kde si prostě potřebujeme během výpočtu „pamatovat" určitou hodnotu, která bude použita jinde.

3.5 Zápis komplexních čísel

Samozřejmě MATLAB se neomezuje na reálná čísla, ale pracuje i s jejich nadmnožinou, čísly komplexními, která jsou 2-rozměrná a obecně v algebraickém tvaru zapsána jako *a+bi*, kde *a* je reálná složka komplexního čísla a *b* imaginární (*i* je imaginární jednotka, která je definována $i^2 = -1$). Geometrická interpretace je pomocí goniometrických funkcí. MATLAB umožní jednoduše definovat komplexní číslo jako samostatné, či prvek matice, vektoru, poradí si i s komplexními funkcemi. Zápis komplexního čísla můžeme realizovat např. *3+2i* či *4-3j* (i, j budou chápány jako komplexní jednotka). Na příkladu si ukážeme součet 2 komplexních matic 2×2.

==Př. 11==

```
c1 = [2+3i 1-2i; 6 -4+i]; % komplexní matice
c2 = [7+3i -2i; 6 2+i]; % komplexní matice
secti = c1+c2

secti =

  9.0000 + 6.0000i   1.0000 - 4.0000i
  12.0000            -2.0000 + 2.0000i
```

Obecně pro práci s komplexními čísly nabízí MATLAB základní funkce – *real* (reálná část), *imag* (imaginární část), *abs* (absolutní hodnota), *angle* (výpočet fáze), *conj* (komplexně sdružené číslo). Např. pro číslo -3+4i bude komplexně sdružené číslo -3-4i.

Shrnutí: MATLAB není omezen na množinu reálných čísel, pracuje stejně i s čísly komplexními. Jejich 2-D zavedení na reálnou a imaginární část má za následek složitější maticový datový typ. MATLAB má v sobě spoustu integrovaných funkcí pro operace nad komplexními čísly.

4 Matice, vektory a izolované hodnoty

Na začátku jsme si řekli, že MATLAB vše chápe jako matici a na Př. 4 jsme ukázali obecný zápis matice *m*×*n*. Taktéž zápis vektoru, tedy 1D pole. Obecně na matici *m*×*n* pohlížíme takto:

$$I(m,n) = \begin{bmatrix} p_{11} & \cdots & p_{m1} \\ \vdots & \ddots & \vdots \\ p_{1n} & \cdots & p_{mn} \end{bmatrix}$$

a takto je MATLAB interpretuje. I samotné číslo 5 tedy MATLAB vyhodnotí jako matici [*5*]. Vektory zapisujeme stejně, tedy v hranatých závorkách. Matice i vektory mohou být reálné i komplexní.

Shrnutí: I číslo je reprezentováno jako matice typu 1x1.

4.1 Maticové operace

S maticemi můžeme provádět celou řadu operací, nejen základní úkony, ale i pokročilé operace díky vestavěným funkcím. Můžeme například zjistit vlastní čísla a vektory, stopu matice, hodnost, determinant, atd. Všechny tyto funkce má MATLAB integrované. Příklad:

==Př. 12==

```
A1 = [3 4; 6 -2] % matice 2x2
A2 = det(A1)
A3 = eig(A1)
A4 = diag(A1) % vypíše diagonálu matice
A5=inv(A1)

A1 =

     3     4
     6    -2


A2 =

   -30


A3 =

     6
    -5
```

```
A4 =

     3
    -2


A5 =

    0.0667    0.1333
    0.2000   -0.1000
```

Ukažme si ještě jeden zajímavý zápis matic. Definujeme matice V a W a poté vytvoříme matici VW, která bude sloučením V a W.

```
V = [0 -2 3];
W = [4 -5 7];
VW = [V W]

VW =
     0     2    -3     4    -5     7
```

či takto:

```
V_W = [V ;W]

V_W =
     0     2    -3
     4    -5     7
```

Shrnutí: MATLAB poskytuje základní i pokročilé operace s maticemi. Užitečnou funkcí je konkatenace matic, tedy možnost spojení více matic do jedné. Je možné to provést různými způsoby. Spousta funkcí pro maticové operace je k dispozici a nezáleží, zda se jedná o matice reálné či komplexní.

4.2 Vektorové operace

Vektor o *n* složkách lze jednoduše samozřejmě jako speciální případ matice, tedy $m \times 1$. V kap. 2.3 jsme nastínili, jak zapsat vektor a jak se chová v MATLABu. Nyní si ukážeme další možnost. Definujme nějaký vektor *t*:

```
t = [4 3 7 -9 1];
```

MATLAB nám umožní z tohoto vektoru vytvořit další vektory ve sloupcích, které budou nějakou operací původního vektoru *t*.

```
t2 = [x 2*x x/2]
t2 =
    4.0000    8.0000    2.0000
    3.0000    6.0000    1.5000
    7.0000   14.0000    3.5000
   -9.0000  -18.0000   -4.5000
    1.0000    2.0000    0.5000
```

Shrnutí: Vektory jsou chápány správně jako matice mx1 a je možno použít tvorbu vektorů z předchozích, např. do sloupců. S vektory můžeme provádět operace podobně jako na maticích. Zápis vektoru je jako zápis matic, akorát že neoddělujeme pomocí ; řádky, neboť je pouze jeden.

4.3 Izolované hodnoty

Někdy však nastane případ, že potřebujeme zapsat několik hodnot, které ale nemají vyjadřovat matici ani vektor, budou to nějaké izolované hodnoty. To si ukážeme na následujícím příkladě s polem 50 náhodných čísel generovaných z intervalu ⟨0; 1⟩ dle normálního rozdělení.

==Př. 13==

```
>> nah=rand(1,50) % 20 náhodných čísel pomocí funkce rand
nejv=max(nah) % vyber maximum z pole
nejm=min(nah)
prum=mean(nah)
odch=std(nah)

nah =

  Columns 1 through 11

    0.7112    0.2217    0.1174    0.2967    0.3188    0.4242
0.5079    0.0855    0.2625    0.8010    0.0292

  Columns 12 through 22

    0.9289    0.7303    0.4886    0.5785    0.2373    0.4588
0.9631    0.5468    0.5211    0.2316    0.4889

  Columns 23 through 33
```

```
       0.6241      0.6791      0.3955      0.3674      0.9880      0.0377
0.8852    0.9133    0.7962    0.0987    0.2619

     Columns 34 through 44

       0.3354      0.6797      0.1366      0.7212      0.1068      0.6538
0.4942    0.7791    0.7150    0.9037    0.8909

     Columns 45 through 50

       0.3342    0.6987    0.1978    0.0305    0.7441    0.5000


   nejv =

       0.9880


   nejm =

       0.0292


   prum =

       0.4984


   odch =

       0.2836
```

V příkladu jsme vytvořili vektor, tedy pole o 20 náhodných hodnotách. Můžeme však tyto hodnoty mít izolované, např. výběr maxima či minima z pole.

Shrnutí: Pozor na izolované hodnoty, nechovají se jako vektor! Ale stále s nimi můžeme pracovat jako s řadou čísel a zjišťovat tak např. statistické ukazatele. Je-li vektor či matice příliš velká, MATLAB zobrazí výsledek po 10 sloupcích (columns).

5 Funkce, grafy a jejich vykreslování do Figure oken

Jak jsme již uvedli v **Př. 7**, MATLAB nedokáže „jen tak“ vykreslit graf funkce. Je nutno definovat nejprve meze osy *x*, poté zapsat samotnou funkci a příkazem plot (či jiným z knihovny grafů) vykreslit. Obecně lze funkce rozdělit na diskrétní a spojité. Spojitou funkcí byla i funkce z **Př. 7** *y=sin(x+2)*. Diskrétní pak může být například zaznamenané hodnot teploty. Ty je pak lépe vykreslit například grafem typu bar (sloupečky). Ukažme si nyní modifikaci **Př.7**., kdy nemusíme přímo definovat funkci *y*.

==Př. 14==

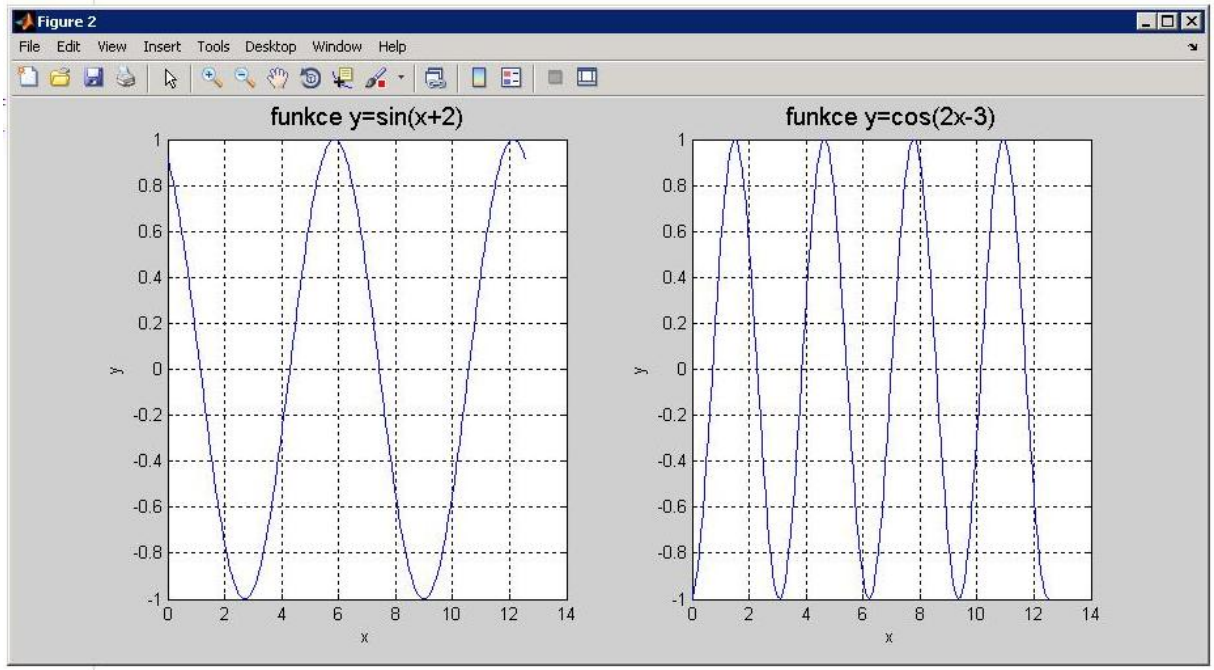
```
   x = 0:pi/50:4*pi; % definujeme rozsah a dělení x osy, tedy <0; 4pi> s dělením pi/50
   y = sin(x+2); % definujeme funkci
   plot(x,y) %vykreslíme funkci do Figure okna

   % a totéž udělá následující zápis
   x = 0:pi/50:4*pi; % definujeme rozsah a dělení x osy, tedy <0; 4pi> s dělením pi/50
   plot(x, (sinx+2)) %vykreslíme funkci do Figure okna
```

Analogickou extenzí je vykreslování 3D grafů pro funkce 2 proměnných, přidáním rozsahu osy *y* a zápisem funkce *z*. 3D grafy lze renderovat, osvětlovat a animovat. Dobrou ukázku 3D grafu je na Př. 8 v kap. 2.4. Další možnosti pak nabízí použití síťovaných a vrstevnicových (contour) grafů. K těmto účelům slouží funkce *meshgrid*. I v případě 3D grafů stále platí stejné možnosti, jako např. přidání popisků os, názvu, zobrazení souřadnic, užití *subplot* funkce a další možnosti, které byly ukázány na předchozích případech 2D grafiky. Stále se jedná o vykreslení do Figure okna, které má stále stejné možnosti úprav.

Podívejme se nyní ještě na možnost vykreslení více objektů do jednoho okna Figure pomocí funkce *subplot*. To spočívá v rozdělení okna na matici m×n a zadání pozice, kam se má objekt vykreslit, tedy syntaxe je *subplot(mnp)*. Musíme však definovat Figure jako okno. Vezměme si funkci z Př. 7 a přidejme druhou funkci *y=cos(2*x-3)*, kterou vykreslíme vedle původního grafu.

==Př. 15==

```
   x = 0:pi/50:4*pi;
   y = sin(x+2);
    % nyní definujme novou fci y2
   y2=cos(2*x-3);
   figure, % zavedení okna Figure a vykreslení grafů y, y2 do stejného okna vedle sebe
   subplot(121), plot(x,y),title('funkce y=sin(x+2)','fontsize',14), xlabel('x'),ylabel('y')
   subplot(122), plot(x,y2),title('funkce y=cos(2x-3)','fontsize',14), xlabel('x'),ylabel('y')
```



Obrázek 5.1: Dvě vykreslené funkce vedle sebe funkcí subplot

Tak stejně by mohly vedle sebe být např. 2 grafy průměrných teplot v lednu a červenci na různých místech. Pokud bychom chtěli 4 grafy do matice 2×2, definovali bychom funkci *subplot* takto:

```
4grafy=figure, % i okno Figure se dá uložit jako proměnná
subplot(221),...
subplot(222),...
subplot(223),...
subplot(224),...
```

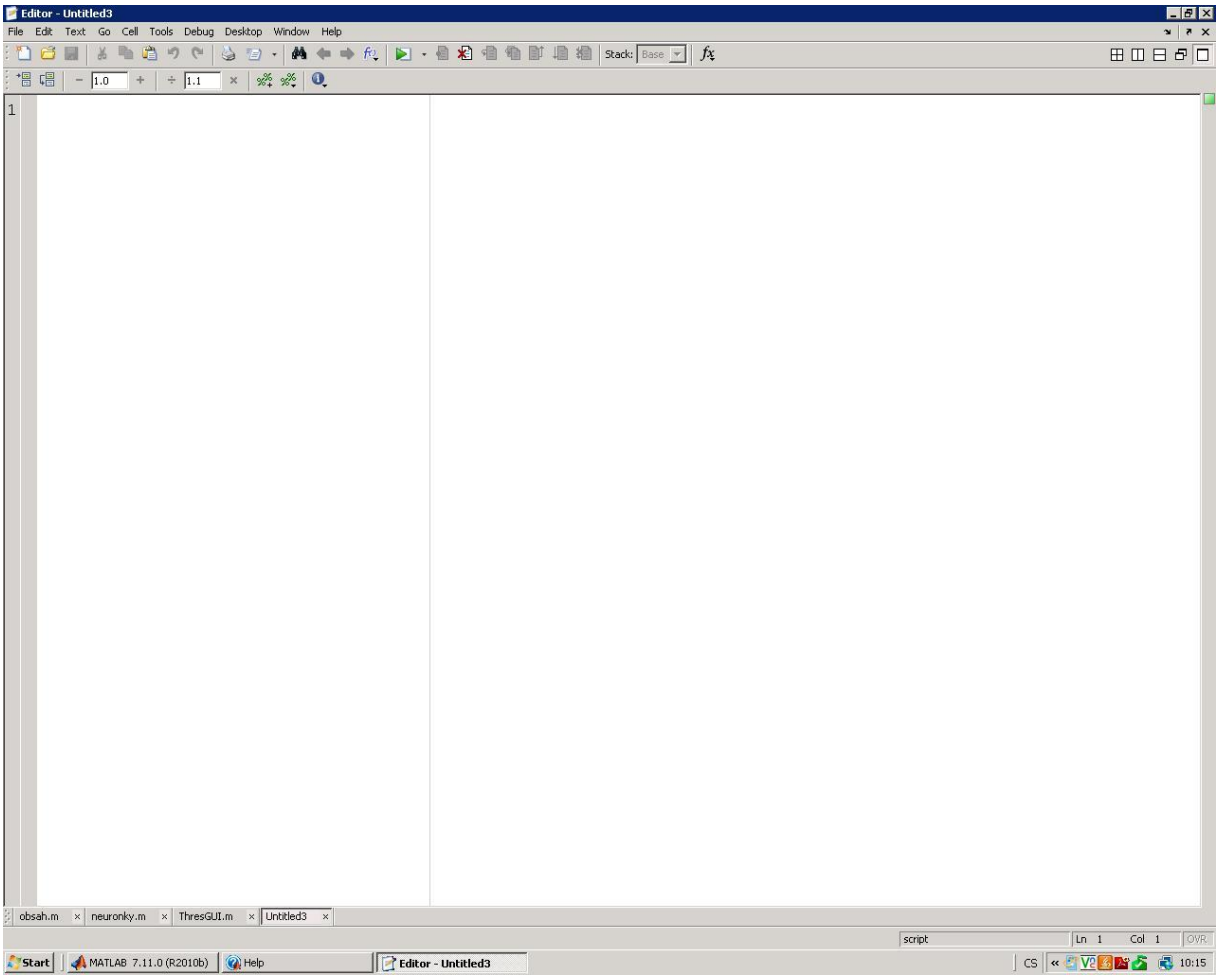
Příkazy *title* , *xlabel* , *ylabel* a *text* lze graf doplnit popisy (titulek, popisek x-osy, popisek y-osy, a text v grafu). Speciálním atributem je *name*, který slouží pro text záhlaví okna Figure. Výsledný graf lze vytisknout nebo exportovat do souboru (tlačítka na panelu nástrojů Figure).

Shrnutí: Vše se vykresluje do Figure oken. Můžeme použít funkci subplot pro vykreslení více objektů do jednoho okna, kde subplot(xxx) určuje xxx pozici v matici mxn. Funkce hold on zase umožní kreslit do stejných os více grafů. Rozsah os je nutno a priori definovat rozsahem, případně krokem. Grafiku ve Figure okně možno doplnit textem, legendou, nadpisem, popisky os, atd. Z Figure okna je možné uložit výstup ve formě FIG souboru či bitmapového obrazu.

6 Editor a M soubory

Zatím jsme uvažovali všechny výpočty pouze v základním prostředí, tedy okně MATLAB. Nikam jsme neuložili svou práci jako soubor, tak jako například text v textovém editoru pro použití později.

Jak jsme zmínili na začátku, v podstatě můžeme pracovat ve 2 režimech – přímo v příkazovém řádku MATLAB a v Editoru. Editor je silná součást MATLABu, ne-li skoro důležitější jak základní okno. Editor je zkrátka editorem, jako každý jiný, například textový či vývojové prostředí. Nové prázdné okno Editoru spustíme příkazem File-New či ikonou New na panelu nástrojů.



Základem jsou tzv. M soubory s příponou M. Do Editoru tak napíšeme celý program, v MATLABu se označuje jako skript. M soubor není nic jiného než posloupnost příkazů a funkcí, například jako kdybychom programovali v klasických jazycích jako C či třeba C#. Editor umožní si uložit celou naši práci a především skript spustit. Můžeme tak vytvořit celý program a ten později spustit znovu. Spuštění skriptu provedeme nejjednodušeji stiskem klávesy F5 nebo tlačítkem  na panelu nástrojů Editoru.

Důležitá poznámka – v Editoru se nikdy nevypisují vypočítané hodnoty, ty uvidíme v základním okně MATLABu. Taktéž se v základním okně vypisují chybové hlášky při spuštění skriptu. Editor je skutečně pouze editor našich programů.

Ukažme si praktický příklad vytvoření nějakého skriptu s uložením jako M soubor. Vytvořme si program, který vykreslí do jednoho okna Figure ve 2 podoknech grafy základních goniometrických funkcí *f1=sin(x)*, *f2=cos(x)*, *f3=tan(x)* a k nim do 2. podokna k nim funkce hyperbolické, tedy *f4=sinh(x)*, *f5=cosh(x)* a *f6=tanh(x)*. Všechny budou v rozsahu 0-8pi. Využijeme tak již popsané funkce *subplot* a *hold on*.

==Př. 16==

```
x = 0:pi/50:8*pi; % rozsah x-osy
% definice funkcí
f1=sin(x);
f2=cos(x);
f3=tan(x);
f4=sinh(x);
f5=cosh(x);
f6=tanh(x);

figure,
subplot(211),
plot(x, f1)
hold on % podrží aktuální graf proti přepisu novým
plot(x, f2)
hold on
plot(x, f3),
title('základní goniometrické funkce','fontsize',14),
xlabel('x'),ylabel('f(x)')

subplot(212),
plot(x, f4)
hold on
plot(x, f5)
hold on
plot(x, f6),
title('hyperbolické goniometrické funkce','fontsize',14),
xlabel('x'),ylabel('f(x)')
```

Celý tento skript (program) můžeme jednoduše uložit pomocí File-Save či ikonkou na panelu nástrojů. Otevře se běžný dialog k ukládání a uložíme si program a můžeme jej kdykoli znovu otevřít a spustit. Další možností k příkladu je například výpočet konkrétních funkčních hodnot goniometrických funkcí – takto:

```
sin(pi/2) + cos(2*pi)
ans =
```

```
     2

asin(0.5)
ans =
    0.5236
```

Jiný příklad užití Editoru si ukážeme na maticových výpočtech, kde se výsledky nezobrazí jako grafika, ale v příkazovém okně MATLABu. Základem bude vytvoření dvou matic, jedna rozměru 5×5 (pro předvedení výpočtu vlastních čísel) a druhá rozměru 4×6 pro ukázku transpozice. Budeme výsledky ukládat do proměnných a volat je. Matice budou generovány funkcí *rand* (náhodná matice o daném rozměru), tedy výsledky budou při každém spuštění jiné.

==Př. 17==

```
m1=rand(5,5);
m2=rand(4,6);
% zobrazení matic v hlavním okně
m1
m2

m2t=transpose(m2); % transpozice matice m2
m2t
m1e=eig(m1); % spočte vlastní čísla a vektory matice m1
m1e % vypíše je

% operace na maticích m1 a m2
m1krat=m1*m1;
m1krat
m1plus=m1+m1;
m1plus
m2skalar=5*m2;
m2skalar
```

V hlavním okně uvidíme například následující výstupy (dle generovaných matic *rand*).

```
m1 =
    0.8147    0.0975    0.1576    0.1419    0.6557
    0.9058    0.2785    0.9706    0.4218    0.0357
    0.1270    0.5469    0.9572    0.9157    0.8491
    0.9134    0.9575    0.4854    0.7922    0.9340
    0.6324    0.9649    0.8003    0.9595    0.6787
```

```
m2 =
    0.7577    0.1712    0.0462    0.3171    0.3816    0.4898
    0.7431    0.7060    0.0971    0.9502    0.7655    0.4456
    0.3922    0.0318    0.8235    0.0344    0.7952    0.6463
    0.6555    0.2769    0.6948    0.4387    0.1869    0.7094

m2t =
    0.7577    0.7431    0.3922    0.6555
    0.1712    0.7060    0.0318    0.2769
    0.0462    0.0971    0.8235    0.6948
    0.3171    0.9502    0.0344    0.4387
    0.3816    0.7655    0.7952    0.1869
    0.4898    0.4456    0.6463    0.7094

m1e =
   3.2037
   0.5684
  -0.1026 + 0.5388i
  -0.1026 - 0.5388i
  -0.0457

m1krat =
    1.3164    0.9614    0.9676    1.0427    1.2492
    1.5213    1.1350    1.5754    1.5032    1.8462
    2.0937    2.3843    2.5910    2.6654    2.3472
    2.9873    2.2809    2.6699    2.5017    2.4191
    2.7964    2.3417    2.8111    2.6409    2.4855

m1plus =
    1.6294    0.1951    0.3152    0.2838    1.3115
    1.8116    0.5570    1.9412    0.8435    0.0714
    0.2540    1.0938    1.9143    1.8315    1.6983
    1.8268    1.9150    0.9708    1.5844    1.8680
    1.2647    1.9298    1.6006    1.9190    1.3575

m2skalar =
    3.7887    0.8559    0.2309    1.5855    1.9078    2.4488
    3.7157    3.5302    0.4857    4.7511    3.8276    2.2279
    1.9611    0.1592    4.1173    0.1722    3.9760    3.2316
    3.2774    1.3846    3.4741    2.1937    0.9344    3.5468
```

Shrnutí: Editor nám dává do rukou mocný nástroj pro programování celých aplikací v MATLABu. Základem je tzv. M-skript, který je vlastně napsaným programem. Ten může obsahovat funkce, výpočty, vykreslování grafů, použití podmínek a cyklů. M-skript je možno kdykoli příště spustit v Editoru a program se spustí.

7 Základy programování skriptů a GUI

V předchozí kapitole jsme si představili Editor, součást pro praktickou tvorbu skriptů. Ukázali jsme si 2 příklady, které ukázali, co vše je možné. Ovšem to zdaleka není vše.

MATLAB nám umožní využívat klasické programátorské konstrukce, jako cykly FOR a WHILE, dále podmínky a jejich větvení, logické podmínky, využívat funkce a deklarace globálních proměnných. Pomocí GUIDE (Graphical User Interface Development Editor) můžeme navrhovat celé aplikace s GUI.

7.1 GUI a GUIDE

Pomocí modulu GUIDE je možné tvořit grafické rozhraní aplikace. Každá aplikace s GUI má dvě části – programování funkčnosti a programování GUI. GUIDE je podobný jako například MS Visual Studio, kdy si přetahujeme ovládací prvky na plochu a těm poté přiřazujeme funkčnost, tedy události. Generuje automaticky základní ovládací zdrojový kód (podobně jako např. HTML head), vzhled GUI ukládá do FIG souboru a funkcionalita je v M-skriptu. Obsahuje všechny prvky *uicontrol*, kterými lze aplikaci ovládat, tedy tlačítka, posuvníky, atd. Zájemce o detailní popis GUIDE odkazuji na webové reference či knihu (1).

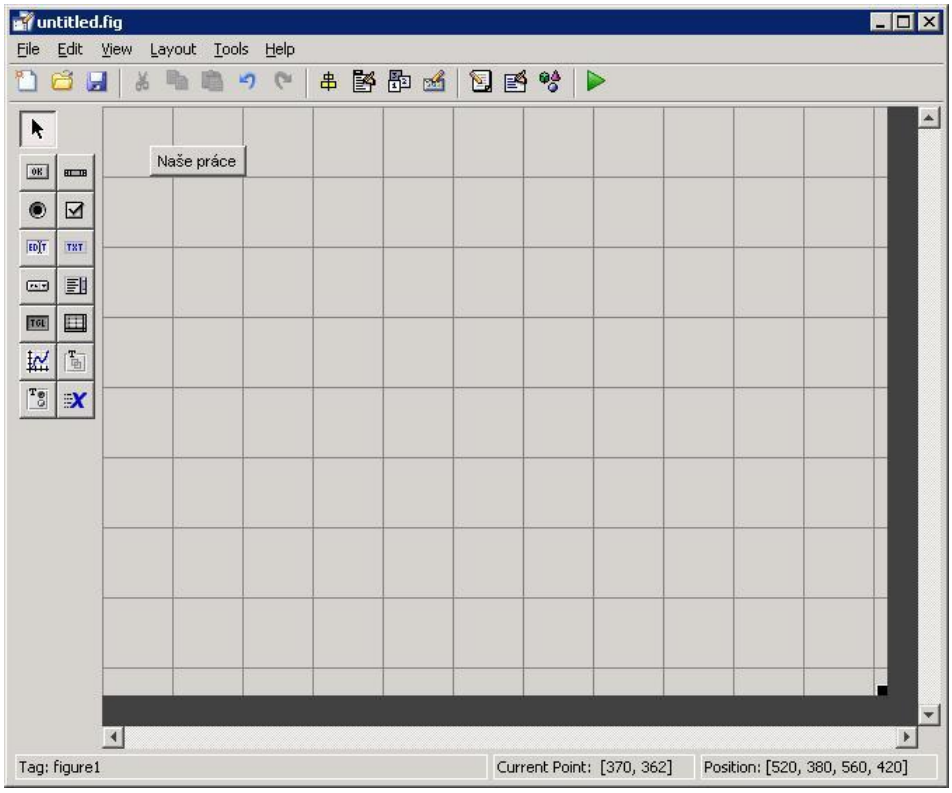
Shrnutí: GUIDE dává k dispozici možnost tvorby GUI k M-skriptům. Je tak možno vytvořit běžnou uživatelskou aplikaci s tlačítky a dalšími prvky. Práce v GUIDE je podobná jako v mnohých moderních IDE, kde pomocí drag-and-drop přetahujeme prvky do okna a v M-skriptu pak programujeme funkcionalitu (události prvků).

7.2 Praktické programování s praktickou aplikací

Ukážeme si praktickou ukázku programování na příkladu, kdy využijeme základní, výše zmíněné konstrukce s návrhem jednoduchého GUI s tlačítky.

Prostředí GUIDE spustíme napsáním *guide* do příkazové řádky. Budeme dotázání, zda chceme nové GUI či otevřít existující. Zvolíme nové, tedy záložku *Create New GUI*. Objeví se prázdné okno (coby aplikace) a vlevo dostupné ovládací prvky.

Podobně jako v jiných IDE, můžeme vizuálně umisťovat prvky do okna. Zvolme tlačítko (Push Button) a přetáhněme do plochy okna. Tlačítko bude mít výchozí název *Push Button*, který můžeme změnit, pokud klikneme na něj sekundárním tlačítkem a zvolíme *Property Inspector*, otevře se panel vlastností prvku, změníme položku *String* na *Naše práce*. Tlačítko změní svůj název na tento. Návrh tlačítka v prostředí GUIDE vidíme na obrázku níže.



Obrázek 7.1: Návrh GUI v GUIDE

Deklarace tlačítka je výchozí jako *pushbutton1*. Tímto jsme obstarali vizuální návrh prvku. Nyní přistoupíme k události, tedy definujeme co se má stát při stisku tlačítka, tedy aplikační logiku. Klikneme-li opět sekundárním tlačítkem na tlačítko a zvolíme Open M-File Editor, otevře se nám známý Editor s novou deklarací tlačítka takto:

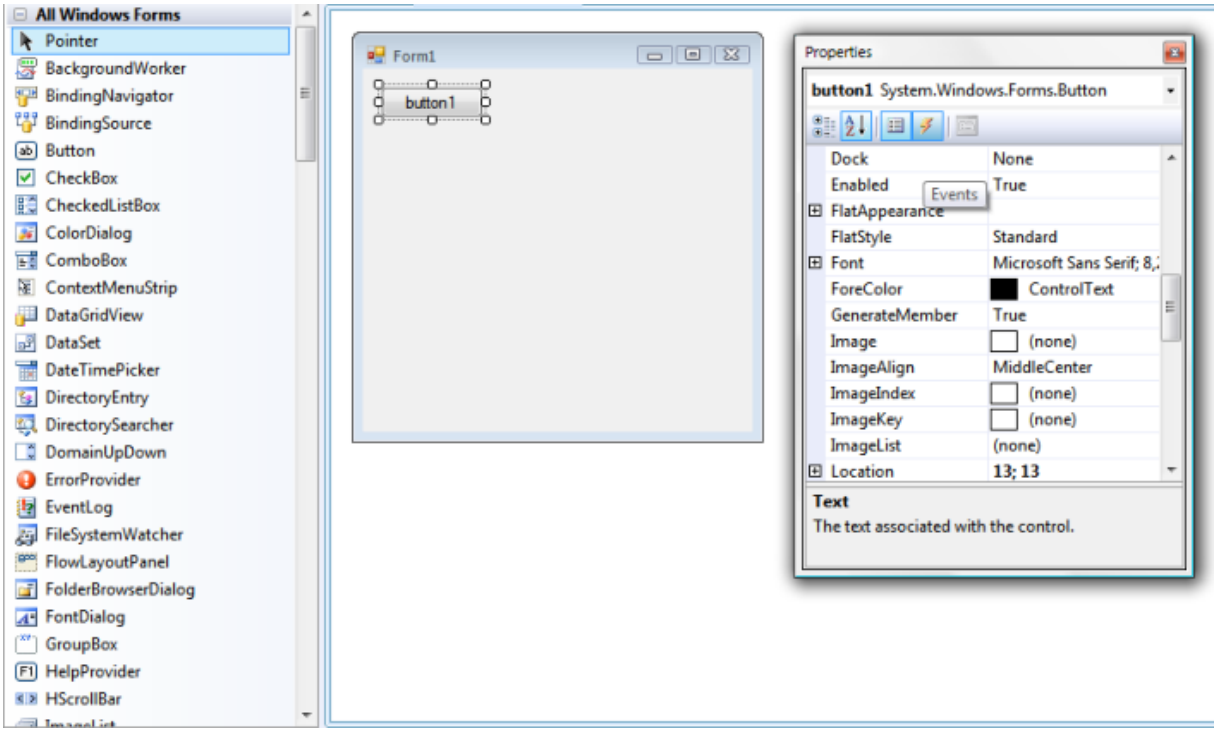
```
function pushbutton1_Callback(hObject, eventdata, handles)
```

Callback je událost, která bude definována pro stisk (push) tlačítka. Naprogramujeme jednoduchý skript, který se vykoná po stisku tlačítka. Na něm ukážeme princip programovacích struktur, podmínka IF byla nastíněna v Př. 8. Další programovací struktury budou rozebrány v následujícím příkladě.

Před příkladem si dovolím malou poznámku. Vizuální návrh aplikace se podobá práci v jakémkoli vývojovém prostředí, např. Visual Studio, kde navrhujeme aplikace Windows Forms. Máme také k dispozici možnost vizuálního návrhu (Designer), nastavování vlastností prvků (Properties), tedy podobně jako v MATLABu. MATLAB je také orientován jako OOP, tedy objektově orientovaný jazyk. Dokonce syntaxe události bude obdobná, srovnejte následující kód deklarace tlačítka v C# s výše uvedeným řádkem v MATLABu, rozdíl je pouze v uzavření do bloku:

```
private void button1_Click(object sender, EventArgs e)
      {
          // zde budou události tlačítka

      }
```

Ještě srovnejme s výše uvedeným obrázkem prostředí GUIDE s vizuálním návrhem Windows Forms aplikace v C#, opravdu způsob práce je velmi podobný.



Obrázek 7.2: Podobnost návrhu GUI s Visual C# Express IDE

Takže kdo vytvářel jakoukoli okenní aplikaci v libovolném vývojovém prostředí, GUIDE v MATLABu jej nijak nepřekvapí.

Vraťme se k vývoji aplikace v GUIDE. Po definici události Callback pro tlačítko pushbutton1, což je událost stisku tohoto tlačítka, budeme programovat samotnou funkcionalitu. Vše si ukážeme na příkladu, který se sestává ze 3 částí – maticové operace na matici náhodných čísel, užití logické funkce rovnosti matic a generování vektoru cyklem FOR.

==Př. 18==

```
   % 1. část generování vhodné matice s náhodnými čísly a maticové
operace
   mx1=rand(10,10) % čtvercová matice 10x10 s náhodnými čísly
   mx2=inv(mx1);
   mx3=transpose(mx1);
   mx2
   mx3

   % výpočet determinantu matice řád n s užitím minorů matice
   dt=det(mx1);
   dt
   % výpis hlášky dle podmínky determinantu
```

```
   if dt>0
   str = 'Determinant matice je kladny';
   msgbox(str,'Porovnani cisla','warn')
   else
   str = 'Determinant zadane matice je zaporny';
   msgbox(str,'Determinant','warn')
   end

   % nyní se čeká na uživatelův stisk OK
   % zjištění prázdnosti pole (matice), výsledkem bude vždy 0
(false)
   isempty(mx1)

   % 2. část s konkrétními maticemi
   1m = [3 4 1 9; 6 -3 5 7] % matice 4x2
   2m = [3 6 1 10; 6 -2 5 7] % matice 4x2
   3m = [3 4 1 9; 6 -3 5 7] % matice 4x2

   % logická funkce isequal vracející hodnotu 0/1 pro rovnost matic
   1a2=isequal(1m,2m) % vrátí 0
   1a3=isequal(1m,3m) % vrátí 1, matice jsou stejné
   2a3=isequal(2m,3m) % vrátí 0

   % 3. část vektor pomocí for cyklu
   % generování vektoru s inkrementováním složek po 2
   for j=1:2:31, %vektor s prvky j od 1 po 31 lichých čísel
      j
   end
```

Shrnutí: Tvorba GUI aplikací pomocí GUIDE je obdobná jako v jiných programovacích jazycích. MATLAB nabízí využití paradigmatu OOP, tedy objektově-orientovaného programování. Příklad ukázal v praxi užití podmínek i cyklů. Zvláštní skupinou funkcí jsou tzv. IS funkce vracející logickou hodnotu 0/1 na dotaz vlastnosti. V nových verzích MATLABu nejsou IS funkce podporovány! Což je velká škoda vzhledem k jednoduchosti a exaktnosti použití např. zjištění rovnosti funkcí isempty. Porovnávat např. 2000-složkové vektory už nebude tak jednoduché.

7.2.1 Podmínky, logické funkce a cykly

Zastavme se u několika funkcí u našeho programu. V 1. části jsme již vysvětlili potřebné konstrukce v předchozích příkladech, včetně IF podmínky. Obecná syntaxe je *IF neco ELSE neco jineho END*, tedy

```
   if podminka
```

```
prikaz ci funkce
else
jiny prikaz
end
```

Všimněme si však funkce *isempty*. Ta patří do skupiny logických funkcí, které vracejí na výstupu vždy pouze boolean hodnotu 0/1 dle pravdivosti. Jedná se o dotazovací funkce, v tomto příkladě se ptáme, zda matice $m\times1$ je prázdná. Název je vlastně anglickou otázkou „Is empty?“. Dostaneme na výstupu 0, matice $m\times1$ není nikdy prázdná. Ve 2. části opět použita logická funkce, tentokrát dotaz na rovnost matic, viz komentáře.

V poslední části máme použitý cyklus FOR. I ten je možno v MATLABu běžně používat a blok cyklu musí být ukončen pomocí *end*. V praxi můžeme většinou použít FOR ve 2 zápisech, viz návaznost na **Př. 17**.

=Př. 19==

```
for z=0:50, % vypíše 50 hodnot z
  z
end

for y=0:5:50, % vypíše pouze 10 hodnot – 0,5,10,…,50
  y
end
```

Syntaxe cyklu FOR tedy je *for x=a:s:b*, kde *a* je počáteční hodnota, *s* je krok a *b* koncová hodnota. Vypustíme-li s, nastaví se výchozí krok *s*=1.

Vraťme se k Př. 6, kde jsme generovali tytéž hodnoty. Avšak tam jsme načetli strukturu jako vektor, takže jsme mohli sledovat statistické ukazatele. Pomocí předchozího cyklu FOR v Př. 18 jsme takto generovali pouze izolované hodnoty, které se nechovají jako pole.

Zmiňme se ještě o použití jiného druhu podmínky než IF, podmínky WHILE. Tato podmínka obecně říká, že příkaz (skript) v podmínce se bude vykonávat tak dlouho, dokud podmínka platí. Ukažme si to na praktickém příkladu:

==Př. 20==

```
n = 1; % počáteční hodnota proměnné n
while 2*n < 20 % dokud bude hodnota součinu 2*n menší než 20,
provede se n+1
n = n + 1;
end
n % výpis n
```

Na tomto poměrně jednoduchém příkladu jsme ukázali, že MATLAB běžné používá programovací konstrukce, jako jsou podmínky a cykly. Tyto konstrukce mohou být i vnořené.

Poznámka k logickým is funkcím – v nejnovějších verzích MATLABu již nejsou k dispozici. Tyto funkce se oblibou používaly ke zjištění datových typů či jiné logické podmínky k dalšímu zpracování. Nyní je nutno tyto funkce nahradit např. IF podmínky testující určitou vlastnost.

Shrnutí: MATLAB umožňuje použít veškeré známé programovací konstrukce, jako IF podmínky, či cykly FOR a WHILE. Nezapomínejme na použití kroku u cyklu FOR, viz Př. 19. Syntaxe je jednoduchá a zřejmá, u cyklu FOR zadáváme : jako meze, např. for 1:50.

7.3 Užití debug módu s krokováním

V praxi se často setkáme se situací, kdy náš vytvořený skript, tedy program, nefunguje jak má. Tedy, buď se nespustí vůbec a obdržíme chybovou hlášku ihned v příkazové řádce, nebo, což je horší, aplikace ukončí svůj běh v nějakém bodě, respektive v nějaké části kódu. MATLAB sice také v tomto případě zobrazí chybu, ale často nevíme přesně, čím je to způsobeno. V zásadě rozlišujeme tyto základní chyby:

- syntaktické – chybná syntaxe funkce, příkazu, parametrů
- sémantické – správná syntaxe, avšak program nedělá co má, tedy co chceme my

Vezměme si praktický příklad syntaktické chyby:

==Př. 21==

```
Q1=zers(4,4) % funkce zers neexistuje, chtěli jsme napsat zeros(4,4) pro generování nulové matice
```

Tato chyba bude ihned detekována a MATLAB vypíše chybu na konkrétním řádku. Vezměme si však druhý případ:

```
Q2=zeros(4,4) % vytvoří se nulová matice a přesto program nefunguje
```

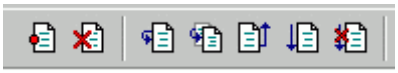
Program nemusí udělat to, co my chceme. To je sémantická chyba, příkaz je napsán správně, ale program nefunguje. Náš program například vyžaduje matici 5×5, nikoli 4×4.

Vezměme si druhý příklad odečítání čísel. Napíšeme správně program na odečítání dvou čísel, ale nechceme záporný výsledek. Program například vyžaduje jako vstup pouze kladná čísla. Což opět je sémantická chyba.

K ošetření těchto chyb MATLAB jako mnohá jiná vývojová prostředí, nabízí tzv. debug mód, tedy režim krokování. V anglické literatuře bývá často používán termín debug mode. Smyslem tohoto krokování je detekovat chybu, při které aplikace nepracuje jak má. MATLAB nabízí tzv. breakpoints, což je řádek v kódu, který můžeme zvolit. Aplikaci spustíme a testujeme, zda se zastaví na tomto bodě či nikoli. Takto můžeme tzv. krokovat a najít sémantickou chybu. V praxi jsem se setkal především s nesprávně použitými datovými typy, chybějící deklarací globální proměnné, atd. Například tak programátor zjistí, že v nějakém kroku se proměnná nastavila na jiný datový typ, byla

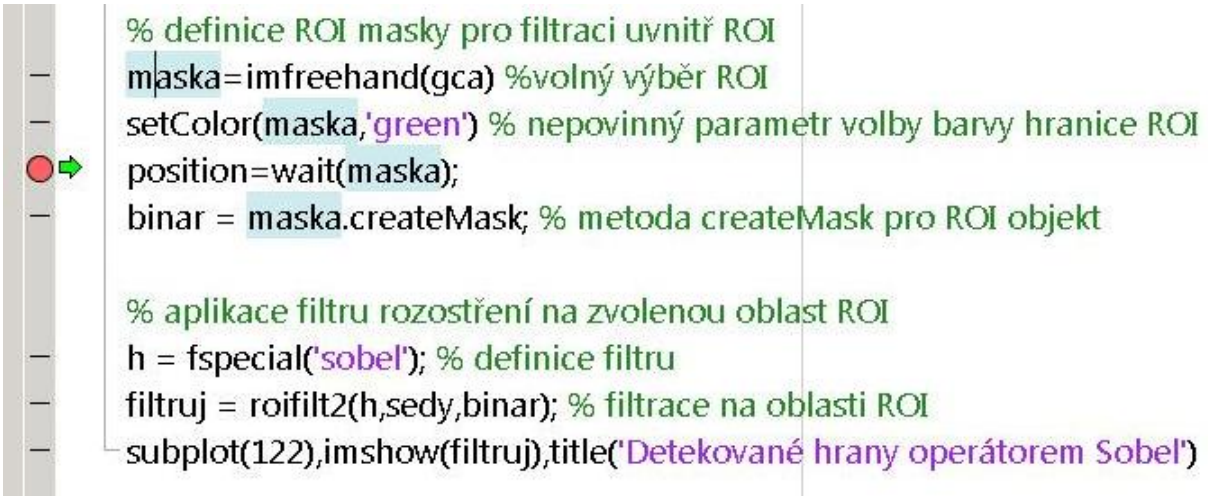
vynulována či jinak změněna. A program v tu chvíli nemá platný vstup a nepracuje. Pomocí krokování je tak možno tyto chyby odhalit.

Nastavení tzv. breakpoints, tedy řádků, kde se má aplikace krokovat, se provádí nejpohodlněji přes panel nástrojů v Editoru.



Obrázek 7.3: Tlačítka pro debug režim krokování

Těmito tlačítky pohodlně můžeme aplikaci pustit v tzv. debug módu s laděním pomocí krokování. Poté aplikaci spustíme. Následující obrázek ukazuje nastavení breakpoints, tedy místo, kde se aplikaci při běhu zastaví a my můžeme ladit krokováním kód, zobrazovat si jednotlivé proměnné.



Obrázek 7.4: Nastavení tzv. breakpoint na určitý řádek v kódu, kde chceme aplikaci ladit

Shrnutí: Odhalení syntaktických chyb ohlídá prostředí samotné, když kontroluje syntaxi. Odhalit sémantické chyby, kdy program sice funguje, ale nedělá co má, je možné pomocí debug módu, tedy krokování. Jako jiné moderní IDE i v Editoru lze použít krokování a nastavit tzv. breakpoints a sledovat po částech chod programu. Častou chybou bývá například chybějící deklarace globálních proměnných. Režim krokování umožní aplikaci spustit do určitého místa, tedy řádku v kódu a sledovat např. datový typ proměnných, atd. Takto můžeme aplikace odladit a vyvarovat se sémantických chyb. Viz příklad s maticí zeros.

7.4 Programování složitějších aplikací

MATLAB nám umožní kromě jednoduchých skriptů programovat ucelené, složité aplikace, které mohou obsahovat stovky proměnných, funkcí, vstupů a výstupů. Je možné pracovat s globálními proměnnými, vlastními funkcemi. K informaci o počtech parametrů slouží funkce *nargin* a *nargout*. S těmi se setkáme při praktickém programování aplikací, kdy definujeme počet vstupních a výstupních parametrů deklarované funkce. Je také možné programovat celé vlastní knihovny funkcí a vytvořit funkční, spustitelnou aplikaci.

Prakticky neexistují limitace, pouze vlastní hardware počítače, tedy především RAM paměť. Pro skutečně náročné aplikace provozované na výpočetních clusterech, nabízí MATLAB přidaný Parallel Computing Toolbox.[2]

Při vlastním programování složitějších aplikací nezapomínejme především na logiku a přehlednost kódu. Dále pozor na přepisování proměnných, o kterém jsme se zmínili hned na začátku! Funkce mohou být volány mezi sebou, může být v jedné funkci definováno stovky proměnných. V případě programování GUI je každý ovládací prvek reprezentován také funkcí s deklarací události (event) a Callback (viz 7.2).

7.5 Návaznost na kompilaci do spustitelného souboru *

Vytvoříme-li nějaký program jako M-soubor, budeme někdy potřebovat tento program předvést na počítači, kde MATLAB není instalovaný. Bereme-li v potaz prostředí operačního systému MS Windows, bude naším cílem vytvořit spustitelný EXE soubor.

Malá poznámka: Kompilace je proces, který z napsaného kódu v nějakém jazyce vytvoří spustitelnou aplikaci, např. EXE soubor či Java aplikaci. To většinou zajišťuje vývojové prostředí (např. NetBeans, Visual Studio, Dev-C++ a další dle programovacího jazyka), které obsahuje i kompiler. Napsaný kód nejprve zkontroluje na výskyt syntaktických chyb jazyka (viz kap. 7.3 a ladění kódu) a zkompiluje do aplikace. Např. v jazyce C# do generuje tzv. MSIL kód, který se poté přeloží pomocí .NET, u Javy zase z tzv. bajtového kódu.

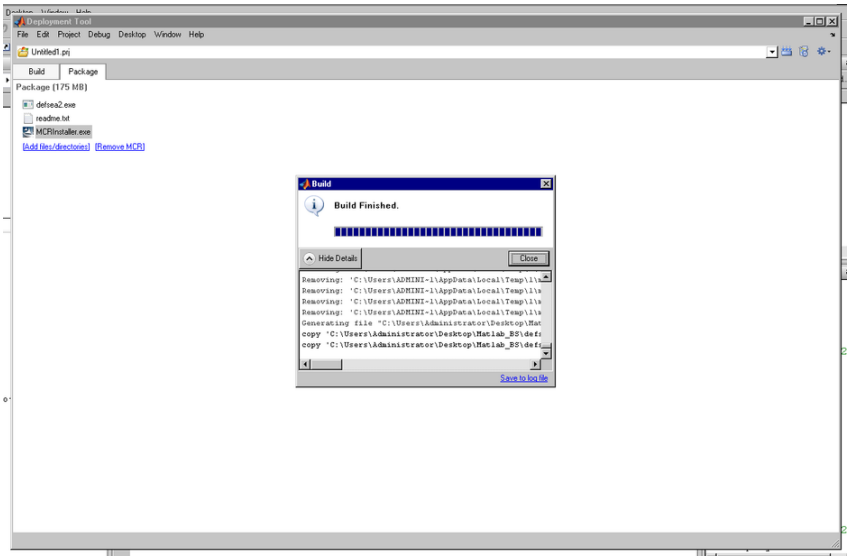
Vytvoření EXE souboru z M-skriptu umožní MATLAB Compiler[3], který však není přímou součástí MATLABu a je distribuován jako Toolbox.

Pomocí nástroje *Deployment Tool* (deploytool) z GUI v prostředí MATLAB vytvořit samostatnou aplikaci EXE, jejíž GUI bylo vytvořeno v GUIDE a funkcionalita je M-skript. Uživatel však potřebuje ke spuštění programu komponentu *MATLAB Compiler Runtime* (MCR)[4], která se přibalí jako instalační balíček k aplikaci. V prostředí Deployment Tool si jednoduše přidáme do projektu všechny soubory, které program využívá (M, FIG a další soubory) a můžeme nastavit, které Toolboxes jsou využitelné v naší aplikaci. Obrázek ukazuje prostředí nástroje *Deployment Tool* s hotovou kompilací projektu.

[2] http://www.mathworks.com/help/toolbox/distcomp/f3-6010.html

[3] http://www.mathworks.com/products/compiler/

[4] http://www.mathworks.com/products/compiler/mcr/index.html



Obrázek 7.5: Ukončení kompilace do EXE aplikace

Dovolím si zde ještě jedno upozornění, které je důležité. Je nutné podotknout, že některé interní příkazy MATLABu nebudou fungovat ve zkompilované aplikaci, tudíž výsledek může být nepoužitelný. Omezení se týká např. Neural Network Toolbox, vybraných příkazů v Image Processing Toolbox a další. Detaily ohledně těchto limitací nalezne čtenář přímo na webu společnosti MathWorks v dokumentu „Limitations About What May Be Compiled“.[5]

Existují ještě verze kompilátoru pro vytvoření .NET či Java aplikace, jež jsou dalšími samostatnými Toolboxy, nazývají se *MATLAB Builder*. Konkrétně to jsou MATLAB Builder NE[6] pro tvorbu .NET aplikací, MATLAB Builder JA[7] pro Java aplikace, které mají výhodu nezávislosti na cílové platformě. Dále pak MATLAB Builder EX[8] pro tvorbu add-in programů do aplikace Microsoft Excel.

Shrnutí: Matlab Compiler umožní jednoduchou cestou M-skript zkompilovat do běžného EXE souboru. Nezapomeňme, že není přímou součástí MATLABu! Jedná se o modul jako kterýkoli jiný Toolbox. Existují ještě speciální kompilery pro .NET, Javu a Excel plug-iny. Některé funkce v M-skriptu nemusí ve zkompilovaném programu fungovat! Doporučuji si projít výše zmíněný dokument o limitacích. Na cílovém počítači musí být instalován Matlab Compiler Runtime (MCR), který se přibalí ke kompilovanému programu.

[5] http://www.mathworks.com/help/toolbox/compiler/br2cqa0-2.html

[6] http://www.mathworks.com/products/netbuilder/

[7] http://www.mathworks.com/products/javabuilder/

[8] http://www.mathworks.com/products/matlabxl/

8 Import a export dat, spolupráce s dalšími programy

Důležitou vlastností MATLABu je jeho otevřenost a možnosti spolupráce s dalšími programy. MATLAB umožní ukládat výstupy (hodnoty, grafy) do různých formátů a naopak umí importovat data z jiného zdroje. Nejčastějším případem bude export/import do/z textových formátů a tabulkových editorů. MATLAB je bohatě vybaven pro tyto úkoly.

Mezi časté potřeby se vyskytne načtení textového souboru či souboru z tabulkového editoru. MATLAB tohle vše umí. Další příjemnou vlastností je možno uložit obsah Figure okna do interního formátu FIG, ale především jako obrázek do libovolného bitmapového formátu. Pro příklad: mějme někde uložený (pro jednoduchost předpokládejme cestu C:/) XLS soubor s daty, například hodnoty krevního tlaku měřené v čase. Chceme tyto hodnoty načíst do MATLABu a uložit do proměnné *tk*. K tomu poslouží funkce *xlsread*.

```
tk=xlsread(´c:/soubor.xls´);
```

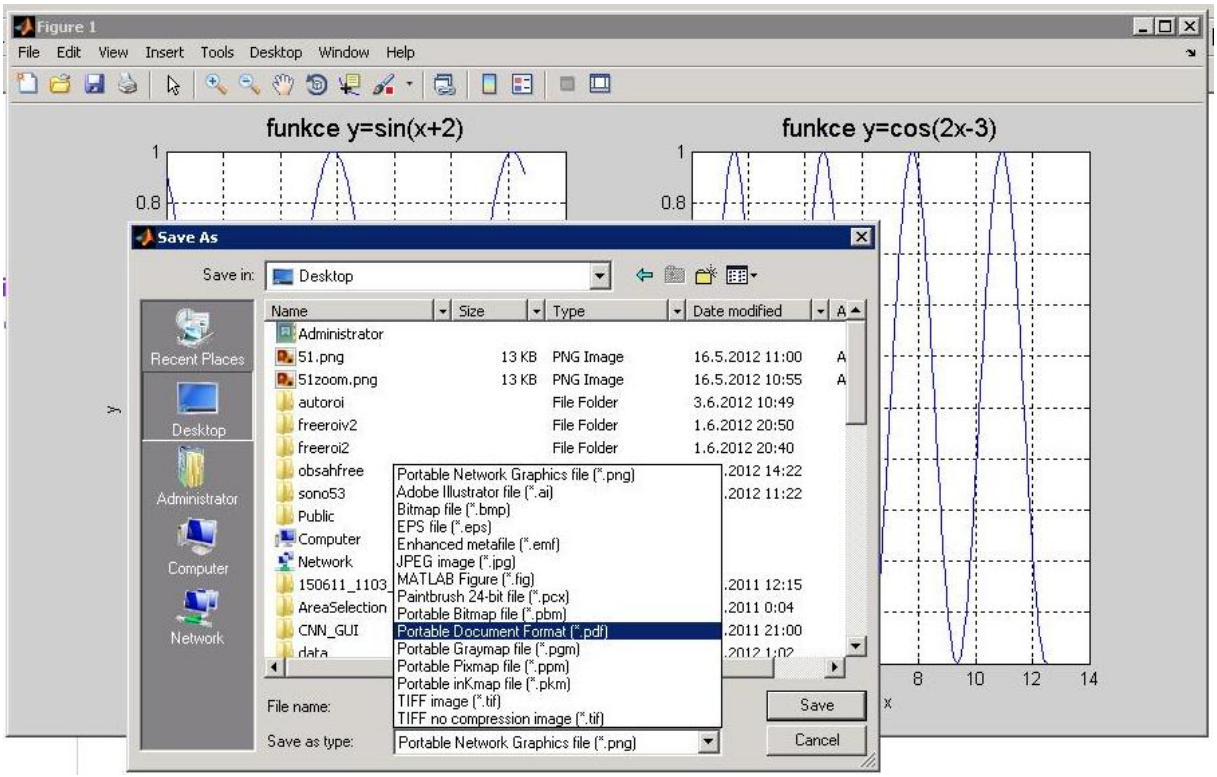
MATLAB zobrazí data. V případě neexistujícího souboru obdržíme chybovou hlášku. Cesta musí být v uvozovkách dle příkladu.

Opačně, využijme vektoru *A* z **Př. 3** a chceme tento vektor uložit do textového souboru. To poskytuje funkce *fwrite* a *dlmwrite* (zápis hodnot do ASCII textového souboru, hodnoty oddělené zvoleným oddělovačem). Zápis do XLS souboru zařídí funkce *xlswrite*. Nezadáme-li do parametrů jinak, zápis se provede od buňky A1 do prvního řádku. Uložení vektoru *A* z **Př. 3**:

```
xlswrite('c:/vystupA.xls',A) % do XLS souboru
dlmwrite('c:/vystupA.txt', A, ';') % zápis do TXT souboru s
oddělovačem hodnot ;
```

Výchozím oddělovačem je čárka (,).

Obsah Figure okna lze taktéž uložit. Využijme předchozího případu dvou funkcí a tento výstup chceme uložit. Po stisku tlačítka Save (ikona diskety) na panelu nástrojů okna Figure, máme možnost obsah uložit v různých grafických formátech, interním FIG formátu či dokonce do PDF. Velmi užitečné, zpracováváme-li mnoho dat a chceme výstupy ukládat.



Obrázek 8.1: Možnosti exportu grafiky z Figure okna do PDF a dalších formátů

V rámci spolupráce MATLABu s dalšími programy existuje mnoho dalších rozšíření a doplňků. Zajímavý je však *Maple Toolbox for MATLAB*[9] pro prostředí Maple. Tento Toolbox umožní skvělé propojení numericky orientovaného MATLABu a se symbolickými výpočty a DTP tvorbou v prostředí Maple. Volně dostupný matematický nástroj Scilab[10] taktéž umí číst M-skripty a některé jednodušší skripty i správně spustit.

Je tak vidět, že MATLAB je dobře vybaven ke spolupráci s dalšími programy a k přenosu dat, jejich importu a exportu.

Shrnutí: MATLAB není uzavřený systém. Díky tomu je možné načítat externí data, např. z textových souborů, souborů XLS, obrázky a jiné soubory. Naopak MATLAB umožní výstupy ukládat v různých formátech. Zajímavý je také Maple Toolbox for MATLAB, který propojí špičkové matematické nástroje Maple a MATLAB. Samozřejmostí je uložení výstupu do souboru z Figure okna a to i v PDF.

[9] http://www.maplesoft.cz/toolbox-for-matlab

[10] http://www.scilab.org/

9 Další rozšíření systému MATLAB – moduly Toolbox a Simulink *

Jak již bylo řečeno na začátku, systém MATLAB je plně modulární a díky modulům, tzv. Toolbox, je možno rozšířit MATLAB o tisíce nových funkcí a možností. V nabídce najdeme desítky těchto Toolbox[11] modulů, mezi ně patří např. modul pro zpracování obrazu a videa, umělé neuronové sítě, ekonomiku, statistiku, symbolickou analýzu, aerotechniku, RC modely, bioinformatiku, tvorbu spustitelných aplikací, práce s databázemi a mnohé další oblasti využití. Seznam všech rozšíření Toolbox je k dispozici na webu výrobce.

9.1 Simulink

Dalším rozšířením je Simulink[12]. Je nadstavbou systému MATLAB, která umožní tvorbu fyzikálních, dynamických systémů s možností 2D/3D vizualizace a animace. Jedná se o výkonné prostředí pro simulace fyzikálních dějů, elektrotechnických schémat, zpracování videa, analýzu a zpracování signálu a mnohá další odvětví. Simulink je orientován nikoli výpočetně, numericky, ale na vizuální modelování těchto složitých dynamických systémů. Základem modelování jsou tzv. blocksety pro určitou oblast, bloky reprezentující elementární dynamické systémy. Propojením signálových vstupů a výstupů těchto bloků vznikají modely složitých systémů. Skupinu bloků lze uzavřít do subsystému a určit externí vstupy a výstupy této skupiny. Dále lze pracovat s takovouto skupinou jako se základním blokem. Simulink je vynikající nástroj pro ty, kteří potřebují propojit výpočetní výkon systému MATLAB s možnostmi 2D/3D vizualizace a animace dynamických systémů, od regulační techniky po fyzikální dynamické systémy či např. hydrauliku.

I pro Simulink existují další specifická rozšíření, jako Simscape, SimHR, SimMechanics, Aerospace Blockset, SimRF a další dle oblasti využití.

Díky systému MATLAB a jeho desítkami modulárních rozšíření se již léta provádí nejrůznější vědecké výzkumy, programují aplikace, které jinak by byly realizovatelné daleko složitěji.

[11] http://www.mathworks.com/products/

[12] http://www.mathworks.com/products/simulink/

10 Image Processing Toolbox *

Jedním z často užívaných Toolbox modulů je modul pro zpracování obrazu, Image Processing Toolbox (dále jen IPT). Nabízí širokou škálu efektivních nástrojů, jednak těch, které můžeme najít i v běžných grafických editorech, tak pokročilých nástrojů k analýze obrazu.

Samotný MATLAB nedokáže žádný obrázek ani načíst, takže chceme-li pracovat s obrazovými daty, je IPT nutností. Podívejme se nejprve stručně, na co je IPT zaměřen a co poskytuje:

- **čtení mnoha typů rastrových i vektorových grafických formátů**
- **podpora HDR a DICOM obrazů**
- **základní úpravy obrazu, konverze mezi barevnými hloubkami**
- **binární prahování šedotónových obrazů**
- **lineární i nelineární filtrace, DFT, IDFT, filtry k detekci hran**
- **analýza obrazu a jeho statistika**
- **ROI-based operace – operace na definované oblasti zájmu (ROI) v obraze**
- **interaktivní nástroje pro úpravy, zobrazení mřížky pixelů**
- **morfologické operace založené na geometrii obrazu**
- **výpočty korelací mezi obrazy, metody rekonstrukce obrazu**
- **zobrazení histogramu a jeho úpravy, včetně HDR, RGB rozkladu**
- **extrakce kanálů z obrazu, konverze barevných prostorů**
- **geometrické transformace – ořez, translace, rotace**
- **čtení metadat z obrazu**
- **a mnohé další**

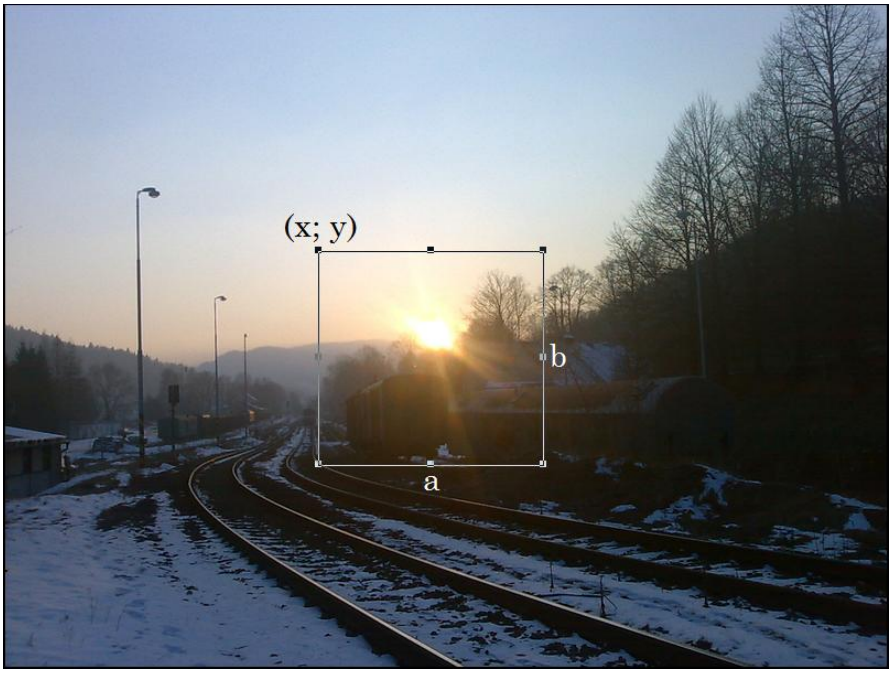
IPT pokrývá skutečně širokou škálu možností pro zpracování obrazu. Poradí si navíc i se specifickými formáty, takže není například problém analyzovat medicínská data ve formátu DICOM.

Připomeňme fakt z úvodní kapitoly 2.1, že vše v MATLABu je matice. Ani obraz není výjimkou. Jakýkoli načtený obraz je již samozřejmě v diskrétní formě, tedy reprezentován maticí pixelů, kde každý pixel je vyjádřen buď vektorem RGB, hodnotou intenzity či pouze binární hodnotou 0/1 u binárního obrazu (např. ROI masky). Ukažme si na příkladě funkce *imcrop*, která vyřeže z obrazu určitou část.

=Př. 22==

```
   % načtení libovolného obrázku pomocí běžného dialogu Otevřít a
uložení do proměnné IM1
   [filename, user_canceled] = imgetfile;
   IM1=imread(filename);

   % zobrazení obrázku ve Figure s titulkem
   figure, imshow(IM1), title('Toto je původní načtený obrázek
určený pro ořez');

   % ořezání obrazu na 100x100 pixelů s levým horním rohem (50; 50)
```

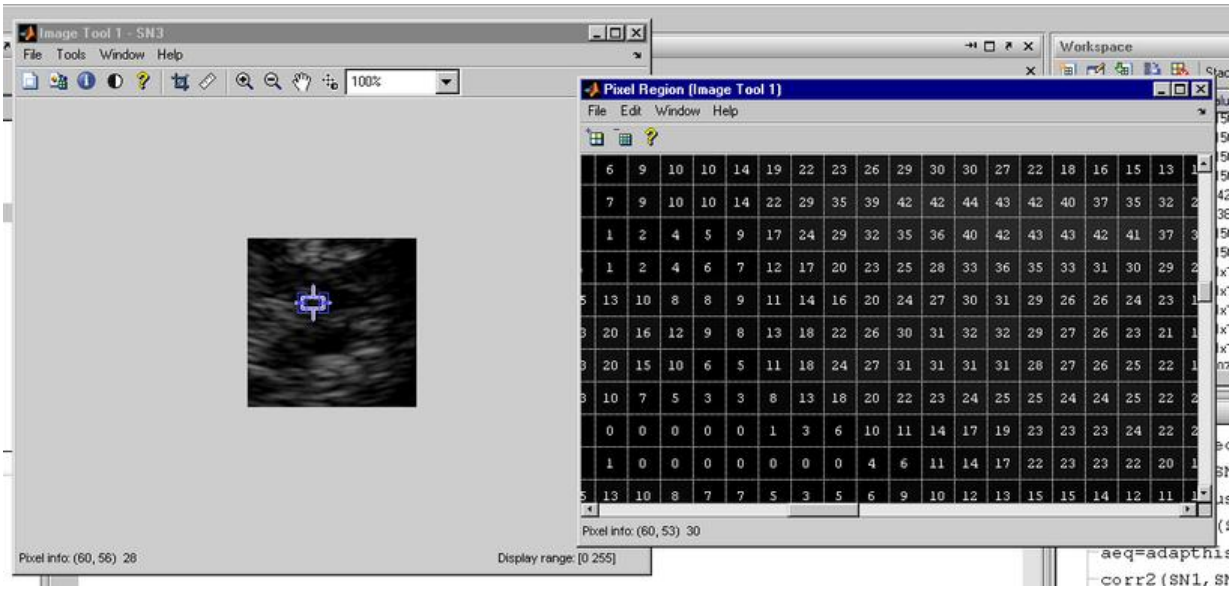
```
   IM2=imcrop(IM1,[50 50 100 100]); % ořez obrazu IM1 a parametry
ořezu
   figure, imshow(IM2), title('Ořezaný obrázek dle parametrů funkce
imcrop');

   Ve funkci imcrop vystupuje 4-složkový vektor, kde první 2 složky
určují pozici horního levého rohu ořezu a další dvě délku a šířku
v pixelech. Obecná syntaxe je tedy

   imcrop(IM1,[x y a b])
```



Obrázek 10.1: Ořez obrazu s parametry rozměru a počátku ořezu

Ponecháme-li funkci *imcrop* bez parametrů, můžeme ručně zvolit obdélník výřezu.

Nejedná se o nic jiného, že z matice reprezentující rozlišení obrazu $m \times n$ byla vyřezána určitá část dle vektoru funkce *imcrop*. Stále se však v matematické podstatě jedná o výřez prvků z obrazové matice. Takže stále platí to, že vše je matice, tedy i obraz v diskrétní podobě. Podívejme se na příklad ultrazvukového snímku části mozkového kmene se zobrazením hodnot pixelů v matici obrazu. Zpracování medicínských dat je možné díky plné podpoře DICOM formátu užívaného v moderní diagnostické medicíně.



Obrázek 10.2: Obraz reprezentovaný jako diskrétní matice pixelů s hodnotami intenzity

Ke zpracování obrazu před digitalizací slouží jiný Toolbox a tím se nebudeme vůbec zabývat. Pro IPT je vstupem diskrétní podoba obrazu.

10.1 Praktický příklad využití IPT

V následujícím praktickém příkladě na využití IPT si ukážeme filtraci oblasti zájmu ROI na načteném ultrazvukovém obrazu DICOM. Příklad dobře ilustruje možnosti použití MATLABu s IPT ke zpracování medicínských dat. Nebudeme se však vůbec zabývat tím, jaká data jsou zpracovávána. Postup bude následující:

1. **načtení libovolného DICOM obrazu a jeho zobrazení ve Figure**
2. **ořez dle zadaných parametrů velikosti okna**
3. **převod tohoto ořezaného obrazu z RGB do 8-bit, pokud bude nutné (funkce isgray)**
4. **filtrace daného obrazu a konečného zobrazení filtrovaného okna**

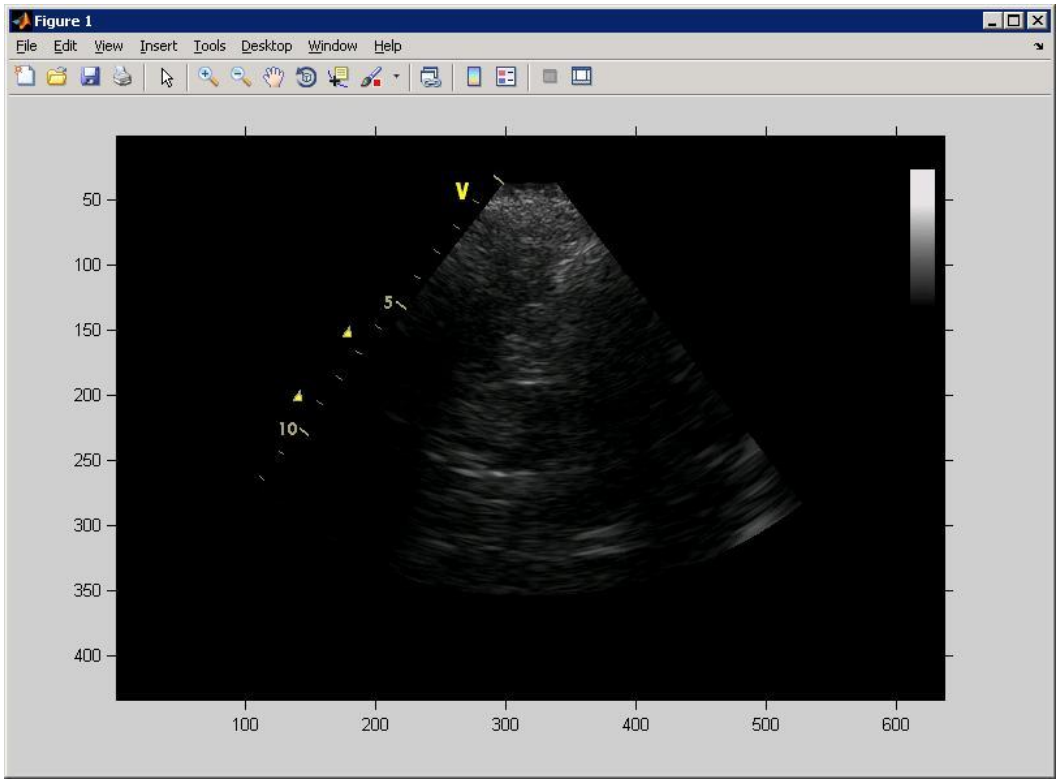
V praxi tak ukážeme několik funkcí najednou, které můžeme použít v rámci IPT nejen pro medicínská data.

==Př. 23==

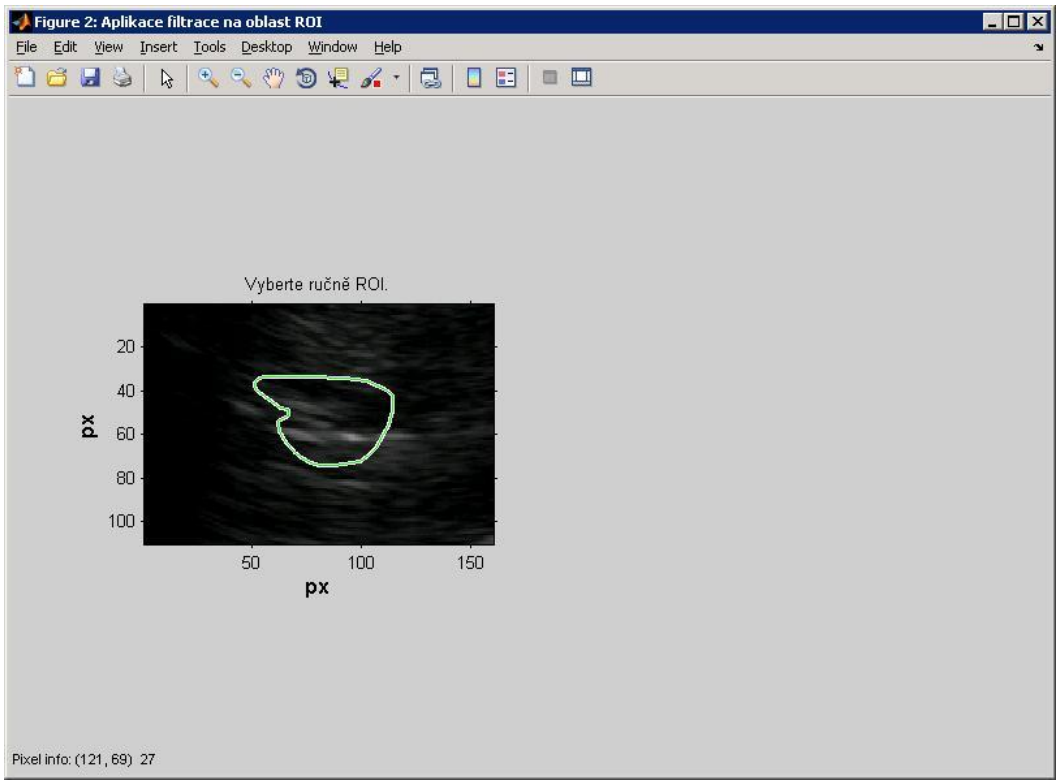
```
function pushbutton1_Callback(hObject, eventdata, handles)
% načtení DICOM obrazu ze souboru
[filename, user_canceled] = imgetfile;
imgor=dicomread(filename);
plk=figure; imshow(filename);
info = dicominfo(filename); % metadata
xdr=info.Width; % šířka obrazu
ydr=info.Height; % výška obrazu
```

```
   % zobrazení do Figure s popisky os, názvu a záhlaví okna dle Name
   GH=figure('Name','Aplikace filtrace na oblast ROI');
   imshow(imgor, 'InitialMagnification',100,'Border','loose'),
title('Tento obraz bude ořezán','fontweight','b'),
ylabel('pixely','fontsize',12','fontweight','b'),
   xlabel('pixely','fontsize',12,'fontweight','b')

   % ořezání obrazu do okna o zadané velikosti z extrahovaných
metadat
   img = imcrop(imgor,[200 200 xdr/4 ydr/4]);

   % převod z RGB do 8-bit pomocí logické funkce isgray
     if isgray(img)==1 %RGB input test

sedy=img;subplot(121),imshow(sedy,'InitialMagnification',100),title(
'Vyberte ručně oblast
ROI.'),xlabel('px','fontsize',12','fontweight','b'),ylabel('px','fon
tsize',12','fontweight','b'), impixelinfo, zoom
     else
sedy=rgb2gray(img);subplot(121),imshow(sedy,'InitialMagnification',1
00),title('Vyberteručně
ROI.'),xlabel('px','fontsize',12','fontweight','b'),ylabel('px','fon
tsize',12','fontweight','b'), impixelinfo, zoom
   end

   % definice ROI masky pro filtraci uvnitř ROI
   maska=imfreehand(gca) %volný výběr ROI
   setColor(maska,'green') % nepovinný parametr volby barvy hranice
ROI
   position=wait(maska);
   binar = maska.createMask; % metoda createMask pro ROI objekt

   % aplikace filtru rozostření na zvolenou oblast ROI
   h = fspecial('sobel'); % definice filtru
   filtruj = roifilt2(h,sedy,binar); % filtrace na oblasti ROI
   subplot(122),imshow(filtruj),title('Detekované hrany operátorem
Sobel')
```

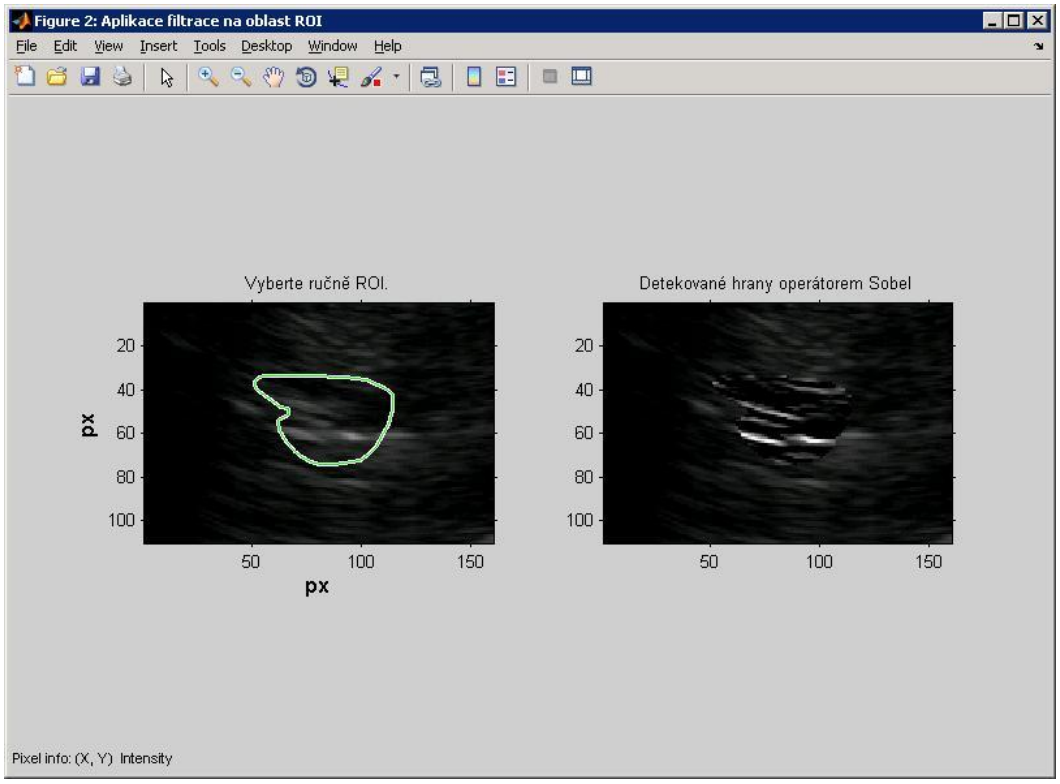
Následující obrázky zachycují načtený celý obraz, poté výběr ROI na výřezu a výslednou filtraci detekcí hran pomocí Sobelova operátoru.



Obrázek 10.3: Načtený DICOM obraz



Obrázek 10.4: Ruční výběr ROI



Obrázek 10.5: Aplikovaná filtrace na zvolenou ROI oblast obrazu

Tento krátký exkurz k užití IPT ukázal v praxi, jakým způsobem se dá MATLAB využít s instalovaným IPT ke zpracování medicínských obrazů. To je samozřejmě jen malá ukázka toho, co je možné s IPT použít.

Definice ROI není možná pouze pomocí volného kreslení funkcí *imfreehand*, ale také pomocí předpřipravených tvarů ROI, jako je polygon, kružnice, obdélník a další, kde se ROI vytvoří pomocí binární masky.

Obecně se filtrace v ROI provádí pomocí funkce *roifilt2*. Ta umožní 2D filtrace na obraze, resp. na vybraném ROI, viz předchozí příklad. Může se jednat o běžné operace, například zvýšení jasu, nebo použít některou z předpřipravených filtrací pomocí funkce *fspecial*. Obecná syntaxe ROI filtrace je tak, jak byla uvedena v předchozím případě, tedy

```
h = fspecial('sobel'); % definice filtru
filtruj = roifilt2(h,sedy,binar); % filtrace na oblasti ROI
```

Tedy obecná syntaxe filtru: *roifilt2(filtr,na_obraz,maska_roi).*

Funkce *fspecial* nabízí filtrace pro detekci hran, vyhlazení obrazu, rozostření, průměrování a další. Obecně mluvíme o filtracích na konvolučním jádře, které je definováno jakou operace konvoluce na původním obraze a tomto jádře.

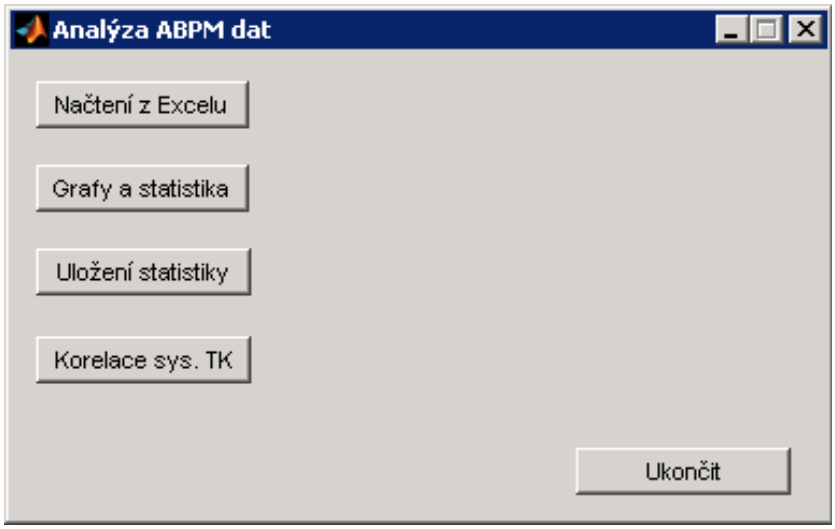
Shrnutí: Image Processing Toolbox je výkonným modulem pro zpracování obrazu a jeho analýzu. Mezi hlavní přednosti patří široká škála podporovaných grafických formátů, včetně medicínských DICOM dat. Nabízí základní i pokročilé nástroje pro retuše, analýzu a filtrace digitálního obrazu. Příklady, které byly součástí kapitoly, ukázaly možnosti výřezu obrazu a zpracování medicínských

DICOM obrazů na vybrané ROI části. Připomeňme, že také obraz je v MATLABu (a i ve své podstatě) reprezentován jako diskrétní reálná matice mxn, kde rozměr určuje počet pixelů, tedy rozlišení obrazu. Výřez tak není ničím jiným než maticovou operací.

11 Praktický příklad užití MATLABu

V této poslední kapitole si ukážeme praktický příklad použití MATLAB k analýze medicínských dat. Vstupem budou diskrétní hodnoty krevního tlaku[13], měřené metodou ABPM[14]. Tento vstupní vektor je časovou řadou hodnot krevního tlaku. Vezmeme náhodně vybraného pacienta a u něho budeme zkoumat systolický TK, diastolický TK a tepovou frekvenci. Tato analýza kardiovaskulárních dat je užitečná pro sledování případných defektů, jako je arytmie či hypertenze. Metoda ABPM je využitelná pro analýzu krevního tlaku. Vstupním souborem je 96 měření v průběhu 24 hodin s časovým intervalem 15 minut, data tak tvoří diskrétní časovou řadu.

Vstupní data budou načtena z XLS souboru jako tři vektory – *sys*, *dia*, *tep*. Poté budou tyto hodnoty vykresleny do grafu a zjištěny statistické ukazatele, jako je minimum, maximum, průměrná hodnota a směrodatnou odchylku. Na konec načteme data systolického TK u jiného pacienta a provedeme výpočet korelačního koeficientu. Téměř všechny použité funkce jsme již zmínili v předchozích příkladech. Celý program bude realizován jako jednoduchá aplikace s 5 tlačítky. Rozebereme program s komentáři.



Obrázek 11.1: Ukázková aplikace s jednoduchým GUI

Popíšeme nyní funkce jednotlivých tlačítek, každé z nich je deklarováno jako v příkladu z kapitoly 7.1.

Načtení z Excelu – načtení vstupních vektorů dat *sys*, *dia* a *tep*

```
   function pushbutton1_Callback(hObject, eventdata, handles)
   % načtení vstupních vektorů TK z XLS souboru a uložení do
proměnných
   sys=xlsread('c:/systk.xls')
   dia=xlsread('c:/diatk.xls')
   tep=xlsread('c:/tep.xls')
```

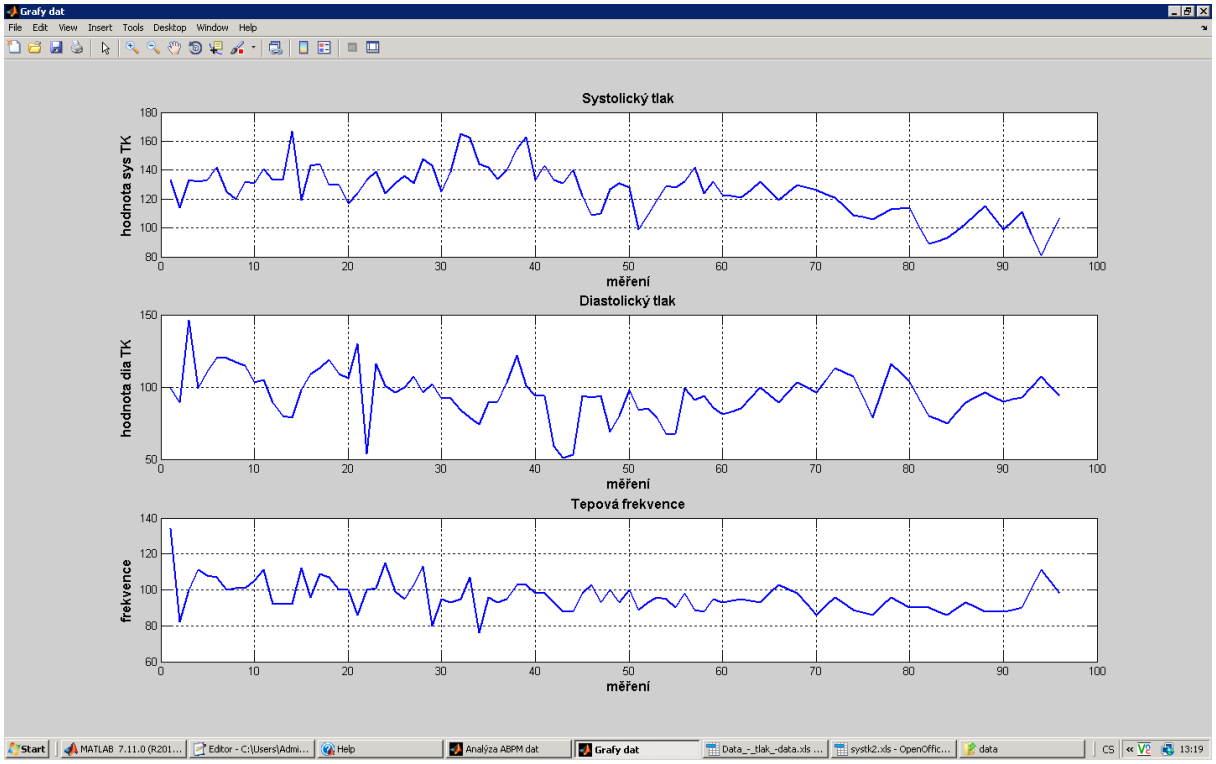
[13] http://krevni-tlak.info/

[14] http://zdravi.e15.cz/news/check-pro?id=151570&seo_name=priloha-lekarske-listy

Grafy a statistika – vykreslení těchto hodnot do grafu a výpočet statistických dat

```
   function pushbutton2_Callback(hObject, eventdata, handles)
   global sys dia tep % nutno deklarovat globálně pro další užití

   figure('Name','Grafy dat'), % záhlaví Figure okna s definovaným
názvem
   subplot(311), plot(sys, 'LineWidth',2), title('Systolický
tlak','fontsize',12,'fontweight','b'),
xlabel('měření','fontsize',12,'fontweight','b'), ylabel('hodnota sys
TK','fontsize',12,'fontweight','b'), grid on

   % statistické veličiny a vykreslení dalších grafů pomocí subplot
   min(sys)
   mas=max(sys)
   avs=mean(sys)
   sos=std(sys)

   subplot(312), plot(dia, 'LineWidth',2), title('Diastolický
tlak','fontsize',12,'fontweight','b'),
xlabel('měření','fontsize',12,'fontweight','b'), ylabel('hodnota dia
TK','fontsize',12,'fontweight','b'), grid on
   min(dia)
   mas=max(dia)
   avs=mean(dia)
   sos=std(dia)

   subplot(313), plot(tep, 'LineWidth',2), title('Tepová
frekvence','fontsize',12,'fontweight','b'),
xlabel('měření','fontsize',12,'fontweight','b'),
ylabel('frekvence','fontsize',12,'fontweight','b'), grid on
   min(tep)
   mas=max(tep)
   avs=mean(tep)
   sos=std(tep)
```

Ukázka vykreslených grafů



Obrázek 11.2: Vykreslené hodnoty TK a tepové frekvence z ABPM dat

Možnost uložit grafy je pod ikonkou diskety ve Figure okně, viz kap. 8 o exportu dat.

Uložení statistiky – možnost uložit statistická data do CSV souboru

```
   function pushbutton3_Callback(hObject, eventdata, handles)
   global mis mas avs sos mid mad avd sod mite mate avte sote sys
dia tep
   antlak=[mis mas avs sos mid mad avd sod mite mate avte sote]; %
vektor hodnot

   % volání systémového dialogu Save As
   [filename, pathname] = uiputfile('*.csv','Uložení do CSV');
   fullname = fullfile(pathname,filename);

   % pomocí funkce csvwrite uloží do CSV souboru stat. hodnoty
   csvwrite(fullname, antlak)
```

Korelace sys. TK – korelační koeficient systolického TK mezi 2 pacienty

```
   function pushbutton4_Callback(hObject, eventdata, handles)
   global sys
   sysv=[sys];
   sys2=xlsread('c:/systk2.xls')
   sys2
   sysv2=[sys2];
```

```
korel=corr(sysv, sysv2)
msgbox('Korelační koeficient je:' )
```

Ukončit – ukončí aplikaci

```
function pushbutton5_Callback(hObject, eventdata, handles)
delete(handles.figure1)  % smaže vše a ukončí aplikaci
close all % zavře všechny okna
```

V tomto praktickém příkladu jsme ukázali, jak je možné MATLAB využít k analýze kardiovaskulárních dat TK a tepové frekvence. Samozřejmě mohli bychom načítat soubor libovolný pomocí dialogového okna (jak bylo ukázáno při ukládání CSV výstupu), provést další statistickou analýzu, atd. Možností je v podstatě nekonečně, co MATLAB může nabídnout. Příklad ukázal praktické použití bez dalších modulů, ukázal také vytváření GUI v praxi. Každé tlačítko obsluhuje událost kliku, ve které jsou definované příkazy a funkce, které se vykonají.

Dovolím si poznámku o globálních proměnných, o kterých sice již byl řeč v kapitole 3.4, ale teprve nyní jsme ukázali jejich význam v praxi. Důležité pravidlo zní – globální proměnnou musíme deklarovat globálně již v prvním výskytu. V praxi to znamená, kdybychom u tlačítka pro uložení měli napsanou globální deklaraci, ale u předchozího nikoli, program nebude fungovat. To si můžeme vyzkoušet, odstraníme-li řádek definice global.

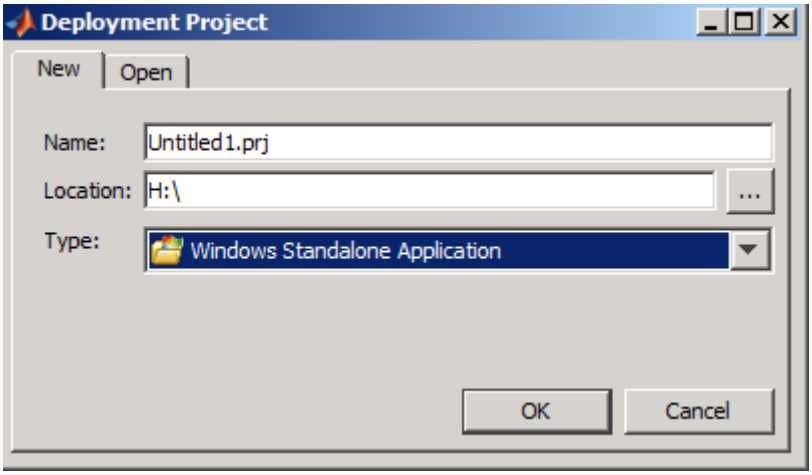
Tlačítko, které ukončí aplikaci, ve své události nese 2 příkazy – *delete(handles,figure1)* a *close all*. První vymaže obsah Figure a poté zavře všechna okna. Takto je aplikace ukončena.

Shrnutí: Kapitola s tímto praktickým příkladem ukázala použití MATLABu pro analýzu dat, které byly načteny externě. MATLAB umí spolupracovat s mnoha externími formáty dat a umožní uživateli je zpracovávat. Taktéž jsme v praxi viděli tvorbu jednoduchého GUI s 5 tlačítky a v kódu jsme ukázali, jakým způsobem se pracovat s ovládacími prvky jako Callback funkce. Pozor na globální proměnné! Je nutno je globálně deklarovat od prvního výskytu, pokud víme předem, že se budou používat např. v události jiného tlačítka! Jinak bychom poté obdrželi prázdný datový typ této proměnné. Globální proměnné uchovají hodnotu proměnné při prvním jejím užití.

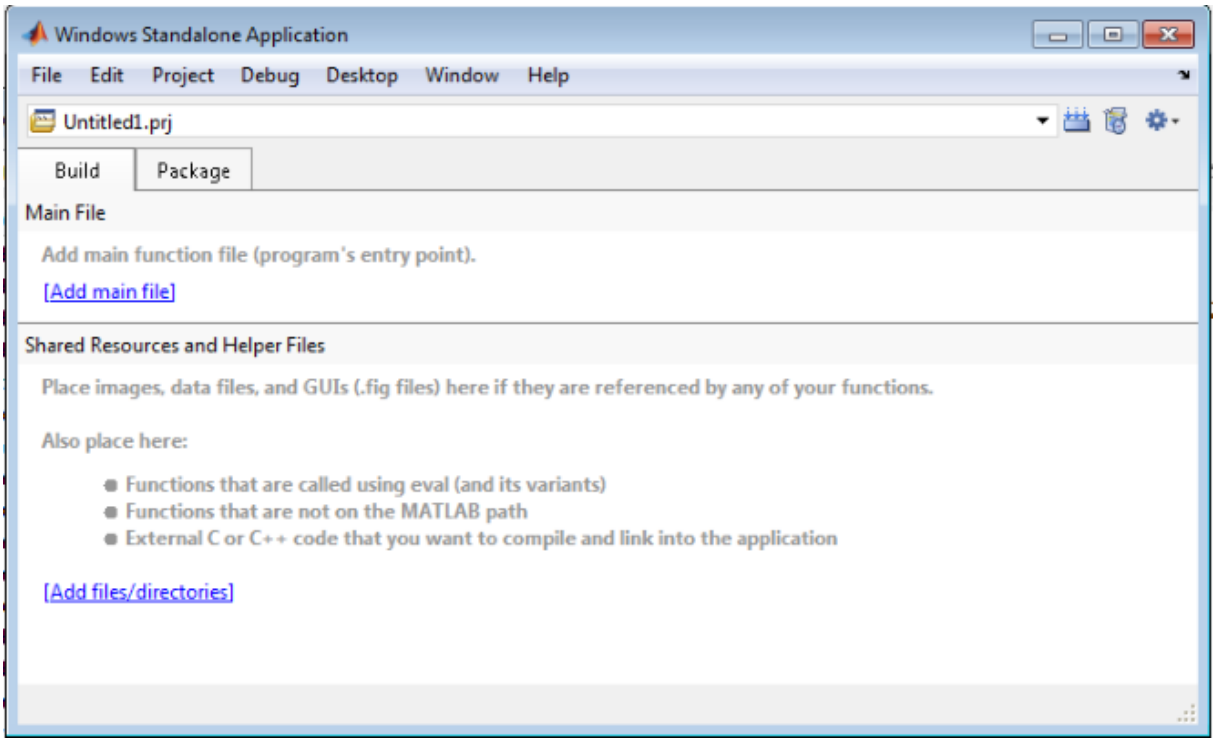
11.1 Kompilace do EXE souboru *

Jak bylo popsáno v kap. 7.5, je možno (je-li dostupný) pomocí *Matlab Compileru* náš program nechat zkompilovat do spustitelného souboru pomocí *Deployment Tool*.

Prvním krokem je spuštění kompilačního nástroje příkazem *deploytool* v příkazové řádce. Poté nás vybídne k vybrání hlavního souboru, tedy náš M-soubor (není-li uložen, MATLAB bude vyžadovat jeho uložení). Poté je možno nastavit, které moduly Toolbox jsou v aplikaci použity, klidně je možno v našem případě všechny vypnout. MATLAB používá svůj interní kompilátor, v nastavení však můžeme zvolit i jiný externí. Připomínám opět, že Matlab Compiler není standardní součástí. V prvním kroku kompilace volíme typ projektu Windows Standalone Application a projekt pojmenujeme. Následující obrázek zachycuje krok výběru projektu a jeho jména, poté volbu hlavního souboru pro typ projektu Windows Standalone Application (EXE aplikace):



Obrázek 11.3: Volba názvu projektu, umístění a typu



Obrázek 11.4: Přidání hlavního souboru do kompilace a vedlejších

Kliknutím na *Add main file* zvolíme M-soubor ke kompilaci. V některých případech budou vyžadovány další soubory, knihovny, externí kódy, atd. To řeší *Add files/directories*. V našem případě postačí plně první volba.

Shrnutí: Kompilaci do EXE souboru umožní Matlab Compiler, který však není součástí MATLABu samotného. MATLAB užívá interní kompilátor a je možno nastavit, které moduly jsou v aplikaci využívány. Hlavním kompilovaným souborem je M-skript, další je možné přidat dle potřeby do kompilace.

12 Užitečné funkce

Na závěr se seznámíme s několika užitečnými funkcemi, které zpříjemní naši práci v MATLABu a bude také efektivnější. Nejde o výpočetní příkazy, ale pomocné příkazy a funkce. Jak rychle vyvolat nápovědu příkazem help, bylo popsáno v první kapitole, je tedy možné použít nápovědu ke konkrétnímu příkazu.

Mezi další užitečné funkce patří:

- ***clc*** **– rychle vymaže vše z příkazového řádku MATLABu**
- ***guide*** **– vyvolání editoru GUI**
- ***exit*** **nebo** ***quit*** **– ukončení MATLABu**
- ***whos*** **– vypíše všechny aktuálně uložené proměnné ve Workspace vč. datového typu**
- ***clear*** **– vymaže vše z Workspace; s příkazem opatrně, zruší všechny deklarace**
- ***ver*** **– rychlé zobrazení nainstalované verze MATLABu, knihoven a Toolboxů**
- ***demo*** **– přístup k demo ukázkám funkcí**

13 Na co nezapomínat…

V textu jsme zmínili některá upozornění a důležité poznámky, na něž není radno zapomínat:

- syntaxe je case-sensitive – tedy rozdíl mezi proměnnou x a X
- každá proměnná je matici se svým rozměrem a datovým typem
- nechcete-li okamžitě vypočíst hodnotu a jen uložit proměnnou, na konci nutný ;
- u součinu nutno použít operátor – 2*x+2 namísto 2x+2
- pozor na přepis proměnných! MATLAB nevaruje a přepíše rovnou
- textové popisky, názvy, cesty k souborům – zkrátka vše co není proměnnou je v uvozovkách 'takto'
- nezaměnit přiřazovací operátor = za relační rovnost ==

Závěrem

Tento sylabus měl za úkol seznámit uživatele s prostředím MATLAB, které je chápáno jednak jako vynikající matematický software pro nejširší použití, ale také jako výkonné programovací prostředí s možností tvorby aplikací s GUI.

Jednotlivé kapitoly jsou členěny tak, aby text byl určen začátečníkům, tak pokročilejším uživatelům. Čtenář se tak seznámí na úvod s prostředím systému MATLAB, porozumění syntaxi a chápání prvků jako matice. Dále se seznamuje s operátory, maticemi, deklaracemi, poté s programovacími strukturami, které se používají stejně jako v „klasických“ programovacích jazycích. Na závěr ukazuje taktéž jak vytvořit celou funkční aplikaci s GUI na praktických příkladech.

Text je provázán příklady, které se vždy vážou k aktuální kapitole a jsou uvedeny poznatky ohledně správné syntaxe a také upozornění, jaká chyba by mohla vzniknout při jiném zadání, na pohled byť stejného příkazu. Na praktických příkladech je ukázána syntaxe, včetně komentářů v kódu. Nechybí ani popis možností vykreslování 2D a 3D grafiky včetně ukázek a vysvětlení funkce Figure oken.

Samostatná kapitola je věnována Image Processing Toolbox, kde na praktických příkladech ořezu obrazu a filtrace ROI medicínských obrazů ukazuje možnosti využití modulu pro zpracování obrazu.

Text je zaměřen na praktické použití systému MATLAB s příklady provázanými k jednotlivým částem. Student není zatěžován hutným textem, ale spíše především upozorněn na syntaxi, jak se daná funkce či příkaz používá, komentované příklady vše ukazují v praxi. U vybraných příkladů jsou i výpisy výstupu, zejména tam, kde by nemusel být výpis zcela zřejmý či se ukazuje nějaká konkrétné vlastnost zápisu. Každá kapitola je ukončena shrnutím poznatků a důležitých připomínek, na co nezapomínat. Pro zájemce o hlubší poznání systému doporučuji níže uvedené webové zdroje či literaturu, která je uvedena na samém závěru textu.

Seznam doporučené literatury

Doňar B., Zaplatílek K.: MATLAB pro začátečníky, BEN, 2003, 2. aktualizované vydání, ISBN 80-7300-175-6.

Doňar B., Zaplatílek K.: MATLAB – tvorba uživatelských aplikací, BEN, 2004, ISBN 80-7300-133-0.

obecný úvod

http://www.humusoft.cz/produkty/matlab/matlab/

http://upload.wikimedia.org/wikibooks/cs/d/d5/Matlab_%28z%C3%A1kladn%C3%AD_informace%29.pdf

http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf (PDF verze MATLAB Primer)

pracovní prostředí

http://www.mathworks.com/help/techdoc/learn_matlab/bqxudm4-1.html

základní logika matic a syntaxe

http://www.mathworks.com/help/techdoc/learn_matlab/bs7uym1.html

http://online.redwoods.cc.ca.us/instruct/darnold/LinAlg/matrices/context-matrices-s.pdf

základy programování

http://www.mathworks.com/help/techdoc/learn_matlab/f4-2525.html

http://hore.dnom.fmph.uniba.sk/personal/jamrichova/MatLab/Programovanie.pdf

Simulink

http://www.kirp.chtf.stuba.sk/~cirka/vyuka/matlab/kap13.php

http://www.kirp.chtf.stuba.sk/~cirka/vyuka/matlab/files/sl_using.pdf

http://www.mathworks.com/products/simulink/

Poděkování

Na tomto místě bych chtěl poděkovat panu doc. Ing. Petru Čermákovi Ph.D., za podnětné připomínky, rovněž také kolegům z ústavu informatiky za rady a důležité informace, bez kterých by tento text nevznikl.