M-soubory (pokračování)

Typy M-souborů:

- skripty (probrány minule)
- uživatelské funkce, resp. knihovní funkce (dnes hlavní téma)

Uživatelské funkce

Máme-li používat jeden určitý postup (algoritmus) pro vícero různých situací (např. pro různé hodnoty proměnných), není zrovna praktické používat skripty, neboť pokaždé musíme opravit hodnoty proměnných ve skriptu, uložit příslušný M-soubor a skript spustit (resp. zadat vstupy s využitím funkce `input`, čímž "zaneřádíme" Command History). Řešení nabízejí funkce.

Funkce jsou M-soubory, které mají přesně definovanou strukturu (viz Vytvoření funkce). Funkce akceptují **vstupní parametry**, které mohou mít při každém spuštění jinou hodnotu. Každá funkce má své **vlastní pracovní prostředí**, které je odděleno od pracovního prostředí Command Window. Všechny proměnné ve funkci jsou *lokální* (existují jen po dobu spuštění funkce)! To znamená, že:

- nemůžeme použít žádné jiné proměnné než ty, které funkci předáváme (pomocí vstupních parametrů), nebo ty, které si funkce sama vytvoří,
- všechny proměnné (i ty, které obsahují vypočtené výsledky) po skončení funkce zaniknou. Jedinou možnost, jak může funkce předat své výsledky "ven", představují **výstupní proměnné**. Výstupních proměnných může být několik, takže v MATLABu název "funkce" není správný (z matematického hlediska).

Počet vstupních i výstupních parametrů funkce se určuje v definici funkce (tj. při jejím vytváření).

Pokud funkce nemá žádné vstupní parametry, můžeme namísto funkce vytvořit skript (probráno minule) - výsledek se liší především v tom, že po skončení skriptu *zůstanou* všechny jím vytvořené proměnné ve Workspace.

Poznámka - typy funkcí

A. **Funkce uložené na disku** (funkce v M-souborech) - použitelné při každém spuštění MATLABu
   - *primární funkce* = první funkce zapsaná v daném M-souboru. Pod ní mohou být uvedeny i další funkce, tzv. subfunkce. Primární funkci zavoláme z Command window (nebo z jiných M-funkcí) pomocí jména souboru, ve kterém je uložena (je doporučeno, aby se primární funkce jmenovala stejně jako M-soubor!).
   - *subfunkce* = funkce viditelné jen z dané (primární) funkce nebo z ostatních subfunkcí v daném M-souboru. Subfunkce jsou součástí stejného M-souboru a nezáleží u nich na pořadí (pouze primární funkce musí být uvedena jako první). Každá subfunkce má své vlastní workspace (pokud chceme sdílet proměnné, musíme je deklarovat ve všech dotčených (sub)funkcích jako globální nebo je předávat jako vstupy). Subfunkce jsou volány přednostně (před knihovními funkcemi atp. - viz Pořadí volání). Subfunkce mohou obsahovat nápovědu, která je dostupná příkazem `>> help soubor/subfunkce`
   - *privátní funkce* = fce uložené v adresáři s názvem `private` (nepřidávejte adresář do "vyhledávacích cest"!). Jsou spustitelné POUZE z funkcí, jež se nacházejí v rodičovském adresáři (pro jiné funkce nejsou viditelné => může existovat víc privátních fcí se stejným

jménem, ale v různých adresářích `private`). Jejich volání má přednost před knihovními funkcemi a M-funkcemi v aktuálním adresáři. Privátní funkce mohou obsahovat nápovědu - ta je pak přístupná pomocí `>> help private/privátní_funkce`

- od MATLABu 7.0 lze psát *vhnízděné funkce* (nested functions): uvnitř funkce je definice jiné funkce, VŠECHNY funkce pak musí končit `end` (nebudeme používat)
- při práci s objekty (cca v polovině semestru) se setkáme ještě s *přetíženými funkcemi* a *konstruktorem*

B. **Funkce uložené v paměti** - použitelné jen v aktuálně běžíci instanci MATLABu

- *anonymní funkce* vytvářené příkazem `fce = @(arglist) expr` - viz Anonymní funkce
- *inline funkce* vytvářené příkazem `g = inline(expr,arg1,arg2,...)` - viz Inline funkce

Nyní se budeme zabývat prací s funkcemi v M-souborech.

Vytvoření funkce

1. **Nastavíme pracovní adresář** MATLABu = adresář, do kterého chceme funkce ukládat a odkud je budeme spuštět (např. příkazem `cd`) - nejlépe na lokálním disku (síťové disky se občas odpojují). Musíme mít právo zápisu do daného adresáře!
   Výhoda nastavení pracovního adresáře PŘED spuštěním M-editoru spočívá v tom, že při uložení bude tento adresář automaticky nabízen jako výchozí.
2. **Otevření M-souboru**
   Funkce je M-soubor, a proto si nejprve musíme otevřít nový soubor: například pomocí menu File→New→M-file. Tím se otevře okno M-editoru/Debuggeru s prázdným souborem.
   Poznámka: pokud chceme nějakou existující funkci opravit/prohlížet, použijeme menu File→Open.
3. **Zápis funkce** (struktura M-souboru obsahujícího funkci)
   První řádek obsahuje *definici funkce*, po něm mohou následovat řádky s *nápovědou k funkci* (komentáře) a zbytek souboru tvoří příkazy (*kód funkce*, algoritmus) potřebné k výpočtu výstupů funkce za použití jejích vstupů.

   ■ **Definice funkce**
   tvoří první řádek M-souboru. Má tvar:

```
function [výstupy]= jmeno_funkce (vstupy)
```

klíčové slovo	výstupy funkce	název funkce	vstupní parametry funkce

   výstupy:
   - je-li jich víc, oddělují se čárkou
   - je-li jen jeden, hranaté závorky nejsou nutné
   - funkce nemusí mít žádný výstup - pak má první řádek tvar `function jmeno_fce(vstupy)`

   jmeno_funkce:
   - mělo by vystihovat její činnost
   - musí splňovat **pravidla pro název proměnné**, jinak se funkci nepodaří spustit!
   - nesmí se shodovat s žádným názvem její lokální proměnné!
   - maximální délka názvu funkce je dána číslem, které vrací funkce `namelengthmax` (většinou 63 znaků)

   vstupy:
   - je-li jich víc, oddělují se čárkou

- funkce nemusí mít žádný vstup (pak má první řádek tvar **`function`** *`jmeno_fce`*) - v tomto případě je volání funkce stejné jako volání nějakého skriptu (pouze názvem), avšak funkce má své lokální pracovní prostředí (workspace)!

Za definičním řádkem se nepíše středník.

Po definičním řádku může být uvedena nápoveda k funkci a pod ní samotný kód funkce.

Na konci M-souboru může být (v novějších verzích MATLABu, cca po 7.1?) uvedeno klíčové slovo **`end`**, ale v případě, že je v M-souboru pouze jedna funkce, tak není nutné.

Příklady definic funkcí:

```
function [s]=soucet(a,b)                - funkce s jedním výstupem a dvěma vstupy
function [p,z]=deleni(delenec,delitel) - funkce se dvěma vstupy i výstupy
function f=faktorial(n)                 - funkce s jedním výstupem i vstupem
function graf(x,y)                      - funkce bez výstupu, se dvěma vstupy
function kresli                         - funkce bez výstupu a bez vstupů
function [t,u,v]=kresli2                - funkce se 3 výstupy, bez vstupů
```

Nápověda k funkci

není povinnou součástí funkce, ale měla by být vytvořena, protože usnadňuje používání dané funkce. Za nápovědu k funkci jsou považovány takové řádky bezprostředně následující za definičním řádkem (tj. od 2. řádku dolů), které začínají znakem `%` (tj. souvislý "blok" komentářů). Nápověda k funkci končí jakýmkoli řádkem, který nezačíná znakem `%` (tj. i třeba prázdným řádkem).

První řádek nápovědy (anglicky zvaný `H1 line`) by měl začínat jménem funkce (od MATLABu 7.0 navíc ve správné velikosti písmen) a vystihovat její činnost, protože je vypisován příkazem **`>> lookfor slovo`** (slouží pro výpis všech funkcí obsahujících v prvním řádku své nápovědy dané slovo) nebo při výpisu nápovědy k funkcím nějakého adresáře **`>> help adresář`** (například `>> help C:\temp\MATLAB`).
Další řádky nápovědy by měly obsahovat definici funkce, popis vstupů a výstupů funkce a také příklad jejího použití (volání).

Pokud je v M-souboru funkce obsažena nápověda, zobrazíme ji příkazem **`>> help jméno_funkce`**, stejně jako nápovědu ke knihovním funkcím MATLABu.

Kód funkce

(algoritmus; příkazy) začíná hned za nápovědou a obsahuje posloupnost příkazů, pomocí nichž funkce vypočítá své výstupy.
Poznámky:

- všechny **výstupy funkce** musí být po skončení provádění jejího kódu **vytvořeny**, jinak je po spuštění funkce (s odebráním výstupu/ů) ohlášena chyba typu `??? Output argument "jméno" (and maybe others) not assigned during call to "jméno_fce".`!
- všechny přiřazovací příkazy unvitř funkce by měly být **ukončeny středníkem** (aby funkce neobtěžovala okolí výpisem mezivýsledků, tj. obsahu pomocných proměnných)
- pro předčasné ukončení funkce lze použít příkaz **`return`**, před nímž musí být vytvořeny všechny výstupy (přiřazovacím příkazem)
- pokud je potřeba, aby funkce předčasně skončila s chybou (tj. s výpisem chyby do Command Window), tak lze použít funkci **`error(řetězec)`**, jejímž parametrem je text

chybového hlášení (např. `error('Chyba: vstup musí být celočíselný!')`). V tomto případě funkce nevrací žádné výstupy (ani kdyby některé již byly ve kunkci vytvořeny)

- kromě vytvoření nápovědy k funkci je vhodné používat také komentáře v kódu funkce (u jednotlivých příkazů), protože až funkci ukážete kolegům nebo ji budete studovat za půl roku, už si nevzpomenete, co znamenal "tenhle divnej řádek"
  doc. Kukal: *"Program bez komentářů je jako velbloud bez hrbů - nedá se na něj dlouho dívat."*
- složité problémy je vhodné rozložit na podproblémy - složitou funkci rozdělíme na několik jednoduchých, které se vzájemně volají

Příklad 1: funkce, která vypočítá součet dvou čísel:

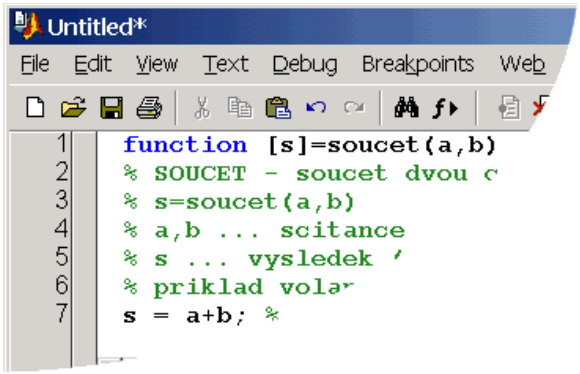
```
function [s]=soucet(a,b)
% SOUCET - soucet dvou cisel
% s=soucet(a,b)
% a,b ... scitance
% s ... vysledek (soucet)
% priklad volani: s=soucet(10,-2.5)
s = a+b;
```

Všimněte si:
- nápověda k funkci obsahuje vše potřebné pro správné použití funkce
- kód funkce zabere sice jen jediný řádek, ale to k vypočítání výstupu s stačí
- přiřazovací příkaz uvnitř funkce je ukončen středníkem, aby funkce neobtěžovala okolí svými pomocnými výpočty
- příkazem `>> help soucet` získáme nápovědu k funkci (včetně jejího jména)
- příkaz `>> type soucet` zobrazí celý obsah souboru `soucet.m`

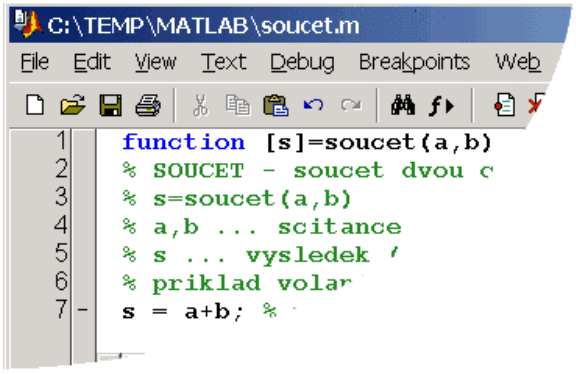
Příklad 2: funkce, která vypočítá délku přepony pravoúhlého trojúhelníku. Funkce obsahuje jen minimální nápovědu:

```
function [prep]=prepona(odvesna1,odvesna2)
% PREPONA - vypocet prepony pravouhleho trojuhelniku
prep = (odvesna1^2 + odvesna2^2)^(1/2); % pouziti Pythagorovy vety
```

4. **Uložení funkce**
   Napsanou funkci musíme před spuštěním **uložit** na disk jako M-soubor. Použijeme např. menu File ➞ Save, kde:
   - zkontrolujeme pracovní adresář (funkci můžeme uložit i jinam, ale potom bude nutno před jejím spuštěním přepnout adresář)
   - jako název souboru zadáme jméno funkce (v případě, že jsme funkci vytvářeli v novém souboru, tak jméno souboru už je *předvyplněno*), protože **název M-souboru se musí shodovat s názvem dané funkce**, jinak by funkce nešla spustit (MATLAB hledá jméno souboru a v něm pak příslušnou funkci)!



Obr. 1: zatím neuložená funkce v M-editoru



Obr. 2: uložená funkce v M-editoru

Všimněte si: po uložení M-souboru **zmizí hvězdička** za jeho názvem v titulkovém pruhu okna M-editoru. **Pokud za názvem souboru svítí hvězdička**, znamená to, že **není uložen**, a tedy **při spuštění je použita nějaká předchozí verze funkce** uložená v paměti MATLABu => **záhadné chyby**!

Spuštění funkce

Pokud jsme vytvořili vlastní funkci, můžeme ji spustit z Command Window (anebo z jiných funkcí či skriptů). Příkaz pro spuštění funkce vypadá obecně takto:

`>> [vystupy]=jmenofce(vstupy)`, pokud chceme uložit výsledky do nějakých proměnných (na konci lze uvést středník, aby se výsledky nevypisovaly), nebo takto:

`>> jmenofce(vstupy)`, pokud nechceme výsledky ukládat do proměnných (není-li za tímto příkazem uveden středník + funkce má definován alespoň jeden výstup, pak vznikne proměnná `ans`.

Příklady volání funkcí:

```
>> s=soucet(7,14);  s odebráním výstupu a bez jeho výpisu
>> vysledek=soucet(7,14)  s odebráním výstupu i jeho vypsáním
>> x=7; y=14; soucet(x,y)  bez odebrání výstupu, ale vypsáním výsledku (pomocí ans)
>> soucet(3.1,8.14);  bez odebrání výstupu, bez vypsání výsledku (v podstatě zbytečná akce)
```

Základním předpokladem pro volání správné funkce je, aby funkce byla "viditelná", tj. umístěna buď v pracovním adresáři, anebo v adresáři, jenž je součástí "vyhledávacích cest" MATLABu.

"Vyhledávací cesty" MATLABu (search path)

Pokud chcete volat funkci, která není v pracovním adresáři (nebo je v pracovním adresáři, ale používá funkce z jiného adresáře), je nutné takový adresář přidat do "vyhledávacích cest" MATLABu. K dočasnému přidání adresáře do "vyhledávacích cest" (tj. v rámci běžící instance MATLABu) slouží příkaz

`>> addpath absolutní_cesta`

Pozn.: adresář lze natrvalo do "vyhledávacích cest" přidat pomocí menu File→Set Path...

Součástí "vyhledávacích cest" jsou všechny adresáře obsahující jednotlivé knihovny MATLABu, a dále pak adresáře všech instalovaných toolboxů.

Chyby při volání funkce a jejich příčiny

"Divné" chyby vznikají kvůli tomu, že **funkce není uložena** (pokud jsme ji opravovali) => před jakýmkoli spuštěním **funkci uložte** a pro jistotu zkontrolujte, že v M-editoru za jménem nesvítí hvězdička!
Pozn.: někdy pomůže k odstranění takové chyby smazání všech proměnných i funkcí z pracovního prostředí (`>> clear all`).

Nejčastější chyby:

chyba	pravděpodobná příčina
`??? Undefined function or variable 'jméno_funkce'.`	jméno funkce je napsáno chybně (překlep) nebo není nastaven správný pracovní adresář
`??? Input argument "jméno" is undefined.`	voláme funkci s menším počtem vstupů, než je vyžadován => zadejte všechny vstupy
`??? Error using ==> jméno_fce` `Too many input arguments.`	zadali jsme moc vstupů, funkce jich vyžaduje méně
`??? Error using ==> pokus` `Too many output arguments.`	odebíráme víc výstupů, než má funkce definováno
`??? Output argument "jméno" (and maybe others) not assigned during call to "jméno_fce".`	chceme odebrat výstup, který funkce nevytvořila (většinou se jedná překlep ve jménu výstupní proměnné v kódu funkce nebo opomenutí přiřadit do VŠECH výstupů výslednou hodnotu)

Rada: pokud je po spuštění funkce ohlášena chyba, tak nejprve ověřte, zda je funkce uložena, dále zda je pracovní adresář nastaven správně, dále zda je správně zadáno jméno funkce a jestli souhlasí počet vstupních/výstupních parametrů.

Pokud není po spuštění funkce hlášena chyba, ale funkce **nevrací správné výsledky**, musíme krokovat (viz níže).

Další důležité poznámky ke spouštění funkcí

- funkci, která má alespoň jeden vstup, nelze spuštět z M-editoru příkazem Debug→Run (klávesová zkratka F5) jako skript, protože v rámci tohoto příkazu nelze zadat vstupy funkce. Musíme ji spouštět vždy příkazem v Command Window
- pokud při psaní *jmenofce* nedodržíte velikost písmen shodnou s definicí, je od MATLABu 7.0 vypisováno varování (v budoucích verzích budou pravděpodobně názvy funkcí case-sensitivní)
- počet vstupních argumentů se musí shodovat s její definicí (výjimkou jsou funkce obsahující proměnný počet vstupů - `varargin`)
- pořadí vstupních argumentů při volání je zavazující - jsou zpracovávány v pořadí daném definicí funkce
- výstupy z funkce nejsme povinni odebírat (pokud nenapíšeme za voláním funkce středník, tak se první z výstupů vypíše pomocí `ans`)
- pokud odebíráme výstupy funkce, měl by být dodržen jejich počet (funkce obsahující `varargout` mohou vracet různé výsledky v závislosti na počtu právě odebíraných výstupů `nargout`)
- neznáme-li počet vstupů ani výstupů, necháme si zobrazit nápovědu k funkci (`>> help jmenofce`). Pokud autor neopatřil funkci nápovědou, zbývá jediná možnost - nechat si vypsat zdrojový kód (`>> type jmenofce`)
- funkce, jejichž vstupem jsou POUZE řetězce, můžeme volat "na způsob příkazu" (Passing String Arguments to Functions), tj. bez použití závorek a apostrofů, avšak v tomto případě MATLAB nemůže vrátit žádné výstupy:
  `>> nazvy Praha Brno`
  je MATLABem interpretováno jako
  `>> nazvy('Praha','Brno')`
  Tento způsob volání nelze použít u funkcí, jejichž argumenty nejsou řetězce!
  Doporučení: volejte funkce standardním způsobem (kulaté závorky).

Co se děje při volání funkce

Když voláme funkci (uloženou v M-souboru), tak si ji MATLAB parsuje do pseudokódu. Tento pseudokód je uchován v paměti po celou dobu spuštění MATLABu, aby byla práce rychlejší (nemusí se parsovat při každém spuštění funkce). Vytvořený pseudokód by se měl nahradit novým, pokud je M-soubor změněn a **uložen**. Občas se však stane, že pseudokód není vyměněn! V tom případě je potřeba "smazat funkci z paměti" následujícím způsobem:

Odstranění funkce z paměti

`clear` *`jméno_funkce`*	odstraní pseudokód funkce z paměti - pouze danou funkci
`clear functions`	odstraní všechny M-funkce z paměti
`clear all`	odstraní všechny proměnné a M-funkce z paměti (POZOR: přijdeme o všechny proměnné!)

Pořadí volání

1. **Proměnné**: MATLAB nejprve zjistí, zda použitý název představuje proměnnou - pokud ji nalezne ve workspace, tak ji použije (tzn. NENÍ volána funkce, i kdyby byla v aktuálním adresáři!).
   Příklad: v akt. adresáři existuje funkce `ahoj` s jedním vstupem (= kolikrát se vypíše "ahoj"). Poté vytvoříme proměnnou `>> ahoj = 3.25`. Následné volání `>> ahoj(5)` způsobí chybu, protože není volána funkce, nýbrž proměnná (snažíme se přistoupit k 5. prvku proměnné `ahoj`). K funkci budeme moci přistupovat, až smažeme proměnnou `ahoj` z paměti.
2. **Subfunkce** mají přednost před všemi M-funkcemi (dostupnými přes "vyhledávací cesty") a přetíženými metodami.
3. **Privátní funkce** jsou volány, pokud aktuálně není k dispozici subfunkce s daným jménem.
4. **Konstruktor třídy** je nadřazen "obyčejným" M-funkcím. Např. když v aktuálním adresáři existuje `trida.m` a spolu s ní také `@trida/trida.m` (neboli konstruktor `trida.m` v podadresáři `@trida`), bude volán pouze konstruktor.
5. **Přetížené metody** jsou volány, pokud je "nepředběhly" subfunkce nebo privátní funkce. To, jaká metoda bude volána, závisí na vstupních argumentech (k jaké třídě objekt patří).
6. **Funkce v aktuálním adresáři** mají přednost před ostatními dostupnými funkcemi.
7. **Funkce umístěné jinde** (ve "vyhledávacích cestách") jsou volány, pokud žádný z předchozích bodů neuspěl.

Krokování (ladění) funkcí/skriptů

Pokud jsou ve funkci chyby (např. nedává správné výsledky) a nemůžeme jejich příčinu najít pouhým přečtením jejího kódu (nebo vypisováním dílčích mezivýsledků, resp. dočasným odstraněním některých středníků), použijeme Debugger ("odvšivovač"), který nám nabízí možnost **krokování**:

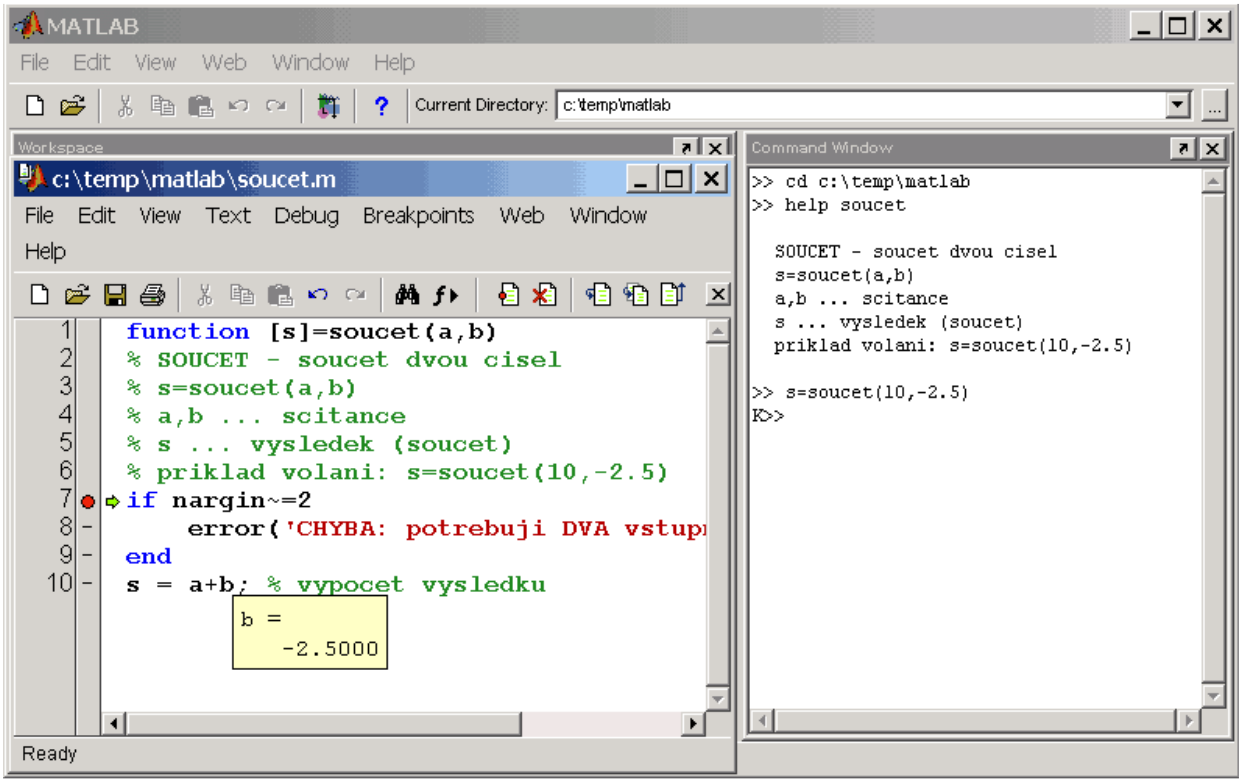
1. v M-editoru nastavíme breakpoint (menu Debug→→Set/Clear Breakpoint, resp. F12) na nějakém řádku s příkazem
2. spustíme funkci s jejími parametry (z Command window) - v Command Window se objeví `K>>` a v M-editoru svítí zelená šipka před řádkem, který bude následně zpracován
3. jednotlivé příkazy funkce spouštíme (krokujeme) v M-editoru pomocí menu Debug→Step (klávesová zkratka F10; krokuje jen aktuální funkci) nebo Debug→Step In (kláves. zkratka F11; otevře a krokuje i funkci volanou uvnitř krokované funkce) - výsledky příkazů kontrolujeme pomocí

myši (pohyb kdekoli v kódu nad názvem proměnné, jejíž obsah nás zajímá) nebo pomocí okna Workspace (kde by měl být nastaven zásobník/Stack pro krokovanou funkci)

4. výsledkem celého snažení je nalezení chyby, její oprava a uložení opravené funkce. Nechceme-li dále krokovat, odstraníme breakpointy, jinak opakujeme body 2 až 4

Hesla semestru (podle doc. Kukala):

- Kdo nekrokuje s námi, krokuje proti sobě.
- Pokud funkce musí fungovat, ale přesto nefunguje, krokuj.
- Pokud funkce vypadá jako zcela nefunkční, ale přesto funguje, krokuj.



Obr. 3: krokování funkce (MATLAB 6.5)

Zobrazení nápovědy k funkci

Pokud chceme vědět, jak funkci spustit nebo jak pracuje (a funkce byla vyvtořena s nápovědou), můžeme použít příkaz `>> help jmenofce`, kde *jmenofce* je název M-souboru bez přípony.
Jinou možností je pak příkaz `>> doc jmenofce`, čímž docílíme téhož efektu, jako když napíšeme jméno funkce (v Command Window nebo v M-editoru/Debuggeru), označíme ho (třeba dvojklikem myši) a pomocí pravého tlačítka myši zobrazíme kontextové menu, kde zvolíme položku "Help on selection": v obou případech se otevře okno Help Browseru s nápovědou k funkci; tato nápověda vypadá u knihovních funkcí lépe (např. obsahuje obrázky) než při použití příkazu `help`.

Zobrazení kódu funkce

Kód funkce (tj. obsah M-souboru) vidíme v M-editoru/Debuggeru, ale můžeme jej zobrazit také v Command Window příkazem `>> type jmenofce`, kde *jmenofce* je název funkce (název M-souboru bez přípony).
Poznámka: kód můžeme zobrazit i u některých knihovních funkcí (speciálně u těch, které nejsou

optimalizovány, "vestavěny" do jádra - tzv. built-in funkce) - zkuste:

```
>> type sin % built-in, kód se nezobrazí
>> type mean % kód by se měl zobrazit
```

Poznámky k vytváření funkcí:

Rekurze

MATLAB umožňuje psát rekurzivní funkce (tj. volat funkci v jejím kódu je povoleno).

Globální proměnné

Kromě lokálních proměnných lze ve funkcích používat *proměnné globální*. Deklarují se v kódu funkce pomocí příkazu **`global proměnná`**. Tímto způsobem lze sdílet proměnné mezi funkcemi. Globální proměnná existuje v základním pracovním prostředí (base workspace) a je sdílena v lokálním pracovním prostředí každé funkce, která globální proměnnou využívá (promítá se tam jakákoli změna její hodnoty).

Příkazem `clear global` *`proměnná`* (uvnitř funkce) smažeme globální proměnnou z globálního pracovního prostoru (base workspace).
Příkazem `clear` *`proměnná`* (uvnitř funkce) smažeme globální odkaz z lokálního pracovního prostředí funkce, tedy nadále již nebude možné ovlivnit hodnotu globální proměnné z této funkce.

Zjišťování počtu skutečných vstupních a výstupních argumentů funkce

Pokud chceme, aby naše funkce reagovala na různý počet vstupních parametrů (tj. na počet vstupů při konkrétním volání z Command Window), můžeme v jejím kódu použít příkaz **`nargin`** (ve skutečnosti je to knihovní funkce MATLABu). Výsledkem volání `nargin` je nezáporné celé číslo představující počet vstupních parametrů, s nimiž byla daná funkce spuštěna. Pokud tedy někdo zavolá naši funkci s menším počtem vstupních argumentů, můžeme je zpracovat jinak než v případě plného počtu vstupů (např. nastavit chybějícím vstupům nějakou implicitní hodnotu).
Pozor: pořadí vstupních parametrů v definici funkce je zavazující (tedy pokud si myslíme, že vynecháme druhý ze čtyř vstupů, funkce to pochopí tak, že jsme vynechali ten čtvrtý!).

Pro zjištění skutečného počtu výstupních parametrů (který je uveden v době spouštění funkce) máme funkci **`nargout`**. Pracuje se s ní podobně jako s funkcí `nargin`.

Příklad 3 - vylepšíme funkci `soucet`, aby sama hlásila chybu, když ji někdo zavolá pro méně než dva parametry:

soucet.m

```
function [s]=soucet(a,b)
% SOUCET - soucet dvou cisel
% s=soucet(a,b)
% a,b ... scitance
% s ... vysledek (soucet)
% priklad volani: s=soucet(10,-2.5)
if nargin~=2 % nespravny pocet vstupu?
   error('CHYBA: k vypoctu potrebuji DVA vstupy!') % vypis chyby a konec funkce
end % konec if
s = a+b; % vypocet vysledku
```

POZOR: `nargin` je shora omezen počtem skutečně uvedených vstupů v definici funkce. Ve výše uvedeném příkladu tedy nemá cenu kontrolovat, zda `nargin~=2`, protože ve skutečnosti to funguje jen pro `nargin<2`.

Funkce s proměnným počtem vstupů

V případě, že potřebujeme vytvořit funkci, která nemá omezený počet vstupů (lze ji volat s libovolným počtem vstupních parametrů nebo úplně bez parametrů), použijeme namísto zápisu jmen (omezeného počtu) proměnných zápis jediného vstupu **`varargin`**. Skutečné vstupy funkce jsou pak přístupné pomocí této proměnné. `varargin` je tzv. pole buněk (cell array), a zpracování skutečných vstupů je pak možné např. v cyklu, např.:

```
function pokus(varargin)
% POKUS - prom. pocet vstupu
for i=1:nargin % nebo i=1:length(varargin)
  disp(varargin{i}) % vypis hodnot vsech vstupu
end
```

Funkce s proměnným počtem výstupů

Funkce může vracet tolik výstupů, kolik je uvedeno při jejím volání (libovolný počet). Použití: **`varargout`** jako jediného výstupu, zpracování např. v cyklu (všechny výstupy je nutno ve funkci naplnit pomocí proměnné `varargout`, což je opět pole buněk), např.:

```
function varargout = pokusven(k)
% POKUSVEN - prom. pocet vystupu
% vstupem je cislo, ktera udava nasobek cisel 1,2,...,p (p je pocet odebranych vystupu)
for i=1:nargout
  varargout{i} = k*i; % inicializace vsech vystupu
end
```

"Ukazatel" na funkci (function handle)

"Ukazatel" na funkci lze s výhodou použít v případě, kdy potřebujeme ve funkci zavolat jinou funkci, jejíž jméno zatím neznáme, resp. chceme zavolat jednu ze skupiny funkcí (které mají shodný počet vstupů a výstupů, ale odlišnou logiku). V kódu funkce pak využijeme volání cizí funkce pomocí jejího "ukazatele" ("ukazatel" na volanou funkci pak samozřejmě musí být vstupem vytvářené funkce).
Nejčastěji to využijeme v případě, kdy vytváříme nějaký univerzální nástroj, např. pro výpočet určitého integrálu nějaké funkce v daném intervalu.

Vytvoření "ukazatele" na funkci

K vytvoření "ukazatele" použijeme zavináč před jménem funkce (bez cesty): **`ukazatel = @název_fce`**

Funkce musí být v době vytváření ukazatele "viditelná" (tj. v aktuálním adresáři nebo ve vyhledávacích cestách MATLABu).

MATLAB si do "ukazatele" uloží (mapuje) informace o dané funkci. Při vytváření ukazatele MATLAB bere v úvahu:

- obor působnosti (rozsah, *scope*) - funkce musí být "viditelná", tj. v aktuálním adresáři nebo ve vyhledávycích cestách,
- prioritu (*precedence*) - MATLAB vybere funkci, která má nejvyšší přednost - viz Pořadí volání,

- přetěžování (*overloading*) - týká se práce s objekty, tj. případů, kdy třída obsahuje metodu stejného jména jako nějaká knihovní funkce pro standardní datový typ (např. pro double nebo char). Podrobnosti naleznete v nápovědě (vyhledejte *handle overload* a čtěte *Additional Information on Function Handles*).

Pokud například změníte pracovní adresář (kde se nachází původní funkce, čímž tato přestane být "viditelná"), tak ukazatel stále pracuje s původní funkcí, i kdyby v novém pracovním adresáři byla funkce se stejným jménem. Tedy mapování na funkci zůstává zachováno i poté, co funkce opustí obor působnosti MATLABu (*"a function handle retains that same mapping even if its corresponding function goes out of scope"*).

"Ukazatel" na funkci lze ukládat a opětovně nahrát (s využitím příkazů `save` a `load`), avšak pro správné fungování "ukazatele" musí být dodrženy stejné podmínky jako v době jeho vytvoření - tedy např. funkce musí existovat na stejném místě jako minule, musí být správně nastavený pracovní adresář apod.

Volání funkce pomocí "ukazatele"

1. prostým zápisem ***`ukazatel(parametry)`*** - doporučený způsob, funguje od MATLABu 7.0. Pokud funkce nemá žádné vstupní parametry, musíme i tak za "ukazatel" napsat prázdné závorky!
2. s použitím funkce **`feval`** - tento způsob volání je zachován z důvodu kompatibility se staršími verzemi MATLABu; můžeme použít jednu ze dvou variant volání:
   `[y1,...,yk] = feval(ukazatel,x1,...,xn)` nebo
   `[y1,...,yk] = feval(název_funkce_jako_řetězec,x1,...,xn)` (jen do verze 6.5; název funkce musí být jednoduchý, tj. bez cesty k souboru)
   - `x1,...,xn` jsou všechny vstupní parametry volané funkce
   - `y1,...,yk` jsou odebírané výstupy

   Poznámka: funkce volaná pomocí `feval` druhým způsobem (tj. názvem) musí být buď knihovní funkcí MATLABu, anebo uživatelskou funkcí uloženou v M-souboru (v aktuálním adresáři).

Následujících 6 zápisů je ekvivalentních:

```
>> y = sin(x) % obyčejné volání

>> fhand=@sin; y = fhand(x) % od MATLABu 7.0, doporučeno

>> fhand=@sin; y = feval(fhand,x)

>> y = feval(@sin,x)

>> y = feval('sin',x)

>> nazev='sin'; y = feval(nazev,x) % ve starších verzích velmi obvyklé
```

Příklad:
máme funkci `polynom1` uloženou v souboru `polynom1.m`, která má jeden vstup a jeden výstup:

`polynom1.m`

```
function v=polynom1(u)
% polynom1 - funkcni hodnota polynomu 3. stupne v bode 'u'

v = 2*u^3 - u^2 + 15*u -8;
```

Tuto funkci můžeme volat pro různé hodnoty nezávisle proměnné, např. těmito způsoby:

```
>> y = polynom1(30.2) % přímé volání
>> fhand=@polynom1; x=7/4-0.5; y = feval(fhand,x) % nepřímé volání pomocí "ukazatele"
>> y = feval(@polynom1,-13)
>> y = feval('polynom1',-4.5) % už není doporučeno
>> nazev='polynom1'; x=5.21; y = feval(nazev,x) % už nedoporučeno
```

Práce s "ukazatelem" na funkci

Funkce pracující s "ukazateli" na funkce:

funkce	**popis**
`functions`	vrací informace popisující "ukazatel" na funkci `>> f = @sin; functions(f)`
`func2str`	konverze "ukazatele" na řetězec (jméno funkce) `>> func2str(f)`
`str2func`	konverze jména funkce (řetězec) na "ukazatel" `>> str2func('sin')`
`save`	uložení "ukazatele" z aktuálního workspace do MAT-souboru
`load`	načtení "ukazatele" funkce z MAT-souboru do akt. workspace
`isa`	zjistí, zda proměnná je "ukazatel" na funkci `>> isa(f,'function_handle')`
`isequal`	porovnání: jedná se o dva "ukazatele" na stejnou funkci? `>> g = @cos; isequal(f,g)`

Sdružování "ukazatelů" do polí

MATLAB nepodporuje ukládání "ukazatelů" na funkce do matic/vektorů, ale je dovoleno vytvářet pole buněk obsahující "ukazatele" na funkce. V rámci pole buněk lze všechny "ukazatele" zpracovat pomocí cyklu **`for`**.

Příklad:

```
>> C = {@sin, @cos, @tan}; % pole buněk
>> f = C{2};
>> y = f(pi/3); % volání funkce cos(pi/3)
```

Kromě polí buněk můžeme také ukládat "ukazatele" do struktur (naučíme se v druhé části semestru).

Anonymní funkce ("ukazatel" na anonymní funkci)

Anonymní funkce poskytují rychlou cestu pro vytvoření jednoduchých funkcí bez nutnosti vytvářet M-soubor. Anonymní funkce lze vytvářet buď z příkazového řádku MATLABu, nebo v jakémkoli M-souboru (funkci nebo skriptu).

Vytvoření anonymní funkce

Anonymní funkci (přesněji "ukazatel" na bezejmennou funkci) vytvoříme příkazem

```
fhandle = @(arglist) expr
```

- `expr` (výraz) reprezentuje tělo funkce - kód musí být jednořádkový (jeden příkaz),
- `arglist` je seznam vstupních parametrů oddělených čárkou,
- znak `@` je povinnou součástí anonymní funkce - bez "ukazatele" by nemohla existovat.

Anonymní funkce nejčastěji vrací **jeden výstup** (tím může být i matice nebo vektor), ale pokud v jejím kódu použijeme např. funkci `xlsread`, tak můžeme odebírat více výstupů (maximálně však tolik, kolik je taková funkce schopna vrátit).

Typy proměnných používaných v anonymní funkci jsou dva:

- vstupní argumenty - jejich hodnoty se liší podle konkrétního volání anonymní funkce,
- proměnné uvedené v těle výrazu - tyto proměnné existují v době vytváření anonymní funkce ve workspace a mají nějakou hodnotu. Jejich hodnota zůstává v rámci anonymní funkce *konstantní*, i když je některá z těchto proměnných ve workspace změněna.
  Výhoda: anonymní funkci lze volat i z jiných funkcí a bude fungovat správně.
  Nevýhoda: chceme-li změnit hodnotu "nevstupní proměnné", musíme znovu definovat celou anonymní funkci.

  Příklad:

```
>> a=4; b=-2; % proměnné ve workspace
>> primka = @(x) a*x+b; % anonymní funkce
>> y = primka(0); % vyhodnocení přímky v bodě x=0 (y=-2)
>> a=8;
>> y = primka(2); % vyhodnocení přímky v bodě x=2 ('a' je stále 4, tj. y=6)
```

Spuštění anonymní funkce

Spuštění je stejné jako u "ukazatelů" na jiné funkce: za název "ukazatele" uvedeme do kulatých závorek vstupní parametry oddělené čárkou.

```
fhandle(args)
```

Pokud funkce nemá žádný vstup, musíme uvést prázdné závorky (jinak nevoláme funkci, ale vypisujeme informace o "ukazateli")!

Příklady anonymních funkcí spolu s jejich voláním:

```
>> f1 = @() disp('ahoj'); % funkce bez vstupu
>> f1() % volání

>> f2 = @(x) x^2-2*x+3; % funkce s 1 vstupem
>> f2(5); % volání (ans=18)
>> y = f2(-1); % volání s odebráním výstupu (y=6)

>> f3 = @(a,b) [a b a-b a+b]; % funkce se 2 vstupy, vracející vektor
>> v = f3(5,2); % volání s odebráním výstupu
```

Sdružování anonymních funkcí do polí

Anonymní funkce lze ukládat do pole buněk (podobně jako u "ukazatelů" na jiné funkce), např.:
```
>> A = {@(x)x.^2, @(y)y+10, @(x,y)x.^2+y+10}
```
a potom je volat:
```
A{1}(4) + A{2}(7)
```

POZOR: při vytváření pole buněk znamená mezera oddělovač prvků! Proto buď odstraňte všechny mezery z anonymních funkcí, anebo použijte kulaté závorky kolem anonymní funkce, anebo nejprve uložte anonymní funkce do proměnných a tyto pak použijte jako prvky pole buněk. Tedy:
`>> A = {@(x)x^2, @(y)y+10, @(x,y)x.^2+y+10}` nebo
`>> A = {(@(x) x^2), (@(y) y + 10), (@(x,y) x^2 + y + 10)}` nebo
`>> a1 = @(x) x^2; a2 = @(y) y + 10; a3 = @(x,y) x^2 + y + 10; A = {a1, a2, a3}`

Výhody a nevýhody anonymních funkcí

Výhody: rychlé vytvoření, není potřeba M-soubor.

Nevýhody: jednořádkový kód, omezené možnosti výstupu, ztráta funkce po ukončení MATLABu, nemá nápovědu, nemá smysl je krokovat.

Inline funkce

Hodně podobné anonymním funkcím jsou tzv. inline funkce (*inline function object*). Odlišný je způsob jejich vytváření a datový typ - nejedná se o "ukazatel" na funkci.

Vytvoření inline funkce

Inline funkci vytvoříme pomocí knihovní funkce `inline`. Jsou tři možnosti volání:

- **`jmenofce = inline(expr)`** - vytvoří inline funkci na základě výrazu `expr` (řetězec). Výraz musí obsahovat jen jednopísmenné proměnné! Vstupní proměnnou(é) MATLAB určí automaticky jako malé písmeno nejbližší písmenu *x* (kromě *i* a *j*). Není-li nalezeno žádné písmeno, použije se *x*;
- **`jmenofce = inline(expr,arg1,arg2,...)`** - umožňuje zadat inline funkci s více než jedním vstupem, resp. s vícepísmennými názvy proměnných;
- **`jmenofce = inline(expr,n)`** - vytvoří inline funkci s *n* vstupy (*n* je skalár!), které se jmenují `x`, `P1`, `P2` atd.

Práce s inline funkcemi

Spuštění inline funkce: **`jmenofce(parametry)`**

Zjištění jmen vstupních parametrů: **`argnames(jmenofce)`** (vrací pole buněk, kde každý prvek je řetězec představující jméno jednoho vstupu)

Převod inline funkce **na řetězec**: **`char(jmenofce)`** (identické s následující funkcí `formula`)

Zjištění výrazu použitého v inline funkci: **`formula(jmenofce)`**

Vektorizace inline funkce (využijeme, až se naučíme pracovat s vektory - za 2 týdny): **`vectorize(jmenofce)`** (vrací inline funkci; vloží tečku před operátory `*`, `/` a `^`)

Příklad:

```
>> f = inline('x^2-2*x+3') % vytvoření inline funkce s jedním vstupem
f =
     Inline function:
     f(x) = x^2-2*x+3
>> y = f(10) % spuštění funkce (y = 83)
>> argnames(f) % seznam vstupů (ans = {'x'})
>> char(f) % řetězec s kódem inline funkce
>> formula(f) % kód inline funkce
>> f2 = vectorize(f) % vektorizace => f2(x) = x.^2-2.*x+3

>> g = inline('2*x + 3*y - z') % vytvoření inline funkce se třemi vstupy
g =
     Inline function:
     g(x,y,z) = 2*x + 3*y - z
>> argnames(g)

>> g2 = inline('2*x + 3*y - beta', 'y', 'x', 'beta') % 3 vstupy, přesně dané pořadí
g2 =
     Inline function:
     g2(y,x,beta) = 2*x + 3*y - beta
>> argnames(g2)
```

Optimalizace zdrojového kódu

Při optimalizaci zdrojového kódu funkcí můžeme sami udělat následující:

- vektorizace cyklů
- předalokování polí (známe-li předem rozměry vektoru/matice, lze využít funkci `ones` či `zeros`; známe-li předem rozměry pole buněk, lze využít funkci `cell`)
- efektivnější využívání paměti (mazat nepotřebné proměnné)
- odstraňování nepotřebných funkcí z paměti (`clear function`)
- ukládat složitě vypočítané hodnoty do poměnných (pokud je nutné jejich opětovné použití)

Ke zvýšení výkonu funkce lze také použít nástroj **Profiler**.

Nástroj `profiler`

Profiler slouží k časové analýze funkce/skriptu. Umožňuje zjistit, které úseky kódu (např. volání jiných funkcí) spotřebovávají nejvíce času. Použití:

1. `>> profile viewer % spustí grafické prostředí Profileru`
2. v řádku *Run this code:* napsat příkaz pro spuštění funkce (i s parametry)
3. po stisknutí ENTERu se v hlavní části okna Profileru objeví časový přehled o funkci

Další informace naleznete v nápovědě: (záložka Content) MATLAB --- Desktop Tools and Development Environment --- Tuning and Managing M-Files --- Profiling for Improving Performance.

Knihovní funkce aneb Co už je hotovo

Než začnete vytvářet uživatelské funkce, vyplatí se vám seznámit se s funkcemi, které jsou již vytvořeny (tzv. *knihovní funkce*). Můžete tím zefektivnit svou práci. Knihovní funkce jsou sdruženy do tzv. knihoven samotného MATLABu nebo do tzv. *toolboxů* (toolboxy se dokupují samostatně).

Ještě jednou si tedy připomeneme systém nápovědy MATLABu, protože z následujících důvodů se vyplatí jej využívat:

- potřeba použít neznámou funkci (resp. příkaz)
- záměr použít dosud nevyužívané možnosti známých funkcí/příkazů
- potřeba osvěžit si dlouho nepoužívané znalosti
- potřeba inspirovat se příbuznými funkcemi (uvedeny v části "See also")
- snaha zjistit celkové možnosti MATLABu v tom kterém směru

Hledání názvu funkce podle klíčového slova

Všechny knihovní funkce mají nápovědu (v angličtině). Pokud tedy potřebujeme zjistit, zda pro nějakou činnost existuje v MATLABu již hotová funkce, můžeme hledat

- pomocí Help Browseru (záložka Search) nebo
- pomocí příkazu `>> lookfor hledane_slovo`
  Příkaz hledá v popisech funkcí a u každé z nalezených funkcí vypisuje první řádek jejich nápovědy (tzv. H1 line).
  Hledání pomocí `lookfor` může trvat delší dobu (nechcete-li čekat, stiskněte CTRL+C).

Nápověda k funkci

Podrobnou nápovědu k funkci, jejíž název známe, získáme:

- příkazem `>> help nazevfce` - výsledek se vypíše do Command Window
- příkazem `>> doc nazevfce` - otevře se Help Browser (od MATLABu 6.5)
- pomocí záložky Search v Help Browseru (menu Help...; od MATLABu 6)
- kliknutím pravým tlačítkem myši na název funkce + výběr položky "Help on selection" (od MATLABu 6)

Poznámka: Help Browser je dostupný, pokud jsou nainstalovány soubory nápovědy! Soubory nápovědy jsou WWW stránky, takže jsou uživatelsky příjemnější než textová nápověda (příkaz `help funkce`).
Ve starších vezrích MATLABu je namísto Help Browseru tzv. Helpdesk (spustí se příkazem `helpdesk`).

Výpis kódu funkce

Pokud knihovní funkce není vestavěná (built-in function), lze si prohlédnout její kód: např. `>> type jmenofce`. Built-in funkce jsou optimalizované a jejich kód není přístupný (např. `sin`).

A nyní už přehled některých užitečných funkcí:

Funkce pro zjišťování rozměrů proměnných

význam funkce	název	příklady

počet řádků a sloupců
(funkci lze volat s jedním vstupem a dvěma výstupy nebo dvěma vstupy (druhý vstup je skalár) a jedním výstupem)
	`size`	`>> [radky,sloupce]=size(N)`
		`>> % srovnejte:`
		`>> radky2=size(N,1)`
		`>> sloupce2=size(N,2)`

délka vektoru	`length`	`>> l=length(u)`

Funkce pro detekci stavu, resp. datového typu

Tyto funkce analyzují buď celou proměnnou, anebo její prvky (v případě vektorů či matic):

význam funkce	název	příklady

je proměnná číselná?	`isnumeric`	`>> isnumeric([NaN 2 -Inf])`
		`>> isnumeric('ahoj')`
		`>> ischar('ahoj')`
		
		`>> isreal([5 -3i 1 0])`
		
		`>> isnan(1./[-1 0 2 5])`
		`>> isprime([2 7 8 9 10 23])`
		`>> isfinite([NaN 2 -Inf])`

je proměnná řetězec?	`ischar`	
je proměnná pole buněk?	`iscell`	
je proměnná stuktura?	`isstruct`	
je proměnná objekt?	`isobject`	
je (celá) proměnná reálná?	`isreal`	
je PRVEK "nečíslo"?	`isnan`	
je PRVEK nekonečno?	`isinf`	
je PRVEK konečné číslo?	`isfinite`	
je PRVEK prvočíslo?	`isprime`	

Matematické funkce

Rozdělení podle jejich chování k maticím

Většina proměnných v MATLABu jsou matice komplexních čísel (typu *m*x*n*, přičemž *m*,*n*>0). Některé funkce vyžadují vstupní argumenty pouze určitého typu (např. jen skaláry), jiným na typu vstupních parametrů "nezáleží" a ke svým vstupním proměnným typu *m*x*n* se chovají třemi základními způsoby: skalárně (tj. po prvcích), vektorově (tj. po sloupcích) nebo maticově (podle pravidel lineární algebry). Lze je tedy rozdělit na:

a. **skalární funkce**
   - vstupem může být skalár, vektor nebo matice
   - funkce je aplikována *na každý prvek* svého vstupu

- výsledek má stejné rozměry jako vstup
- přehled těchto funkcí naleznete v knihovně elementárních funkcí: `>> help elfun`
- **goniometrické funkce**

význam funkce	název v MATLABu	příklady
sinus	`sin`	`>> sin([0 pi/2; pi/4 2*pi])`
kosinus	`cos`	`>> cos([0 pi/2; pi/4 2*pi])`
tangens	`tan` (ne `tg`!)	`>> tan([0 pi/2; pi/4 2*pi])`
kotangens	`cot`	`>> cot([0 pi/2; pi/4 2*pi])`
tytéž funkce ve stupních	`sind`, `cosd`, `tand`, `cotd`	
inverzní funkce	`asin`, `acos`, `atan`, `acot`	
hyperbolické funkce	`sinh`, `cosh`, `tanh`, `coth`	
...a další		

- **exponenciální a logaritmické funkce**

význam funkce	název v MATLABu	příklady
exponenciální funkce e^x	`exp`	`>> y = exp(1) % číslo e`
k-tá mocnina čísla *x*	operátor `^`	`>> y = 13^2`
funkce 2^x	`pow2`	`>> y = pow2(10) % 1024, též 2^10`
odmocnina	`sqrt`	`>> y = sqrt(150) % cca 12.2, též 150^0.5`
přirozený logaritmus ln(x)	`log`	`>> y = log(10) % cca 2.3`
dekadický logaritmus log (x)	`log10`	`>> y = log10(100) % 2`
dvojkový logaritmus $\log_2$ (x)	`log2`	`>> y = log2(10) % cca 3.32`
...a další		

- **funkce pro práci s komplexními čísly**

význam funkce	název	příklady

reálná část	`real`	`>> real(5-3i)`
		`>> real([5+i -2i 3i 1 2-i])`
		`>> imag(5-3i)`
		`>> abs(-2+4.5i)`
		`>> angle(i) % pi/2`
		`>> z=complex(5,-11)`
		`>> conj(z) % z'`

imaginární část	`imag`	
fázový úhel (od -π do π)	`angle`	
absolutní hodnota	`abs`	
komplex. číslo z dvojice reálných	`complex`	
komplexně sdružené číslo	`conj`	

- **zaokrouhlovací funkce**

význam funkce	název	příklady

zaokrouhlení na nejbližší celé číslo	`round`	`>> round([-5.7 -2.23 3.45 10.9])`
		`>> fix([-5.7 -2.23 3.45 10.9])`
		`>> floor([-5.7 -2.23 3.45 10.9])`
		`>> ceil([-5.7 -2.23 3.45 10.9])`

zaokrouhlení směrem k nule ("odseknutí" desetinné části)	`fix`	
zaokrouhlení k minus nekonečnu	`floor`	
zaokrouhlení k plus nekonečnu	`ceil`	

- **další funkce**

význam funkce	název	příklady
znaménko	`sign`	`>> sign([-5.7 -2.23 3.45 10.90])`
absolutní hodnota	`abs`	`>> y = abs(-12.3)`

zbytek po dělení (znaménko podle dělitele)	`mod`	`>> mod(10,3)`
		`>> mod(10,-3)`
		`>> mod(-10,-3)`
		`>> mod(-10,3)`
		`>> mod(34,0)`

zbytek po dělení	`rem`	zkuste výše uvedené pro `rem`

b. **vektorové funkce**

- vstupem je vektor nebo matice
- funkce je aplikována *na celý vektor* nebo *na každý sloupec matice*
- výsledkem je řádkový vektor typu 1x*n*
- přehled těchto funkcí: **`>> help datafun`**

význam funkce	název	příklady

suma (součet prvků vektoru)	`sum`	`>> min([0 pi/2; pi/4 2*pi]) %matice`
		`>> min([1 2 8 -15.3 16/2]) %vektor`
		`>> min([1 2 8 -15.3 16/2 3+2i]) % pozor, jde o komplex. vektor => nejprve abs`
		`>> V=[1,2;5,3;-3,0.8]`
		`>> m1=mean(V)`
		`>> m=min(min(V)) % m=min(m1)`
		`>> s=sum(V)`

minimum	`min`	
maximum	`max`	
průměr	`mean`	
směrodatná odchylka	`std`	`>> std([1.3 1.7 0.9 1.4])`
rozptyl	`var`	`>> var([1.3 1.7 0.9 1.4])`
seřazení hodnot	`sort`	`>> sort([1.3 1.7 0.9 1.4])`

c. **maticové funkce**

- vstupem je většinou algebraická matice, někdy vektor
- funkce jsou aplikovány na *celou matici*
- výsledkem je skalár, vektor nebo matice (podle pravidel lineární algebry)
- přehled těchto funkcí: `>> help matfun`

význam funkce	název v MATLABu	příklady

determinant	`det`	`>> M=[1,2,3;-3,4,1;0,8,4]`
		`>> d=det(M)`
		`>> I=inv(M)`
		`>> h=rank(M)`
		`>> a=1:7; b=5:11;`
		`>> dot(a,b) % SROVNEJTE: a'*b`

hodnost matice	`rank`	
norma matice	`norm`	
inverzní matice	`inv`	
pseudoinverze	`pinv`	
...a další		

Funkce pro vytváření matic `>> help elmat`

Funkce, které vytvářejí matice/vektory, nám mohou usnadnit práci se zadáváním matic/vektorů. Tyto funkce mají většinou jeden anebo dva vstupní parametry:

- příkazem `[matice] = nazevfce(m,n)` vznikne obdélníková matice typu *m*x*n*,
- příkazem `[matice] = nazevfce(n)` vznikne čtvercová matice řádu *n* (typu *n*x*n*).

význam funkce	název	příklady

matice obsahující nuly	`zeros`	`>> N=zeros(5,2)`
		`>> E=eye(7)`
		`>> R=rand(4,3)`
		`>> G=magic(4)`
		`>> sloupce=sum(G)`
		`>> radky=sum(G')`
		`>> diagonala=sum(diag(G))`

matice obsahující jedničky	`ones`	
jednotková matice	`eye`	
matice náhodných čísel od 0 do 1	`rand`	
magický čtverec (má shodné součty řádků, sloupců a diagonál)	`magic`	

Poznámka: práci s maticemi se budeme věnovat podrobněji cca za 3 týdny.

Funkce pro práci s datem a časem `>> help timefun`

význam funkce	název	příklady
aktuální datum a čas (vrací 1 číslo)	`now`	`>> date` `>> clock`
aktuální datum a čas (vrací vektor)	`clock`	
aktuální datum (vrací řetězec)	`date`	
převod data na řetězec	`datestr`	`>> datestr(now)`
kalendář na daný měsíc	`calendar`	`>> calendar` `>> eomday(2000,2)`
poslední den v měsíci	`eomday`	

stopky - začátek měření	`tic`	`% způsoby měření uplynulého času:`
		`>> tic, M=magic(32); toc`
		`>> t=cputime; M=magic(1200); e=cputime -t`
		`>> t=clock; M=magic(135); etime(clock,t)`
		Pozn.: údaje se liší při každém volání téhož příkazu!

stopky - konec měření	`toc`	
uplynulý čas procesoru (od spuštění MATLABu)	`cputime`	
uplynulý čas (mezi 2 časovými vektory)	`etime`	
pauza (v sekundách)	`pause`	

Další funkce si budeme uvádět podle potřeby.