

Skripty a funkce v MATLABu

cykly, uživatelsky definované funkce a další

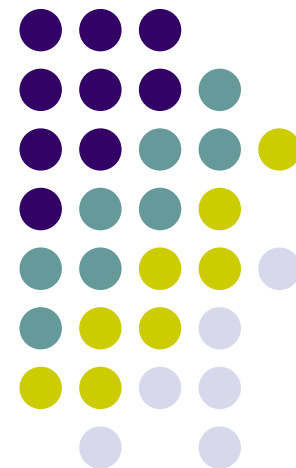
přednáška předmětu A2B99MAA, 11.týden



Miloslav Čapek

Pavel Hamouz

K13117, B2-819



Programování v MATLABu



- **MATLAB** je plnohodnotné programovací prostředí \Rightarrow umožňuje vytvářet **skripty a funkce**.
- **MATLAB** obsahuje celou řadu **nástrojů** pro efektivní návrh, vývoj a spravování rozsáhlých segmentů kódu.
- **MATLAB** implementuje **OO přístup** (toto paradigma není součástí přednášky).



Obsah přednášky

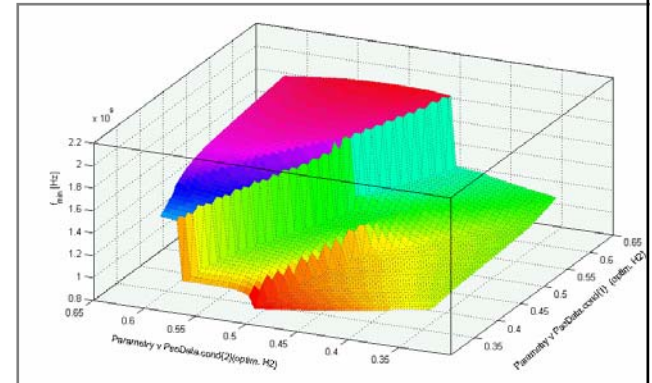


- **skripty** v Matlabu
- fibonacci.m (**příklad** skriptu)
- skript **startup.m**

I.

- **funkce** v Matlabu
- skript x funkce
- **hlavička** funkce, vstupní a výstupní parametry
- **typy** funkcí (nested, anonymní, ...)
- reference funkcí pomocí **handle** objektu
- **lazení** funkcí, nástroj **profiler**

II.



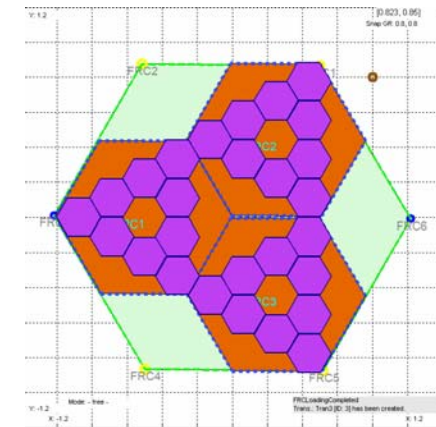
- **cykly** (for, while)
- využití cyklů v MATLABu
- **Dotazy**, diskuze
- **Literatura**
- Program cvičení

III.

Skripty v MATLABu



- **posloupnout** příkazů
(takové, které by šlo zapsat po jednom do příkaz. řádky)
- přípona **.m** (např. *fibonacci.m* – viz dále)
- mohou odkazovat na další m-soubory (skripty i funkce)
- skript i funkce mohou být volány **rekurzivně**
- M-soubory mohou být vytvořeny libovolným textovým editorem
- zpravidla využíváme editor Matlabu (příkaz *edit*)
- typický **příklad**: zpracování naměřených dat



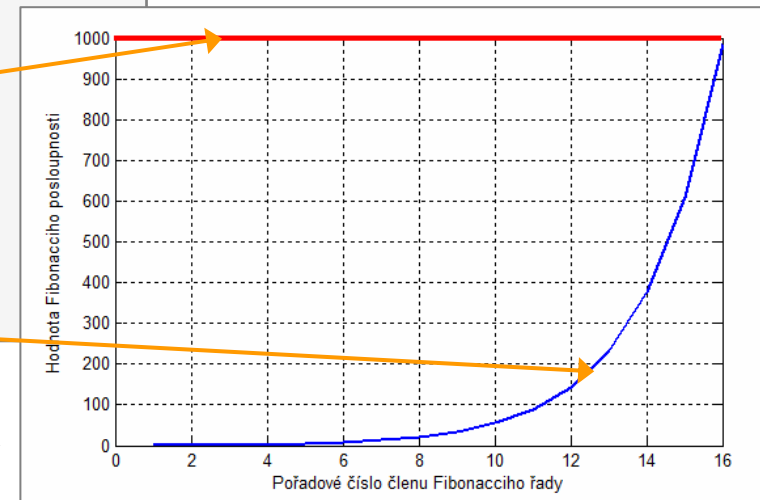
fibonacci.m – příklad skriptu



```
>> fibonacci % start z prikaz. radky
```

PŘ.

```
% M-skript pro vypocet Fibonacciho cisel  
% fibonacci.m  
f = [1 1]; i = 1;  
while f(i) + f(i+1) < 1000  
    f(i+2) = f(i) + f(i+1);  
    i = i + 1;  
end  
plot(f)
```



- Matlab **sekvenčně** vykonává příkazy

Data ve skriptu



- s daty je operováno **globálně** → vše je uloženo (načítá se) v globálním pracovním prostoru
- proměnné zůstávají v pracovním prostoru po **skončení** výpočtu
- vhodné pro dlouhé posloupnosti příkazů

```
>> fibonacci % start z prikaz. radky
```

- když Matlab hledá soubor *fibonacci.m*, platí následující:
 - ❑ proměnná ⇒ skript nesmí být proměnnou
 - ❑ vestavěná funkce ⇒ skript nesmí být built-in funkcí (např. *sin*)
 - ❑ lokální funkce ⇒ v daném adresáři
 - ❑ privátní funkce ⇒ **[private]** podsložka aktuálního adresáře
 - ❑ funkce MEX, dll, .p, nebo .m v aktuálním adresáři
 - ❑ funkce MEX, dll, .p, nebo .m ve vyhledávací cestě Matlabu (*path*)



Skript startup.m, komentáře



- tento skript se spouští **vždy** při startu Matlabu
- lze si do něj zadat konstanty, nastavení pracovního adresáře (*cd...*) nebo cesty k vlastním souborům (*addpath...*)
- umístění *startup.m*:
... \MATLAB\R2XXXx\toolbox\local\startup.m
- Komentáře ve skriptech:

```
clear; clc;
%{
1) Toto je viceradkovy komentar..
%}
a = 52;
c = a*pi; % 2) ... a jednoduchy komentar
d = c + ...
    ... 3) i zde lze komentovat (bez %)
25;
```



Práce se skripty



Funkce	Popis
disp(X)	Zobrazí výsledek s potlačením jména proměnné (X)
echo	Řídí zobrazování příkazů ve spuštěném skriptu
input	Uživatelský vstup
keyboard	Dočasné předání řízení klávesnici. Návrat do skriptu příkazem return.
pause	Přeruší skript a čeká na stisk libovolné klávesy.
pause(n)	Přeruší skript a čeká n sekund.
waitforbuttonpress	Přeruší skript a čeká na stisk libovolné klávesy nebo tlačítka myši.



Funkce v MATLABu



- efektivnější (a přehlednější) než skripty
- definovaný vstup a výstup, dostupnost komentáře
- nezbytná je hlavička funkce
- mohou obsahovat libovolný ASCII text vyhovující syntaxi Matlab jazyka
- mohou být volány z pracovního okna, nebo z prostoru jiné funkce (v obou případech musí být funkce v dostupná)
- každá funkce má vlastní pracovní prostor, vzniká při zavolání funkce a zaniká s poslední řádkou kódu funkce



Typy funkcí (M-soubory)



- vestavěné (buil-in)
 - ❑ nejsou uživateli přístupné k editaci
 - ❑ lze je pouze volat k výpočtům
 - ❑ optimalizované a uložené v jádře
 - ❑ zpravidla se jedná o často využívané (= elementární) funkce

sin, cos, abs, uipushtool, ...
- funkce v knihovnách Matlabu (zejm. adresáři toolbox)
 - ❑ „tematické“ (problémově zaměřené) funkce
 - ❑ je možné je editovat (nedoporučuje se!) / měnit – viz pdetool(box)

pdeinit, ...
- funkce vytvořené uživatelem
 - ❑ plně přístupné a editovatelné
 - ❑ funkčnost není garantována
 - ❑ povinné části: hlavička funkce
 - ❑ doporučené součásti funkce: popis funkce, datum poslední editace / verze, nepřehledné řádky je vhodné komentovat

vypocObsahuKruhu, ...

Typy funkcí (M-soubory)



```
function functA  
%FUNCTA - neobvykle, byt mozne
```

```
function functB(parIN1)  
%FUNCTB - napr. funkce s GUI vystupem, tiskem apod.
```

```
function parOUT1 = functC  
%FUNCTC - priprava dat, pseudonahodnych dat atd.
```

```
function parOUT1 = functD(parIN1)  
%FUNCTD - plnohodnotna funkce
```

```
function parOUT1 = functE(parIN1, parIN2)  
%FUNCTE - plnohodnotna funkce
```

```
function [parOUT1 parOUT2] = functF(parIN1, parIN2)  
%FUNCTF - plnohodnotna funkce s vice parametry
```



Volání funkce



Příklad:

```
function [parOUT1 parOUT2 parOUT3] = functG(parIN1,parIN2,parIN3)
%FUNCTG - 3 vstupy, 3 vystupy
```

- jakoukoliv funkci v Matlabu lze volat s **méně vstupními parametry** než je celkový počet v hlavičce (na rozdíl od jiných prog. jazyků)
- jakoukoliv funkci v Matlabu lze volat **pro méně výstupních parametrů** než je celkový počet v hlavičce (na rozdíl od jiných prog. jazyků)

```
>> [par01 par02] = functG(pIN1,pIN2,pIN3)
>> [par01 par02 par03] = functG(pIN1)
>> functG(pIN1,pIN2,pIN3)
>> [par01 par02 par03] = functG(pIN1,pIN2,pIN3)
```

→ vše je správně, pokud je na to funkce uzpůsobena (viz dále)



Práce s hlavičkou funkce:



Pomocné funkce:

Funkce	Popis
<code>varargin</code>	struktura na vstupu, pokud neznáme přesný počet vstup. parametrů
<code>varargout</code>	jako výše, avšak na výstupu
<code>nargin</code>	počet aktuálních vstupních parametrů do funkce
<code>nargout</code>	počet aktuálních výstupních parametrů z funkce

- funkce `varargin` a `varargout` používáme v hlavičce i v těle funkce
- funkce `nargin` a `nargout` používáme jen v těle funkce

Např.:

```
function [parOUT1 parOUT2] = functH(varargin)
%FUNCTH - funkce s promennym poctem vstup. parametru
```

```
function varargout = functI(parIN1, parIN2)
%FUNCTI - funkce s promennym poctem vystup. param.
```



Hlavička a parametry funkce (příklad)



Příklad:

```
function [parOUT1 parOUT2 parOUT3] = functG(varargin)
```

3 =
-1 =
1 =
struktura

```
>> nargout('functG') % pocet vystup. param.  
>> nargin('functG') % pocet vstup. param.  
% vraci -1, pokud je pocet promenny (varargin)  
>> nargin('sin') % pocet vstup. param. funkce sin  
>> pin{1} = 15; pin{2} = magic(3); pin{3} = 'hello';  
>> [po1 po2 po3] = functG(pin);
```

PŘ1.

```
function [...] = functG(varargin)  
% ... uvnitr funkce:  
! 3 = length(varargin{:})  
1 = nargin
```

PŘ2.

Požadují-li **pouze** 1. a 3. výstup. parametr, lze od verze R2009b:

```
>> [par01 ~ par03] = functG(14,magic(3),'test')
```

Typy souborů s funkcemi



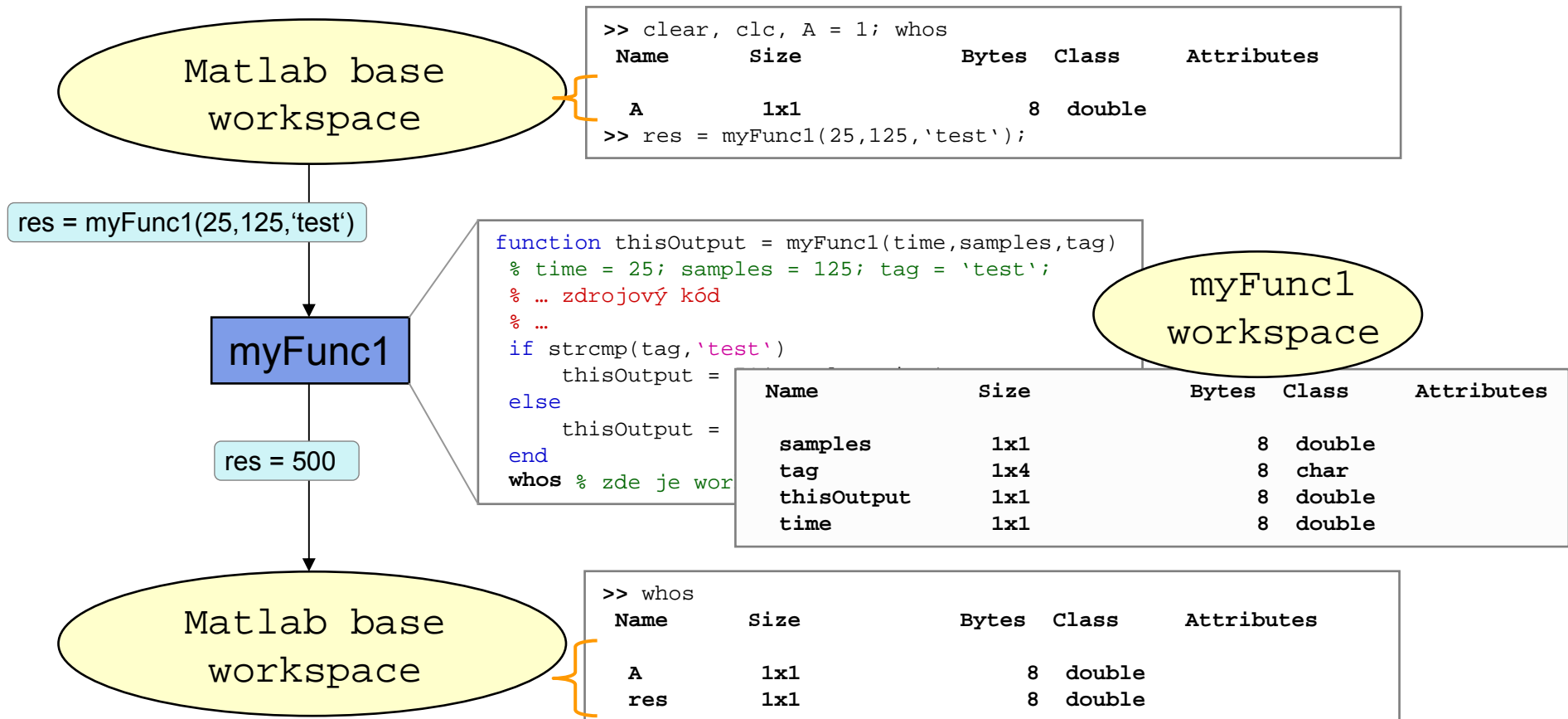
- M-soubory
- P-soubory (kompilované M-soubory, Matlab jim dává přednost)
- zkompilované soubory (.exe atp.)
 - pomocí nástroje Matlab Compiler
- MEX a dll soubory
- funkce jazyka C / Javy



Datový prostor funkce



- každá funkce disponuje vlastním pracovním prostorem (pro další účely – bez rozdílu typu: hlavní/vedlejší/atd.)



Datový prostor funkce - poznámka



- Když je funkce volána, vstupní proměnné nejsou kopírovány do pracovního prostoru funkce, pouze jsou jejich hodnoty funkci zpřístupněny.

Avšak je-li libovolná vstupní proměnná změněna, dojde k jejímu zkopírování do pracovního prostoru funkce. Z důvodu ušetření paměti a urychlení výpočtu je proto výhodnější z velkého pole nejprve vyjmout příslušné prvky a ty modifikovat, než přímo modifikovat velké pole a tím vyvolat jeho kopírování do pracovního prostoru funkce.

Pokud se použije stejná proměnná pro vstup i výstup, dojde k okamžitému kopírování této proměnné do pracovního prostoru funkce.

- Pro funkce platí všechny **zásady programování**, které byly probrány dříve (přetypování, alokace prostoru, indexování atd.)



Datový prostor funkce – poznámka č.2



- Voláme- li funkci **rekurzivně**, vytvoří se pro každé volání vlastní pracovní prostor.
- Sdílení proměnných pro více pracovních prostorů → **globální proměnné**. Jejich využívání se snažíme omezit na nezbytné minimum.
- Další možností, jak přistoupit do jiného pracovního prostoru, než v aktuální funkci, je příkaz **evalin** (viz dále).



Pravidla pro tvorbu M-souborů



- Jméno M-souboru a (hlavní) funkce by mělo být **stejné**.
- Jméno M-souboru může mít max. **31** znaků. Toto maximum může být omezeno systémem. Znaky nad tento limit Matlab ignoruje.
- Jméno funkce může začínat pouze písmenem. Až za ním mohou následovat číslice, písmena a podtržítka. Stejné pravidlo platí pro proměnné. Jména funkcí i proměnných jsou cAsE sEnSiTiVe.
- První řádka M-funkce se nazývá **řádka deklarace** (česky též hlavička funkce), musí obsahovat klíčové slovo *function* následované syntaxí funkce v obecném tvaru.
- První souvislý komentář za hlavičkou funkce slouží jako **nápovědný text pro funkci**.



Komentáře v M-funkci



- předpokládejme funkci:

hlavička funkce

nápověda pro funkci,
zobrazena na dotaz:
help myFcn1

```
function [dataOut idx] = myFcn1(dataIn, method)
%MYFCN1 Umožňuje vypočítat součet prvku matice.
% příklady volání, autor, další podobné funkce
% další části nápovědy
```

1. řádka (tzv. H1 řádka),
tuto řádku vyhledává příkaz
lookfor. Zpravidla obsahuje
jméno funkce velkými
písmeny a stručný popis
účelu funkce

```
matX = dataIn(:,1);
% nyní sečteme členy
SumX = sum(matX);
% CELL režim (musí být zapnut)
disp(num2str(SumX));
```

tělo funkce

KOMENTUJTE!

**% Komentáře zásadním způsobem
% zvyšují srozumitelnost funkce!!!**

Běh funkce



- Zpracování M-souboru končí poté, co je zpracována poslední řádka souboru, nebo pokud Matlab narazí na příkaz **return**.
- Funkce je ukončena také poté, co Matlab narazí na příkaz **error**. Tuto funkci využíváme, signalizujeme-li nesprávné užití funkce (zpravidla při inicializaci na počátku funkce).

zde je funkce ukončena

```
function res = myFcn2(matrixIN)
if isempty(matrixIN)
    error('matrixINcannotbeempty');
end
normMat = matrixIN - max(max(matrixIN));
```

- Funkce může podávat varování pomocí příkazu **warning**.
- Běh funkce lze dále ovlivnit pomocí ladících prostředků (dále).



Typy možných funkcí v M-souboru



Typ funkce	Popis
hlavní funkce	hlavička na první řádce, platí zásady výše
vedlejší funkce	volána hlavní funkcí, vlastní pracovní prostor
zanořená (nested) funkce	funkce je umístěna uvnitř funkce hlavní nebo vedlejší
handle funkce	využíváme reference na funkci (mySinX = @sin)
anonymní funkce	podobné výše (myGoniomFcn = @(x) sin(x)+cos(x))
OOP funkce	specifický přístup (konstruktor, metody set a get a další)

- jakákoliv **funkce** v Matlabu **může volat skript**, ten je potom vyhodnocován v pracovním prostoru volající funkce, nikoliv v základním pracovním prostoru Matlabu (jako obvykle)



Hlavní (primary) funkce



- za hlavní funkci je považována **první** funkce v M-souboru
- pod touto funkcí může být libovolný počet vedlejších funkcí
- zpravidla je hlavní funkce v M-souboru taková funkce, kterou jako jedinou můžeme volat z Matlabu, nebo z jiné funkce

Příklad primary funkce:

```
function y = average(x)
% AVERAGE Mean of vector elements.

y = sum(x)/length(x);      % Actual computation
```

Příklad volání funkce:

```
>> av = average([12 60 42]);
```

- jméno souboru většinou odpovídá jménu hlavní funkce (avšak nemusí tomu tak být - pak pro volání využíváme jméno souboru)

Vedlejší funkce



- první funkce v M-souboru je hlavní funkce
- všechny další jsou funkce vedlejší
- vedlejší funkce se ,vidí' navzájem, všechny vedlejší funkce potom ,vidí' funkce hlavní
- každá vedlejší funkce má vlastní pracovní prostor
- nápověda vedlejších funkcí není dostupná z pracovního okna

```
function [avg, med] = newstats(u) % primarni funkce  
n = length(u);  
avg = mean(u, n);  
med = absol(u);
```

```
function a = mean(v, n) % vedlejsi funkce  
a = sum(v)/n;
```

```
function m = absol (v) % vedlejsi funkce  
m = abs(v);
```



Nested funkce



- v Matlabu lze vložit funkci do jiné funkce → pak ji označujeme jako **vnořenou** (nested)
- do nested funkce lze vnořit další funkci
- funkce **nelze** vložit do těla funkčního bloku (if, else, elseif, switch, for, while, try a catch)

```
function x = A(p)
% jednoduchá
% nested funkce

...
    function y = B(q)
        ...
    end

...
end
```

```
function x = A(p)
% více jednoduchých
% nested funkcí

...
    function y = B(q)
        ...
    end

    function z = C(r)
        ...
    end

...
end
```

```
function x = A(p)
% vícenásobná
% nested funkce

...
    function y = B(q)
        ...
        function z = C(r)
            ...
        end
    end

...
end
```



Nested funkce



- při využití nested funkce/í musíme na konci každé funkce (nezáleží zda nested, primary atp.) použít klíčové slovo **end**
- nested funkci lze volat z funkce:
 - ❑ bezprostředně nad ní
 - ❑ na stejné úrovni
 - ❑ na (libovolné) nižší úrovni
- z nested funkce lze vytvořit handle (viz dále)
- **krom svého pracovního prostoru, má nested funkce přístup i k pracovnímu prostoru všech funkcí, do kterých je vnořena**

```
function x = A(p)
    function y = B(q)
        ...
        function z = C(t)
            ...
            end
        end
    ...
    end
    function u = D(r)
        ...
        function v = E(s)
            ...
            end
        end
    ...
    end
...
end
```

Privátní funkce



- v podstatě se jedná o vedlejší funkce
- jsou umístěny v podsložce [private] hlavní funkce
- k privátním M-souborům mají přístup pouze M-soubory z nejbližšího nadřazeného adresáře
- často se [private] využívá při větších aplikacích, nebo pokud chceme omezit viditelnost souborů uvnitř umístěných

Tyto funkce mohou volat pouze funkce *parTCM*, *preTCM* a *postTCM*.

parTCM volá funkce v [private]

```
...\TCMapp\  
private\  
    eigFcn.m  
    impFcn.m  
    rwgFcn.m  
parTCM.m  
preTCM.m  
postTCM.m
```

Handle funkce



- nejedná se o funkci jako takovou
- **handle** = reference na zvolenou funkci
- vlastnosti handlu umožňují volat i funkci, která není jinak viditelná
- s proměnnou, do níž uložíme handle (zde fS) můžeme volně nakládat

vytvoření handlu
(reference na funkci, o
jejíž umístění se dále
nemusíme starat)

```
>> fS = @sin;  
>> fS(pi/2)  
ans =  
    1
```

```
>> whos
```

Name	Size	Bytes	Class	Attributes
ans	1x1	8	double	
fS	1x1	32	function_handle	

Anonymní funkce



- umožňuje vytvořit funkci podobnou strukturu, bez nutnosti vytvářet m-file

PŘ1.

```
>> sqr = @(x) x.^2; % vytvoření anonymní funkce (handle)
>> vys = sqr(5); % x ~ 5, vys = 5^2 = 25;
```

má charakter
handle funkce

parametry

funkce
(funkční předpis)

PŘ2.

```
>> A = 4; B = 3; % parametry A,B musí být definovány
>> sumAxB = @(x, y) (A*x + B*y); % *zkuste si whos
>> vys2 = sumAxB(5,7); % x ~ 5, y ~ 7
vys2 = % vys2 = 4*5+3*7 = 20+21 = 41
41
```

- lze využívat i anonymní funkce bez vstupu
- umožňuje práci s poli anonymních funkcí
- více informací viz nápověda Matlabu



Příkaz evalin a assignin



- **evalin**: podobně jako funkce **eval**, zde se však uživatel může rozhodnout, v **jakém** pracovním prostoru bude řetězec vyhodnocen
 - volací prostor ('caller'): prostor, z něhož byla volána současná funkce
 - základní prostor ('base'): pracovní prostor příkazového okna Matlabu
- **assignin**: umožňuje přiřadit výsledek určitého výrazu v současném pracovním prostoru do proměnné v jiném pracovním prostoru

```
strg = '3*x+2';  
A = evalin('caller',strg); % vyhodnoti ve volanem prac. prostoru a  
% vysledek vrati do promenne A v soucasnem pracovnim prostoru  
B = evalin('base',[strg '-2']); % vyhodnoti (strg-2) v zakladnim  
pracovnim prostoru a vysledek vrati do promenne A v soucas. p.p.  
% tedy platí: B = A - 2;  
assignin('caller','finalRes1',A); % priradi obsah promenne A ze  
% soucasneho prostoru do promenne finalRes1 ve volanem p.p.  
assignin('base','finalRes2',B); % obsah B do zakl. p.p.
```



Příkaz inputname



- umožňuje určit jména proměnných použitých při volání funkce

volání funkce:

```
>> y = myFunc1(xdot, time, sqrt(25));
```

uvnitř
funkce:

```
function output = myFunc1(par1,par2,par3)
...
p1str = inputname(1); % p1str = 'xdot';
p2str = inputname(2); % p2str = 'time';
p3str = inputname(3); % p3str = '';
...
```

vrací prázdné pole,
protože sqrt(25) není proměnná

Lazení funkce



- Můžeme nalézt **dva typy** chyb:
 - a) syntaktické – pomůže odhalit M-Lint při tvorbě programu, příp. je odhalí kompilace funkce
 - b) chyby za běhu programu (run-time error) – jsou hůře odhalitelné, využíváme následující strategie:
 - 1) odstraníme **středníky**
 - 2) přidáme na vhodná místa **disp** a sledujeme mezivýstup
 - 3) změníme funkci na **skript**
 - 4) použijeme **ladící prostředky** Matlabu
(funkce *dbstop*, *dbclear*, *dbstatus*, *dbtype* a *dbstep*; všechny jsou dostupné i z menu Matlab editoru)



Nástroj profile



- slouží k analýze běhu programu

Příkaz	Popis
<code>profile function</code>	analyzuj funkci <i>function</i>
<code>profile report</code>	zobraz zprávu o analýze
<code>profile reset</code>	resetuje výsledky (vhodné před startem nového rozboru)
<code>profile off</code>	vypne nástroj profile – znemožní analýzu
<code>profile on</code>	zapne nástroj profile
<code>profile clear</code>	vymaže data z předchozí analýzy
<code>profile done</code>	ukončí analýzu

pozastavení, vymazání a opětovný start služby

start testované
funkce

```
>> profile off, profile clear, profile on  
>> ResTable = EvalInFem('stat',FRC_B);  
>> profile report
```

zobrazí zprávu (další slide)



Nástroj profile – uživatelské rozhraní



```

306
307 %% Stand-alone solution (once)
1 308 if strcmp(request,'stat')
1 309     if nargin == 3
310         [results fem report] = evalFcn(Input,parl); % ,han
1 311     else % nargin = 2 ('stat' Structure)
1 312         [results fem report] = evalFcn(Input); % ,handle
1 313     end
< 0.01 1 314     if report.event == 1 % error event
315         results.done = report.tag;
316         checkError(report.mess);
317         return
318     end
319
320 %% Multi-solution (for PSO)
321 elseif strcmp(request,'pso')
322     if ~isfield(Input,'type') || ~strcmp(Input.type,'psopt')
323         ~(nargin == 4)
324         err = '#1: Error: Bad input data (x ''psopt'' type
325         results.score = Inf;
326         results.done = -1;
327         fem = NaN;
328         checkError(err);
329         return
    
```

EvalInFem>evalFcn (1 call, 13.690 sec)
 Generated 27-Apr-2010 17:32:45 using cpu time.
 M-subfunction in file d:\FEL - work\Matlab\mfiles\EvalInFem.m
[Copy to new window for comparing multiple runs](#)

Refresh

Show parent functions
 Show busy lines
 Show child functions
 Show M-Lint results
 Show file coverage
 Show function listing

Parents (calling functions)

Function Name	Function Type	Calls
EvalInFem	M-function	1

Lines where the most time was spent

Line	Line Number	Code	Calls	Total Time	% Time	Time Plot
312	494	fem.sol=femeig(fem, ...	1	2.674 s	19.5%	█
177	492	fem=multiphysics(fem);	1	2.295 s	16.8%	█
186	493	fem.xmesh=meshextend(fem);	1	2.082 s	15.2%	█
178	526	drawPostFcn(fem,solNo);	1	2.076 s	15.2%	█
164	433	fem = makeGeomFcn(FRC);	1	1.433 s	10.5%	█
All	All other lines			3.129 s	22.9%	█
Tot	Totals			13.690 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
femeig	M-function	1	2.668 s	19.5%	█
multiphysics	M-function	1	2.295 s	16.8%	█
EvalInFem>drawPostFcn	M-subfunction	1	2.076 s	15.2%	█



Cykly



- **opakování** určité činnosti vícekrát
- v Matlabu známe 2 typy cyklů:
 - ❑ **for** (nejčastěji využíván, známe počet opakování)
 - ❑ **while** (známe podmínku, za které (ne)pokračovat)
- více než v jiných případech, je vhodné v kontextu cyklů dodržovat **programátorské zásady**:
 - ❑ alokujeme prostor (matice) dostatečné velikosti
 - ❑ cykly by měly být řádně ukončeny
 - ❑ mnohdy stačí vhodně upravit pole (1D → 2D, 2D → 3D pomocí funkce *repmat* a operaci provést maticově, řádově mnohem rychlejší nežli využití cyklů)
 - ❑ u *while* cyklu zajistíme splnitelou podmínku
 - ❑ vždy si položíme otázku: Je cyklus skutečně nezbytný??
 - ❑ a další obecně platné zásady (viz programování)



Cyklus FOR



- pro předem daný počet opakování skupiny příkazů

```
for i = vyraz
    prikazy
end
```

- `vyraz` je matice (může být vícerozměrná). Sloupce této matice jsou postupně přiřazovány proměnné `i` a následně jsou provedeny příkazy.

```
function chNum = myFunc1(MeshStruct,options)
...
chNum = zeros(MeshSize,FreqSize); % alokace
for k = 1:FreqSize
    chNum(:,k) = eigFcn(ZMatrix,MeshStruct,Freq(k));
end
```



Cyklus WHILE



- opakuje sadu příkazů v těle cyklu v závislosti na logické podmínce

```
while vyraz
    prikazy
end
```

- příkazy se opakují tak dlouho, dokud jsou všechny prvky ve výrazu (`vyraz` může být i vícerozměrná matice) nenulové (pokud jde o relační výraz, pak dokud jsou rovny *true*)
- pokud `vyraz` není skalár, můžeme ho redukovat pomocí funkcí *any* nebo *all*



Příkaz break a continue



- umožňují opustit (**break**), nebo naopak pokračovat (**continue**) v cyklu
- **break** ani **continue** nejsou definovány vně cyklu (zde použijte příkaz **return**)
- v případě zanořených cyklů je příkaz **break** účinný pouze v cyklu, kde se vyskytuje

Příklad ukazuje načítání souboru `fft.m` pomocí `while` cyklu. Příkaz `break` je využit k ukončení cyklu ihned, jak funkce nalezne prázdnou řádku.

```
fid = fopen('fft.m', 'r');
s = '';

while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) || ~ischar(line), break, end
    s = sprintf('%s%s\n', s, line);
end
disp(s);

fclose(fid);
```



Využití cyklů v Matlabu



- Bude doplněno



Literatura



- K.Zaplatílek, B.Doňar: MATLAB - Matlab pro začátečníky
- K.Zaplatílek, B.Doňar: MATLAB - Tvorba uživatelských aplikací
- Mathworks: Programming (dokumentace – pdf)
- Mathworks: Programming Tips (dokumentace – pdf)
- J. Kiusalaas: Numerical Methods in Engineering with Matlab
- S.R.Otto, J.P.Denier: An Introduction to Programming and Numerical Methods in Matlab
- B.R.Hunt, R.L.Lipsman, J.M.Rosenberg: A Guide to Matlab

- fórum na www.mathworks.com
- internet, google



Děkuji za pozornost

`miloslav.capek@fel.cvut.cz`

