Seminář 5

Programování II

- funkce
- chyby
- optimalizace

Funkce

Základní funkce

syntaxe:

```
function [vystupni argumenty] = nazev_funkce(vstupni argumenty)

% Komentar popisujici funkci (lookfor)

% ostani komentare (help)

telo funkce

% klasicke komentare

end
```

- zvláštní soubor (`.m`) - funkce se musí jmenovat stejně jako soubor
- na konci skriptu - je viditelná pouze v rámci skriptu, lokální funkce

Vytvořte funkci `funkce1()`, která bude brát dva argumenty a bude vracet menší a větší z předaných argumentů. Napište komentář popisující funkci a vyzkoušejte `help` na tuto funkci.

```
a = 1
```

```
a = 1
```

```
b = -4
```

```
b = -4
```

```
[mensi, vetsi] = funkce1(a, b)
```

```
mensi = -4
vetsi = 1
```

```
help funkce1
```

```
  funkce1 Vraci minimalni a maximalni hodnotu ze dvou argumentu
     Funkce jako vstup bere dve cisla a vraci mensi a vetsi z nich
```

```
```

Pokud bychom volali jen `funkce1(a,b)` a výstup nenavázali na proměnné, hodnota prvního výstupního argumentu je navázána na `ans` (vypsána). *Vyzkoušejte.*

`lookfor klicove_slovo`

Pokud funkce nevrací žádnou hodnotu vypadá hlavička následovně:

`function nazev_funkce(vstupni argumenty)`

Pokud funkce nebere žádné vstupní argumenty, je možné ji volat bez závorek

`nazev_funkce() i nazev_funkce`

!Pozor! uživatelsky definované funkce mají přednost před vestavěnýma funkcema (přepíšou vestavěné funkce). *Vyzkoušejte si vytvořit funkci min, která bude vracet největší prvek.*

Ve skutečnosti je přednost následující: proměnné, nested funkce, lokální funkce, funkce v current folder, funkce v search path.

Vyzkoušejte si vytvořit proměnnou min a otestujte si, jak se bude matlab chovat. (`help min`*)*

Hledání funkce -- nejprve se hledá v aktuálním skriptu, aktuální složce (current folder) a pak případně v search path (Set Path), ve skriptu je možné přidat cestu k souborům (`addpath`)

Variadické funkce

Námi definované funkci můžeme předat i méně argumentů, než je definováno vstupními argumenty. Aby funkce pracovala správně, musíme však v těle funkce zbylé argumenty nastavit. Pro zjištění počtu předaných argumentů použijeme proměnnou `nargin`.

Podívejte se na `funkce2()`. Funkce vrací součet předaných argumentů. S tím, že bere až 3 vstupní argumenty.

```
funkce2()
```

```
Nespravny pocet argumentu
ans = 0
```

```
funkce2(1)
```

```
ans = 1
```

```
funkce2(1, 1)
```

```
ans = 2
```

```
funkce2(1, 1, 1)
```

```
ans = 3
```

```
funkce2(1, 1, 1, 1)
```

```
Error using funkce2
Too many input arguments.
```

Zcela variadické funkce

Pokud chceme zcela variadickou funkci, použijeme jako výpustku klíčové slovo `varargin.` Předané argumenty budou navázány na tuto proměnnou. Proměnná je typu `cell array` (o něm budeme mluvit později). Pro převod na matici použijeme `cell2mat().`

Podívejte se na funkci `funkce3()`, která vrací součet předaných argumentů. Mimo to také vypisuje jejich počet.

```
funkce3()
```

```
Pocet predanych argumentu : 0
ans = 0
```

```
funkce3(1)
```

```
Pocet predanych argumentu : 1
ans = 1
```

```
funkce3(1, 1)
```

```
Pocet predanych argumentu : 2
ans = 2
```

```
funkce3(1, 1, 1)
```

```
Pocet predanych argumentu : 3
ans = 3
```

```
funkce3(1, 1, 1, 1)
```

```
Pocet predanych argumentu : 4
ans = 4
```

Napište funkci, která bude brát argument `n` a libovolný počet nepovinných argumentů. Funkce vrátí `n` tý nepovinný argument.

Funkce vracející proměnlivý počet argumentů

Už jsme si ukázali, že funkce mohou vracet různý počet argumentů. Například `size()`

```
M = zeros(10,20);

% vraci vektor
velikost = size(M)
```

```
velikost = 1×2
    10    20
```

```

% vraci hodnoty m a n
[m, n] = size(M)
```

```
m = 10
n = 20
```

Ve funkci můžeme pomocí `nargout` zjistit počet očekávaných výstupních argumentů a podle toho vypočítat jen některé výstupní argumenty.

Vytvořte funkci `je_prvek(p, vektor)`, která bude vracet informaci o tom, zda je prvek `p` ve vektoru `vektor`. V případě, že budou očekávány dva výstupní argumenty, vrátí i index nalezeného prvku (pokud nenalezen vrátí třeba 0).

Můžeme vytvořit i funkci s libovolným počtem výstupních argumentů. K tomu využijeme proměnnou `varargout`. Protože je tato proměnná `cell array`, vrátíme se k ní později.

Inline funkce - starý přístup

Syntaxe: `inline(vyraz, argumenty)`

```
f = inline('x^2', 'x');
f(5)
```

```
ans = 25
```

Anonymní funkce

Syntaxe: `@(argumenty)vyraz`

```
f = @(x)x^2
```

```
f = function_handle with value:
    @(x)x^2
```

```
f(5)
```

```
ans = 25
```

Ukazatele na funkce

Uvození symbolem `@`

```
f = @sin
```

```
f = function_handle with value:
    @sin
```

Funkce vyšších řádů

Funkce, které berou jako argument funkci, nebo funkci vrací. Už víme, jak získat ukazatel na funkci i jak, jí pak tuto funkci zavolat. Podívejte se na funkci `funkce4()`.

```
f = @(x)x^2;
b = funkce4(f,5)
```

```
b = 25
```

```
%
```

```
b = funkce4(@(x)x^2,5)
```

```
b = 25
```

```
%
b = funkce4(@sin,pi)
```

```
b = 1.2246e-16
```

Vytvořte funkci, která bude brát argument `n` a vrátí funkci, která bude vracet `n` tou mocninu předaného čísla.

Nested funkce (vnořené funkce)

Nested funkce je funkce, která je definovaná uvnitř jiné funkce.

Synaxe:

```
function [vystupni argumenty] = rodicovska_funkce(vstupni argumenty)
telo funkce
function [vystupni argumenty] = nested_funkce(vstupni argumenty)
telo funkce
end
end
```

Nested funkce není možné definovat uvnitř cyklů, větvení nebo `try-catch` konstrukce.

Nested funkce mohou obsahovat další nested funkce.

Nested funkce mohou používat a modifikovat proměnné rodičovské funkce. (ale ne naopak)

Podívejte se na funkce `funkce5()` a `funkce6()`.

```
funkce5()
```

```
x = 2
```

```
funkce6()
```

```
x = 2
```

Rodičovská funkce může obsahovat více nested funkcí. Ty však spolu nesdílí proměnné.

Podívejte se na funkci `funkce7()`.

```
funkce7()
```

```
x = 1
x = 2
```

Výstupní argumenty nested funkce nejsou viditelné v rodičovské funkci.

Podívejte se na funkci `funkce8()`.

```
funkce8()
```

```
Unrecognized function or variable 'y'.

Error in funkce8 (line 8)
    y
```

Vytvořte funkci, která bude jako vstup brát dva argumenty `a` a `b` a vrátí ukazatel na nested funkci `linearni`, která bude počítat lineární funkci `ax + b`.

Optimalizace

Správné použití Matlabu

Vektorizace, prealokace paměti

```
% jednoduchy zpusob
for i=1:m
  for j=1:n
    M(i,j) = 0;
  end
end

% vektorizace
M(1:m, 1:n) = 0;
```

Hledání neoptimálních částí - Profilování

Ve scriptech (ne v live scriptech) můžeme hledat neoptimální části kódu pomocí `Run and time`. Analyzovat kód můžeme i v live scriptech pomocí profileru (ten se spustí i v případě volby `Run and time`).

Vyzkoušelte profiler pro `skript_profilovani.m`

```
skript_profilovani
```

Ve cviční pak tento kód optimalizujte.

Ladění

Chyby

syntaktické

Matlab je odhalí a zvýrazní (viz dříve)

```
 %fprintf(ahoj svete)
```

Logické

Sémantické chyby, které jsou způsobeny špatným algoritmem.

Vypište všechna lichá čísla od 1 do 10.

```
for i = 1 : 10
    if rem(i, 2) == 0
        disp(i);
    end
end
```

Chyby za běhu programu

```
soubor = input('zadejte nazev souboru: ', 's');
fid = fopen(soubor, 'r');
a = fread(fid);
```

```
Error using fread
Invalid file identifier. Use fopen to generate a valid file identifier.
```

Debugger

Klasické nástroje, jako v jiných IDE.

Uvažujme, že máme naprogramovanou funkci `comb(k, n)`, která vrací počet k prvkových kombinací čísla n.

```
pocet = comb(3, 4)
```

```
pocet = 4
```

Funkce, ale nefunguje správně pro následující vstup.

```
pocet = comb(2, 4)
```

```
pocet = 24
```

Analyzujte funkci a najděte, kde je chyba.

Try a catch

I když jsme opravili chybu algoritmu, funkce nepracuje úplně, jak by měla.

```
pocet = comb(4, 2)
```

```
Error using factorial
N must be an array of real non-negative integers.

Error in comb (line 6)
    c = factorial(n - k);
```

Funkci můžeme opravit přidáním testu, zda není k větší než n.

```
n = 4;
k = 3
```

```
k = 3
```

```
if n > k
    comb(k,n)
else
    comb(n,k)
end
```

```
ans = 4
```

Případně můžeme použít `try-catch` blok, který znáte z jiných programovacích jazyků.

```
try
    comb(n,k)
catch
    comb(k,n)
end
```

```
ans = 4
```