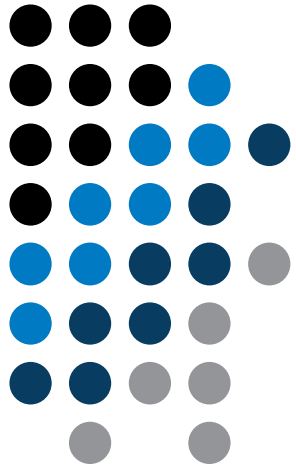
A0B17MTB – Matlab

Část #7

Miloslav Čapek
miloslav.capek@fel.cvut.cz

Filip Kozák a Viktor Adler

Katedra elektromagnetického pole
B2-626, Dejvice

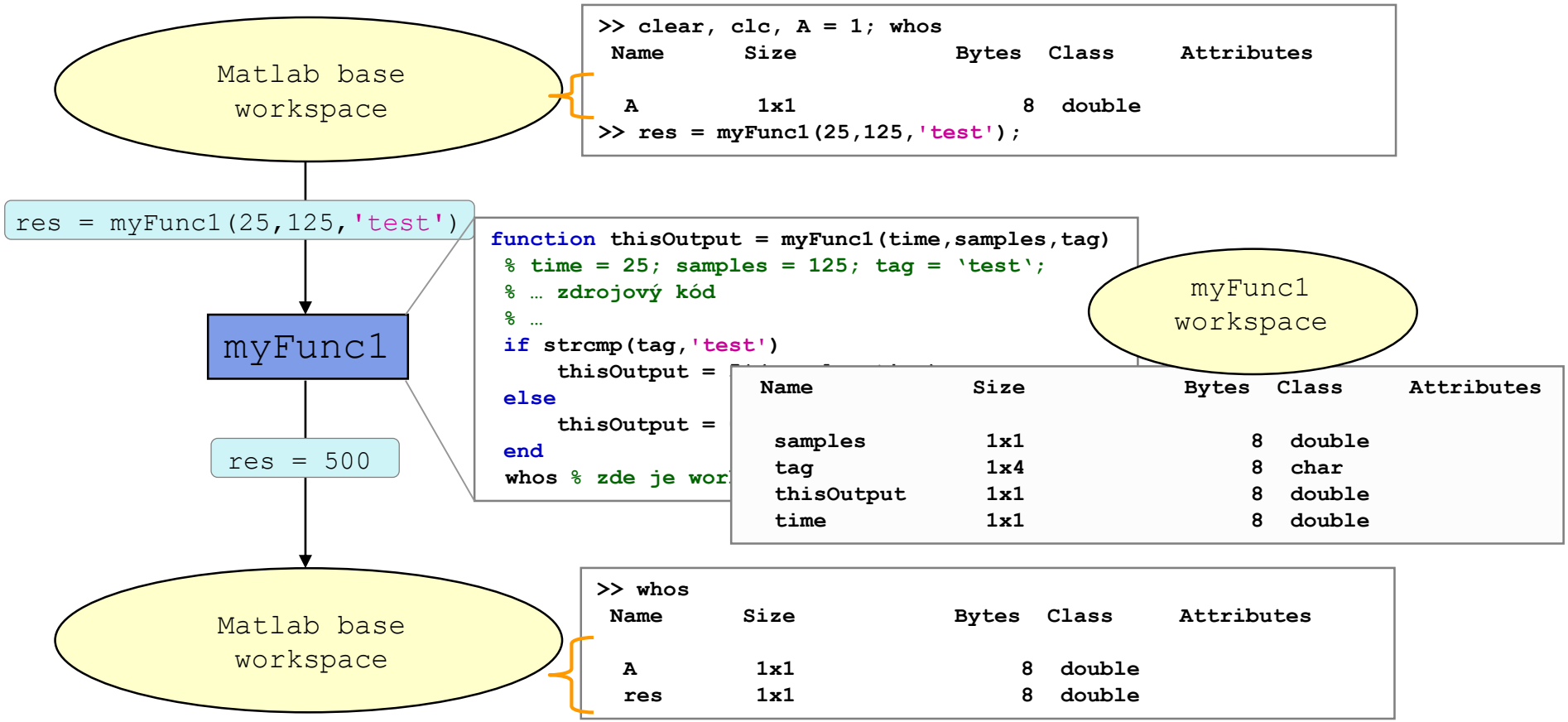


Naučíte se …

Funkce #2

Pracovní prostor funkce

- každá funkce disponuje vlastním pracovním prostorem



Datový prostor funkce #1

- 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
  - (za předpokladu, že je vstupní proměnná ve skriptu jakkoliv modifikována, jinak jsou proměnná vstupující do funkce a proměnná vystupující z funkce referencí na totožná data)

Datový prostor funkce #2

- pro funkce platí všechny zásady programování, které byly probrány dříve (přetěžování, přetypování, alokace prostoru, indexování atd.)
  - v případě přetížení built-in funkce lze stále použít `builtin`
- voláme-li funkci rekurzivně, vytvoří se pro každé volání vlastní pracovní prostor
  - pozor na přílišný nárůst počtu pracovních prostorů (hlídá Matlab, < 500)
- sdílení proměnných pro více pracovních prostorů
  → globální proměnné
  - pozor s jejich využíváním (není doporučeno), nejsou zpravidla potřeba

Běh funkce

- kdy je funkce ukončena?
  - Matlab interpret se dostane až na poslední řádku
  - interpret narazí na klíčové slovo `return`
  - interpret narazí na chybu (lze vyvolat i pomocí příkazu `error`)
  - stisknuto CTRL+C

```
function res = myFcn2(matrixIn)

if isempty(matrixIn)
    error('matrixInCannotBeEmpty');
end
normMat = matrixIn - max(max(matrixIn));

if matrixIn == 5
    res = 20;
    return;
end
```

Počet vstup. a výstup. proměnných

- počet proměnných do funkce vstupujících a z funkce vystupujících určí funkce `nargin` a `nargout`
- tyto funkce mj. umožňují navrhnout hlavičku funkce tak, aby byl umožněn různý počet vstupů / výstupů

```
function [out1, out2] = myFcn3(in1, in2)

if nargin == 1
    % do something
elseif nargin == 2
    % do something
else
    disp('Bad inputs!');
end
```

Počet vstup. a výstup. proměnných

500 s ↑

- upravte funkci `fibonacci.m` tak, aby umožňovala více možných vstupů a výstupů:
  - funkce lze volat bez vstupních proměnných
    - pak je řada generována tak, že poslední prvek je menší než 1000
  - funkci lze volat s jedním vstupním parametrem `in1`
    - pak je řada generována tak, že poslední prvek posloupnosti je menší než `in1`
  - funkci lze volat se dvěma vstupními parametry `in1`, `in2`
    - pak je generována tak, že poslední prvek posloupnosti je menší než `in1` a zároveň první 2 členy řady jsou udány vektorem `in2`
  - funkci lze volat bez výstupních proměnných nebo s jednou výstupní proměnnou
    - pak vrací vygenerovanou řadu
  - funkci lze volat s dvěma výstupními proměnnými
    - pak vrací vygenerovanou řadu a objekt třídy `Line`, který je vykreslen v grafu

```
hndl = plot(f);
```

Počet vstup. a výstup. proměnných

Typy funkcí z hlediska syntaxe

Typ funkce	Popis
hlavní	hlavička na první řádce, platí zásady výše, jediná z m-souboru viditelná zvenčí
vedlejší (subfunction)	volána hlavní funkcí, má vlastní pracovní prostor (workspace), všechny další funkce krom hlavní, lze i do `[private]`
zanořená (nested)	funkce je umístěna uvnitř funkce hlavní nebo vedlejší, vidí WS funkce nad sebou
handle	využíváme reference na funkci (`mySinX = @sin`)
anonymní	podobné výše (`myGoniomFcn = @(x) sin(x)+cos(x)`)
OOP	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)
- pořadí vedlejších funkcí není důležité (logická návaznost!)
- nápověda lokálních funkcí není dostupná příkazem `help`

Funkce – vedlejší funkce

- vedlejší funkce mohou být volány pouze funkcí hlavní
  - všechny tyto funkce mohou (měly by) končit klíčovým slovem `end`
  - využíváme pro opakující se funkcionalitu uvnitř funkce hlavní (pomáhá nám s atomizací celé úlohy, která je poté přehlednější)
  - funkce vedlejší se „vidí“ navzájem, mají vlastní pracovní prostory
  - velmi často využíváme při programování GUI pro zpracování jednotlivých akcí grafických prvků (callbacks)

```
function x = model_ITUR901(p,f)
% main function body
% ...
% ...
end

function y = calc_parTheta(q)
% function body
end
```

Funkce – nested funkce

- zanořené funkce (nested) jsou vytvořeny uvnitř jiné funkce
  - umožňuje nám to využít pracovní prostor funkce nadřazené a pracovat efektivně s (zpravidla malým) pracovním prostorem funkce zanořené
  - funkce nelze vložit do těla funkčních bloků (`if-else-elseif` / `switch-case` / `for` / `while` / `try-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
```

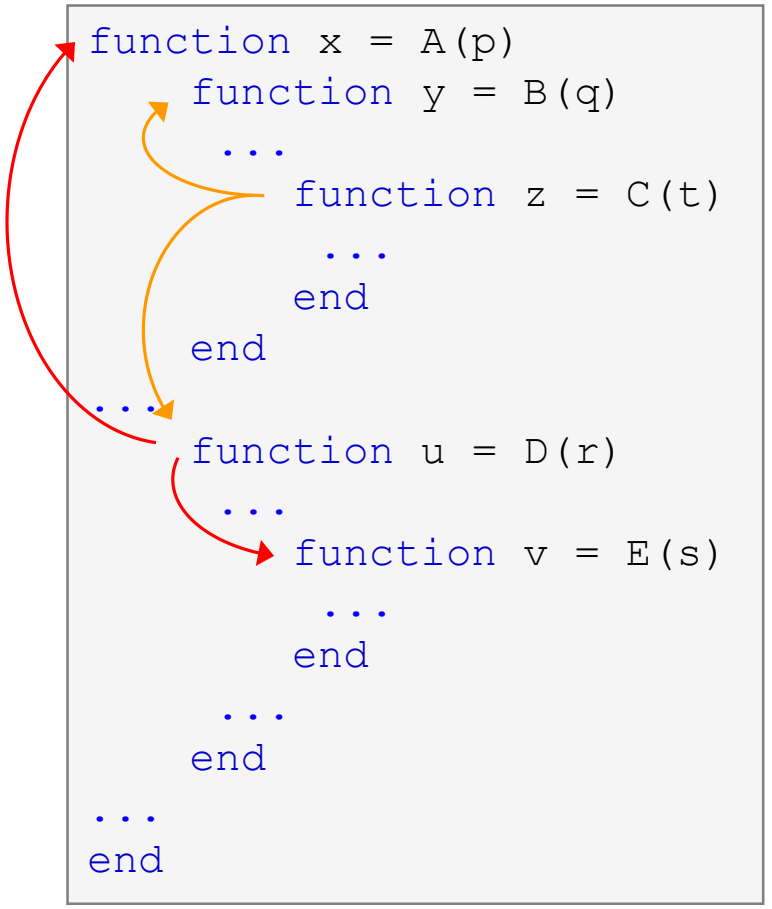
```
function x = A(p)
% vícenásobná
% nested funkce
...
    function y = B(q)
      ...
        function z = C(r)
          ...
        end
      ...
    end
...
end
```

Funkce – nested funkce: volání

- krom svého pracovního prostoru, má zanořená funkce přístup i k pracovnímu prostoru všech funkcí, do kterých je vnořena

- zanořenou funkci lze volat z funkce:
  - bezprostředně nad ní
  - na stejné úrovní
  - na (libovolné) nižší úrovni

- ze zanořené funkce lze vytvořit handle
  - viz dále



Privátní funkce

- v podstatě se jedná o vedlejší funkce
- jsou umístěny v podsložce `[private]` hlavní funkce

- k privátním funkcím mají přístup pouze funkce z nejbližšího nadřazeného adresáře
  - často se `[private]` využívá při větších aplikacích, nebo pokud chceme omezit viditelnost souborů uvnitř umístěných

Tyto funkce mohou být volány pouze funkcemi: `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
- typicky jsou potřeba jako vstupní parametr do jiných funkcí

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

```
>> whos
 Name        Size                Bytes  Class                Attributes

  ans         1x1                     8  double
  fS          1x1                    32  function_handle
```

Anonymní funkce

- umožňují vytvořit handle na funkci, která není definovaná v samostatném souboru
  - definice funkce musí jít napsat jako jediný vykonatelný výraz

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

  - anonymní funkce může mít i více vstupních parametrů

```
>> A = 4; B = 3; % parametry A,B musí být definovány
>> sumAxBy = @(x, y) (A*x + B*y); % definice fuknce
>> vys2 = sumAxBy(5,7);     % x = 5, y = 7
% vys2 = 4*5+3*7 = 20+21 = 41
```

  - anonymní funkce si kromě funkčního předpisu uchovává i potřebné proměnné

```
>> A = 4;
>> multAx = @(x) A*x;
>> clear A
>> vys3 = multAx(2);
% vys3 = 4*2 = 8
```

- `>> doc Anonymous Functions`

Funkce – pokročilé techniky

- neznáme-li vůbec počet vstupních nebo výstupních proměnných, využijeme `varargin` a `varargout`
  - je nutné upravit hlavičku funkce
  - rovněž si musíme vstupní / výstupní proměnné přiřadit z proměnných `varargin` / `varargout`

```
function [parOut1, parOut2] = funcA(varargin)
%% proměnný počet vstupních parametrů
```

```
function varargout = funcB(parIn1, parIn2)
%% proměnný počet výstupních parametrů
```

```
function varargout = funcC(varargin)
%% proměnný počet vstupních i výstupních parametrů
```

```
function [parOut1, varargout] = funcC(parIn1, varargin)
%% proměnný počet vstupních i výstupních parametrů
```

Funkce – `varargin`

- typické využití: funkce s mnoha volitelnými parametry / vlastnostmi
  - např. GUI (kupř. funkce jako `stem`, `surf` atp. obsahují `varargin`)
- proměnná `varargin` je vždy typu `cell`, i když obsahuje pouze jedinou položku
- funkce `nargin` v těle funkce vrací počet parametrů vstupujících do funkce při volání
- funkce `nargin(fx)` vrací počet vstup. parametrů v hlavičce funkce
  - při použití `varargin` v hlavičce vrátí zápornou hodnotu

```
function plot_data(varargin)

nargin
celldisp(varargin)

par1 = varargin{1};
par2 = varargin{2};
% ...
```

Proměnný počet vstupních parametrů

500 s ↑

- vytvořte novou funkci s hlavičkou v následující podobě:

```
function plot_data(data, varargin)
```

přičemž proměnné ve `varargin` jsou vždy po dvojicích (vlastnost a její hodnota)

- ověřte nejprve, že v proměnné `data` jsou číselné údaje (funkce `is*`)
- pokud ano, vykreslete data, pokud ne vypište chybu a funkci ukončete

```
hndl = plot(data);
```

- poté postupně (dokud není `varargin` prázdný) nastavujte jednotlivé vlastnosti na zadané hodnoty:

```
set(hndl, vlastnost, hodnota);
```

- podporované vlastnosti viz >> `doc line` nebo následující slajd

Proměnný počet vstupních parametrů

- vybrané vlastnosti, které lze využít pro testování funkce:
  - zamyslete se, jak funkci upravit, aby dokázala zachytit případné chybné vlastnosti (neexistující / špatně zadané / se špatnou hodnotou)?

vlastnost	hodnota
`Color`	[R G B]
`LineWidth`	0.1 – …
`Marker`	`'o'`, `'*'`, `'x'`, …
`MarkerSize`	0.1 – …
a další …	

```
>> plot_data(magic(3),...
       'Color',[.4 .5 .6],'LineWidth',2);
>> % nebo
>> plot_data(sin(0:0.1:5*pi),...
       'Marker','*','LineWidth',3);
```

Funkce – `varargout`

- proměnný počet výstupních proměnných
- princip je stejný jako u funkce `varargin`
  - nezapomeňte, že výstupní proměnné z funkce jsou v cellu
- využíváme spíše ojediněle

```
function [s, varargout] = sizeout(x)
nout = max(nargout, 1) - 1;
s = size(x);
for k = 1:nout
   varargout{k} = s(k);
end
```

```
>>  [s, rows, cols] = sizeout(rand(4, 5, 2))
% s = [4 5 2], rows = 4, cols = 5
```

Výstupní argument `varargout`

180 s ↑

- upravte funkci `fibonacciFcn.m` tak, aby měla pouze jeden výstupní parametr `varargout` a její funkcionalita zůstala zachovaná

Vyhodnocení výrazu v jiném WS

- funkce `evalin` („vyhodnoť v“) lze využít k vyhodnocení výrazu v jiném pracovním prostoru než je pracovní prostor, kde výraz existuje

- krom aktuálního pracovního prostoru lze využít prostory
  - `base`: základní pracovní prostor Matlab
  - `caller`: prostor funkce nadřazené (ze které byla funkce současná volána)
- nelze využívat rekurzivně

```
>> clear; clc;
>> A = 5;
>> vysl = eval_in
% vysl = 12.7976
```

```
function out = eval_in
%% no input parameters (A isn't known here)

k   = rand(1,1);
out = evalin('base', ['pi*A*', num2str(k)]);
```

Rekurze

- Matlab umožňuje rekurzi (= tj. funkce může volat sebe sama)
  - defaultně je omezen počet rekurzí na 500
  - počet rekurzí lze změnit, příp. zjistit současné nastavení:

```
>> get(0, 'RecursionLimit')
```

  - rekurze je součástí některých užitečných algoritmů (např. adaptivní kompozitní Simpsonova integrace)

Počet kroků rekurze

360 s ↑

- napište jednoduchou funkci schopnou volat samu sebe, vstupním parametrem je `rek = 0`, která se s každou rekurzí zvedá o 1
  - vypisujte růst hodnoty `rek`
  - na jakém čísle se růst zastaví
  - promyslete kde všude je rekurze nezbytná…

```
...
...
...
...
...
```

```
>> test_function(0)
```

Třída `inputParser` #1

- umožňuje pohodlně testovat parametry vstupující do funkce
- zvláště výhodná je k vytváření funkcí s mnoha vstupními parametry s dvojicemi `'parameter', value`
  - velmi typické pro grafické funkce

```
>> x = -20:0.1:20;
>> fx = sin(x)./x;
>> plot(x, fx, 'LineWidth', 3, 'Color', [0.3 0.3 1], 'Marker', 'd',...
 'MarkerSize', 10, 'LineStyle', ':')
```

- metoda `addParameter` umožňuje vložit volitelný parametr
  - musí se nastavit výchozí hodnota parametru
  - funkce pro testování validity není povinná
- metoda `addRequired` definuje název povinného parametru
  - při volání funkce musí být vždy zadán na správném místě

Třída `inputParser` #2

- funkce vykreslí kružnici nebo čtverec s určitým rozměrem, barvou a tloušťkou čáry

```
function drawGeom(dimension, shape, varargin)
p = inputParser; % instance of inputParser
p.CaseSensitive = false; % parameters are not case sensitive
defaultColor = 'b'; defaultWidth = 1;
expectedShapes = {'circle', 'rectangle'};
validationShapeFcn = @(x) any(ismember(expectedShapes, x));
p.addRequired('dimension', @isnumeric); % required parameter
p.addRequired('shape', validationShapeFcn); % required parameter
p.addParameter('color', defaultColor, @ischar); % optional parameter
p.addParameter('linewidth', defaultWidth, @isnumeric) % optional parameter
p.parse(dimension, shape, varargin{:}); % parse input parameters

switch shape
   case 'circle'
      figure;
      rho = 0:0.01:2*pi;
      plot(dimension*cos(rho), dimension*sin(rho), ...
         p.Results.color, 'LineWidth', p.Results.linewidth);
      axis equal;
   case 'rectangle'
      figure;
      plot([0 dimension dimension 0 0], ...
         [0 0 dimension dimension 0], p.Results.color, ...
         'LineWidth', p.Results.linewidth)
      axis equal;
end
```

Funkce `validateattributes`

- podle různých kritérií zkontroluje správnost zadaného parametru
  - často se používá se spojení s třídou `inputParser`
  - kontrola jestli matice má rozměr 2x3, je třídy `double` a obsahuje pouze celá kladná čísla:

```
A = [1 2 3;4 5 6];
validateattributes(A, {'double'}, {'size',[2 3]})
validateattributes(A, {'double'}, {'integer'})
validateattributes(A, {'double'}, {'positive'})
```

  - lze použít i zápis, kdy jsou všechny testované třídy a atributy napsané do jedné celly:

```
B = eye(3)*2;
validateattributes(B, {'double', 'single', 'int64'},...
   {'size',[3 3], 'diag', 'even'})
```

- pro kompletní výčet možností >> `doc validateattributes`

Původní jména vstupních proměnných

- pomocí funkce `inputname` lze určit název vstupních proměnných před zavoláním funkce
  - uvažujme následující volání funkce:

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

  - uvnitř funkce potom:

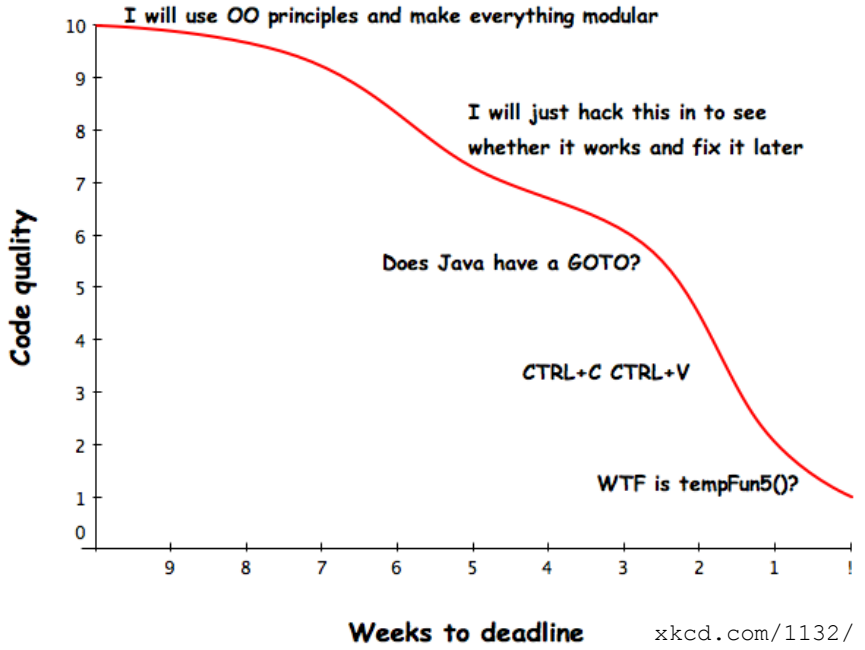
```
function output = myFunc1(par1, par2, par3)

% ...
p1str = inputname(1);     % p1str = 'xdot';
p2str = inputname(2);     % p2str = 'time';
P3str = inputname(3);     % p3str = '';
% ...
```

Tvorba funkcí – tipy

- hledisko efektivity – čím častěji budeme funkce využívat, tím lépe by měla být implementována
  - škálování kódu
  - vhodné verifikovat vstupní proměnné
  - vhodné alokovat předběžné výstupní proměnné
  - odladění chyb
  - optimalizace doby běhu funkce

- zásada atomizace – ideálně by daná funkce měla počítat pouze jednu věc, každá funkcionalita má být implementována pouze jednou



xkcd.com/1132/

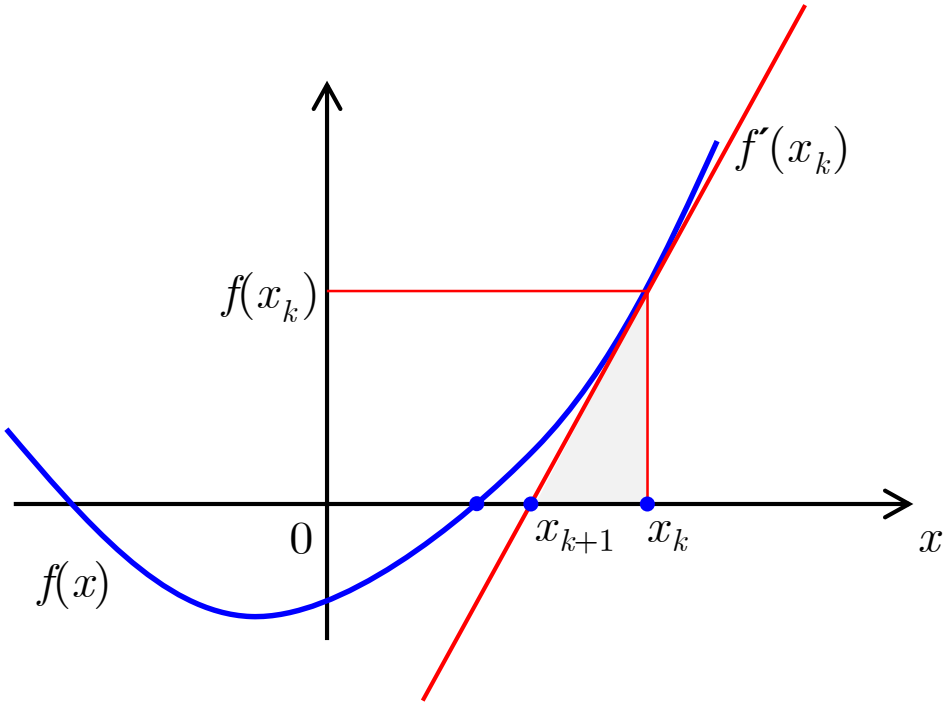
Probrané funkce

`@`	handle a anonymní funkce	
`varargin, varargout`	proměnný počet vstupních / výstupních proměnných	
`evalin, assignin`	vyhodnocení příkazu / přiřazení v jiném pracovním prostoru	•
`inputname`	název vstupních proměnných v nadřazeném pracovním prostoru	

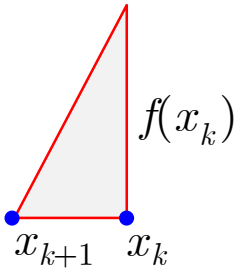
Cvičení #1

- nalezněte neznámé x pro rovnici typu $f(x)=0$
  - využijte Newtonovy metody

- Newtonova metoda:



$$f'(x_k)=\frac{\Delta f}{\Delta x}\approx\frac{\mathrm{d}f}{\mathrm{d}x}$$



$$f'(x_k)=\frac{\Delta f}{\Delta x}=\frac{f(x_k)-0}{x_k-x_{k+1}}$$

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

Cvičení #2

- nalezněte neznámé x pro rovnici $f(x)=0$ pomocí Newtonovy metody

- klasický postup implementace:
  (1) matematický model
    - uchopení problému, jeho formální řešení
  (2) pseudokód
    - návrh konzistentního a efektivního kódu
  (3) Matlab kód
    - převod do syntaxe Matlabu
  (4) otestování
    - zpravidla na příkladu se známým řešením
    - využijte… (originální příklad)

Cvičení #3

- nalezněte neznámé x pro rovnici $f(x)=0$ pomocí Newtonovy metody
- návrh pseudokódu:

(1) dokud $(x_k - x_{k-1})/x_k \geq \text{err}$ a současně $k < 20$ dělej:

(2) $$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

(3) $$\text{disp}\left(\left[k \quad x_{k+1} \quad f(x_{k+1})\right]\right)$$

(4) $$k = k+1$$

- musíte zajistit korektní vstup do (`while`) cyklu
- pro vyhodnocení $f(x_k)$, $f'(x_k)$ si vytvořte novou funkci
- je potřeba spočítat $f'(x_k)$, využijeme numerické diference:

$$f'(x_k) \approx \Delta f = \frac{f(x_k+\Delta) - f(x_k-\Delta)}{2\Delta}$$

Cvičení #4

600 s ↑

- nalezněte neznámé x pro rovnici $f(x)=0$ pomocí Newtonovy metody
  - výše rozpracovanou metodu implementujte v Matlabu a najděte neznámé x pro funkci $x^3+x-3=0$
  - Newtonova metoda bude ve formě skriptu, který bude volat funkci:

```
function fx = optim_fcn(x)

fx = x^3 + x - 3;
```

```
clear all; close all; clc;

% zadání proměnných
% xk, xk1, err, k

while podm1 a_soucasne podm2
    % nastav xk1 z xk
    % vypocti f(xk)
    % vypocti df(xk)
    % vypocti xk1
    % zvys hodnotu k
end
```

Cvičení #5

```
function fx = optim_fcn(x)

fx = x^3 + x - 3;
```

- jaká jsou omezení Newtonovy metody
  - v souvislosti s existencí více kořenů
- lze metodu využít pro komplexní hodnoty *x* (argumentu funkce)?

Cvičení #6

600 s ↑

- pomocí funkce `integral` spočítejte integrál z proudu $Q = \int I(t)dt$ v intervalu $t \in \langle 0,1 \rangle$s. Proud základní frekvenci $f = 50$ Hz a časový průběh

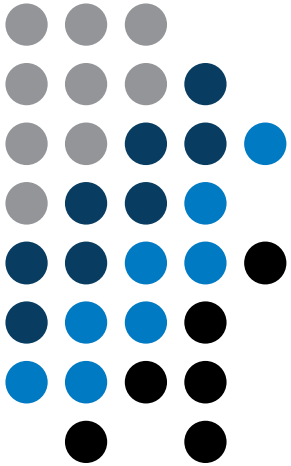
$$I(t) = 10\cos(2\pi f t) + 5\cos(4\pi f t)$$

  - problém řešte pomocí handle funkce

  - pomocí anonymní funkce

Děkuji!

ver. 3.5 (24/03/2015)
Miloslav Čapek
miloslav.capek@fel.cvut.cz



Materiál vytvořen v rámci předmětu A0B17MTB.