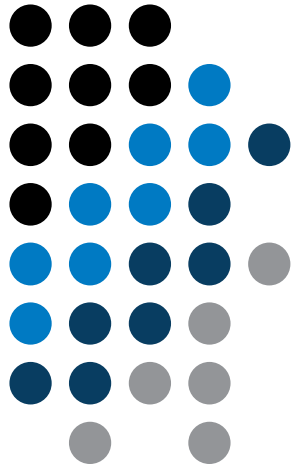
A0B17MTB – Matlab

Část #8

Miloslav Čapek
miloslav.capek@fel.cvut.cz

Filip Kozák, Viktor Adler

Katedra elektromagnetického pole
B2-626, Dejvice



Naučíte se …

Textové řetězce

`eval`, `feval`

Matlab `path`

Textové řetězce v Matlabu

- textový řetězec = pole (vektor, případně matice nebo cell) znaků
  - v Matlabu se snažíme pracovat bez diakritiky
- řetězec vytvoříme pomocí apostrofů

```
>> st = 'Hello, world!'
```

  - řetězec produkují krom toho i funkce (např. `>> char(65)`)
- každý znak v řetězci je jedním prvkem v poli a vyžaduje 2B
  - jde o datový typ `char`
- pokud potřebujeme apostrof i uvnitř řetězce, zdvojíme ho:

```
>> pt = 'That''s it!'
```

Textové řetězce – zásady

- pokud má řetězec víc než 1 řádek, musí mít stejný počet sloupců

```
>> st = ['karel'; 'pepa ']
```

- jinak (zpravidla) používáme datového typu cell, kam řetězce ukládáme:

```
pt = {'karel', 'pepa', 'a vsichni ostatni', 'včetně háčků'}
```

- zda je daná proměnná typu `char` zjistíme takto

```
>> ischar(st)
>> iscellstr(pt)
```

Textové řetězce – přetypování

- velice často musíme převádět z číselného kódu na řetězec a zpět, např.
  - `double` → `char`
  - `char` → `double`
  - `char` → `uint16`

```
>> tx = char([65:70])

>> B = double(tx)

>> C = uint16(tx)

>> whos
```

- operace s textovými řetězci jsou podobné jako v případě čísel. polí
  - platí zejm. pro indexaci!

```
>> S1 = 'test'; S2 = '_b5';
>> S3 = [S1 S2]
>> size(S3), size(S3')
>> S4 = [S3(3:5) 'end']
```

Textové řetězce

200 s ↑

- vytvořte si libovolný textový řetězec
  - zjistěte jeho délku
  - zkuste ho konvertovat do double
  - vyzkoušejte si indexovat vybrané části tohoto řetězce

- dotazy???

Textové řetězce – převod čísel #1

- převod čísla v řetězci (`char`) na skutečné číslo (`double`):
  - převod více čísel (funkce `str2num`):

```
>> str2num('[1 2 3 pi]')
>> str2num('[1, 2;3 4]')
```

```
>> str2num('[1 2 3 pi]')

ans =

    1.0000    2.0000    3.0000    3.1416

>> str2num('[1, 2;3 4]')

ans =

     1     2
     3     4
```

  - převod jednoho čísla na `double` (`str2double`):

```
>> str2double('1 +1j')
>> str2double('-0.5453')
```

  - pozor na možné chyby, které musí být případně v kódu ošetřeny

```
>> str2num('1a')
ans =
     []
```

```
>> str2num('1+1j')
```

```
>> str2num('1 +1j')
```

```
>> str2double('[1 2 3 pi]')
ans =
    NaN
```

```
>> str2num('1+1j')

ans =

   1.0000 + 1.0000i
```

```
>> str2num('1 +1j')

ans =

   1.0000 + 0.0000i   0.0000 + 1.0000i
```

Textové řetězce – převod čísel #2

- často potřebujeme číselný výsledek převést zpět na řetězec

```
>> num2str(pi)
>> num2str(pi, 10)
```

```
>> disp(['hodnota pi je: ' num2str(pi, 5)]);
```

- pro účely výpisu však vhodnější využít funkce `sprintf`
  - lépe totiž umožňuje kontrolovat výstup

```
>> st = sprintf('hodnota pi je: %0.5f\n', pi);
>> st
```

Textové řetězce – další převody

- známe mj. ještě tedy funkce

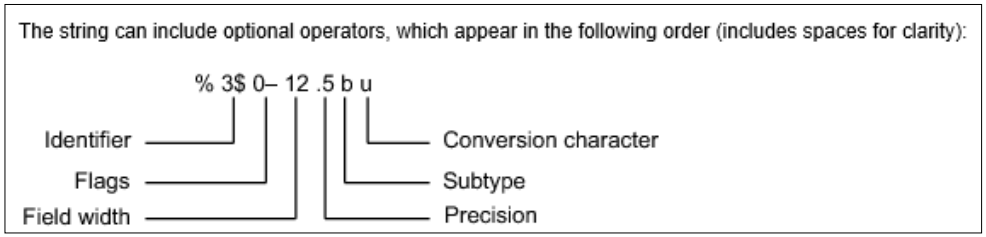
Funkce	Popis
`int2str`	převede celé číslo (integer) na text; pokud nejde o celočíselný údaj, je předtím zaokrouhlen
`mat2str`	převede matici na textový řetězec
`hex2num, num2hex`	převede hexadecimální číselný údaj ve formátu `char` na číslo (a opačně)

- např.

```
>> mat2str(magic(3))
```

Textové řetězce – formátování

- funkce `sprintf` vygeneruje textový řetězec s daným formátováním
  - více viz `>> doc sprintf`
  - lze i `disp(sprintf(...))`



- funkce `fprintf` textový řetězec vypisuje
  - na obrazovku (`fid` = 1 / 2)
  - do souboru (`fid` získáme např. pomocí funkce `fopen`, více později)

```
>> st = sprintf('hodnota pi je: %2.3e\n\n', pi);
>> fprintf(st) % příp. rovnou fprintf('...', pi);
```

```
>> fprintf(fid, st)
```

Textové řetězce

450 s ↑

- pomocí nápovědy pro `sprintf` připravte následující textové řetězce:
  - I.

```
ans =

Hodnota pi je: 3.14159, hodnota 5*pi je: 15.70796
```

  - tj. obě čísla jsou vypsána na 5 platných desetinných čísel
  - II.

```
ans =

Toto je 50%
```

  - tj. zobrazte znak procenta, výraz obsahuje 3 odřádkování
  - III.

```
ans =

Toto je sada měření: test_A
```

  - tj. vložte do řetězce proměnnou, jejíž hodnota je řetězec `'test_A'`

Textové řetězce

200 s ↑

- zamyslete se nad rozdíly mezi `disp` a `fprintf` (`sprintf`)
  - popište rozdíly
  - jakou funkci a v jakých případech použijete?

Přetypování (obecně) – poznámka

- Matlab si datové typy proměnných určuje sám
  - sám si případně i proměnné přetypuje
- jednoduchá / dvojitá přesnost: `single()` / `double()`
- pokud však explicitně vyžadujete daný typ a nezadali jste ho při tvorbě proměnné, můžete tuto proměnnou přetypovat:
  - funkce `cast`: provede přetypování, hodnoty jsou případně zmenšeny
  - funkce `typecast`: provede přetypování a zachová velikost původní proměnné z pohledu paměti, ale hlavně bitovou hodnotu
  - více viz dokumentace

Malá / velká písmena

- `lower` převede všechna písmena řetězce na malá písmena

```
>> lower('Vše BuDe MALÉ')
% ans =
% vše bude malé
```

- `upper` převede všechna písmena řetězce na velká písmena

```
>> str = 'vše bude velké';
>> str = upper(str)
% str =
% VŠE BUDE VELKÉ
```

  - na PC podpora symbolů ze znakové sady Latin 1
  - jiné platformy: ISO Latin-1 (ISO 8859-1)
  - $\Rightarrow$ podpora české diakritiky

Textové řetězce – vyhledávání

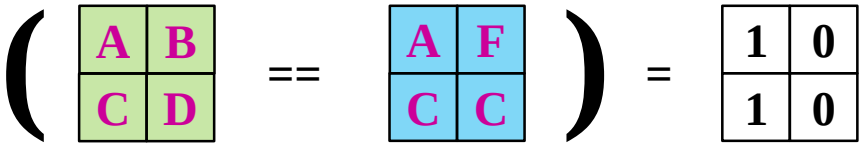
- `strfind` nalezne daný řetězec uvnitř jiného
  - vrací indexy (pozice)
  - hledá i vícenásobný výskyt
  - je CaSe sEnSiTiVe
  - umožňuje hledat i mezery atp.

```
>> lookFor = 'a';
>> res = strfind('tato kniha', lookFor);
res =
     2    10
```

Textové řetězce – porovnávání

- dva řetězce lze porovnat pomocí funkce `strcmp`
  - funkci často využijete uvnitř `if-else` / `switch-case` konstrukcích
  - výsledek je `true` nebo `false`
  - lze porovnávat i řetězec vs. `cell` řetězců, příp. `cell` vs. `cell`

```
>> strcmp('tel', 'A')
>> strcmp('tel', 'tel')
>> strcmp('test', {'test', 'A', '3', 6, 'test'})
>> strcmp({'A', 'B'; 'C', 'D'}, {'A', 'F'; 'C', 'C'})
```



Textové řetězce – spojování řetězců

- textové řetězce lze spojit dohromady pomocí funkce `strjoin`
  - pracujeme nad proměnnou typu `cell` (viz dále)
  - lze vybrat oddělovač (jinak je jím mezera)

```
>> cl = {'A', 'B', 'C', 'D'}
>> strjoin(cl)
>> strjoin(cl, ',')
```

```
>> cl = {'A','B','C','D'}

cl =

    'A'    'B'    'C'    'D'

>> strjoin(cl)

ans =

A B C D

>> strjoin(cl,',')

ans =

A,B,C,D
```

- `fullfile` spojí jednotlivé složky do souborové cesty
  - mezi jednotlivé položky jsou přidány lomítka (\)

```
>> folder1 = 'Matlab';
>> folder2 = 'project_one';
>> file    = 'run_process.m';
>> fpath = fullfile(folder1, folder2, file);
```

```
fpath =

Matlab\project_one\run_process.m
```

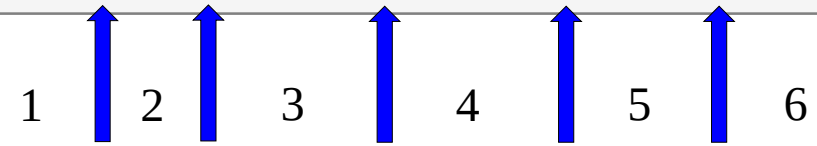
  - použijeme dále při exportu a práci s GUI

Textové řetězce – separace řetězců

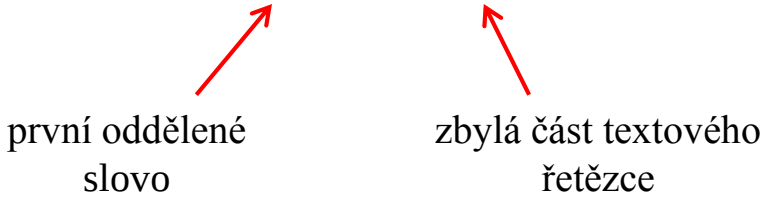
- funkce `deblank` odstraní přebývající mezery na konci řetězce
- funkce `strtrim` odstraní mezery na začátku i na konci řetězce

- chceme-li rozdělit libovolný řetězec, užijeme funkce `strtok`
  - lze volit libovolný oddělovač

```
>> this_str = 'some few little little small words'
```




```
>> [token, remain] = strtok(this_str, ' ');
```



Textové řetězce – separace řetězců

- funkce `regexp` umožňuje vyhledávání v textovém řetězci s pomocí regulárních výrazů
  - funkce má složitější syntax, ale má i nesmírné možnosti!!
    - Př.: hledáme všechna slova začínající na `wh`, pokračující samohláskami `a` nebo `e`, a obsahující 2 znaky:

```
>> that_str = 'what which where whose';
>> regexp(that_str, 'wh[ae]..', 'match')
```

    - Př.: hledáme indexy (pozice), kde začínají a končí slova obsahující `a` nebo `o`

```
>> that_str    = 'what which where whose';
>> [from, to] = regexp(that_str, '\w*[ao]\w*')
```

  - pro další detaily viz `>> doc regexp` → `Input Arguments`
  - kombinací s funkcemi výše umožňuje vytvořit klasický tokenizer

Textové řetězce

600 s ↑

- vyzkoušejte si příkazy níže a zkuste dopředu odhadnout co se stane …

```
>> str2num('4.126e7')
>> str2num('4.126A')
>> D = '[5 7 9]';
>> str2num(D)
>> str2double(D)
>> int2str(pi + 5.7)
>> A = magic(3);
>> mat2str(A)
>> disp([15 pi 20-5i]);
>> disp(D);
>> B = 'MaTLaB';
>> lower(B)
```

```
>> C = 'cik cak cet  ';
>> findstr(C, 'cak')
>> deblank(C)
>> [tok remain] = strtok(C, ' ')
>> [st se] = regexp(C, 'c[aeiou]k')
>> [st se] = regexp(C, 'c[ei][kt]')
>> regexp(C, '[d-k]')
>> fprintf('Vysledek je %3.7f', pi);
>> fprintf(1, 'Enter\n\n');
```

```
>> disp(['Vysledek: ' num2str(A(2, 3)) 'mm']);
>> fprintf(1, '% 6.3f%% (procent)\n', 19.21568);
>> fprintf('Bude to: %3.7fV\n', 1e4*(1:3)*pi);
>> fprintf('A=%3.0f, B=%2.0f, C=%1.1f\n', magic(3));
>> fprintf('%3.3e + %3.3f = %3.3f\n', 5.13, 13, 5+13);
>> fprintf(1, '%s a %s\n\n', B, C([1:3 5:7]));
```

Textové řetězce – porovnávání

300 s ↑

- funkce na porovnání textových řetězců (CaSe SeNsItIvE) je `strcmp`
  - zkuste najít podobnou funkci, která však ignoruje velikost písmen (case insensitive)
  - zkuste najít funkci, která pracuje podobně jako funkce výše (tj. porovnává řetězce bez ohledu na velikost písmen), ale která porovná pouze prvních `n` písmen
  - zamyslete se nad alternativními možnostmi k funkci `strcmp`

Textové řetězce

300 s ↑

- odstraňte z následujícího řetězce všechny mezery
  - zkuste si vzpomenout na logické indexování
  - případně využijte libovolné funkce Matlabu

```
>> s = 'toto je velka kniha'
```

Textové řetězce

420 s ↑

- napište skript/funkci, která rozloží následující větu na jednotlivá slova
  - upravte kód tak, aby na konec vypsal kolikrát se ve větě opakuje `'is'`
  - vypište slova samostatně, vč. pořadí slova ve větě (použijte `fprintf`)

```
This-sentence-is-for-testing-purposes-only.
```

Textové řetězce

420 s ↑

- napište skript/funkci, která rozloží následující větu na jednotlivá slova
- problém lze řešit ještě elegantněji pomocí funkce `textscan`
  - řešení však není splněno zcela (chybí pořadí písmen)

`eval` – řetězec jako příkaz

- motivace:

```
>> st  = 'sqrt(abs(sin(x).*cos(y)))';
>> x   = 0:0.01:2*pi;
>> y   = -x;
>> fxy = eval(st);
>> plot(x, fxy);
```

  tj. máme textový řetězec, který obsahuje vykonavatelné příkazy
  - jeho zpracování nám zajistí funkce `eval`
- využijeme zejm. při práci s GUI (načítání příkazů, které zadal uživatel do GUI, zpracování tzv. callback funkcí aj.)
- `eval` má jisté nevýhody, pro které vždy zvažujeme její použití:
  - blok kódu s `eval` není kompilován (zpomalení)
  - kód uvnitř textového řetězce může cokoliv přepsat
  - syntax uvnitř textového řetězce není kontrolována, hůře se i čte
- v jakých případech lze `eval` nahradit, viz dokumentace k funkci
  - např. ukládání souborů s pořad. číslem (`data1.mat`, `data2.mat`, …)

evalc

- v některých případech je potřeba nejen provést příkaz, který Matlabu zadáváme ve formě textového řetězce, ale i výsledek tohoto příkazu uložit pro pozdější využití

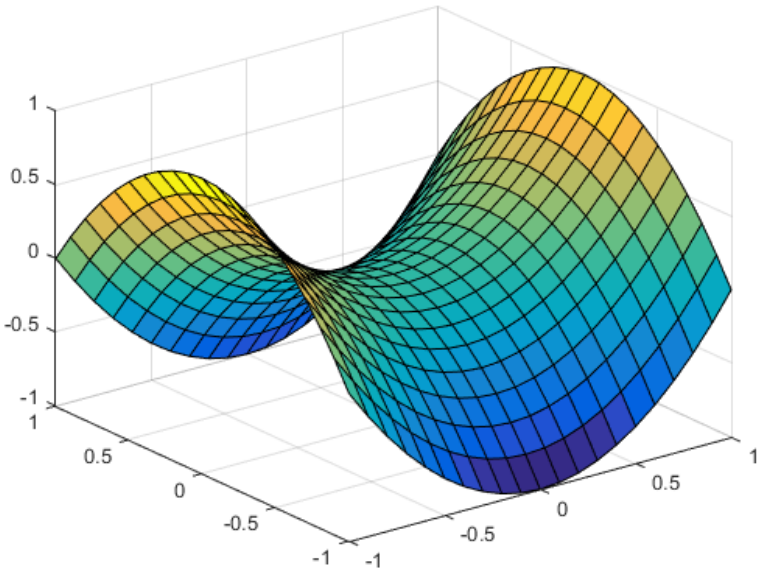
- k tomu slouží funkce `evalc` („*eval with capture*“)

```
>> CMD = evalc(['var = ' num2str(pi)]);
>> CMD

CMD =


var =

    3.1416



>> whos
  Name       Size              Bytes  Class     Attributes

  CMD        1x20                 40  char
  var        1x1                   8  double
```

`feval` – vyhodnocení handle funkce

- funkce je určena pro vyhodnocování handle funkcí
  - jednoduše řečeno, tam kde `eval` vyhodnocuje textový řetězec, tam `feval` vyhodnocuje funkci zadanou pomocí handle
  - uvažujme tuto úlohu:

$$f(x,y) = x^2 + y^2, \quad x, y \in \langle -1, 1 \rangle$$




```
>> hFcn    = @(x,y) x.^2 - y.^2;
>> x       = -1:0.1:1;
>> y       = x;
>> [X, Y] = meshgrid(x, y);
```

```
>> fxy     = hFcn(X, Y);
>> surf(X, Y, fxy);
```

```
>> fxy     = feval(hFcn, X, Y);
>> surf(X, Y, fxy);
```

Newtonova metoda – modifikace

600 s ↑

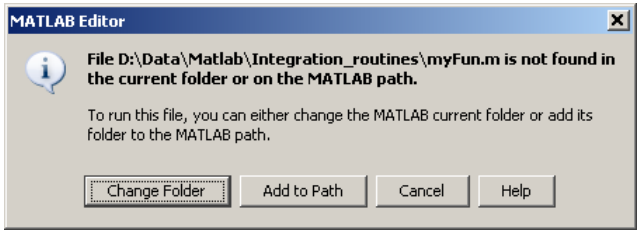
- upravte naší Newtonovu metodu pro nalezení kořene polynomu tak, že půjde zadat libovolný polynom ve formě handle funkce
  - skript viz `elmag.org/cs/Matlab/harmonogram` → 7. týden
  - po dokončení ověřte tím, že naleznete kořeny těchto polynomů:

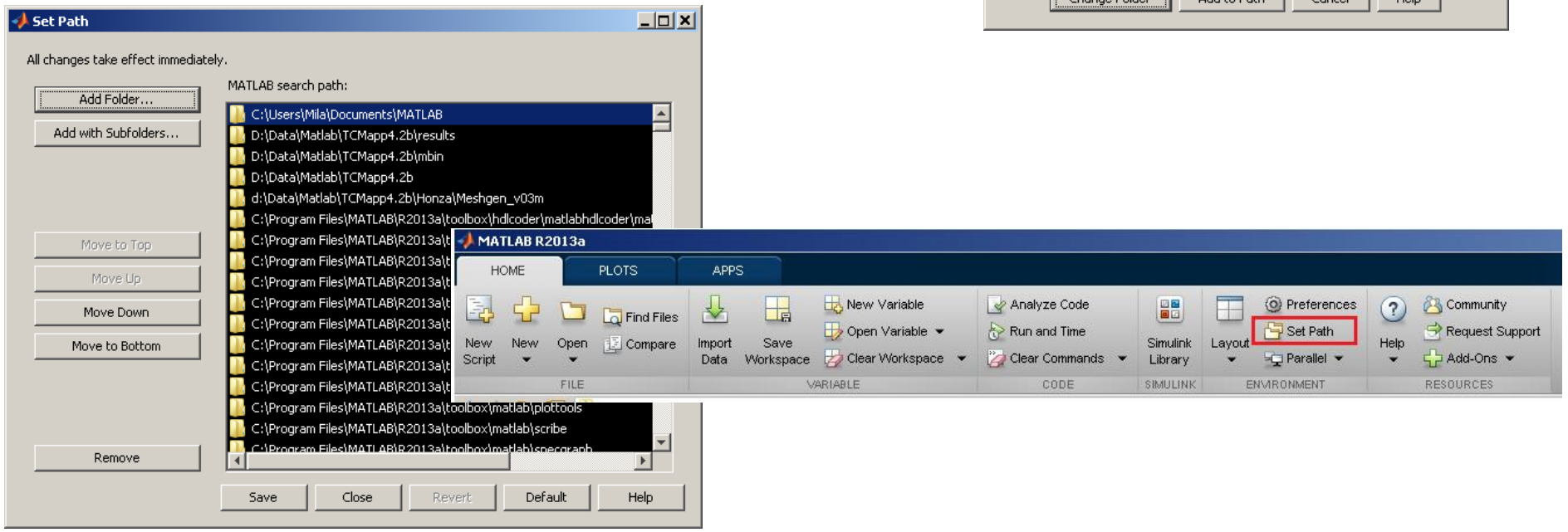
$$x - 2 = 0, \quad x^2 = 1$$

  - výsledek ověřte pomocí funkce `roots`

Matlab `path`

- do kterých adresářů Matlab vidí: `>> path`
- více viz `>> doc path`
- `addpath`: přidá daný adresář do path
- `rmpath`: odebere daný adresář z path





Volání funkce – pořadí

- jak Matlab hledá Vaší funkci:
  - jde o proměnnou
  - funkce importované pomocí `import`
  - zanořená (nested) příp. poté vedlejší funkce uvnitř dané funkce
  - privátní funkce
  - funkce (metoda) příslušné třídy příp. poté konstruktor příslušné třídy
  - funkce v daném adresáři
  - funkce kdekoliv v dosahu Matlabu (`path`)
- uvnitř daného adresáře je priorita různých přípon následující:
  - vestavěná (built-in) funkce
  - `mex` funkce
  - Simulink funkce (`slx` / `mdl`)
  - `p`-soubory
  - `m`-soubory

Funkce `exist`

- funkce zjistí, zda zkoumané slovo odpovídá existující(mu)

  (=1) proměnné v Matlab Workspace

  (=5) zabudované funkce (built-in)

  (=7) adresář

  (=3) mex/dll funkce/knihovna

  (=4) mdl-soubor

  (=6) p-soubor

  (=2) m-soubor známý Matlabu (vč. Vašich funkcí, vidí-li je Matlab)

  (=8) třída

  (seřazeno podle priority, v závorce návratová hodnota)

```
>> type = exist('sin')    % type = 5
>> exist('ukol1', 'var')   % is the file ukol1 ...
>> exist('ukol1', 'dir')   % the variable / ...
>> exist('ukol1', 'file') % directory / file?
```

Na čem je Váš m-file závislý?

- pokud kód kompilujete, posíláte kolegům atp., je vhodné otestovat, zda mají všechny potřebné související soubory (funkce)

- používáme funkci `depfun`
  - Př.1: funkce sinus `(sin)`

```
>> depfun('sin')
============================================================
depfun report summary:
------------------------------------------------------------
-> trace list:           1 files  (total)
                         1 files  (total arguments)
                         0 files  (arguments off MATLABPATH)
                         0 files  (argument duplicates on MATLABPATH)
------------------------------------------------------------
Notes: 1. Use argument  '-quiet' to not print this summary.
       2. Use arguments '-print','file' to produce a full
          report in file.
       3. Use argument  '-all' to display all possible
          left hand side arguments in the report(s).
============================================================

ans =

    {''}
```

- podobně máme funkci `depdir`

depfun

- od verze Matlabu R2014b je funkce `depfun` nahrazena jako
  - `matlab.codetools.requiredFilesAndProducts`

- Př.2: Newtonova metoda

```
>> depfun('newton_method')
============================================================
depfun report summary:
------------------------------------------------------------
-> trace list:          30 files  (total)
                         1 files  (total arguments)
                         0 files  (arguments off MATLABPATH)
                         0 files  (argument duplicates on MATLABPATH)
------------------------------------------------------------
Notes: 1. Use argument  '-quiet' to not print this summary.
       2. Use arguments '-print','file' to produce a full
          report in file.
       3. Use argument  '-all' to display all possible
          left hand side arguments in the report(s).
============================================================
```

```
ans =

'd:\Data\Matlab\newton_method.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\@opaque\char.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\@opaque\double.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\@opaque\toChar.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\datatypes\num2cell.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\elmat\fliplr.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\elmat\repmat.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\@opaque\disp.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\@opaque\display.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\@opaque\evalc.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\ans.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\lang\details.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\ops\@opaque\eq.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\ops\@opaque\ne.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@cell\strcat.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@cell\strfind.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@cell\strjust.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\findstr.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\isspace.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\private\fromOpaq
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\strcmp.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\strfind.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\@opaque\strncmp.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\blanks.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\cellstr.m'
'C:\Program Files\MATLAB\R2013a\toolbox\matlab\strfun\int2str.m'
```

Tvorba funkcí – tipy

- jak naznačit, že daná funkce / skript počítá?
  - níže několik možností, vyzkoušejte…

```
fprintf('START\n    ');
for n = 1:100
    fprintf(1, '\b\b\b\b%3.0f%%', n);
    pause(0.05);
end
fprintf('\nEND\n');
```

```
T = ['/' '-' '\'];
fprintf(2, 'START\n\n');
for n = 1:100
    fprintf(1, '\b%c', T(mod(n, 3)+1));
    pause(0.05);
end
fprintf('\b');
fprintf(2, 'END\n');
```

```
fprintf(2, 'START\n');
for n = 1:100
    fprintf(1, '*');
    pause(0.05);
end
fprintf(1, '\n');
fprintf(2, 'END\n');
```

  - později si ukážeme i grafické možnosti!

Matlab – přípony souborů

přípona	popis
`.fig`	Matlab obrázek
`.m`	skript / funkce / třída
`.mat`	binární soubor s daty
`.mdl, .slx`	model Simulinku
`.mdlp, .slxp`	chráněný model Simulinku
`.mexa64, .mexmaci64, .mexw32, .mexw64`	mex knihovny
`.mlappinstall`	APP soubor – instalátor
`.mlpkginstall`	podpůrný balíček – instalátor
`.mltbx`	soubor toolboxu – instalátor
`.mn`	MuPAD notebook
`.mu`	MuPAD kód
`.p`	chráněný Matlab kód

Probrané funkce

`char, uint16, ...`	přetypování / tvorba proměnných daného typu	●
`single, double`	jednoduchá / dvojitá přesnost	
`ischar, iscellstr`	test na textový řetězec / řetězec v cell proměnné	
`int2str, mat2str, hex2num, num2hex`	převody (celá čísla – řetězce, hexadecimální – IEEE double)	
`str2double`	řetězec na double	
`sprintf, fprintf`	formátování řetězce, výpis řetězce	●
`cast, typecast`	přetypování (bez / se zachováním velikosti)	
`lower, upper`	převod řetězce na malá / velká písmena	
`strfind, strcmp, strjoin, fullfile`	vyhledávací, porovnávací, spojovací funkce pro řetězce	●
`deblank, strtrim, strtok`	odstranění mezer, ořezávání řetězce, rozdělování řetězce	●
`regexp, textscan`	vyhledávání v řetězci (vč. regulárních výrazů)	●
`eval, feval`	vyhodnocení řetězce / vyhodnocení handle funkce	●
`path, exist, depfun`	přístup Matlabu do adresářů, existence proměnné (souboru)	

Příklad #1, #2

450 s ↑

- zjistěte, kolik mezer je v textu „*how are you?*“
  - prohledejte tuto přednášku / nápovědu a najděte vhodnou funkci

- převeďte následující řetězec na malá písmena a určete počet znaků

```
>> st = 'MATLAB is CaSe sEnSiTiVe!!!';
```

Příklad #3

300 s ↑

- vytvořte funkci na výpočet objemu, obsahu a tělesové úhlopříčky následujících těles: kvádr, válec
  - hlavní funkce `teleso.m` obsahuje verifikace vstupních proměnných (typ, rozměr) a rozhodnutí, zda se jedná o kvádr (argumenty `'kvadr'`,a,b,c) nebo válec (`'valec'`,r,h)
- vedlejší funkce `kvadr()` a `valec()` vypočítají dané údaje

Příklad #3

Příklad #4

600 s ↑

- vytvořte tzv. tokenizer (analyzátor textu), který
  - pomocí funkce `input` načte text `retezec`, který uživatel zadá
  - načte oddělovač `oddel` (pozor, mezera vyžaduje více pozornosti!!)
  - rozdělte `retezec` na jednotlivé části podle `oddel`
  - uložte jednotlivé části zvlášť do proměnné typu cell
  - analyzujte kolik má které slovo základních samohlásek (a/e/i/o/u), tento údaj uložte a na konci vypište, společně s výpisem všech rozdělených slov
  - celý skript / funkce musí mít všechny příkazy ukončené středníkem!

Příklad #4

- vytvořte tokenizer (analyzátor textu)
  - řešení s využitím `strtok`

Příklad #4

- vylepšené řešení pomocí `strsplit`

Příklad #5

600 s ↑

- zkuste vytvořit velice jednoduchý konvertor délek, bude zadána délka `x` v `'mm'`, `'cm'`, `'in'`, `'inch,` (proměnná `units`), délka v palcích lze označit `'in'` i `'inch'`. Podle zadaného textového řetězce jednotek bude převeden rozměr do [mm]
  - jakou rozhodovací konstrukci využijete?
  - přidejte za závěr kontrolní výpis z jakých jednotek do [mm] byl údaj konvertován a jaký je výsledek

```
x     = 15;
units = 'in';
% doplňte zbytek
```

Příklad #5

Konvertor jednotek elegantně

- využíváme datové typu `struct` a jeho vlastností
  - jednotlivá pole ve struktuře lze indexovat pomocí proměnných typu `char`

```
function result = convertLength(in_val, in_unit, out_unit)

% supported units for conversion
conversion.in   = 1e4/254; % en.wikipedia.org/wiki/Imperial_units
conversion.inch = conversion.in;
conversion.mm   = 1e3;
conversion.cm   = 1e2;
conversion.m    = 1;

% are the units supported?
if ~isfield(conversion, in_unit)
    error('convertor:nonExistentUnit', ['Unknown unit: ' in_unit]);
end

% calculation
result = in_val * conversion.(out_unit) / conversion.(in_unit);
```

Děkuji!

ver. 3.5 (20/02/2015)
Miloslav Čapek
miloslav.capek@fel.cvut.cz