Fakulta elektrotechniky
a informatiky

Úvod do Matlabu

Prednáška č. 5

- **Riadiace príkazy (if, else a elseif)**
- Príkazy slučiek (while, for a parfor)
- Vlastné funkcie

Ing. Ondrej Kováč, PhD.

Príkazy riadenia– Úvod

Tak ako v iných programovacích jazykoch, aj Matlab má v sebe implementované dobre známe príkazy:

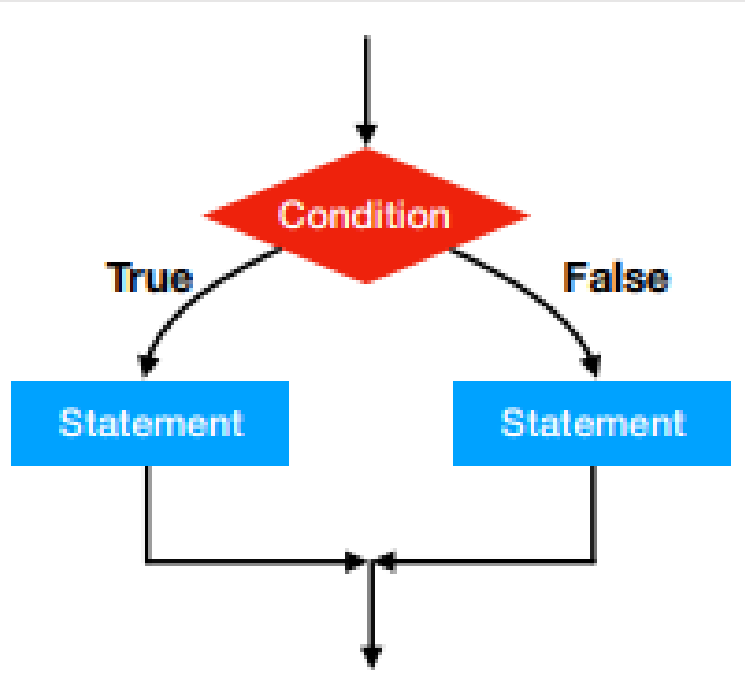
- **If – else - elseif**
- **switch**

Navyše v Matlabe oproti klasickému jazyku C je tiež implementovaný príkaz , ktorý je určený na manažment chýb a neočakávaných ukončení programu

- **try-catch**

Príkazy riadenia– if – else - elseif

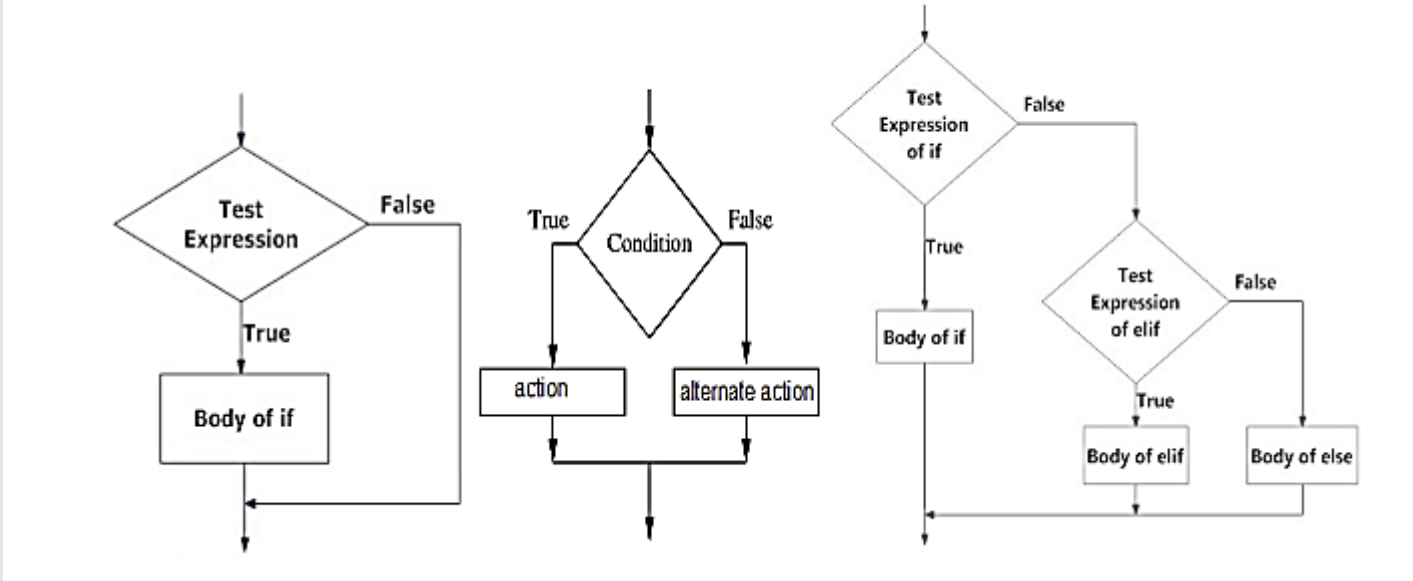
Príkaz **if** je veľmi dobre známy príkaz. Je ho možné použiť v jeho neúplnej forme, kedy sa niečo vykoná, iba ak je splnená podmienka, alebo s použitím **else** úplnej forme, kedy sa vykoná niečo iné ak podmienka splnená nie je. V Matlabe navyše môžeme **if** ďalej vetviť pomocou **elseif.**




```
a = input('zadaj hodnotu a:');
b = input('zadaj hodnotu b:');
if a == 1
   c = 1;
elseif b == 1
   c = 2;
else
   c = 3;
end
```

na rozdiel od jazyka C, v Matlabe nepoužívame zátvorky!

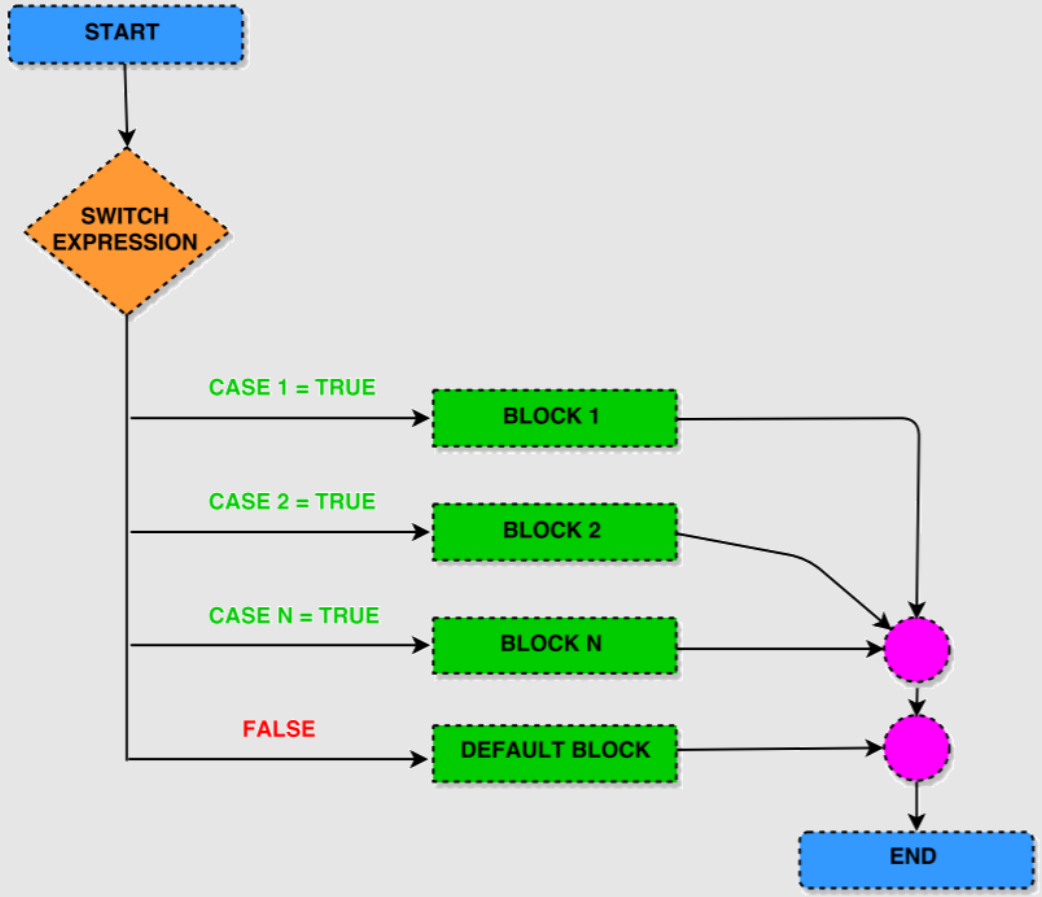
Príkaz riadenia alebo aj slučky je ukončený príkazom **end**.



Príkazy riadenia– switch - case

Príkaz **switch** predstavuje prepínač. **Switch je vhodné použiť vtedy, ak sledovaná premenná môže nadobudnúť viacero definovaných stavov.** Vhodný je napríklad pre tvorbu jednoduchého menu. Jednotlivé možnosti sú definované pomocou prípadov – **case**, príkaz switch umožňuje aj základnú voľbu pomocou príkazu **otherwise**.

```
volba = input('zadaj hodnotu 1-3 a:');
switch volba
     case 1
          disp('Volba 1');
     case 2
          disp('Volba 2');
     case 3
          disp('Volba 3');
     otherwise
           disp('Volba mimo povoleny rozsah');
end
```



Príkazy riadenia– try – catch

- Táto dvojica príkazov je vynikajúcim riešením, pokiaľ chce programátor riadiť to, ako sa bude navrhnutý program chovať v prípade chýb. V prípade chyby, Matlab okamžite ukončí vykonávanie programu a do príkazového riadku vypíše chybu.
- Programátor dokáže množstvo chýb, ktoré sa môžu vyskytnúť predvídať svoj program musí poriadne ošetriť **(if – else - elseif)!**
- Svoj program môže tiež upraviť tak, že ak dôjde chybe, beh programu sa nepreruší, ale operácia, ktorá spôsobila chybu sa vykoná inak.

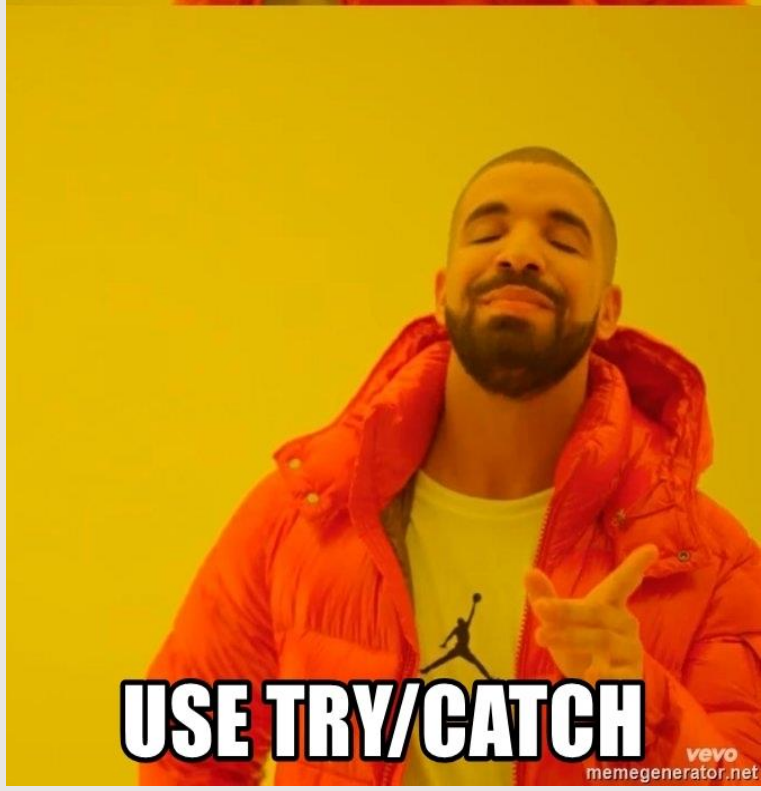
```
try
  a  = [1;'ahoj'];
catch ME
  disp(['Nastala chyba:',ME.message]);
end
b = 1
```

```
a  = [1;'ahoj'];
b = 1
```

Výstup príkazového riadku

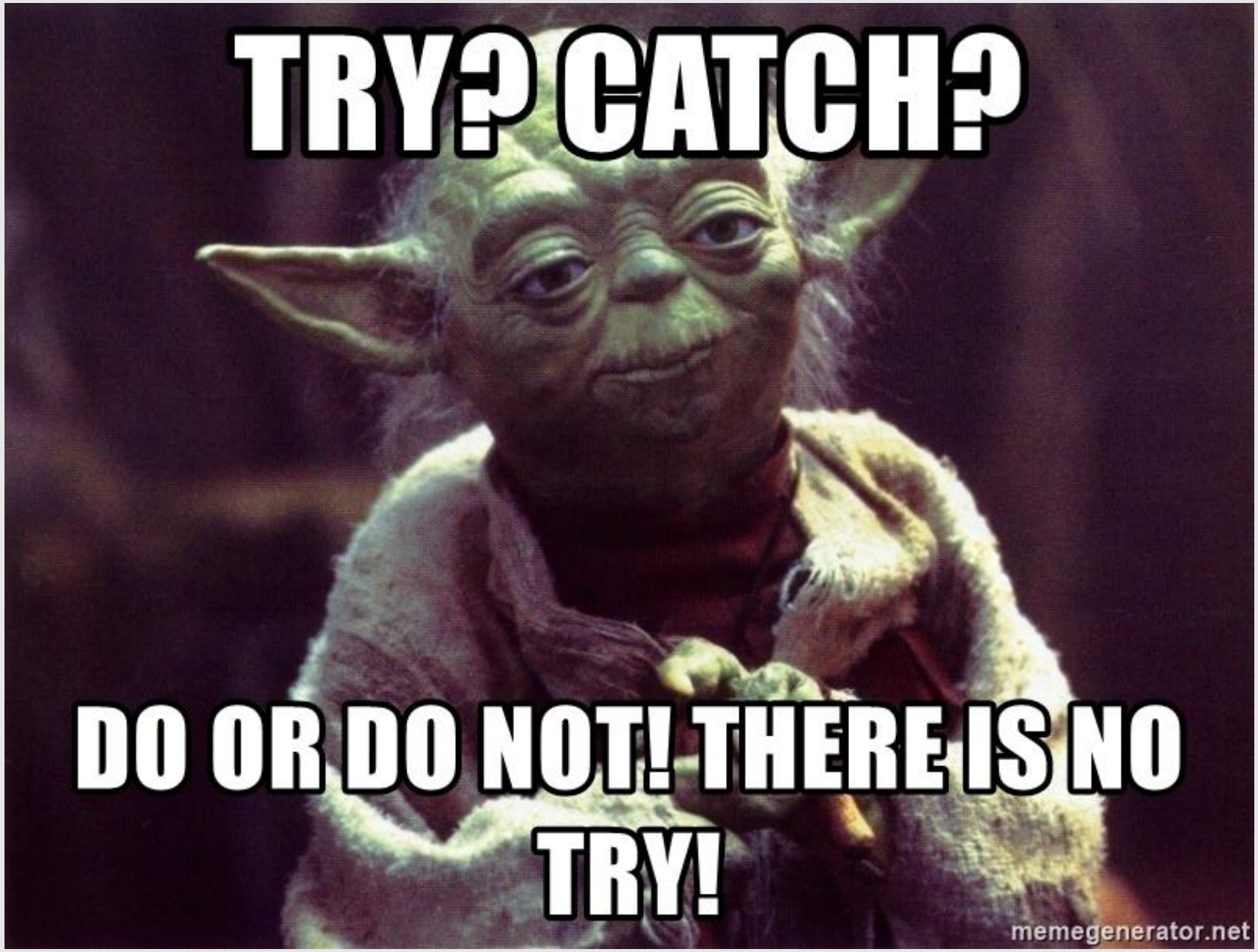
```
Nastala chyba : Dimensions of matrices
being concatenated are not consistent.
b =
     1
```

```
Error using vertcat
Dimensions of matrices being
concatenated are not consistent.
```



Príkazy riadenia– try – catch

try – catch je naozaj mocný nástroj, no používajte ho s rozumom, **hlavne pri vývoji**. Skúste sa mu **vyhnúť v produkčnom** resp. finálnom riešení.



Fakulta elektrotechniky
a informatiky

Úvod do Matlabu

Prednáška č. 5

- Riadiace príkazy (if, else a elseif)
- **Príkazy slučiek (while, for a parfor)**
- Vlastné funkcie

Ing. Ondrej Kováč, PhD.

Slučky/cykly – Úvod

Pod príkazom slučky resp. cyklu rozumieme taký príkaz, ktorý zabezpečí opakovanie krokov programu dovtedy kým nie je splnená podmienka jeho ukončenia. Poznáme dva, aj z iných programovacích jazykov, dobre známe príkazy cyklu a to **while** a **for**.

Matlab umožňuje aj paralelné výpočty pomocou príkazu **parfor**.

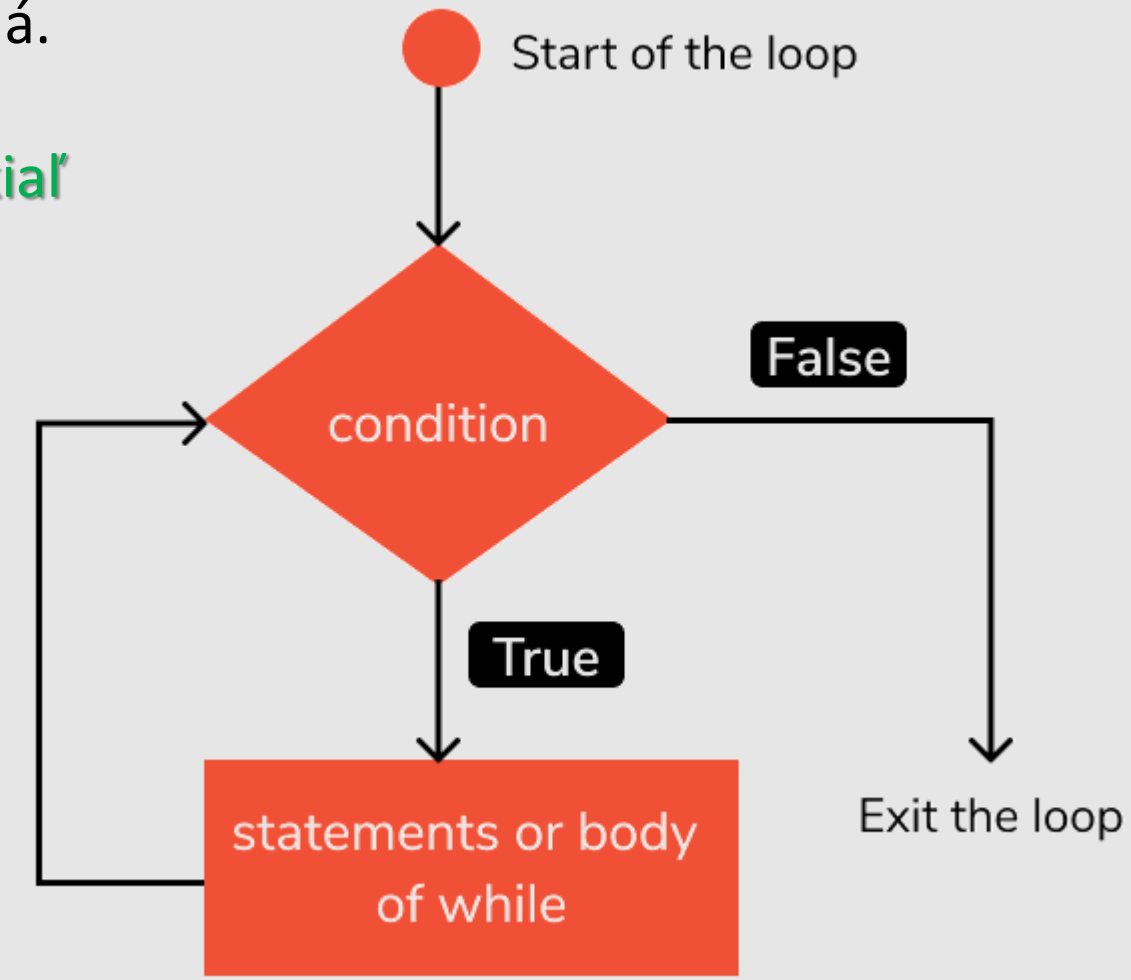
Každú slučku **for** možno vykonať pomocou **while** a naopak.

Slučky/cykly – while

- Príkaz **while** sa používa hlavne v tom prípade, kedy nie je známe, koľko krát sa cyklus vykoná.

- Telo príkazu while sa vykonáva dovtedy dokiaľ je podmienka pravdivá.

```
1
2 -     a = 0;
3 -   while a < 10
4 -         disp(a);
5 -         a = a + 1;
6 -   end
```



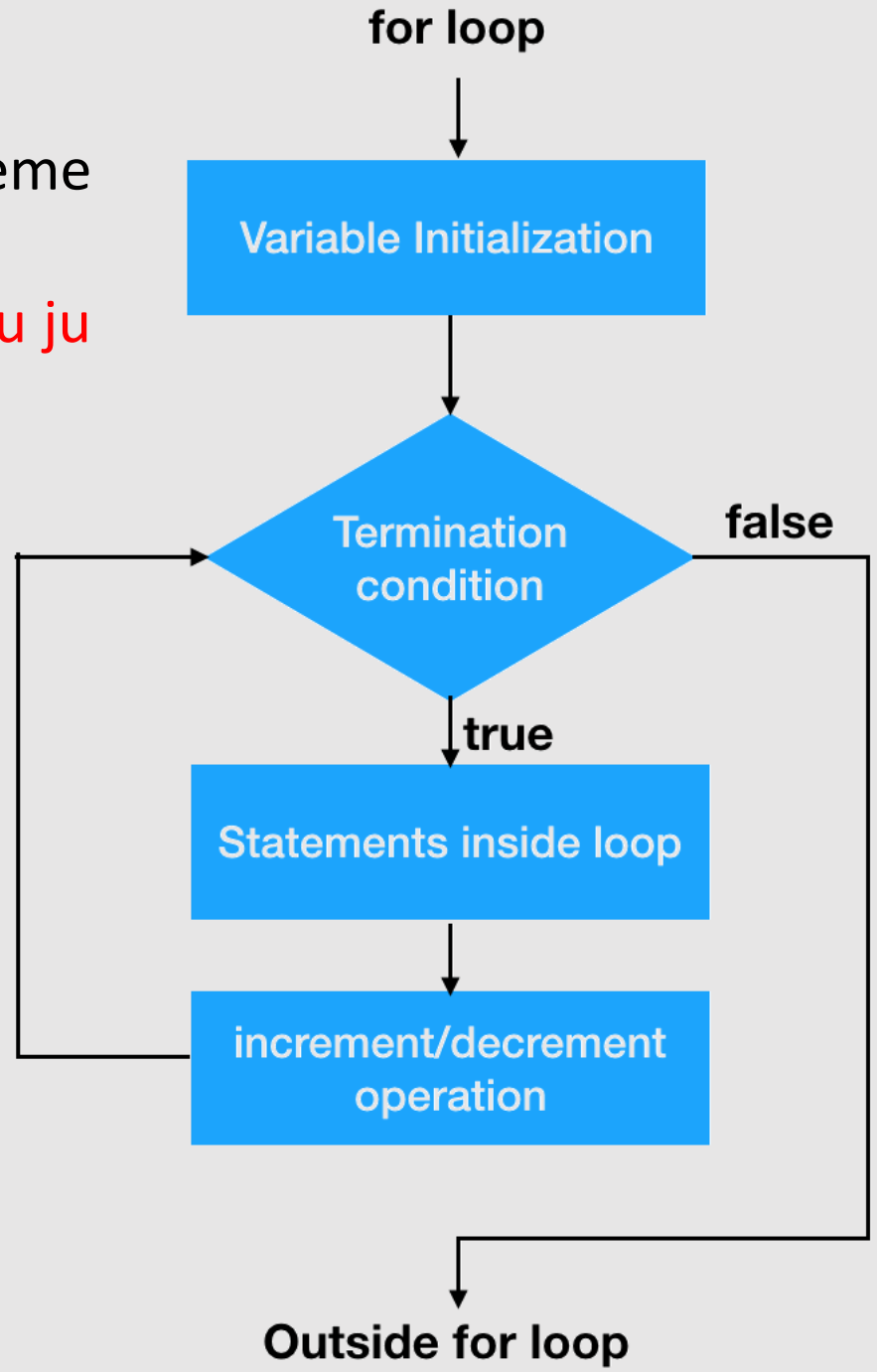
Slučky/cykly – for

- Príkaz **for** je vhodné použiť tam, kde jednoznačne vieme koľko krát sa má vykonať.
- Premenná sa inkrementuje automaticky a v tele cyklu ju neinkrementujeme!

```
1     % For s inkrementáciou
2     for a = 0:10
3         disp(a);
4     end
5
6     % For s inkrementáciou s krokom 2
7     krok = 2;
8     for a = 0:krok:10
9         disp(a);
10    end
11
12    % For s dekrementáciou  s krokom 2
13    for a = 10:-krok:0
14        disp(a);
15    end
```

Krok inkrementácie nemusí byť rovný iba jednotke

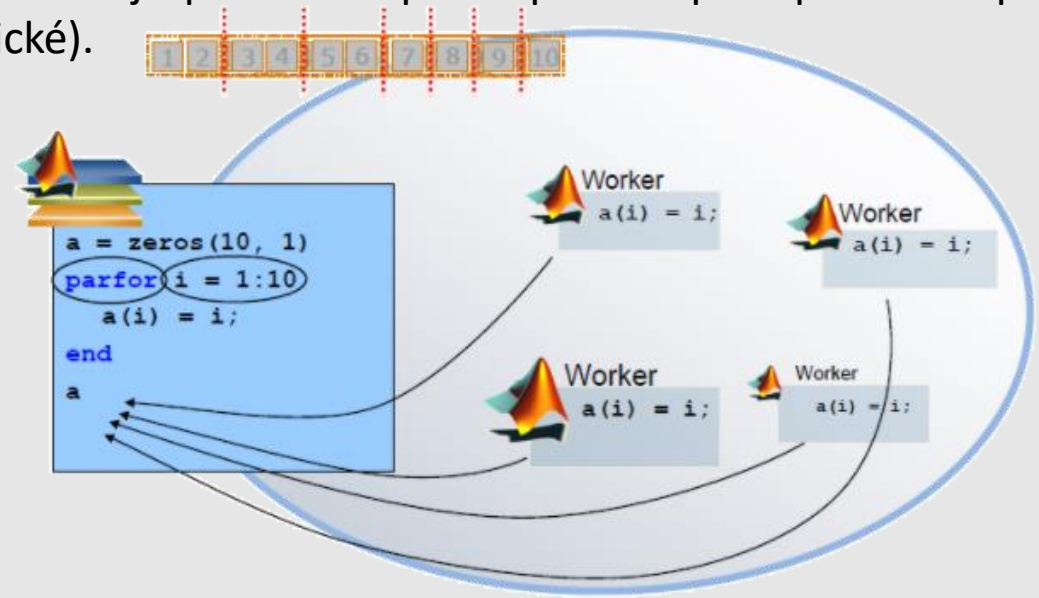
Pri dekrementácii je potrebné nastaviť záporný krok!



Slučky/cykly – paralelné výpočty – parfor

Matlab štandardne vykonáva príkazy sekvenčne na jednom jadre CPU. Príkaz parfor, ktorý je implementovaný v paralel computing toolboxe, umožňuje vykonávať výpočty paralelne na viacerých jadrách procesora. Pre použitie príkazu parfor však platia isté pravidlá.

- Iterácie cyklu sú vykonávané paralelne, **ale v náhodnom poradí.**
- **Iterácie musia byť po sebe idúce kladné celočíselné hodnoty.**
- **Telo cyklu musí byť nezávislé. Iterácie musia byť na sebe nezávislé.**
- **Cyklus parfor nemožno použiť vo vnútri iného parfor cyklu.**
- Pred spustením príkazu je potrebné spustiť parallel pool pomocou príkazu parpool (automatické).



Pool of MATLAB Workers

```
1     % Paralelný výpočet s funkciou parfor
2 -   parfor i=1:10
3 -       disp(i);
4 -   end
```

Command Window

```
>> parforloop
     1

     4

     3

     7

     8

     9

     2

     6

     5

    10
```

Slučky/cykly – paralelné výpočty – parfor

Matlab štandardne vykonáva príkazy sekvenčne na jednom jadre CPU. Príkaz parfor, ktorý je implementovaný v paralel computing toolboxe, umožňuje vykonávať výpočty paralelne na viacerých jadrách procesora. Pre použitie príkazu parfor však platia isté pravidlá.

- **Telo cyklu musí byť nezávislé. Iterácie musia byť na sebe nezávislé.**

```
2 -    clc
3 -    a = ones([1 8]);
4 -    for i=2:8
5 -        a(i) = i*a(i-1);
6 -    end
7
8 -    disp(a);
```

Pre príkaz **for** toto nepredstavuje problém.

Command Window

```
     1     2     6    24   120   720  5040 40320

fx >>
```

Slučky/cykly – paralelné výpočty – parfor

Matlab štandardne vykonáva príkazy sekvenčne na jednom jadre CPU. Príkaz parfor, ktorý je implementovaný v paralel computing toolboxe, umožňuje vykonávať výpočty paralelne na viacerých jadrách procesora. Pre použitie príkazu parfor však platia isté pravidlá.

- **Telo cyklu musí byť nezávislé. Iterácie musia byť na sebe nezávislé.**

```
2 -    clc
3 -    a = ones([1 8]);
4 -    parfor i=2:8
5 -        a(i) = i*a(i-1);
6 -    end
7
8 -    disp(a);
```

Pre príkaz **parfor** je to problém.

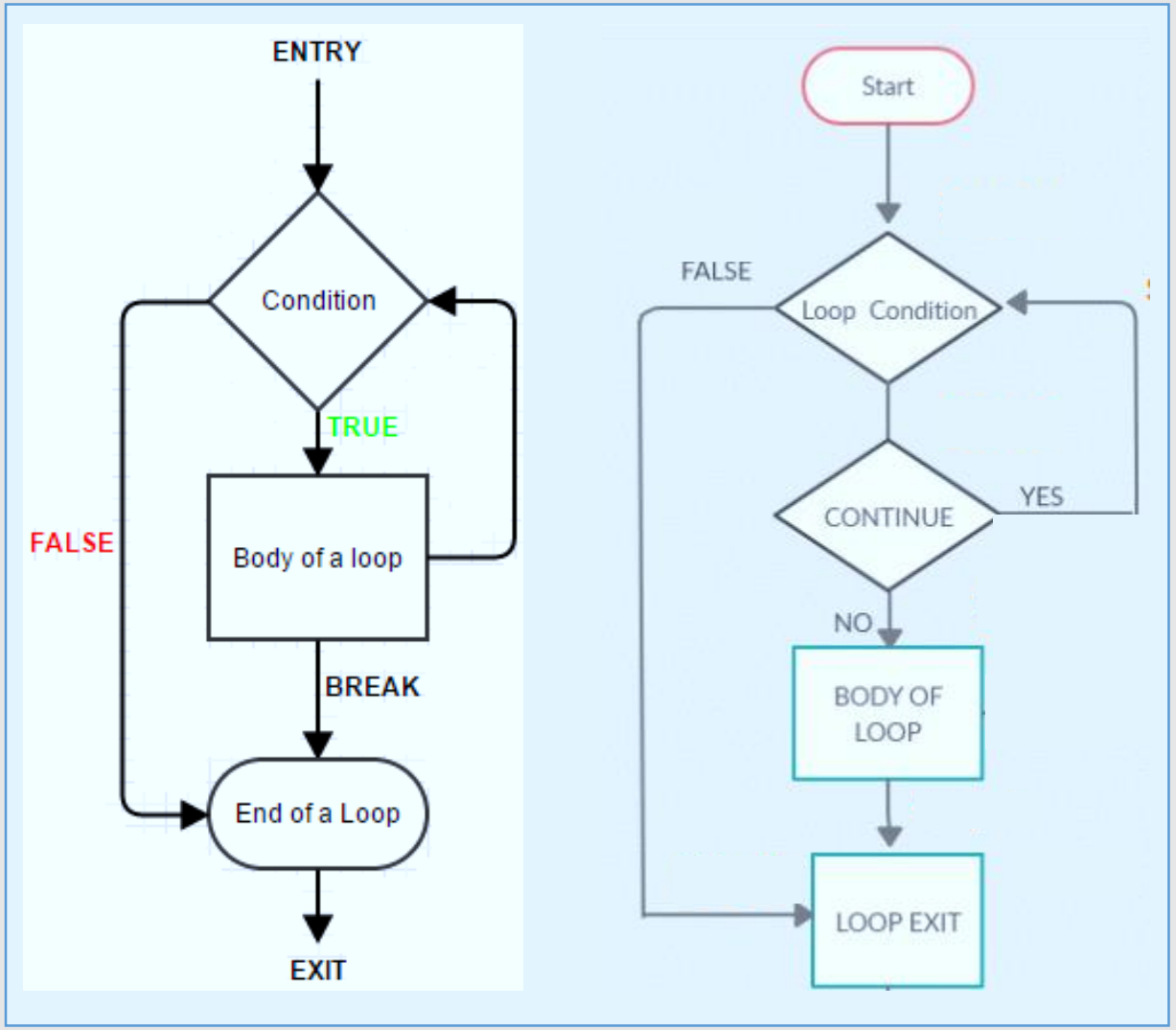
Command Window

```
Error using parforloop (line 4)
Error: Unable to classify the variable 'a' in the body of the parfor-loop. For more information, see Parallel for Loops in
MATLAB, "Solve Variable Classification Issues in parfor-Loops".

fx >>
```

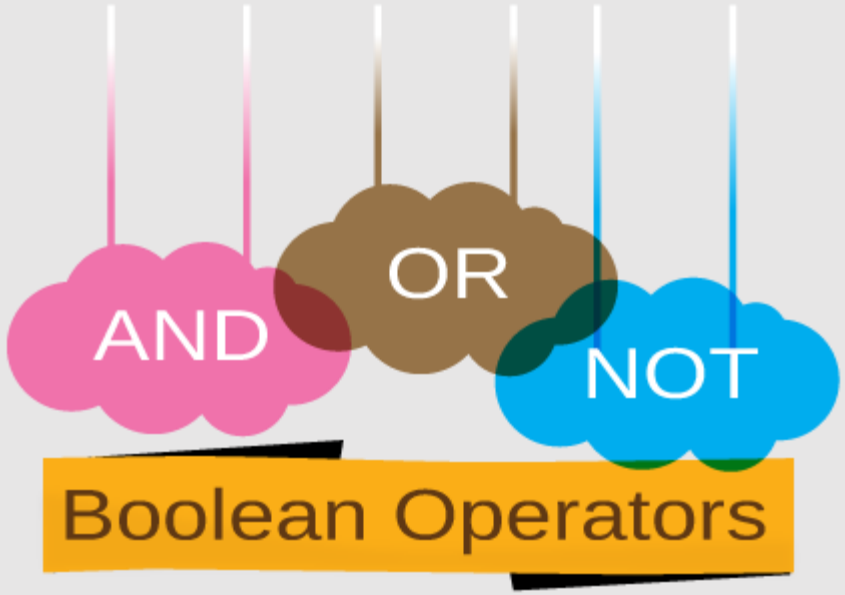
Slučky/cykly – Prerušenie slučky a vynechanie iterácie

- Vykonávanie slučky sa ukončuje vtedy, ak je splnená podmienka jej ukončenia.
- Niekedy je však žiadúce ukončiť slučku skôr, ako je táto podmienka splnená. Na prerušenie behu slučky z jej tela je možné použiť príkaz **break**.
- V prípade, že je potrebné vynechať jednu alebo viac iterácii, je možné použiť príkaz **continue**.
- Po vykonaní príkazu **continue** sa slučka posúva na začiatok ďalšej iterácie.
- V prípade potreby je možné vykonávanie slučky príkazom **pause(*trvanie_v_sekundách*)** pozastaviť na definovanú dobu, alebo bez uvedenia času bude slučka pozastavená až do momentu stlačenia tlačidla enter.



Slučky/cykly – Logické operácie

Príkazy riadenia a cyklu sú ukončované (ne)splnením podmienky. Len pre zopakovanie, rozlišujeme nasledovné logické operácie:



Operátor	Funkcia operátora
A == B	A je zhodné s B
A ~= B	A je rôzne od B
A < B	A je menšie ako B
A > B	A je väčšie od B
A <= B	A je menšie nanajvýš rovné B
A >= B	A je väčšie nanajvýš rovné B
~A	Neplatí A (A je negované)
||	Alebo
&&	Zároveň

Fakulta elektrotechniky
a informatiky

Úvod do Matlabu

Prednáška č. 5

- Riadiace príkazy (if, else a elseif)
- Príkazy slučiek (while, for a parfor)
- **Vlastné funkcie**

Ing. Ondrej Kováč, PhD.

Vlastné funkcie

- Na predchádzajúcej prednáške sme sa zaoberali skriptami a ich ladením. Skripty sú vhodné vtedy, ak chceme vykonávať nejakú časť kódu resp. opakovať niekoľko príkazov za sebou bez toho, aby sme to zakaždým zadávali z príkazového riadka.
- **Všetky premenné použité v skripte ostávajú vo workspace dovtedy, pokiaľ ich používateľ nevymaže.**

- Funkcie sa od skriptov líšia tým, že predstavujú ucelený kód, ktorého výsledkom je jedna alebo viacero výstupných premenných.
- Všetky premenné použité v tele funkcie sa po jej ukončení stratia.
- Funkcie majú spravidla viacero vstupných a výstupných premenných.
- Funkcia sa z kódu zavolá v tvare:

[vystup1, vystup2, … , vystupN] = **nazov_funkcie(**vstup1, vstup2, ..., vstupM**)**

- V Matlabe je možné hovoriť o niekoľkých typoch funkcie
  - **Anonymné funkcie**
  - **Funkcia definovaná vo vnútri skriptu** (Matlab v 2016b a novší).
  - **Regulárna funkcia** - definovaná v jednom m-súbore s názvom zhodným s názvom funkcie.
  - **Viaceré funkcie definované v jednom m-súbore**

Vlastné funkcie – Anonymne funkcie

- Anonymné funkcie sú uložené v premennej dátového typu **function_handle**.
- Anonymné funkcie, tak ako aj ostatné funkcie, majú vstupné a výstupné premenné.
- Definovať anonymnú funkciu môžeme priamo vo vnútri skriptu resp. priamo z príkazového riadku.
- Anonymne funkcie majú svoj význam, ale treba si dávať pozor na to, aby nedochádzalo k zvyšovaniu neprehľadnosti kódu!

Majme napríklad skript, v ktorom je potrebné často vykonávať súčet premennej so súčinom dvoch iných premenných. Vtedy je možné použiť anonymnú funkciu.

Command Window

```
>> sucet_sucin = @(x,y,z) (x*y+z);
>> k = sucet_sucin(5,2,2)

k =

    12

>>
```

Workspace

Name	Value	Size	Mean	Class
k	12	1x1	12	double
sucet_sucin	@(x,y,z)(x*y+z)	1x1		function_handle

Definícia anonymnej funkcie pozostáva z troch častí :

- **Názov_funkcie nasledovaný znakom „=“**
- **@(vstupne premenne)**
- **(operácia s premennými)**

Funkcia sa potom zavolá tak ako každá iná funkcia.

Vlastné funkcie – Regulárna funkcia

Ide o najbežnejšie používaný spôsob definície funkcie a tiež je veľmi dobre spätne podporovaná staršími verziami Matlabu.

- Funkcia je definovaná v jednom m-súbore s názvom zhodným s názvom funkcie.
- Môže mať viac vstupných a výstupných premenných.
- **Je to najprehľadnejšia forma kreovania vlastných funkcii.**

Názov funkcie by mal byť zhodný!

- **Funkcia by vždy mala mať hlavičku, kde sú informácie o jej tvorcovi a hlavne je v nej popísané ako sa má používať!**

bublinove_triedenie.m

```
% Funkcia je urcena na zoradenie prvkov ciselneho vektora
%
% Pouzitie:
% Pre vzostupne zoradenie:
% V = bublinove_triedenie(V)
%
% Pre zostupne zoradenie:
% V = bublinove_triedenie(V,'zostupne')
%
% V musi byt cisleny vektor s rozmerom minimalne 2
%
% Autor:
% Ondrej Kovac
% Technical university of Kosice
% (c)2017

function V = bublinove_triedenie(V,zoradenie)
    ln = length(V);
% test vstupnej premennej V
    if ln<2 || ~isnumeric(V) || ~isvector(V)
        error('V musi byt numericky vektor dlzky aspon 2');
    end
    vymena = 1;
    while vymena
        vymena = 0;
        for i=1:ln-1
            if V(i)>V(i+1)
               pom = V(i);
               V(i) = V(i+1);
               V(i+1) = pom;
               vymena = 1;
            end
        end
    end
% Zostupne usporiadanie
     if  strcmp(zoradenie,'zostupne')
       V = flip(V);
     else
       error('zly vstupny parameter');
     end
end
```

Vlastné funkcie – Regulárna funkcia

Vždy keď píšeme funkciu, mali by sme myslieť na to, že tá by mala overovať to, či zadané parametre sú také, aké majú byť **(blbuvzdornosť alebo Idiot-proof).**

As programmers create bigger and better idiot proof programs, so the universe creates bigger and better idiots!

```
17    function V = bublinove_triedenie(V,zoradenie)
18 -      ln = length(V);
19    % test vstupnej premennej V
20 -      if ln<2 || ~isnumeric(V) || ~isvector(V)
21 -          error('V musi byt numericky vektor dlzky aspon 2');
22 -      end
23 -      vymena = 1;
24 -      while vymena
25 -          vymena = 0;
26 -          for i=1:ln-1
27 -              if V(i)>V(i+1)
28 -                  pom = V(i);
29 -                  V(i) = V(i+1);
30 -                  V(i+1) = pom;
31 -                  vymena = 1;
32 -              end
33 -          end
34 -      end
35    % Zostupne usporiadanie
36 -       if  strcmp(zoradenie,'zostupne')
37 -         V = flip(V);
38 -       else
39 -         error('zly vstupny parameter');
40 -       end
41 - end
```

Vlastné funkcie – Funkcia definovaná v súbore so skriptom alebo inou funkciou

- Táto možnosť je podporovaná v Matlabe verzii 2016b a novšej.
- Túto možnosť využijeme vtedy, ak pracujeme s nie príliš rozsiahlym skriptom a potrebujeme definovať jednoduchú funkciu, ktorú už ale nie je vhodné definovať pomocou anonymnej funkcie.

Regulárna funkcia

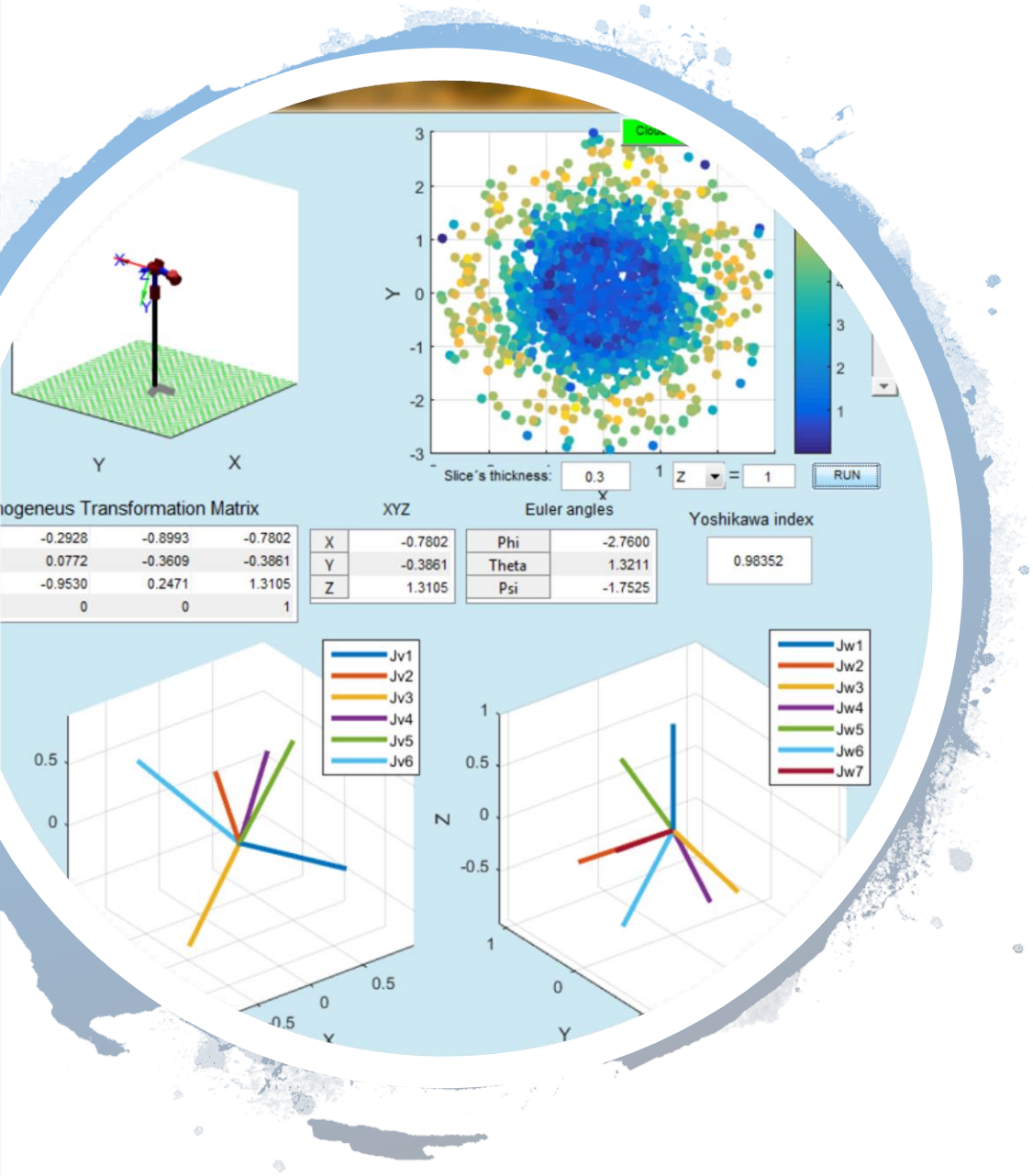
stat2.m

```
1     % Hlavna funkcia
2     function [m,s] = stat2(x)
3         n = length(x);
4         m = priemer(x,n);
5         s = sqrt(sum((x-m).^2/n));
6     end
7
8     % Lokalna funkcia
9     function m = priemer(x,n)
10        m = sum(x)/n;
11    end
```

```
1     % Toto je časť skriptu
2     k = sucet_sucin(5,2,2);
3
4     % Tu je definovana funkcia
5     function k = sucet_sucin(x,y,z)
6         k = x*y+z;
7     end
```

Vlastné funkcie – Niektoré užitočné funkcie

Príkaz	Funkcia príkazu
nargin	Zisti počet vstupných premenných funkcie
nargout	Zisti počet výstupných premenných funkcie
varargin	Vráti zoznam vstupných premenných
varargout	Vráti zoznam výstupných premenných
mfilename	Vráti názov m-súboru funkcie



Nabudúce

- *Návrh grafického používateľského prostredia GUI*
- *Nástroj GUIDE*