INTRODUCERE ÎN SOFTWARE MATEMATIC
CURS #5

Liviu Marin

Facultatea de Matematică și Informatică, Universitatea din București, România

E-mails: marin.liviu@gmail.com; liviu.marin@fmi.unibuc.ro

CUPRINS - CURS #5

1. Programare în MATLAB
   - Operatori relaționali
   - Operatori logici
   - Declarații condiționale
   - Cicluri
   - Cicluri și declarații condiționale imbricate

Programare în MATLAB

Operatori relaționali

Operator relațional	Descriere
$<$	Mai mic (strict)
$>$	Mai mare (strict)
$<=$	Mai mic sau egal
$>=$	Mai mare sau egal
$==$	Egal
$\tilde{}=$	Diferit

Operatori relaționali

- Un operator relațional compară elementele corespunzătoare a două tablouri (în sens extins) de aceeași dimensiune și determină dacă aceste comparări sunt adevărate sau false.
- Rezultatul unei comparări realizate printr-un operator relațional este de tip logic, i.e. 1 (adevărat) sau 0 (fals).
- Operatorii relaționali sunt folosiți ca operatori aritmetici într-o expresie matematică, iar rezultatul poate fi folosit în alte operații matematice, la apelarea tablourilor și, împreună cu alte comenzi MATLAB (de exemplu, `if`, `while`), la controlul fluxului unui program.
- Dacă sunt comparate două numere, rezultatul este de tip logic, i.e. 1 (adevărat) sau 0 (fals).

- Dacă sunt comparate două tablouri, tablourile trebuie să aibă aceleași dimensiuni, compararea se face element cu element, iar rezultatul este un tablou de tip logic de aceleași dimensiuni ca cele ale celor două tablouri comparate, i.e. cu elementele 1 (adevărat) sau 0 (fals).
- Dacă un scalar este comparat cu un tablou, atunci scalarul este comparat cu fiecare element al tabloului, iar rezultatul este un tablou de tip logic de aceleași dimensiuni ca cele ale tabloului care se compară cu scalarul, i.e. cu elementele 1 (adevărat) sau 0 (fals).
- Rezultatul unei operații relaționale cu tablouri este un tablou de tip logic, i.e. cu elementele 1 (adevărat) sau 0 (fals).
- Tablourile numerice cu elemente 0 sau 1 sunt diferite de tablourile logice ale căror elemente sunt 0 sau 1:
  - tablourile numerice nu pot fi folosite la apelarea tablourilor;
  - tablourile logice pot fi folosite la apelarea tablourilor;
  - tablourile logice pot fi folosite în operații aritmetice;

  - la prima utilizare a unui tablou logic în operații aritmetice, tabloul respectiv devine tablou numeric.
- Într-o expresie matematică, operațiile aritmetice au prioritate față de operațiile relaționale. Parantezele pot schimba ordinea operațiilor.
- Operațiile relaționale au aceeași prioritate una față de alta, fiind evaluate de la stânga la dreapta. Parantezele pot schimba ordinea operațiilor.

EXEMPLUL#1: Operatori relaționali

```
% EXEMPLUL#1(a)
a = 5 > 8;        fprintf('a = %i\n', a)
b = 5 <= 10;      fprintf('b = %i\n', b)
c = (6 < 10) + (7 > 8) + (5*3 == 60/4);
fprintf('c = %i\n', c)

% EXEMPLUL#1(b)
x = [15 6 9 4 11 7 14];
y = [8 20 9 2 19 7 10];
z = x >= y;       disp('z = '), disp(z)
v = x == y;       disp('v = '), disp(v)
w = x ~= y;       disp('w = '), disp(w)
f = x - y > 0;    disp('f = '), disp(f)
```


EXEMPLUL#2: Operatori relaționali

```
% EXEMPLUL#2(a)
A = [2 9 4; -3 5 2; 6 7 -4];
disp('A = '),    disp(A)
B = A <= 2;      disp('B = '), disp(B)

% EXEMPLUL#2(b)
r = [8 12 9 4 23 19 10];
s = r <= 10;     disp('s = '), disp(s)
t = r(s);        disp('t = '), disp(t)
w = r(r <= 10); disp('w = '), disp(w)

% EXEMPLUL#2(c)
a = 3 + 4 < 16 / 2;      disp(a)
b = 3 + (4 < 16) / 2;  disp(b)
```

Operatori logici

Operator logic	Descriere
& **Exemplu:** `A & B`	Operează asupra a două argumente (`A` și `B`). **Rezultatul:** adevărat (1) dacă `A` și `B` sunt adevărate; fals (0) în caz contrar.
| **Exemplu:** `A | B`	Operează asupra a două argumente (`A` și `B`). **Rezultatul:** adevărat (1) dacă `A` sau `B` este adevărat; fals (0) în caz contrar.
~ **Exemplu:** `~A`	Operează asupra unui singur argument (`A`). **Rezultatul:** adevărat (1) dacă `A` este fals; fals (0) dacă `A` este adevărat.

Operatori logici

- Operatorii logici acționează asupra numerelor: un număr diferit de zero este adevărat, iar numărul zero este fals.
- Operatorii logici sunt folosiți ca operatori aritmetici într-o expresie matematică, iar rezultatul poate fi folosit în alte operații matematice. Rezultatul unui operator logic poate fi folosit, împreună cu alte comenzi MATLAB (de exemplu, `if`, `while`), la controlul fluxului unui program.
- Operatorii logici și (&) și sau (|) pot avea ambele argumente scalari, ambele argumente tablouri sau un argument scalar și un altul tablou:
  - dacă ambele argumente sunt scalari, rezultatul este un scalar 0 sau 1;
  - dacă ambele argumente sunt tablouri, rezultatul este un tablou de dimensiunea celor două implicate în operația logică și ale cărui elemente sunt 0 și 1, iar operația logică se face element cu element;

  - dacă unul dintre argumente este un scalar și celălalt este un tablou, atunci operația logică se efectuează între scalar și fiecare element al tabloului, iar rezultatul este un tablou de aceeași dimensiune cu cel implicat în operația logică și ale cărui elemente sunt 0 și 1.
- Operatorul logic de negare (~) are un singur argument, care poate fi un scalar sau un tablou:
  - dacă argumentul este un scalar, rezultatul este un scalar 0 sau 1;
  - dacă argumentul este un tablou, rezultatul este un tablou de aceeași dimensiune cu cel implicat în operația logică și ale cărui elemente sunt 0 și 1, iar operația logică se face element cu element.

EXEMPLUL#3: Operatori logici

```
% EXEMPLUL#3(a)
a = 3 & 7;          fprintf('a = %i\n', a)
b = 5 | 0;          fprintf('b = %i\n', b)
c = ~25;            fprintf('c = %i\n', c)
d = 25*((12 & 0) + (~0) + (0|5));
fprintf('d = %i\n', d)

% EXEMPLUL#3(b)
x = [9 3 0 11 0 15];
y = [2 0 13 -11 0 4];
z = x & y;          disp('z = '), disp(z)
v = x | y;          disp('v = '), disp(v)
w = ~(x + y);       disp('w = '), disp(w)
```

- Operatorii logici, relaţionali şi aritmetici pot fi combinaţi în expresii matematice, iar rezultatul unei astfel de expresii depinde de ordinea în care sunt efectuate operaţiile.
- Dacă două sau mai multe operaţii au aceeaşi prioritate, expresia este executată de la stânga la dreapta.

Prioritate	Operaţie
1 (cea mai mare)	Parantezele (de la interior spre exterior pentru imbricate)
2	Ridicarea la putere
3	Negarea logică (~)
4	Înmulţirea şi împărţirea
5	Adunarea şi scăderea
6	Operatorii relaţionali (<, >, <=, >=, ==, ~=)
7	Conjuncţia logică (&)
8 (cea mai mică)	Disjuncţia logică (|)

EXEMPLUL#4: Operatori logici, relaţionali şi aritmetici

```
% EXEMPLUL#4
x =-2;
y = 5;

a = -5 < x < 1;              fprintf('a = %i\n', a)
b = -5 < x & x < 1;          fprintf('b = %i\n', b)
c = ~(y < 7);                fprintf('c = %i\n', c)
d = ~y < 7;                  fprintf('d = %i\n', d)
e = ~((y >= 8) | (x < 1));   fprintf('e = %i\n', e)
f = ~(y >= 8) | (x < 1);     fprintf('f = %i\n', f)
```


Funcţii logice predefinite MATLAB

Funcţie	Descriere
`and(A,B)`	Conjuncţie logică, i.e. echivalent cu `A & B`
`or(A,B)`	Disjuncţie logică, i.e. echivalent cu `A | B`
`not(A)`	Negare logică, i.e. echivalent cu `~A`
`xor(a,b)`	Sau exclusiv. **Rezultatul:** adevărat (1) dacă cel puţin unul dintre argumente este adevărat; fals (0) în caz contrar.
`all(A)`	**Rezultatul:** adevărat (1) dacă toate elementele vectorului `A` sunt adevărate (nenule); fals (0) în caz contrar. Dacă `A` este matrice, fiecare coloană a lui `A` este tratată ca un un vector şi rezultatul este un vector linie de tip logic.
`any(A)`	**Rezultatul:** adevărat (1) dacă unul dintre elementele vectorului `A` este adevărat (nenul); fals (0) în caz contrar. Dacă `A` este matrice, fiecare coloană a lui `A` este tratată ca un vector şi rezultatul este un vector linie de tip logic.
`find(A)`	Dacă `A` este un vector, returnează un vector linie care conţine indicii elementelor nenule ale lui `A`.
`find(A) > d`	Dacă `A` este un vector, returnează un vector linie care conţine indicii elementelor lui `A` mai mari ca `d` (orice operator relaţional poate fi folosit).

Declaraţii condiţionale

- Declaraţiile condiţionale sunt comenzi MATLAB care permit luarea unei decizii de executare sau neexecutare a unui grup de instrucţiuni care urmează după declaraţia condiţională respectivă.
- Într-o declaraţie condiţională se găseşte o expresie condiţională care se evaluează:
  - dacă expresia condiţională este adevărată, atunci grupul de instrucţiuni care urmează după declaraţia condiţională se execută;
  - dacă expresia condiţională este falsă, atunci grupul de instrucţiuni care urmează după declaraţia condiţională nu se execută.
- Declaraţiile condiţionale pot fi parte a unui fişier script sau a unei funcţii MATLAB.

1. Structura condiţională `if-end`

```
if condition
    statement#1
    statement#2
    ..................
    statement#i
end
```

- *condition* – expresie condiţională (logică) în MATLAB a cărei valoare logică (0 sau 1) se calculează;
- *statement#1*, *statement#2*, . . . , *statement#i* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition* este adevărată (1) şi se omit dacă valoarea expresiei condiţionale *condition* este falsă (0).

2. Structura condiţională `if-else-end`

```
if condition
    statement#1.1
    ..................
    statement#1.i
else
    statement#2.1
    ..................
    statement#2.j
end
```

- *condition* – expresie condiţională (logică) în MATLAB a cărei valoare logică (0 sau 1) se calculează;
- *statement#1.1*, . . . , *statement#1.i* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition* este adevărată (1), după care se omit toate declaraţiile până la instrucţiunea `end`;
- *statement#2.1*, . . . , *statement#2.j* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition* este falsă (0).

3. Structura condiţională `if-elseif-else-end`

```
if condition#1
    statement#1.1
    ..................
    statement#1.i
elseif condition#2
    statement#2.1
    ..................
    statement#2.j
else
    statement#3.1
    ..................
    statement#3.k
end
```

- *condition#1*, *condition#2* – expresii condiţionale (logice) în MATLAB ale căror valori logice (0 sau 1) se calculează;

- *statement#1.1*, . . . , *statement#1.i* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition#1* este adevărată (1), după care se omit toate declaraţiile până la instrucţiunea `end`;
- *statement#2.1*, . . . , *statement#2.j* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition#1* este falsă (0) şi valoarea expresiei condiţionale *condition#2* este adevărată (1), după care se omit toate declaraţiile până la instrucţiunea `end`;
- *statement#3.1*, . . . , *statement#3.k* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition#1* este falsă (0) şi valoarea expresiei condiţionale *condition#2* este falsă (0).

Observaţie:

- Unei structuri condiţionale `if-elseif-else-end` i se pot adăuga oricâte instrucţiuni `elseif` sunt necesare.

4. Structura condiţională `if-elseif-end`

Observaţie:

- Într-o structură condiţională `if-elseif-else-end` se poate omite instrucţiunea `else` dacă este necesar aşa ceva, devenind astfel `if-elseif-end`.

```
if condition#1
    statement#1.1
    ..................
    statement#1.i
elseif condition#2
    statement#2.1
    ..................
    statement#2.j
end
```

- *condition#1*, *condition#2* – expresii condiţionale (logice) în MATLAB ale căror valori logice (0 sau 1) se calculează;
- *statement#1.1*, ..., *statement#1.i* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition#1* este adevărată (1), după care se omit toate declaraţiile până la instrucţiunea end;
- *statement#2.1*, ..., *statement#2.j* – seturi de declaraţii în MATLAB care se execută dacă valoarea expresiei condiţionale *condition#1* este falsă (0) şi valoarea expresiei condiţionale *condition#2* este adevărată (1), după care se omit toate declaraţiile până la instrucţiunea end.

5. Structura condiţională `switch-case`

```
switch expression
    case value#1
        statement#1.1
        ..........................
        statement#1.n_1
    case value#2
        statement#2.1
        ..........................
        statement#2.n_2
    ..............................
    case value#(N-1)
        statement#(N-1).1
        ..........................
        statement#(N-1).n_{N-1}
    otherwise
        statement#N.1
        ..........................
        statement#N.n_N
end
```

- *expression* – expresie ce poate fi un scalar, un şir de caractere (string) sau o variabilă a cărei valoare este un scalar sau un şir de caractere (string);
- *value#1*, ..., *value#(N-1)* – scalari sau şiruri de caractere (stringuri) asociate cazurilor respective, unde $k \in \overline{1, N-1}$;
- otherwise – instrucţiune opţională complementară cazurilor considerate;
- *statement#k.1*, ..., *statement#k.*n_k – seturi de declaraţii în MATLAB care se execută dacă *expression* = *value#k*, unde $k \in \overline{1, N}$.

Observaţii:

- Valoarea expresiei *switch expression* este comparată cu valorile *value#1*, ..., *value#(N-1)* asociate cazurilor respective.
- Dacă valoarea expresiei *switch expression* coincide cu cel puţin una dintre valorile *value#1*, ..., *value#(N-1)*, atunci se execută doar comenzile asociate primului caz $k \in \overline{1, N-1}$ pentru care *switch expression* = *value#k*, după care se omit toate celelalte declaraţii până la instrucţiunea end.
- Unui caz $k \in \overline{1, N-1}$ din structura condiţională `switch-case` i se pot atribui mai multe valori sub forma $\{value\#k.1, value\#k.2, \ldots, value\#k.m_k\}$.

EXEMPLUL#5: Structuri condiționale

Un rezervor dintr-un turn de apă are forma unui cilindru de diametru $d_{\text{cilindru}} = 25$ m și înălțime $h_{\text{cilindru}} = 19$ m, deasupra căruia se găsește un trunchi de con răsturnat, cu diametrul bazei mici dat de $d_{\text{con}} = d_{\text{cilindru}}$, diametrul bazei mari dat de $D_{\text{con}} = 46$ m și înălțime $h_{\text{con}} = 14$ m. În interiorul rezervorului, o baliză de control plutește pe apă și indică nivelul apei din rezervor.

Scrieți un script în MATLAB care să determine volumul de apă din rezervor în funcție de poziția balizei dată de distanța h la care se găsește față de sol.

Programul cere utilizatorului să introducă poziția balizei în m, $h > 0$, și volumul de apă din rezervor în m^3.

EXEMPLUL#5: Declarații condiționale

```
% EXEMPLUL#5
% Cilindru
d_cilindru = 25;              % Diametru
r_cilindru = d_cilindru/2;    % Raza
h_cilindru = 19;              % Inaltime
% Con
r_con = r_cilindru;           % Raza baza mica
D_con = 46;                   % Diametru baza mare
R_con = D_con/2;              % Raza baza mare
h_con = 14;                   % Inaltime
% Pozitie baliza
h = input('Pozitia balizei h[m] = ');
% Volumul de apa din rezervor
if h > h_cilindru + h_con
    disp('Pozitia balizei nu poate fi mai mare ca
    inaltimea rezervorului!')
```



```
elseif h < 0
    disp('Pozitia balizei nu poate fi sub nivelul
    solului!')
elseif h <= 19
    V = pi*r_cilindru^2*h;
    fprintf('Volumul de apa din rezervor este de
    %-.2f m cubi\n', V)
else
    r = r_con + (R_con - r_con)*(h - h_cilindru)/
    h_con;
    V = pi*r_cilindru^2*h + ...
        pi*(h - h_cilindru)*(r_con^2 + r_con*r + r
    ^2)/3;
    fprintf('Volumul de apa din rezervor este de
    %-.2f m cubi\n', V)
end
```


Cicluri

- Un ciclu este o altă modalitate de modificare a fluxului unui program.
- Într-un ciclu, o instrucțiune sau mai multe instrucțiuni sunt executate repetitiv de un anumit număr de ori. O astfel de execuție completă a instrucțiunii sau instrucțiunilor respective s.n. pas.
- În cadrul fiecărui pas al unui ciclu, uneia sau mai multor variabile li se alocă noi valori.
- În MATLAB există două tipuri de cicluri:
  - ciclurile `for-end`, în cazul cărora numărul de pași este specificat la începutul ciclului;
  - ciclurile `while-end`, în cazul cărora numărul de pași nu este cunoscut, dar procesul repetitiv se execută până când o condiție este îndeplinită.
- Ciclurile pot fi terminate în orice moment prin intermediul instrucțiunii `break`.

1. Ciclul `for-end`

```
for k = f : s : t
    statement#1
    ..................
    statement#n
end
```

- k – contorul ciclului (numără paşii ciclului);
- f – valoarea contorului ciclului la primul pas;
- s – incrementul contorului ciclului de la un pas la următorul, cu menţiunea că s poate fi pozitiv sau negativ;
- t – valoarea maximă admisibilă a contorului ciclului la ultimul pas dacă $f < t$ şi $s > 0$, respectiv valoarea minimă admisibilă a contorului ciclului la ultimul pas dacă $f > t$ şi $s < 0$;
- *statement#1*, ..., *statement#n* – seturi de declaraţii în MATLAB care se execută la fiecare pas al ciclului.

Observaţii:

- Dacă $f = t$, atunci ciclul este excutat o singură dată.
- Dacă $f > t$ şi $s > 0$ sau $f < t$ şi $s < 0$, atunci ciclul nu este executat.
- Într-o instrucţiune `for`, contorul k poate avea prescrise anumite valori, scrise sub forma unui vector linie, e.g. $k = [7\ 9\ -1\ 3\ 3\ 5]$.
- Fiecărei instrucţiuni `for` îi corespunde o instrucţiune `end`.
- Valoarea contorului ciclului, k, nu este afişată automat, dar aceasta se poate face prin instrucţiunile uzuale de afişare dacă este cazul (de exemplu, la repararea unui cod).
- La finalul ciclului `for-end`, contorului ciclului, k, are ultima valoare care i-a fost alocată.

EXEMPLUL#6: Ciclul `for-end`

Folosind un ciclu `for-end`, scrieţi un fişier script în MATLAB care să calculeze suma primilor n termeni ai seriei

$$\sum_{k=1}^{n} \frac{(-1)^k k}{2^k}.$$

Programul cere utilizatorului să introducă numărul de termeni n ai seriei de mai sus şi trebuie să afişeze rezultatul formatat în virgulă fixă cu 6 zecimale.

Executaţi programul pentru $n = 4$, $n = 10$ şi $n = 20$.

EXEMPLUL#6: Ciclul `for-end`

```
1  % EXEMPLUL#6: Suma primilor n termeni ai unei serii
2  n = input('Introduceti numarul de termeni ai seriei
       n = ');
3  % Initializare
4  S = 0;
5  % Sumare
6  for k = 1 : n
7      S = S + (-1)^k*k/2^k;
8  end
9  % Afisarea rezultatului
10 fprintf('Suma primilor %-i termeni ai seriei este
       %-.6f\n', n, S)
```

EXEMPLUL#7: Modificarea elementelor unui vector

Scrieţi un fişier script în MATLAB care modifică elementele unui vector introdus de utilizator astfel:

(i) dublează elementele vectorului care sunt pozitive şi sunt divizibile cu 3 sau cu 5;

(ii) ridică la cub elementele vectorului care sunt negative şi mai mari sau egale cu -5.

Programul afişează vectorul modificat conform celor de mai sus. Rulaţi programul pentru vectorul

$$[5,\ 17,\ -3,\ 8,\ 0,\ -7,\ 12,\ 15,\ 20,\ -6,\ 6,\ 4,\ -2,\ 16].$$

EXEMPLUL#7: Modificarea elementelor unui vector

```
% EXEMPLUL#7: Modificarea elementelor unui vector
V = input('Introduceti vectorul linie V = ');
n = length(V);      % Lungimea vectorului
% Modificarea elementelor vectorului V
for k = 1 : n
    if V(k) > 0 && (rem(V(k), 3) == 0 || rem(V(k),
    5) == 0)
        V(k) = 2*V(k);
    elseif V(k) < 0 && V(k) >= -5
        V(k) = V(k)^3;
    end
end
for k = 1 : n       % Afisarea rezultatului
    fprintf('V(%-i) = %-.f\n', k, V(k))
end
```


2. Ciclul `while-end`

```
while conditional expression
    statement#1
    ..................
    statement#n
end
```

- *conditional expression* – expresie condiţională (logică);
- *statement#1*, ..., *statement#n* – seturi de declaraţii în MATLAB care se execută la fiecare pas al ciclului.

Observaţii:

- Expresia condiţională *conditional expression* trebuie să includă cel puţin o variabilă.
- Variabilele din expresia condiţională *conditional expression* trebuie să aibă valori alocate atunci când MATLAB execută instrucţiunea `while`.

- La fiecare pas al ciclului, expresia condiţională (logică) *conditional expression* este evaluată şi i se atribuie valorile logice corespunzătoare (0 sau 1).
- Declaraţiile MATLAB *statement#1*, ..., *statement#n* se execută atâta vreme cât expresia condiţională *conditional expression* este adevărată (1).

  În caz contrar, se execută instrucţiunea finală a ciclului `end` şi se iese din ciclu.
- Cel puţin uneia dintre variabilele din expresia condiţională *conditional expression* trebuie să i se aloce o nouă valoare prin declaraţiile MATLAB care se execută la fiecare pas al ciclului, i.e. *statement#1*, ..., *statement#n*.
- Atenţie la evitarea aşa-numitelor "cicluri infinite"!

EXEMPLUL#8: LABORATOR #4, EX#5

Scrieţi un fişier script în MATLAB® care calculează arctg(x), $x \in (-1,1)$, folosind seria

$$\sum_{k=0}^{\infty}(-1)^k \frac{x^{2k+1}}{2k+1} = \text{arctg}(x), \quad x \in (-1,1).$$

Testaţi programul pentru un vector MATLAB® $x \in (-1,1)$ şi comparaţi rezultatele obţinute cu funcţia predefinită MATLAB® `atan`, listând într-un tabel valorile x, arctg(x) calculat de programul de mai sus, arctg(x) dat de funcţia predefinită MATLAB® `atan` şi erorile absolute şi relative corespunzătoare.

EXEMPLUL#8.1: LABORATOR #4, EX#5

```
% EXEMPLUL#8.1: LABORATOR#4, EX#5
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++
clear;  clc;    close all
% Defineste argumentul seriei
x = linspace(-1 + 1.e-5, 1 - 1.e-5, 11);
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Calculul atan(x) cu toleranta epsilon = 1e-6
epsilon = 1e-6;
fprintf('x\t\t\t Serie\t\t\t Exact\t\t\t RelErr\n')
for i=1:length(x)
    sum0 = 0;
    sum1 = x(i);
    k = 1;
    while( abs(sum1 - sum0) > epsilon*abs(sum0) )
        sum0 = sum1;
        term = (-1)^k*x(i).^(2*k+1)./(2*k+1);
```



```
        sum1 = sum0 + term;
        k = k + 1;
    end
% Afisarea rezultatelor
    fprintf('%-+.5e\t %-+.8e\t %-+.8e\t %-.4e\n',
    ...
        x(i), sum1, atan(x(i)), abs(atan(x(i)) -
    sum1)/atan(exp(x(i))) );
end
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```


EXEMPLUL#8.2: LABORATOR #4, EX#5

```
% EXEMPLUL#8.2: LABORATOR#4, EX#5
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++
clear;  clc;    close all
% Defineste argumentul seriei
x = linspace(-1 + 1.e-5, 1 - 1.e-5, 11);
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Calculul atan(x) cu precizia masinii, eps
fprintf('x\t\t\t Serie\t\t\t Exact\t\t\t RelErr\n')
for i=1:length(x)
    sum0 = 0;
    sum1 = x(i);
    k = 1;
    while( abs(sum1 - sum0) > eps*abs(sum0) )
        sum0 = sum1;
        term = (-1)^k*x(i).^(2*k+1)./(2*k+1);
        sum1 = sum0 + term;
```

```
17          k = k + 1;
18      end
19  % Afisarea rezultatelor
20      fprintf('%-+.5e\t %-+.8e\t %-+.8e\t %-.4e\n',
        ...
21          x(i), sum1, atan(x(i)), abs(atan(x(i)) -
        sum1)/atan(exp(x(i))) );
22  end
23  %++++++++++++++++++++++++++++++++++++++++++++++++
```


Cicluri şi declaraţii condiţionale imbricate

- Ciclurile şi declaraţiile condiţionale pot fi incluse în alte cicluri şi/sau declaraţii condiţionale, i.e. pot fi imbricate.
- Nu există o limitare a numărului de cicluri şi/sau declaraţii condiţionale care pot fi imbricate.
- Trebuie avut în vedere că:
  - fiecare ciclu începe cu instrucţiunea `for` sau `while` şi se încheie cu instrucţiunea `end`;
  - fiecare declaraţie condiţională începe cu instrucţiunea `if` sau `switch` şi se încheie cu instrucţiunea `end`.


```
for k = 1 : n
    statement#1.1
    ..................
    statement#1.m
    for ℓ = 1 : m
        statement#2.1
        ..................
        statement#2.n2
    end
    statement#1.(m+1)
    ..................
    statement#1.n1
end
```

- k – contorul ciclului exterior;
- ℓ – contorul ciclului interior;
- *statement#1.1*, ..., *statement#1.*n_1 – seturi de declaraţii în MATLAB care se execută la fiecare pas al ciclului exterior;
- *statement#2.1*, ..., *statement#2.*n_2 – seturi de declaraţii în MATLAB care se execută la fiecare pas al ciclului interior.

EXEMPLUL#9: Generarea unei matrice prin cicluri imbricate

Scrieţi un fişier script în MATLAB care generează elementele unei matrice $\mathscr{M}_{m,n}(\mathbb{R})$ definite astfel:

(i) valoarea fiecărui element de pe prima linie este egală cu numărul coloanei sale;

(ii) valoarea fiecărui element de pe prima coloană este egală cu numărul liniei sale;

(iii) restul elementelor au valoarea egală cu suma valorilor elementului de deasupra sa şi a celui de la stânga sa.

Când este rulat, programul cere utilizatorului să introducă numărul de linii, m, şi numărul de coloane, n, ale matricei.

EXEMPLUL#9: Generarea unei matrice prin cicluri imbricate

```
% EXEMPLUL#9: Generarea unei matrice
m = input('Introduceti numarul de linii m = ');
n = input('Introduceti numarul de coloane n = ');
A = zeros(m, n);    % Initializarea matricei A
% Definirea elementelor matricei A
for k = 1 : m
    for l = 1 : n
        if k == 1
            A(k, l) = l;
        elseif l == 1
            A(k, l) = k;
        else
            A(k, l) = A(k, l-1) + A(k-1, l);
        end
    end
end
```



```
% Afisarea rezultatului
disp(' ')
disp('A = '), disp(A)
```


Instrucțiunea `break`

- În interiorul unui ciclu, instrucțiunea `break` termină execuția acelui ciclu, i.e. MATLAB sare la instrucțiunea finală a ciclului respectiv (`end`) și continuă cu execuția primei comenzi din afara acelui ciclu.
- Dacă instrucțiunea `break` se găsește în interiorul unui ciclu interior unui (imbricat într-un) alt ciclu, atunci se termină execuția ciclului interior (imbricat).
- Dacă instrucțiunea `break` se găsește într-un fișier script, în afara unui ciclu, atunci se termină execuția acelui script.

Instrucțiunea `continue`

- Instrucțiunea `continue` poate fi folosită în interiorul unui ciclu pentru a opri execuția pasului actual și a trece la următorul pas al ciclului respectiv.
- Instrucțiunea `continue` este, de regulă, parte a unei declarații condiționale. Când MATLAB ajunge la instrucțiunea `continue`, nu se mai execută declarațiile rămase în interiorul ciclului la pasul actual, se sare la instrucțiunea finală a ciclului (`end`) și apoi se începe execuția următorului pas al ciclului.