# Programare în MATLAB

## Cuprins



### Fluxul de control

MATLAB are patru structuri de control: instrucţiunea `if`, instrucţiunea de ciclare `for`, instrucţiunea de ciclare `while` şi instrucţiunea `switch`. Cea mai simplă formă a instrucţiunii `if` este

`if` *expresie*

*instrucţiuni*

`end`

unde secvenţa *instrucţiuni* este executată dacă părţile reale ale elementelor lui *expresie* sunt toate nenule. Secvenţa de mai jos interschimbă `x` şi `y` dacă `x` este mai mare decât `y`:

```
x = 5; y = 3;
if x > y
    temp = y;
    y = x;
    x = temp;
end
```

Atunci când o instrucţiune `if` este urmată în aceeaşi linie de alte instrucţiuni, este nevoie de o virgulă pentru a separa `if`-ul de instrucţiunea următoare:

```
if x > 0, x = sqrt(x); end
```

Alternativa se implementează cu `else`, ca în exemplul

```
a = pi^exp(1); c = exp(pi);
if a >= c
    b = sqrt(a^2-c^2)
else
    b = 0
end
```

```
b =
     0
```

În fine, se pot introduce teste suplimentare cu `elseif` (de notat că nu este nici un spaţiu între `else` şi `if`):

```
if a >= c
    b = sqrt(a^2-c^2)
elseif a^c > c^a
    b = c^a/a^c
else
    b = a^c/c^a
end
```

```
b =
   0.234726930419171
```

Ciclul `for` este una dintre cele mai utile construcţii MATLAB, deşi codul este mai compact fără ea. Sintaxa ei este

`for` *variabilă = expresie*

*instrucţiuni*

`end`

De obicei, *expresie* este un vector de forma `i:s:j`. Instrucţiunile sunt executate pentru *variabilă* egală cu fiecare element al lui *expresie* în parte. De exemplu, suma primilor 25 de termeni ai seriei armonice $1/i$ se calculează prin

```
s = 0;
for i = 1:25, s = s + 1/i; end, s
```

```
s =
   3.815958177753507
```

Un alt mod de a defini *expresie* este utilizarea notaţiei cu paranteze pătrate:

```
format short
for x = [pi/6 pi/4 pi/3], disp([x, sin(x)]), end
```

```
    0.5236    0.5000

    0.7854    0.7071

    1.0472    0.8660
```

Ciclurile for pot fi imbricate, indentarea ajutând în acest caz la creşterea lizibilităţii. Editorul-debuger-ul MATLAB poate realiza indentarea automată. Codul următor construieşte o matrice simetrică $5 \times 5$, A, cu elementul $(i,j)$ egal cu $\frac{i}{j}$ pentru $i \geq j$:

```
n = 5; A = eye(n);
for j=2:n
   for i = 1:j-1
      A(i,j)=i/j;
      A(j,i)=i/j;
   end
end
```

Expresia din ciclul `for` poate fi o matrice, în care caz lui *variabilă* i se atribuie succesiv coloanele lui *expresie*, de la prima la ultima. De exemplu, pentru a atribui lui `x` fiecare vector al bazei canonice, putem scrie `for x=eye(n), ..., end`.

Ciclul `while` are forma

`while` *expresie*

*instrucţiuni*

`end`

Secvenţa *instrucţiuni* se execută atât timp cât *expresie* este adevărată. Exemplul următor aproximează cel mai mic număr nenul în virgulă flotantă:

```
x = 1;
while x>0
    xmin = x;
    x = x/2;
end
xmin
```

```
xmin = 4.9407e-324
```

Execuţia unui ciclu `while` sau `for` poate fi terminată cu o instrucţiune `break`, care dă controlul primei instrucţiuni de după `end`-ul corespunzător. Construcţia `while 1, ..., end`, reprezintă un ciclu infinit, care este util atunci când nu este convenabil să se pună testul la începutul ciclului.

(De notat că, spre deosebire de alte limbaje, MATLAB nu are un ciclu „repeat-until”.) Putem rescrie exemplul precedent mai puţin concis prin

```
x = 1;
while 1
    xmin = x;
    x = x/2;
    if x == 0, break, end
end
xmin
```

```
xmin = 4.9407e-324
```

Într-un ciclu imbricat un `break` iese în ciclul de pe nivelul anterior.

Instrucţiunea `continue` cauzează trecerea controlului la execuţia unui ciclu `for` sau `while` următoarei iteraţii, sărind instrucţiunile rămase din ciclu. Un exemplu trivial este:

```
for i=1:10
   if i < 5, continue, end
   disp(i)
end
```

```
     5
     6
     7
     8
     9
    10
```

care afişează întregii de la 5 la 10.

Structura de control cu care încheiem este instrucţiunea `switch`. Ea constă din „`switch` *expresie*” urmată de o listă de instrucţiuni „case *expresie instrucţiuni*”, terminată opţional cu „`otherwise` *instrucţiuni*” şi urmată de `end`. Exemplul următor evaluează $p$-norma unui vector `x` pentru trei valori ale lui $p$:

```
p=2; x=[1;2];
switch(p)
   case 1
       y = sum(abs(x));
   case 2
       y = sqrt(x'*x);
   case inf
       y = max(abs(x));
   otherwise
       error('p poate fi 1, 2 sau inf.')
end
y
```

```
y = 2.2361
```

Funcţia `error` generează un mesaj de eroare şi opreşte execuţia curentă. Expresia ce urmează după `case` poate fi o listă de valori delimitate de acolade. Expresia din `switch` poate coincide cu orice valoare din listă:

```
x = input('Enter a real number: ')
switch x
   case {inf, -inf}
      disp('Plus or minus infinity')
   case 0
      disp('Zero')
   otherwise
      disp('Nonzero and finite')
end
```

Construcţia `switch` din MATLAB se comportă diferit de cea din C sau C++ : odată ce MATLAB a selectat un grup de expresii `case` şi instrucţiunile sale au fost executate, se dă controlul primei instrucţiuni de după `switch`, fără a fi nevoie de instrucţiuni `break`.

## Fişiere M

Fişierele M din MATLAB sunt echivalentele programelor, funcţiilor, subrutinelor şi procedurilor din alte limbaje de programare. Ele oferă următoarele avantaje:

- experimentarea algoritmului prin editare, în loc de a retipări o listă lungă de comenzi;
- înregistrarea permanentă a unui experiment;
- construirea de utilitare, care pot fi utilizate repetat;
- schimbul de fişiere M.

Multe fişiere M scrise de entuziaşti pot fi obţinute de pe Internet, pornind de la pagina de web http://www.mathworks.com. (MATLAB Central)

Un fişier M este un fişier text cu extensia (tipul) .m ce conţine comenzi MATLAB. Ele sunt de două tipuri:

**Fişiere M de tip script** (sau fişiere de comenzi) — nu au nici un argument de intrare sau ieşire şi operează asupra variabilelor din spaţiul de lucru.

**Fişiere M de tip funcţie** — conţin o linie de definiţie `function` şi pot accepta argumente de intrare şi returna argumente de ieşire, iar variabilele lor interne sunt locale funcţiei (înafară de cazul când sunt declarate global).

Un fişier script permite memorarea unei secvenţe de comenzi care sunt utilizate repetat sau vor fi necesare ulterior.

Script-ul de mai jos utilizează numerele aleatoare pentru a simula un joc. Să considerăm 13 cărţi de pică care sunt bine amestecate. Probabilitatea de a alege o carte particulară din pachet este 1/13. Acţiunea de extragere a unei cărţi se implementează prin generarea unui număr aleator. Jocul continuă prin punerea cărţii înapoi în pachet şi reamestecare până când utilizatorul apasă o tastă diferită de r sau s-a atins numărul de repetări (20).

```
rng(sum(100*clock));
for k=1:20
    n=ceil(13*rand);
    fprintf('Cartea extrasa: %3.0f\n',n)
    disp('   ')
    disp('Apasati r si Return pentru a continua')
    r=input('sau orice litera pentru a termina: ','s');
    if r~='r', break, end
end
```

Prima linie resetează de fiecare dată generatorul la o stare diferită.

Primele două linii ale acestui fişier script încep cu simbolul % şi deci sunt linii de comentariu. Ori de câte ori MATLAB întâlneşte un % va ignora restul liniei. Aceasta ne permite să inserăm texte explicative care vor face fişierele M mai uşor de înţeles. Începând cu versiunea 7 se admit blocuri de comentarii, adică comentarii care să se întindă pe mai multe linii. Ele sunt delimitate prin operatorii `%{` şi `%}`. Ei trebuie să fie singuri pe linie, ca în exemplul:

```
%{
Comentariu bloc
pe doua linii
%}
```

Dacă script-ul de mai sus este memorat în fişierul `joccarti.m`, tastând `joccarti` se executa script-ul.

Fişierele M de tip funcţie permit extinderea limbajului MATLAB prin scrierea de funcţii proprii care acceptă şi returnează argumente. Ele se pot utiliza în acelaşi mod ca funcţiile MATLAB existente, cum ar fi `sin`, `eye`, `size`, etc.

```
function [med,abmp] = stat(x)
%STAT Media si abaterea medie patratica a unei selectii
% [MED,ABMP] = STAT(X) calculeaza media si abaterea
% medie patratica a selectiei X
n = length(x);
med = sum(x)/n;
abmp = sqrt(sum((x-med).^2)/n);
```

Sursa MATLAB `stat` dă o funcţie simplă care calculează media şi abaterea medie pătratică a unei selecţii (vector). Acest exemplu ilustrează unele facilităţi ale funcţiilor. Prima linie începe cu cuvântul cheie `function` urmat de argumentele de ieşire, `[med,abmp]` şi de simbolul `=`. În dreapta `=`urmează numele funcţiei, `stat`, urmat de argumentele de intrare, în cazul nostru `x`, între paranteze. (În general, putem avea orice număr de argumente de intrare şi de ieşire.) Numele de funcţie trebuie să fie la fel ca al fişierului `.m` în care funcţia este memorată – în cazul nostru `stat.m`.

A doua linie a unui fişier funcţie se numeşte linie H1 sau help 1. Se recomandă ca ea să aibă următoarea formă: să înceapă cu un %, urmat fără nici un spaţiu de numele funcţiei cu litere mari, urmat de unul sau mai multe spaţii şi apoi o scurtă descriere. Descrierea va începe cu o literă mare, se va termina cu un punct, iar dacă este în engleză se vor omite cuvintele “the” şi “a”. Când se tastează `help nume_functie`, toate liniile, de la prima linie de comentariu pâna la prima linie care nu este de comentariu (de obicei o linie goală, pentru lizibilitatea codului sursă) sunt afişate pe ecran. Deci, aceste linii descriu funcţia şi argumentele sale. Se convine ca numele de funcţie şi de argumente să se scrie cu litere mari. Pentru exemplul `stat.m` avem

```
help stat
```

```
  stat Media si abaterea medie patratica a unei selectii
   [MED,ABMP] = stat(X) calculeaza media si abaterea
   medie patratica a selectiei X
```

Se recomandă documentarea *tuturor* funcţiilor utilizator în acest mod, oricât de scurte ar fi. Este util ca în liniile de comentariu din text sa apară data scrierii funcţiei şi datele când s-au făcut modificări. Comanda `help` lucrează similar şi pe fişiere script.

Funcţia `stat` se apelează la fel ca orice funcţie MATLAB:

```
[m,a]=stat(1:10)
```

```
m = 5.5000
a = 2.8723
```

```
x=randn(1,10);
[m,a]=stat(x)
```

```
m = 0.7049
a = 1.1565
```

O funcţie mai complicată este `sqrtn`, ilustrată mai jos. Dându-se $a > 0$, ea calculează $\sqrt{a}$ cu metoda lui Newton,

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right), x_1 = a.$$

afişând şi iteraţiile.

```
function [x,iter] = sqrtn(a,tol)
%SQRTN   Radical cu metoda lui Newton.
%        X = SQRTN(A,TOL) calculeaza radacina patrata a lui
%        A prin metoda lui Newton(sau a lui Heron).
%        presupunem ca A  >= 0.
%        TOL este toleranta (implicit EPS).
%        [X,ITER] = SQRTN(A,TOL) returneaza numarul de
%        iteratii ITER necesare.

if nargin < 2, tol = eps; end

x = a;
iter = 0;
xdiff = inf;
fprintf(' k             x_k            er. relativa\n')

for k=1:50
    iter = iter + 1;
    xold = x;
    x = (x + a/x)/2;
    xdiff = abs(x-xold)/abs(x);
    fprintf('%2.0f:  %20.16e  %9.2e\n', iter, x, xdiff)
    if xdiff <= tol, return, end
end
error('Nu s-a atins precizia dupa 50 de iteratii.')
```

Dăm exemple de utilizare:

```
format long
[x,it]=sqrtn(2)
```

```
 k             x_k            er. relativa
 1:  1.50000000000000000e+00   3.33e-01
 2:  1.41666666666666650e+00   5.88e-02
 3:  1.41421568627450970e+00   1.73e-03
 4:  1.41421356237468990e+00   1.50e-06
 5:  1.41421356237309490e+00   1.13e-12
 6:  1.41421356237309490e+00   0.00e+00
x =
   1.414213562373095

it =
     6
```

```
[x,it]=sqrtn(2,1e-4)
```

```
 k             x_k            er. relativa
 1:  1.5000000000000000e+00   3.33e-01
 2:  1.4166666666666665e+00   5.88e-02
 3:  1.4142156862745097e+00   1.73e-03
 4:  1.4142135623746899e+00   1.50e-06
x =
   1.414213562374690

it =
     4
```

Acest fişier M utilizează comanda `return`, care dă controlul apelantului. Spre deosebire de alte limbaje de programare, nu este necesar să se pună `return` la sfârşitul unei funcţii sau al unui script.

Funcţia `nargin` returnează numărul de argumente de intrare cu care funcţia a fost apelată şi permite atribuirea de valori implicite argumentelor nespecificate. Dacă apelul lui `sqrtn` nu a furnizat o precizie `tol`, se ia implicit valoarea `eps`. Un fişier M de tip funcţie poate conţine alte funcţii, numite subfuncţii, care pot să apară în orice ordine după funcţia principală (sau primară). Subfuncţiile sunt vizibile numai din funcţia principală sau din alte subfuncţii. Ele realizează calcule care trebuie separate de funcţia principală, dar nu sunt necesare în alte fişiere M, sau supraîncarcă funcţii cu acelaşi nume (subfuncţiile au prioritate mai mare). Help-ul pentru o subfuncţie se poate specifica punând numele funcţiei urmat de "/" şi numele subfuncţiei.

Pentru a crea şi edita fişiere M avem două posibilităţi. Putem utiliza orice editor pentru fişiere ASCII sau putem utiliza MATLAB Editor/Debugger. Sub Windows el se apelează prin comanda `edit` sau din opţiunile de meniu File-New sau File-Open. Sub Unix se apelează doar prin comanda `edit`. Editorul/Debugger-ul MATLAB are diverse faciltăţi care ajută utilizatorul, cum ar fi indentarea automată a ciclurilor şi structurilor de control, evidenţierea sintaxei prin culori, verificarea perechilor de paranteze şi apostrofuri.

Cele mai multe funcţii MATLAB sunt fişiere M păstrate pe disc, dar există şi funcţii predefinite conţinute în interpretorul MATLAB. Calea MATLAB (MATLAB path) este o listă de directori care specifică unde caută MATLAB fişierele M. Un fişier M este disponibil numai dacă este pe calea MATLAB. Drumul poate fi setat şi modificat prin comenzile `path` şi `addpath`, sau prin utilitarul (fereastra) path Browser, care se apelează din opţiunea de meniu Set Path sau tastând `pathtool`. Un script (dar nu şi o funcţie) care nu este pe calea de căutare se poate executa cu `run` urmat de calea completă până la fişierul M. Un fişier M se poate afişa pe ecran cu comanda `type`.

Un aspect important al MATLAB este dualitatea comenzi-funcţii. Înafară de forma clasică, nume, urmat de argumente între paranteze, funcţiile pot fi apelate şi sub forma nume, urmat de argumente separate prin spaţii. MATLAB presupune în al doilea caz că argumentele sunt şiruri de caractere. De exemplu apelurile `format long` şi `format('long')` sunt echivalente.

Începând cu versiunea 7, MATLAB permite definirea de funcţii imbricate, adică funcţii conţinute în corpul altor funcţii. În exemplul care urmează, funcţia `F2` este imbricată în funcţia `F1`:

```
function x = F1(p1,p2)

...
```

```
F2(p2)

 function y = F2(p3)

 ...

 end

...

end
```

Ca orice altă funcţie, o funcţie imbricată are propriul său spaţiu de lucru în care se memorează variabilele pe care le utilizează. Ea are de asemenea acces la spaţiul de lucru al tuturor funcţiilor în care este imbricată. Astfel, de exemplu, o variabilă care are o valoare atribuită ei de funcţia exterioară poate fi citită şi modificată de o funcţie imbricată la orice nivel în funcţia exterioară. Variabilele create într-o funcţie imbricată pot fi citite sau modificate în orice funcţie care conţine funcţia imbricată.

## Argumente de tip funcţie

În multe probleme, cum ar fi integrarea numerică, rezolvarea unor ecuaţii operatoriale, minimizarea unei funcţii, este nevoie ca o funcţie să fie transmisă ca argument unei alte funcţii. Aceasta se poate realiza în mai multe feluri, depinzând de modul în care funcţia apelată a fost scrisă. Vom ilustra aceasta cu funcţia `fplot`, care reprezintă grafic funcţia $f(x)$ peste domeniul implicit $[-5, 5]$. Un prim mod este transmiterea funcţiei printr-o construcţie numită *function handle*. Acesta este un tip de date MATLAB care conţine toate informaţiile necesare pentru a evalua o funcţie. Un function handle poate fi creat punând caracterul @ în faţa numelui de funcţie. Astfel, dacă `fun` este un fişier M de tip funcţie de forma cerută de f`plot`, atunci putem tasta

```
fplot(@fun)
```

`fun` poate fi numele unei funcţii predefinite:

```
fplot(@sin)
```

Function handle a fost introdus începând cu MATLAB 6 şi este de preferat utilizării şirurilor, fiind mai eficient şi mai versatil. Totuşi, ocazional se pot întâlni funcţii care să accepte argumente de tip funcţie sub formă de şir, dar nu sub formă de function handle. Conversia dintr-o formă în alta se poate face cu `func2str` şi `str2func` (vezi `help function_handle`).

Începând cu versiunea 7, MATLAB permite funcţii anonime. Ele pot fi definite în linii de comandă, fişiere M de tip funcţie sau script şi nu necesită un fişier M separat. Sintaxa pentru crearea unei funcţii anonime este

```
f = @(listaarg)expresie
```

Instrucţiunea de mai jos crează o funcţie anonimă care ridică argumentul ei la pătrat:

```
sqr = @(x) x.^2;
```

Dăm şi un exemplu de utilizare

```
sqr(5)
```

```
ans =
    25
```

Să considerăm funcţia `fd_deriv` din sursa de mai jos.. Această funcţie aproximează derivata funcţiei dată ca prim argument cu ajutorul diferenţei divizate

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

```
function y = fd_deriv(f,x,h)
%FD_DERIV  Aproximarea derivatei cu diferenta divizata.
%          FD_DERIV(F,X,H) este diferenta divizata a lui F cu nodurile X si
%          X+ H.  H implicit:  SQRT(EPS).

if nargin < 3, h = sqrt(eps); end
f=fcnchk(f);
y = (f(x+h) - f(x))/h;
```

Când se tastează

```
fd_deriv(@sqrt,0.1)
```

```
ans =
   1.581138771027327
```

primul apel la `f` din `fd_deriv` este equivalent cu `sqrt(x+h)`. Putem utiliza funcţia `sqrtn` în locul funcţiei predefinite `sqrt`:

```
fd_deriv(@sqrtn,0.1)
```

```
 k             x_k            er. relativa
 1:  5.5000000745058064e-01   8.18e-01
 2:  3.6590910694939033e-01   5.03e-01
 3:  3.1960053057932136e-01   1.45e-01
 4:  3.1624558582760998e-01   1.06e-02
 5:  3.1622779007837043e-01   5.63e-05
 6:  3.1622778957764164e-01   1.58e-09
 7:  3.1622778957764164e-01   0.00e+00
 k             x_k            er. relativa
 1:  5.5000000000000004e-01   8.18e-01
 2:  3.6590909090909096e-01   5.03e-01
 3:  3.1960050818746472e-01   1.45e-01
 4:  3.1624556228038903e-01   1.06e-02
 5:  3.1622776651756745e-01   5.63e-05
 6:  3.1622776601683794e-01   1.58e-09
 7:  3.1622776601683794e-01   0.00e+00
ans =
   1.581138771027327
```

Pentru a face `fd_deriv` să accepte expresii de tip şir am inserat

```
f=fcnchk(f);
```

la începutul funcţiei (în acest mod lucrează unele funcţii MATLAB, vezi (Moler, 2004) pentru exemple).

## Număr variabil de argumente

În anumite situaţii o funcţie trebuie să accepte sau să returneze un număr variabil, posibil nelimitat, de argumente. Aceasta se poate realiza utilizând funcţiile `varargin` şi `varargout`. Să presupunem că dorim să scriem o funcţie `companb` ce construieşte matricea companion pe blocuri, de dimensiune $mn \times mn$, a matricelelor $n \times n$ $A_1, A_2, \ldots, A_m$ :

$$C = \begin{bmatrix} -A_1 & -A_2 & \cdots & \cdots & -A_m \\ I & 0 & \cdots & \cdots & 0 \\ & I & \ddots & & \vdots \\ & & \ddots & 0 & 0 \\ & & & I & 0 \end{bmatrix}$$

Soluţia este de a utiliza `varargin` aşa cum se arată în sursa MATLAB care urmează

```
function C = companb(varargin)
%COMPANB   Matrice companion pe blocuri.
%          C = COMPANB(A_1,A_2,...,A_m) este matricea
%          companion pe blocuri corespunzatoare
%          matricelor n-pe-n A_1,A_2,...,A_m.

m = nargin;
n = length(varargin{1});
C = diag(ones(n*(m-1),1),-n);
for j = 1:m
    Aj = varargin{j};
    C(1:n,(j-1)*n+1:j*n) = -Aj;
end
```

Când `varargin` apare în lista de argumente, argumentele furnizate sunt copiate într-un tablou de celule numit `varargin`. Tablourile de celule (cell arrays) sunt structuri de date de tip tablou, în care fiecare element poate păstra date de tipuri şi dimensiuni diferite. Elementele unui tablou de celule pot fi selectate utilizând acolade. Considerăm apelul

```
X = ones(2); C = companb(X, 2*X, 3*X)
```

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | [1,1;1,1] | [2,2;2,2] | [3,3;3,3] |

```
C = 6x6
    -1    -1    -2    -2    -3    -3
    -1    -1    -2    -2    -3    -3
     1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     1     0     0     0
     0     0     0     1     0     0
```

Dacă inserăm o linie ce conţine doar `varargin` la începutul lui `companb` apelul de mai sus produce

```
varargin =

 [2x2 double] [2x2 double] [2x2 double]
```

Deci, `varargin` este un tablou de celule $1\times3$ ale cărui elemente sunt matrice $2\times2$ transmise lui `companb` ca argumente, iar `varargin{j}` este a $j$-a matrice de intrare. Nu este necesar ca `varargin` să fie singurul argument de intrare, dar dacă apare el trebuie să fie ultimul.

Analogul lui `varargin` pentru argumente de ieşire este `varargout`. În sursa MATLAB `momente` este dat un exemplu care calculează momentele unui vector, până la un ordin dorit. Numărul de argumente de ieşire se determină cu `nargout` şi apoi se crează tabloul de celule `varargout` ce conţine ieşirea dorită.

```
function varargout = momente(x)
%MOMENTE Momentele unui vector.
%        [m1,m2,...,m_k] = MOMENTE(X) returneaza momentele de
%        ordin 1, 2, ..., k ale vectorului  X, unde momentul
%        de ordin j este SUM(X.^j)/LENGTH(X).

for j=1:nargout, varargout{j} = {sum(x.^j)/length(x)}; end
```

Ilustrăm cu apelurile funcţiei `momente`

```
m1 = momente(1:4)
```

```
m1 =
    {[2.500000000000000]}
```

```
[m1,m2,m3] = momente(1:4)
```

```
m1 =
    {[2.500000000000000]}

m2 =
    {[7.500000000000000]}

m3 =
    {[25]}
```

## Variabile globale

Variabilele din interiorul unei funcţii sunt locale spaţiului de lucru al acelei funcţii. Uneori este convenabil să creăm variabile care există în mai multe spaţii de lucru, eventual chiar cel principal. Aceasta se poate realiza cu ajutorul instrucţiunii `global`.

Ca exemplu dăm codurile pentru funcţiile `tic` şi `toc` (cu unele comentarii prescurtate). Aceste funcţii pot contoriza timpul, gestionând un cronometru. Variabila globală TICTOC este vizibilă în ambele funcţii, dar este invizibilă în spaţiul de lucru de bază (nivel linie de comandă sau script) sau în orice altă funcţie care nu o declară cu `global`.

```
function tic
%    TIC Start a stopwatch timer.
%        TIC; any stuff; TOC
%    prints the time required.
%    See also: TOC, CLOCK.
global TICTOC
TICTOC = clock;
function t = toc
%    TOC Read the stopwatch timer.
%    TOC prints the elapsed time since TIC was used.
%    t = TOC; saves elapsed time in t, does not print.
%    See also: TIC, ETIME.
global TICTOC
if nargout < 1
    elapsed_time = etime(clock,TICTOC)
else
    t = etime(clock,TICTOC);
end
```

În interiorul unei funcţii, variabilele globale vor apare înaintea primei apariţii a unei variabile locale, ideal la începutul fişierului. Se convine ca numele de variabile globale să fie scrise cu litere mari, să fie lungi şi sugestive.

## Alte tipuri numerice

Tipul de date implicit în MATLAB este tipul de date `double`. Pe lângă acesta, MATLAB furmizează şi alte tipuri de date, având scopul în principal de a realiza economie de memorie. Acestea sunt

- `int8` şi `uint8` – întregi pe 8 biţi cu semn şi fără semn;
- `int16` şi `uint16` – întregi pe 16 biţi cu semn şi fără semn;
- `int32` şi `uint32` – întregi pe 32 biţi cu semn şi fără semn;
- `int64` şi `uint64` – întregi pe 64 biţi cu semn şi fără semn;
- `single` – numere în virgulă flotantă simplă precizie (pe 32 de biţi).

Funcţiile `eye`, `ones`, `zeros` pot returna date de ieşire de tipuri întregi sau `single`. De exemplu,

```
ones(2,2,'int8')
```

```
ans = 2x2 int8 matrix
   1   1
   1   1
```

returnează o matrice $2\times2$ cu elemente de tipul `int8`.

Funcţiile care definesc tipuri de date întregi au acelaşi nume ca şi tipul. De exemplu

```
x = int8(5)
```

```
x =
   5
```

atribuie lui `x` valoarea 5 reprezentată sub forma unui întreg pe 8 biţi. Funcţia `class` permite verificarea tipului unui rezultat.

```
class(x)
```

```
ans = 'int8'
```

Conversia unui număr de tip `double` la un tip întreg se face prin rotunjire la cel mai apropiat întreg:

```
int8(2.7)
```

```
ans =
   3
```

```
int8(2.5)
```

```
ans =
   3
```

Funcţiile `intmax` şi `intmin`, având ca argument un nume de tip întreg, returnează cea mai mare şi respectiv cea mai mică valoare de acel tip:

```
intmax('int16')
```

```
ans =
   32767
```

```
intmin('int16')
```

```
ans =
   -32768
```

Dacă se încearcă conversia unui număr mai mare decât valoarea maximă a unui întreg de un anumit tip la acel tip, MATLAB returnează valoarea maximă (saturation on overflow).

```
int8(300)
```

```
ans =
   127
```

Analog, pentru valori mai mici decât valoarea minimă, se returnează valoarea minimă de acel tip.

Dacă se realizează operaţii aritmetice între întregi de acelaşi tip rezultatul este un întreg de acel tip. De exemplu

```
x=int16(5)+int16(9)
```

```
x =
   14
```

```
class(x)
```

```
ans = 'int16'
```

Dacă rezultatul este mai mare decât valoarea maximă de acel tip, MATLAB returnează valoarea maximă de acel tip. Analog, pentru un rezultat mai mic decât valoarea minimă, se returnează valoarea minimă de acel tip. În ambele situaţii se dă un mesaj de avertisment care se poate inhiba (sau reactiva) cu funcţia `intwarning`.

Dacă `A` şi `B` sunt tablouri de tip întreg, împărţirile în sens tablou, `A./B` şi `A.\B`, se realizează în aritmetica în dublă precizie, iar rezultatul se converteşte la tipul întreg original, ca în exemplul

```
int8(4)./int8(3)
```

```
ans =
   1
```

Se pot combina în expresii scalari de tip double cu scalari sau tablouri de tip întreg, rezultatul fiind de tip întreg:

```
class(5*int8(3))
```

```
ans = 'int8'
```

Nu se pot combina scalari întregi sau tablouri de tip întreg cu scalari sau tablouri de un tip întreg diferit sau de tip `single`.Pentru toate operaţiile binare în care un operand este un tablou de tip întreg iar celălat este un scalar de tip double, MATLAB realizează operaţia element cu element în dublă precizie şi converteşte rezultatul în tipul întreg originar. De exemplu,

```
int8([1,2,3,4,5])*0.8
```

```
ans = 1x5 int8 row vector
   1   2   2   3   4
```

De notat că: MATLAB calculează `[1,2,3,4,5]*0.8` în dublă precizie şi apoi converteşte rezultatul în `int8`; al doilea şi al treilea element din tablou după înmulţirea în dublă precizie sunt `1.6` şi `2.4`, care sunt routunjite la cel mai apropiat întreg, 2.

## Bibliografie


Higham, N. H. (2009). *MATLAB Guide.* Philadelphia: SIAM.

Moler, C. (2004). *Numerical Computing in MATLAB.*Philadelphia: SIAM