

Cuprins

Programare în MATLAB.....	1
1. Fluxul de control.....	1
2. Fișiere M.....	9
3. Argumente funcție.....	19
4. Număr variabil de argumente	24
5. Variabile globale	27
6. Recursivitate.....	29
7. Alte tipuri numerice.....	34
Bibliografie.....	39

Programare în MATLAB

1. Fluxul de control

MATLAB are patru structuri de control: instrucțiunea `if`, instrucțiunea de ciclare `for`, instrucțiunea de ciclare `while` și instrucțiunea `switch`. Cea mai simplă formă a instrucțiunii `if` este

```
if expresie
    instrucțiuni
end
```

unde secvența *instrucțiuni* este executată dacă părțile reale ale elementelor lui *expresie* sunt toate nenule. Secvența de mai jos interschimbă x și y dacă x este mai mare decât y :

```
if x > y
    temp = y;
```

```
y = x;  
x = temp;  
end
```

Atunci când o instrucțiune `if` este urmată în aceeași linie de alte instrucțiuni, este nevoie de o virgulă pentru a separa `if`-ul de instrucțiunea 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
```

În fine, se pot introduce teste suplimentare cu `elseif` (de notat că nu este nici un spațiu între `else` și `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
```

Într-un test `if` de forma „*if condiție1 & condiție2*”, *condiție2* nu este evaluată dacă *condiție1* este falsă (așa-numită evaluare prin scurtcircuit). Acest lucru este util când evaluarea lui *condiție2* ar putea da o eroare—probabil din cauza unei variabile nedefinite sau a unei depășiri de indice.

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

```
for variabilă = expresie  
    instrucțiuni  
end
```

De obicei, *expresie* este un vector de forma `i : s : j`. Instrucțiunile 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 =
```

3.8160

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

```
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 creșterea lizibilității. Editorul-debugger-ul MATLAB poate realiza indentarea automată. Codul următor construiește o matrice simetrică 5 pe 5, A, cu elementul (i,j) egal cu i/j pentru $j \geq i$:

```
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
    instrucțiuni
end
```

Secvența *instrucțiuni* 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
```

Execuția unui ciclu `while` sau `for` poate fi terminată cu o instrucțiune `break`, care dă controlul primei instrucțiuni de după `end`-ul corespunzător. Construcția `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 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.

Instrucțiunea `continue` cauzează trecerea controlului la execuția unui ciclu `for` sau `while` următoarei iterații, sărind instrucțiunile 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 afișează întregii de la 5 la 10.

Structura de control cu care încheiem este instrucțiunea `switch`. Ea constă din „`switch expresie`” urmată de o listă de instrucțiuni „`case expresie instrucțiuni`”, terminată opțional cu „`otherwise instrucțiuni`” și 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
```

Funcția `error` generează un mesaj de eroare și oprește execuția 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
```

```
Enter a real number: x =
    7
```


Nonzero and finite

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

2. Fișiere M

Fișierele M din MATLAB sunt echivalentele programelor, funcțiilor, subrutinelor și 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 fișiere M.

Multe fișiere M scrise de entuziaști pot fi obținute de pe Internet, pornind de la pagina de web <http://www.mathworks.com>. Există și grupul de știri `comp.soft-sys.matlab`, dedicat MATLAB. (Grupurile de știri pot fi citite în mai multe moduri, inclusiv printr-un web browser.) Se pot obține detalii tastând `info` la promptul MATLAB.

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

Fișiere M de tip script (sau fișiere de comenzi) — nu au nici un argument de intrare sau ieșire și operează asupra variabilelor din spațiul de lucru.

Fișiere M de tip funcție — conțin o linie de definiție `function` și pot accepta argumente de intrare și returna argumente de ieșire, iar variabilele lor interne sunt locale funcției (înafară de cazul când sunt declarate global).

Un fișier script permite memorarea unei secvențe 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 cărți de pică care sunt bine amestecate. Probabilitatea de a alege o carte particulară din pachet este 1/13. Acțiunea de extragere a unei cărți se implementează prin generarea unui număr aleator. Jocul continuă prin punerea cărții înapoi în pachet și reamestecare până când utilizatorul apasă o tastă diferită de r sau s-a atins numărul de repetări (20).

```
rand('state',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
```

Linia

```
rand('state',sum(100*clock));
```

resetează de fiecare dată generatorul la o stare diferită.

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

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

Dacă script-ul de mai sus este memorat în fișierul `joccarti.m`, tastând `joccarti` se obține:

joccarti

Cartea extrasa: 11

Apasati r si Return pentru a continua
sau orice litera pentru a termina: Cartea extrasa: 11

Apasati r si Return pentru a continua
sau orice litera pentru a termina: Cartea extrasa: 4

Apasati r si Return pentru a continua
sau orice litera pentru a termina:

Fișierele M de tip funcție permit extinderea limbajului MATLAB prin scrierea de funcții proprii care acceptă și returnează argumente. Ele se pot utiliza în același mod ca funcțiile 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 funcție simplă care calculează media și abaterea medie pătratică a unei selecții (vector). Acest exemplu ilustrează unele facilități ale funcțiilor. Prima linie începe cu cuvântul cheie `function` urmat de argumentele de ieșire, `[med, abmp]` și de simbolul `=`. În dreapta = urmează numele funcției, `stat`, urmat de argumentele de intrare, în cazul nostru `x`, între paranteze. (În general, putem avea orice număr de argumente de intrare și de ieșire.) Numele de

funcție trebuie să fie la fel ca al fișierului `.m` în care funcția este memorată – în cazul nostru `stat.m`.

A doua linie a unui fișier funcție se numește linie H1 sau help 1. Se recomandă ca ea să aibă următoarea formă: să înceapă cu un `%`, urmat fără nici un spațiu de numele funcției cu litere mari, urmat de unul sau mai multe spații și 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” și “a”. Când se tastează `help nume_funcție`, toate liniile, de la prima linie de comentariu până la prima linie care nu este de comentariu (de obicei o linie goală, pentru lizibilitatea codului sursă) sunt afișate pe ecran. Deci, aceste linii descriu funcția și argumentele sale. Se convine ca numele de funcție și 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* funcțiilor utilizator în acest mod, oricât de scurte ar fi. Este util ca în liniile de comentariu din text să apară data scrierii funcției și datele când s-au făcut modificări. Comanda `help` lucrează similar și pe fișiere script.

Funcția `stat` se apelează la fel ca orice funcție MATLAB:

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

`m =`

```

5.5000
a =
2.8723
x=randn(1,10);
[m,a]=stat(x)

m =
0.0013
a =
0.8571

```

O funcție mai complicată este `sqrtn`, ilustrată în sursa 1. Dându-se $a > 0$, ea calculează \sqrt{a} cu metoda lui Newton,

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

afișând și iterațiile.

```

function [x,iter] = sqrtn(a,tol)
%SQRN    Radical cu metoda lui Newton.
%        X = SQRN(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] = SQRN(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:

[x,it]=sqrtn(2)

k	x_k	er. relativa
1:	1.5000000000000000e+000	3.33e-001
2:	1.4166666666666665e+000	5.88e-002

```

3:  1.4142156862745097e+000  1.73e-003
4:  1.4142135623746899e+000  1.50e-006
5:  1.4142135623730949e+000  1.13e-012
6:  1.4142135623730949e+000  0.00e+000
x =
    1.4142
it =
     6
x=sqrtn(2,1e-4)

k          x_k          er. relativa
1:  1.5000000000000000e+000  3.33e-001
2:  1.4166666666666665e+000  5.88e-002
3:  1.4142156862745097e+000  1.73e-003
4:  1.4142135623746899e+000  1.50e-006
x =
    1.4142

```

Acest fișier M utilizează comanda `return`, care dă controlul apelantului. Spre deosebire de alte limbaje de programare, nu este necesar să se pună `return` la sfârșitul unei funcții sau al unui script. Funcția `nargin` returnează numărul de argumente de intrare cu care funcția a fost apelată și permite atribuirea de valori implicite argumentelor nespecificate. Dacă apelul lui `sqrtn` nu a furnizat o

valoare pentru `tol`, atunci `tol` primește valoarea `eps`. Numărul de argumente la ieșire este returnat de funcția `nargout`.

Un fișier M de tip funcție poate conține alte funcții, numite subfuncții, care pot să apară în orice ordine după funcția principală (sau primară). Subfuncțiile sunt vizibile numai din funcția principală sau din alte subfuncții. Ele realizează calcule care trebuie separate de funcția principală, dar nu sunt necesare în alte fișiere M, sau supraîncarcă funcții cu același nume (subfuncțiile au prioritate mai mare). Help-ul pentru o subfuncție se poate specifica punând numele funcției urmat de “/” și numele subfuncției.

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

Cele mai multe funcții MATLAB sunt fișiere M păstrate pe disc, dar există și funcții predefinite conținute în interpretorul MATLAB. Calea MATLAB (MATLAB path) este o listă de directori care specifică unde caută MATLAB fișierele M. Un fișier M este disponibil numai dacă este pe calea MATLAB. Drumul poate fi setat și modificat prin comenzile `path` și `addpath`, sau prin utilitarul (fereastra) path Browser, care se apelează din opțiunea de meniu File-Set Path sau tastând `pathtool`. Un script (dar nu și o funcție) care nu este pe calea de căutare se poate executa cu `run` urmat de calea completă până la fișierul M. Un fișier M se poate afișa pe ecran cu comanda `type`.

Un aspect important al MATLAB este dualitatea comenzi-funcții. În afară de forma clasică, nume, urmat de argumente între paranteze, funcțiile pot fi apelate și sub forma nume, urmat de argumente separate prin spații. MATLAB presupune în al doilea caz că argumentele sunt șiruri de caractere. De exemplu apelurile `format long` și `format('long')` sunt echivalente.

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

```
function x = F1(p1,p2)
...
F2(p2)
    function y = F2(p3)
        ...
    end
...
end
```

Ca orice altă funcție, o funcție imbricată are propriul său spațiu de lucru în care se memorează variabilele pe care le utilizează. Ea are de asemenea acces la spațiul de lucru al tuturor funcțiilor în care este imbricată. Astfel, de exemplu, o variabilă care are o valoare atribuită ei de funcția exterioară poate fi citită și modificată de o funcție imbricată la orice nivel în funcția exterioară. Variabilele create într-o funcție imbricată pot fi citite sau modificate în orice funcție care conține funcția imbricată.

3. Argumente funcție

În multe probleme, cum ar fi integrarea numerică, rezolvarea unor ecuații operatoriale, minimizarea unei funcții, este nevoie ca o funcție să fie transmisă ca argument unei alte funcții. Aceasta se poate realiza în mai multe feluri, depinzând de modul în care funcția apelată a fost scrisă. Vom ilustra aceasta cu funcția `ezplot`, care reprezintă grafic funcția $f(x)$ peste domeniul implicit $[-2\pi, 2\pi]$. Un prim mod este transmiterea funcției printr-o construcție numită *function handle*. Acesta este un tip de date MATLAB care conține toate informațiile necesare pentru a evalua o funcție. Un function handle poate fi creat punând caracterul `@` în fața numelui de funcție. Astfel, dacă `fun` este un fișier M de tip funcție de forma cerută de `ezplot`, atunci putem tasta

```
ezplot(@fun)
```

`fun` poate fi numele unei funcții predefinite:

```
ezplot(@sin)
```

Numele unei funcții poate fi transmis ca un șir de caractere:

```
ezplot('exp')
```

Function handle a fost introdus începând cu MATLAB 6 și este de preferat utilizării șirurilor, fiind mai eficient și mai versatil. Totuși, ocazional se pot întâlni funcții care să accepte argumente de tip funcție sub formă de șir, dar nu sub formă de function handle. Conversia dintr-o formă în alta se poate face cu `func2str` și `str2func` (vezi `help function_handle`). Mai există două moduri de a transmite o funcție lui `ezplot`: ca expresie între apostrofuri,

```
ezplot('x^2-1'), ezplot('1/(1+x^2)')
```

sau ca obiect inline

```
ezplot(inline('exp(x)-1'))
```

Un obiect inline este o funcție definită printr-un șir și care poate fi atribuită unei variabile și apoi evaluată:

```
>> f=inline('exp(x)-1'), f(2)
f =
    Inline function:
    f(x) = exp(x)-1
ans =
    6.3891
```

MATLAB determină și ordonează argumentele unei funcții inline. Dacă acest lucru nu este satisfăcător, argumentele se pot defini și ordona explicit, transmițând lui `inline` parametrii suplimentari:

```
>> f = inline('log(a*x)/(1+y^2)')
f =
    Inline function:
    f(a,x,y) = log(a*x)/(1+y^2)
>> f = inline('log(a*x)/(1+y^2)', 'x', 'y', 'a')
f =
    Inline function:
    f(x,y,a) = log(a*x)/(1+y^2)
```

```
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 = (feval(f,x+h) - feval(f,x))/h;
```

Începând cu versiunea 7, MATLAB permite funcții anonime. Ele pot fi definite în linii de comandă, fișiere M de tip funcție sau script și nu necesită un fișier M separat. Sintaxa pentru crearea unei funcții anonime este

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

Instrucțiunea de mai jos crează o funcție anonimă care ridică argumentul ei la pătrat:

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

Dăm și un exemplu de utilizare

```
a = sqrt(5)
a =
    2.2361
```

Pentru a evalua în corpul unei funcții o funcție transmisă ca parametru se utilizează funcția `feval`. Sintaxa ei este `feval(fun, x1, x2, ..., xn)`, unde `fun` este funcția, iar `x1, x2, ..., xn` sunt argumentele sale. Să considerăm funcția `fd_deriv` din sursa `Error! Reference source not found.`. Această funcție aproximează derivata funcției dată ca prim argument cu ajutorul diferenței divizate

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

Când se tastează

```
>> fd_deriv(@sqrt, 0.1)
ans =
    1.5811
```

primul apel la `feval` din `fd_deriv` este equivalent cu `sqrt(x+h)`. Putem utiliza funcția `sqrtn` (sursa `MATLAB Error! Reference source not found.`) în locul funcției predefinite `sqrt`:

```
>> fd_deriv(@sqrtn, 0.1)
k          x_k          er. relativ:
5.5000000745058064e-001  8.18e-0012:  3.6590910694939033e-001
5.03e-001=  1.5811
```

Putem transmite lui `fd_deriv` un obiect inline, dar o expresie de tip șir nu funcționează:

```
>> f = inline('exp(-x)/(1+x^2)');  
>> fd_deriv(f,pi)  
ans =  
    -0.0063  
>> fd_deriv('exp(-x)/(1+x^2)',pi)  
??Error using==>feval Invalid function name 'exp(-x)/(1+x^2)'  
Error in ==> C:\radu\sane2\FD_DERIV.M  
On line 8 ==> y = (feval(f,x+h) - feval(f,x))/h;
```

Pentru a face `fd_deriv` să accepte expresii de tip șir vom insera

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

la începutul funcției (în acest mod lucrează `ezplot` și alte funcții MATLAB, vezi (Moler, 2004) pentru exemple).

Este uneori necesar să se „vectorizeze” un obiect inline sau o expresie de tip șir, adică să se convertească înmulțirile, ridicările la putere și împărțirile în operații în sens tablou, astfel ca să se poată utiliza argumente vectori și matrice. Acest lucru se poate realiza cu funcția `vectorize`:

```
>> f = inline('log(a*x)/(1+y^2)');  
>> f = vectorize(f)  
f =  
    Inline function:
```

$$f(a, x, y) = \log(a.*x) ./ (1+y.^2)$$

Dacă `fcnchk` se apelează cu un argument suplimentar `'vectorized'`, ca în `fcnchk(f, 'vectorized')`, atunci ea vectorizează șirul `f`.

MATLAB 7 a simplificat modul de apel al argumentelor de tip funcție. Se pot apela funcții referite cu function handle prin interfața de apel standard în loc de `feval`. Astfel, un function handle va fi tratat ca și un nume de funcție

```
fhandle(arg1, arg2, ..., argn)
```

Dacă funcția nu are nici un argument apelul ei are forma

```
fhandle()
```

Linia a doua a funcției `fd_deriv` (sursa `Error! Reference source not found.`) ar fi putut fi scrisă în MATLAB 7 sub forma

$$y = (f(x+h) - f(x)) / h;$$

4. Număr variabil de argumente

În anumite situații o funcție trebuie să accepte sau să returneze un număr variabil, posibil nelimitat, de argumente. Aceasta se poate realiza utilizând funcțiile `varargin` și `varargout`. Să presupunem că dorim să scriem o funcție `companb` ce construiește matricea companion pe blocuri, de dimensiune $mn \times mn$, a matricelor $n \times n$ A_1, A_2, \dots, A_m :

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

```
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
```

Soluția este de a utiliza `varargin` așa cum se arată în sursa MATLAB Error! Reference source not found.. 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 și dimensiuni diferite. Elementele unui tablou de celule pot fi selectate utilizând acolade. Considerăm apelul

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

```
C =  
    -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 conține 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 1×3 ale cărui elemente sunt matrice 2×2 transmise lui `companb` ca argumente, iar `varargin{j}` este a j -a matrice de intrare, A_j . 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 ieșire este `varargout`. În sursa MATLAB **Error! Reference source not found.** este dat un exemplu care calculează momentele unui vector, până la un ordin dorit. Numărul de argumente de ieșire se determină cu `nargout` și apoi se crează tabloul de celule `varargout` ce conține ieșirea dorită. Ilustrăm cu apelurile funcției `momente` din sursa **Error! Reference source not found.**:

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

```
m1 =  
    [2.5000]  
[m1,m2,m3] = momente(1:4)
```

```
m1 =  
    [2.5000]  
m2 =  
    [7.5000]  
m3 =  
    [25]
```

5. Variabile globale

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

```
function varargout = momente(x)
%MOMENTE Momentele unui vector.
%      [m1,m2,...,m_k] = MOMENTE(X) returns 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
```

Ca exemplu dăm codurile pentru funcțiile `tic` și `toc` (cu unele comentarii prescurtate). Aceste funcții pot contoriza timpul, gestionând un cronometru. Variabila globală `TICTOC` este vizibilă în ambele funcții, dar este invizibilă în spațiul de lucru de bază (nivel linie de comandă sau script) sau în orice altă funcție 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 nargin < 1  
    elapsed_time = etime(clock,TICTOC)  
else  
    t = etime(clock,TICTOC);  
end
```

În interiorul unei funcții, variabilele globale vor apărea înaintea primei apariții a unei variabile locale, ideal la începutul fișierului. Se convine ca numele de variabile globale să fie scrise cu litere mari, să fie lungi și sugestive.

6. Recursivitate

Funcțiile pot fi recursive, adică ele se pot autoapela, direct sau indirect. Recursivitatea este un instrument puternic, deși nu toate calculele descrise în manieră recursivă pot fi implementate eficient în mod recursiv.

```
function koch(pl,pr,level)
```

```
%KOCH    Curba Koch generata recursiv.
%         Apel KOCH(PL, PR, LEVEL) unde punctele PL si PR
%         sunt extremitatea stanga si dreapta
%         LEVEL este nivelul de recursivitate.

if level == 0
    plot([pl(1),pr(1)], [pl(2),pr(2)]); % Uneste pl si pr.
    hold on
else
    A = (sqrt(3)/6)*[0 1; -1 0];          % matrice rot./scal.
    pmidl = (2*pl + pr)/3;
    koch(pl,pmidl,level-1)                % ramura stanga

    ptop = (pl + pr)/2 + A*(pl-pr);
    koch(pmidl,ptop,level-1)              % ramura stanga-mijloc

    pmidr = (pl + 2*pr)/3;
    koch(ptop,pmidr,level-1)              % ramura dreapta-mijloc

    koch(pmidr,pr,level-1)                % ramura dreapta

end
```

Funcția `koch` utilizează recursivitatea pentru a desena o curbă Koch și este inspirată din (Higham, 2009). Construcția de bază este înlocuirea unui segment de dreaptă prin patru segmente mai scurte. Partea din stânga sus a figurii 1 arată rezultatul aplicării acestei construcții unei linii orizontale. Imaginea din dreapta jos ne arată ce se întâmplă când fiecare din aceste linii este prelucrată. Imaginile din stânga și dreapta jos ne arată următoarele două niveluri de recursivitate.

Funcția `koch` are trei argumente de intrare. Primele două, `p1` și `pr` dau coordonatele (x,y) ale capetelor segmentului curent și al treilea, `level`, indică nivelul de recursivitate cerut. Dacă `level = 0` se desenează un segment; altfel `koch` se autoapelează de patru ori cu `level` decrementat cu 1 și cu puncte care definesc capetele celor patru segmente mai scurte.

Figura 1 a fost obținută cu următorul cod:

```
p1=[0;0]; %left endpoint
pr=[1;0]; %right endpoint
for k = 1:4
    subplot(2,2,k)
    koch(p1,pr,k)
    axis('equal')
    title(['Koch curve: level = ',num2str(k)], 'FontSize',16)
end
hold off
```

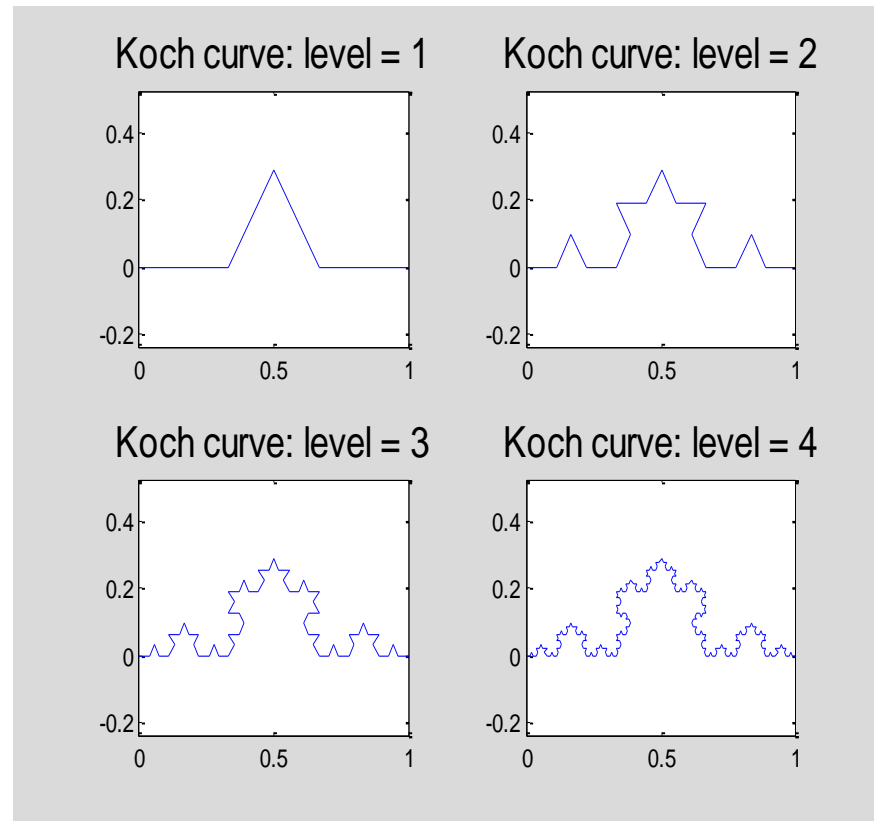


Figure 1: Curbe Koch create cu funcția `koch`.

Apelând `koch` cu perechi de puncte echidistante situate pe cercul unitate se obține o curbă numită fulgul de zăpadă al lui Koch (Koch's snowflake). Codul este dat în sursa `Error! Reference source not found.`. Funcția `snowflake` acceptă la intrare numărul de laturi `edges` și nivelul de recursivitate `level`. Valorile implicite ale acestor parametrii sunt 7 și respectiv 4.

```
function snowflake(edges,level)
if nargin<2, level=4; end
if nargin<1, edges=7; end

clf
for k = 1:edges
    pl = [cos(2*k*pi/edges); sin(2*k*pi/edges)];
    pr = [cos(2*(k+1)*pi/edges); sin(2*(k+1)*pi/edges)];
    koch(pl,pr,level);
end
axis('equal')
s=sprintf('Koch snowflake, level=%d, edges=%d',level, edges);
title(s, 'FontSize',16, 'FontAngle', 'italic')
hold off
```

În figura **Error! Reference source not found.** se dau două exemple, obținute cu
snowflake(7,4)
snowflake(5,3)

Pentru alte exemple de recursivitate, a se vedea funcțiile MATLAB quad și quadl, utilizate pentru integrarea numerică.

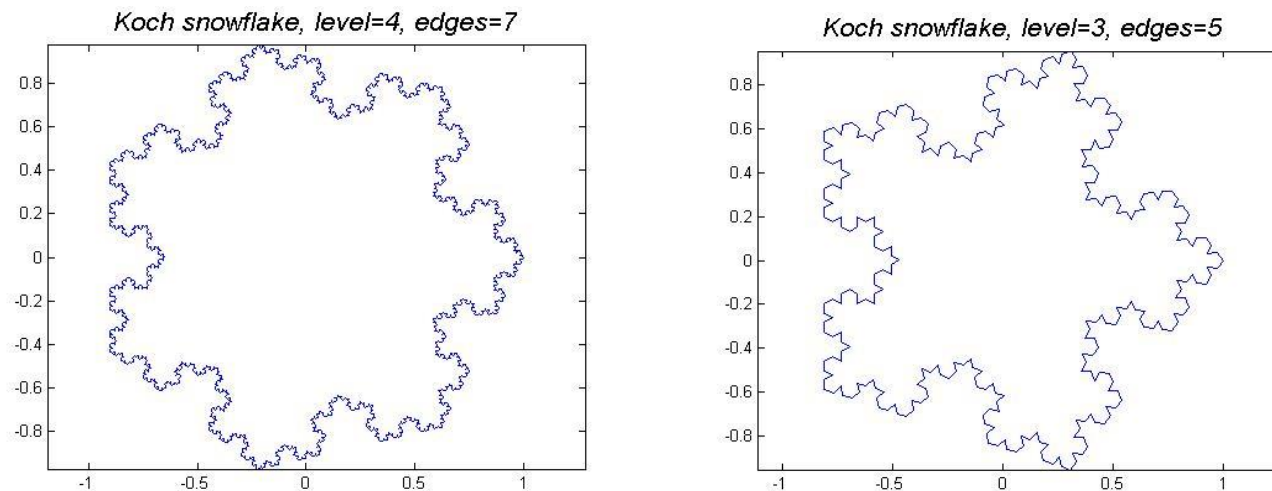


Figure 2: Fulgi Koch obținuți cu snowflakes

7. Alte tipuri numerice

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

- `int8` și `uint8` – întregi pe 8 biți cu semn și fără semn;
- `int16` și `uint16` – întregi pe 16 biți cu semn și fără semn;
- `int32` și `uint32` – întregi pe 32 biți cu semn și fără semn;

- `single` – numere în virgulă flotantă simplă precizie (pe 32 de biți).

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

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

returnează o matrice 2×2 cu elemente de tipul `int8`

```
ans =  
    1    1  
    1    1
```

Funcțiile care definesc tipuri de date întregi au același nume ca și tipul. De exemplu

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

atribuie lui `x` valoarea 5 reprezentată sub forma unui întreg pe 8 biți. Funcția `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
```

Funcțiile `intmax` și `intmin`, având ca argument un nume de tip întreg, returnează cea mai mare și 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 operații aritmetice între întregi de același 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 situații se dă un mesaj de avertisment care se poate inhiba (sau reactiva) cu funcția `intwarning`.

Dacă A și B sunt tablouri de tip întreg, împărțirile în sens tablou, `A./B` și `A.\B`, se realizează în aritmetica în dublă precizie, iar rezultatul se convertește 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 operațiile binare în care un operand este un tablou de tip întreg iar celălalt este un scalar de tip double, MATLAB realizează operația element cu element în dublă precizie și convertește rezultatul în tipul întreg originar. De exemplu,

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

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

Bibliografie

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

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