METODE NUMERICE: Laborator #1
Introducere în Matlab/Octave. Funcții și intructiuni. Fișiere M. Funcții de citire/scriere de tipul C

Titulari curs: *Florin Pop, George-Pantelimon Popescu*

Responsabili Laborator: **Bogdan Țigănoaia, Florin Pop**

Obiective Laborator

În urma parcurgerii acestui laborator, studentul va fi capabil să:

- utilizeze programul Matlab / Octave pentru a rezolva probleme de calcul numeric;
- folosească fișiere `.m`;
- folosească funcții Octave, funcții de citire / scriere de tipul C.

Introducere în Matlab/Octave

Modelarea problemelor matematice în sistemele informatice se poate face cu ajutorul *programelor specializate pentru calcule matematice* (Mathematica[1], MathCAD[2]) sau cu ajutorul *mediilor de programare* (MATLAB[3], Maple[4], Octave[5], Scilab[6]). Vă recomandăm și portalul Wolfram|Alpha[7].

MATLAB (MATrix LABoratory) este un pachet de programe de înaltă performanță, dedicat calculului numeric și reprezentărilor grafice în domeniul științei și ingineriei. Acesta integrează analiza numerică, calculul matriceal, procesarea semnalului și reprezentările grafice. Limbajul MATLAB a fost creat de profesorul Cleve B. Moler de la Universitatea din New Mexico pentru a permite un access ușor la bibliotecile de calcul matricial realizate în Fortran. Octave este implementarea opensource pentru o parte din bibliotecile de funcții MATLAB. Vom prezenta în continuare câteva elemente de bază, ca o introducere în Matlab/Octave. Se presupune că ați instalat deja Octave.

Elementul de bază cu care operează MATLAB-ul este **matricea**. În MATLAB nu se declară varibile (ca în C) și nici nu se folosesc tipuri de date predefinite, deoarece orice set de date manipulat de utilizator este văzut ca o matrice (o zonă continuă de memorie). Toate programele scrise în MATLAB sunt interpetate (nu compilate) și sunt executate linie cu linie. Octave pune la dispoziție o consolă pentru lansarea în execuție a comenzilor. Pentru editarea codului sursă aveți nevoie de un editor de text. Octave pune la dispoziție un astfel de editor prin comanda `edit`.

[1] `http://www.wolfram.com/mathematica/`
[2] `http://www.ptc.com/product/mathcad`
[3] `http://www.mathworks.com/products/matlab/`
[4] `http://www.maplesoft.com/products/maple/`
[5] `https://www.gnu.org/software/octave/`
[6] `http://www.scilab.org`
[7] `http://www.wolframalpha.com`

Deschideţi terminalul Octave dând dublu-clic pe iconiţa Octave sau scriind `octave` în consolă. Rețineți că fișierele externe ulterioare pe care le veți crea trebuie salvate în directorul curent de lucru (comanda `pwd`, similară cu cea din Unix). Semnul > reprezintă promptul Octave (poate fi precedat de diverse afișări, de exemplul `octave:x>`, cifra `x` indicând numărul comenzii din sesiunea curentă de lucru).

```
> pwd
/home/student/Desktop
```

În Octave se pot executa majoritatea comenzilor de consolă. Încercați `dir`, `ls -al`, etc.

Funncția `help` permite obținerea unor informații cu caracter general despre comenzile interne și externe MATLAB. Ea poate fi apelată în mai multe forme:

```
> help                  % oferă informații despre elementele limbajului MATLAB
                        % și a fișierelor .M din directorul curent.
> help <nume-functie>   % oferă informații despre funcția în cauză.
```

Alte funcții folosite pentru controlul general al mediului de lucru sunt:

```
> what     % listează fișierele .m, .mat și .mex din directorul de lucru
> type     % listează fișierul .m menționat
> lookfor  % returnează numele fișierelor care au în prima linie a help-ului
                (linia H1) cuvintele precizate ca argument
> which    % calea în care este localizat un fișier sau o funcție MATLAB
> path     % returnează căile cu are lucrează Octave/MATLAB
> who      % listează variabilele curente din memorie
> whos     % listează variabilele curente, dimensiunile și tipul acestora
> format   % setează formatul de afișare a datelor (short, long, short e,
                long e, hex, plus, bank, rat)
```

Pentru a crea o matrice de tip coloană, tipăriţi:

```
> v = [ 0; 1; 2 ]
```

Pentru a crea o matrice de tip linie, tipăriţi:

```
> l = [ 0  1+5i  2 ]
```

Observați modul de reprezentare al numerelor complexe. Aceasta este unica situație în care se poate eluda semnul de înmulțire *, subînțelegându-se coeficientul numărului imaginar i.

Pentru a crea o matrice de 2 linii și 3 coloane, tipăriţi:

```
> m = [ 0 1 2; 3 4 5 ]
```

Se observă deci că atunci când este întâlnit simbolul ";", programul consideră că începe o linie nouă în matrice. Cazul pentru matrice de `m` linii și `n` coloane se generalizează ușor.

Sunt cazuri când dorim să construim vectori cu multe elemente, cu termeni în progresie aritmetică. Introducerea lor manuală ar lua prea mult, așa că există următoarea comandă care simplifică lucrurile:

```
> v = [ initial : pas : final ]
```

Dați diverse valori pasului (chiar și pas negativ, sau care să nu fie divizor al lui final-initial), valorii inițiale și celei finale. Pentru `pas=1`, comanda poate fi dată doar ca `v = [ initial : final ]`, sau chiar `v = initial : final`. Urmăriți outputul. După ce ați construit diverse matrice de tip coloană, linie sau matrice generală, introduceți comenzile:

```
> length(v)
> size(m)
```

În timp ce `length` returnează lungimea unui vector sau a unei linii, `size` returnează un vector `[nr, nc]`, adică numarul de linii și coloane ale matricei `m`. Pentru mai multe detalii, tastati `help length`.

Ca în C/C++, pare natural să putem accesa un element al unui vector. Aceasta se face simplu prin `v(i)`, unde `i` este indicele. Analog pentru o matice `m`, avem `m(i,j)` pentru elementul de pe linia `i`, coloana `j`.

ATENTIE! Indicierea începe de la 1, și nu de la 0, ca în C/C++!!!

Pentru a extrage o submatrice dintr-o matrice `m`, extragând doar liniile `l1, l2, l3...` și coloanele `c1, c2, c3...`, construim doi vectori `l` si `c` (fie linie, fie coloană, nu are relevanță) și tipărim `m(l, c)`:

```
> m = [0 1 2 ; 3 4 5; -3 -1 10 ]
> m([1 2 3], [2, 3])
> m([1:3], [2, 3])
> v = [1 2 3];
> m(v, [2 3])
```

Am arătat mai multe metode de extragere a unei submatrice. Pentru a nu avea output, folosiți `;` după comanda care doriți să nu genereze output.

Pentru transpunerea unei matrice, se folosește operatorul `'`. Spre exemplu, `m'` va returna transpusa (hermitica) a lui `m`. Operații aritmetice pe matrice:

```
> m + 3    % se adună 3 la toate componentele
> 3 * m    % se înmultesc toate componentele cu 3
> m * n    % se înmultesc două matrice, cu dimensiunile compatibile
> m + n    % se adună două matrice, ca mai sus
> a .* b   % noua matrice are componentele a(i,j)*b(i,j)
```

Operatorii cu `.` se numesc operatori Hadamard (notația vine de la produsul Hadamard, dar s-a extins și altori operatori). Acești operatori se aplică elementelor matricelor, element cu element. Priviți și următorul exemplu:

```
> a = [1 1; 2 3]
> a^2
ans =
    3    4
    8   11
> a.^2
ans =
   1   1
   4   9
```

Instrucțiuni și Funcții Octave. Fișiere M

Funcții și constante

Tipăriți următoarele comenzi și observați efectul:

```
> cos(pi/3)
> sin(pi/4)
> ans
> inf
> eps
> realmax
> realmin
```

Instrucțiuni

Instrucțiunea de decizie `if` are sintaxa generală:

```
if conditie
    ...
endif
```

Bucla `for` are sintaxa generală:

```
for variabila = vector
    ...
endfor
```

Exemplu de program ce calculează media elementelor unui vector. Programul va fi salvat în fișierul cu numele `medie.m` și se va lansa în execuție cu comanda `> medie` (numele sau, fără extensia `.m`).

```
x = [2 3 4 1 -20];
suma = 0;
for var = x
    suma = suma + var;
endfor
disp('Media este')
disp(suma / length(x))
```

Listing 1: Exemplu de program ce calculează media elementelor unui vector.

Bucla `while`. Sintaxa generală se poate observa în următorul exemplu (observați outputul):

```
x = 1.0;
while x < 1000
  x = x*2;
  disp(x);
endwhile
```

Listing 2: Exemplu "while".

Funcții în Octave

O funcție în Octave face același lucru ca în C/C++: primește parametri, execută instrucțiuni, returnează un rezultat. Fiecare funcție trebuie definită într-un fișier separat, iar numele funcției trebuie să coincidă cu numele fișierului (exceptând, bineînțeles, extensia, care este .m). Exemplu de funcție, care face suma a două numere / vectori / matrici :

```
function [ s ] = suma( a, b )
  s = a + b;
endfunction
```

Listing 3: Exemplu functie in Octave

Aceasta trebuie salvată într-un fișier cu numele `suma.m`. Un apel al funcției arată astfel: `> suma(3,2)` (testati si apelul `suma(2:5,3:6)`).

Putem avea și functii void, de tipul `function functie(parametri)`. Se pot returna și mai multi parametri (un vector de parametri), în felul urmator: `function [x y z] = functie(parametri)`.

Funcții de citire/scriere de tipul C

Deschiderea fisierelor

Înaintea citirii și scrierii dintr-un fișier text sau binar acesta trebuie deschis folosind comanda `fopen` cu una din formele:

```
fid = fopen('numefis','mod')
[fid, mesaj] = fopen('numefis', 'mod')
```

Modul (sau permisiunea) poate fi una din alternativele:

```
r, w, a    % numai pentru citire, scriere sau adăugare
r+         % atât pentru citire cât și pentru scriere
```

Dacă operația de deschidere reușește, `fopen` întoarce un întreg nenegativ, numit identificator de fișier (`fid`). Valoarea aceasta este transmisă ca argument altor funcții de I/E care accesează fișierul deschis. Dacă deschiderea fișierului eșuează, întrucât fișierul nu există, `fid` primește valoarea `-1`. Fișierele standard nu trebuie deschise. Fișierul standard de ieșire are identificatorul `fid=1`, iar fișierul standard de eroare `fid=2`.

Deschideți pentru citire un fișier, al cărui nume îl introduceți de la tastatură. Afișați mesajul care specifică dacă operația de deschidere a reușit sau nu.

```
fid = 0;
while fid < 1
    numefis = input('Deschide fisier: ', 's') ;
    [fid, mesaj] = fopen(numefis, 'r');
    if fid == -1
        disp(mesaj);
    end
end
```

Listing 4: Exemplu de program care deschide un fisier

Funcția `input` permite introducerea de date de la tastatură. Șirul de caractere dat ca prim parametru va fi afișat. Al doilea parametru `s` arată că datele introduse sunt caractere.

Funcția `disp` afișează un șir de caractere.

Scrierea datelor formatate în fișiere text

```
contor = fprintf(fid, format, A, ...)
```

Funcția întoarce numărul de octeți transferați. Descriptorii de format sunt aceiași din C. Există descriptorii specifici Matlab:

```
%bx  % afișare valoare double în hexazecimal
%tx  % afișare valoare float în hexazecimal
```

Descriptorii pot fi precedați de caracterele: '−','+',' ','0', cu semnificațiile:

```
'-'  % aliniere stânga
'+'  % afișează întotdeauna cu semn
' '  % inserează un spațiu înaintea valorii afișate
'0'  % pune zerouri în locul spațiilor
```

Citirea datelor formatate din fișiere text

```
A = fscanf(fid, format);
[A, contor] = fscanf(fid, format, dimensiune);
```

Prima formă citește până la sfârșitul fișierului. Cea de-a doua formă citește un număr specificat de dimensiune. Aceasta se specifică sub una din formele:

```
n       % cel mult n numere, caractere sau șiruri
inf     % până la sfârșitul fișierului
[m,n]   % cel mult m*n valori, care completează o matrice pe coloane
```

Funcția `fscanf` este vectorizată și întoarce un argument matrice. Funcția acceptă și valorile `-inf`, `+inf`, `NaN`, pe care le convertește în reprezentările numerice corespunzătoare.

Citirea unei linii din fișierul text până la sfârșitul de linie se face cu funcția `linie = fgetl(fid, LEN)`. Dacă se întâlnește `eof` funcția întoarce `-1`. Parametrul `LEN` indică câte carectere se vor citit. Dacă acesta este omis se va citi până la întâlnirea terminatorului de linie.

```
1 fid = fopen( 'printfile.m' ) ;
2 while 1
3     linie = fgetl(fid);
4     if ~ischar(linie), break, end
5     disp(linie);
6 end
7 fclose(fid);
```

Listing 5: Exemplu de citire a unui fișier text linie cu linie și afișarea lui pe ecran.

Controlul poziției în fișier

Funcția `fseek` ne permite să ne poziționăm oriunde în fișier:

```
stare = fseek(fid, deplasare, origine)
```

Originea ia una din valorile:

```
'bof'  % față de începutul fișierului
'cof'  % față de poziția curentă
'eof'  % față de sfârșitul fișierului
```

Deplasarea este o valoare pozitivă sau negativă exprimată în octeți și raportată la origine. Funcția `ftell` determină poziția curentă în fișier, față de începutul fișierului: `pozitie = ftell(fid)`.

Exportul și importul datelor

Pentru salvarea variabilelor curente cu care se lucrează la încheierea unei sesiuni de lucru se poate utiliza comanda `save file`. Această comandă va salva toate variabilele curente generate de către utilizator într-un fișier dat ca parametru prin `file`. De exemplu:

```
> save date A B x y
```

realizează salvarea variabilelor `A, B, x, y` în fișierul `date.mat`.

Pentru restituirea variabilelor dintr-un fișier `.mat` se folosește comanda `load`.

Vectorizări

Operaţiile cu vectori şi matrice sunt executate în MATLAB mult mai repede decât operaţiile de interpretare a instrucţiunilor şi executare a lor. Obţinem astfel o îmbunătăţire a timpului de execuţie pentru programele scrise. *Vectorizarea* constă în transformarea ciclurilor for şi while, acolo unde este posibil, în operaţii pe vectori sau matrice. De exemplu, soluţia alternativă pentru exemplul:

```
for n = 1:10
    x(n) = sin(n*pi/5)
end
```

este o soluţie vectorizată, mult mai rapidă şi atribuie memorie pentru vectorul `x` o singură dată. Mai întâi se iniţializează vectorul, apoi se foloseşte funcţia `sin` care a fost implementată special pentru calcule vectorizate.

```
n = 1:10;
x = sin(n*pi/5);
```

Vom prezenta în continuare alte exemple de vectorizări în care am folosit foarte mult puterea operatorilor logici şi a funcţiilor MATLAB.

Exemplul 1

```
x=-2:0.5:2;
for i=1:length(x)
    if x(i)>=0
        s(i)=sqrt(x(i));
    else
        s(i)=0;
    end
end
```

Listing 6: Exemplu cu bucla for.

```
x=-2:0.5:2;
s = sqrt(x);
s(x<0) = 0;
```

Listing 7: Exemplu de cod vectorizat.

Aici ne-am bazat pe puterea funcţiei `sqrt` care funcţionează şi pe numere negative (având ca rezultat un număr complex). Am folosit operatorul < care pentru vectori are ca rezultat un alt vector cu unu pe poziţiile care satisfac condiţia şi am utilizat indexarea unui vector prin intermediul altui vector.

Exemplul 2

```
M = magic(3);
for i=1:3,
   for j=1:3,
       if (M(i,j) > 4),
           M(i,j) = -M(i,j);
       end
   end
end
```

Listing 8: Exemplu cu bucla for.

```
ind = find(M > 4);
M(ind)=-M(ind);
```

Listing 9: Exemplu de cod vectorizat.

În acest exemplu metoda care foloseşte bucla for se execută în 23.4956 unităţi de timp iar metoda vectorizată prin intermediul funcţiei find se execută în 2.1153 unităţi de timp, deci de 11 ori mai rapid!

Exemplul 3

Un alt exemplu de vectorizare care foloseşte funcţii tipice operaţiilor cu vectori precum `filter` şi `find` (`findstr`).

```
V = 'Sunt    multe   spatii   albe   in acest   text.';
len = length(V);
i = 1;
while (i<len)
    if (V(i) == ' ' & V(i+1) == ' ')
        for j = i:len-1
            V(j) = V(j+1);
        end
        V(len)=0;
        len = len-1;
    else
        i=i+1;
    end
end
V = char(V)
```

Listing 10: Exemplu cu bucla for.

Secvenţa vectorizată este:

```
V = 'Sunt    multe   spatii   albe   in acest   text.';
ind = find(filter([1 1], 2, V==' ') == 1);
V(ind)= []
```

Listing 11: Exemplu de cod vectorizat.

sau, o a doua metodă:

```
V = 'Sunt    multe   spatii   albe   in acest   text.';
ind = findstr(V,' ');
V(ind) = []
```

Listing 12: Exemplu de cod vectorizat.

Exemplul 4

Folosirea operatorilor Hadamard poate conduce la eliminarea buclelor şi vectorizarea unei secvenţe de program. De exemplu secvenţa:

```
for i=1:n
    for j=1:n
        M(i,j) = A(i,j)/(B(i,j)*C(i,j));
    end
end
```

se poate transforma în urma folosirii operatorilor ./ şi .* în:

```
M = A./(B.*C);
```

Este recomandat ca ori de câte ori folosiţi o funcţie MATLAB în interiorul unei bucle să verificaţi (consultând help-ul MATLAB-ului) dacă aceasta poate fi de folos la vectorizarea calculului. De asemenea ţineţi cont de faptul ca operaţiile logice din instrucţiunile de ramificare pot ajuta la vectorizare.

Aplicaţii

1. Testați programele și funcțiile prezentate în laborator.
2. Creați fișierul `valori.txt` cu valorile funcției $f(x) = 2x + 1$ pe intervalul `[0, 1]` cu pasul `0.1`.
3. Construiți o funcție care să calculeze suma numerelor pare mai mici ca `n` utilizând bucla `for`. Faceți același lucru utilizând bucla `while`. Citirea unui numar de la tastatura se face utilizand comanda `var = input('Introduceti variabila: ')`.
4. Definiți o funcție, fișier `.m` pentru determinarea dimensiunii unui fișier exprimată în octeți. Funcția are ca parametru numele fișierului.
5. Definiți o funcție care citește un fișier text, linie cu linie și întoarce numărul total de apariții ale unui anumit șir de caractere în fișier. Funcția va afișa fiecare linie din fișier, precedată de numărul de apariții ale șirului în linie. Funcția are semnătura: `function y = contor(numefis, sir)`.
6. Scrieți o funcție care citește o matrice pătrată din fișier și verifică dacă matricea are proprietățile unui pătrat magic (suma elementelor pe linii, coloane și diagonale este aceeași).