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

Noțiuni teoretice

Elemente Introductive

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 WolframAlpha[7].

MATLAB (MATrix LABoratory) este un set de pachete 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 acces ușor la bibliotecile de calcul matriceal realizate în Fortran. Octave este implementarea open source pentru o parte din bibliotecile de funcții MATLAB. Vom prezenta în continuare câteva elemente de bază, ca o introducere în MATLAB/Octave.

Elementul de bază cu care MATLAB/Octave operează este matricea. În MATLAB/Octave nu se declară variabile (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/Octave sunt interpretate (nu compilate) și sunt executate linie cu linie.

Se presupune că ați instalat deja Octave. Deschideţi terminalul Octave dând dublu-click pe iconiţa Octave sau scriind `octave` în consolă. Acest mediu de programare pune la dispoziția utilizatorului o consolă pentru lansarea în execuție a comenzilor. Pentru editarea codului sursă aveți nevoie de un editor text, obţinut folosind comanda `edit`.

Rețineți că fișierele 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
```

[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

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

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

```
> help   % oferă informaţii despre elementele limbajului
             % Matlab/Octave şi a fişierelor .m din
             % directorul curent;
> help <functie>   % oferă informaţii despre o funcţie.
```

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

```
> what     % listează fişiere de tipul .m, .mat etc. din
               % directorul de lucru;
> type     % listează fişierul .m menţionat;
> lookfor  % returnează numele fişierelor care au în prima
                % linie a help-ului (linia H1) cuvintele
                % precizate ca argument;
> which    % calea în care este localizat un fişier sau o
                % funcţie Octave;
> path     % returnează căile cu care lucrează Octave;
> who      % listează variabilele curente din memorie;
> whos     % listează variabilele curente, dimensiunile
               % şi tipul acestora;
> format  % setează formatul de afişare a datelor
               % (short, long, short e, long e, hex,
               % plus, bank, rat).
```

Pentru a construi o matrice de tip coloană, folosiţi comanda:

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

Pentru a construi o matrice de tip linie, folosiţi comanda:

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

Observați modul de reprezentare a numerelor complexe. Este unica situație în care se poate exclude semnul de înmulțire $*$, subînțelegându-se coeficientul numărului imaginar i.

Pentru a construi o matrice de 2 linii și 3 coloane, folosiţi comanda:

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

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

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

```
> v = [inițial : pas : final]
```

Dați diverse valori `pasului` (chiar și pas negativ), valorii inițiale și celei finale. Pentru `pas=1`, comanda poate fi dată doar ca `v = [initial : final]` sau chiar `v = initial : final`. Urmăriți rezultatul. După ce ați construit diverse matrici 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 `[nl, nc]`, adică numărul de linii și coloane ale matricei `m`. Pentru mai multe detalii, tastați `help length`.

La fel ca în C/C++, putem accesa un element al unui vector. În acest scop, folosim comanda `v(i)`, unde `i` este indicele elementului accesat. Analog pentru o matrice `m`, folosim `m(i,j)` pentru a accesa elementul de pe linia `i`, coloana `j`.

ATENTIE! Indicii încep de la 1, nu de la 0 ca în C/C++!!!

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

Folosiți operatorul ; dacă doriți ca o comandă să nu afişeze rezultatele.

Pentru transpunerea unei matrice, se folosește operatorul `'`. De exemplu, comanda `m'` va returna transpusa (hermitica) matricei `m`. Putem defini mai multe tipuri de operații aritmetice pe matrice, de exemplu:

```
> m + 3    % se adună 3 la toate componentele matricei m;
> 3 * m    % se înmulțesc toate componentele matricei m cu 3;
> m * n    % se înmulțesc două matrici, cu dimensiunile
               % compatibile;
> m + n    % se adună două matrici, cu dimensiunile
                % compatibile;
> 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 matricilor, element cu element. Considerăm următoarele exemple pentru operatorii Hadamard:

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

Instrucțiuni și funcții Octave.

Funcții și constante

Rulați următoarele comenzi și observați efectul lor:

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

Instrucțiuni

Instrucțiunea de decizie `if` are sintaxa generală:

```
if condiție
    ...
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 folosind comanda `> medie` (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:

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

Listing 2: Exemplu de utilizare a buclei while.

Funcții în Octave

O funcție în Octave are acelaşi comportament ca în C/C++: primește parametri, execută instrucțiuni şi întoarce un rezultat. Fiecare funcție trebuie definită într-un fișier separat, iar numele funcției trebuie să coincidă cu numele fișierului (exceptând extensia `.m`). Exemplu de funcție care calculează suma a două numere/vectori/matrice:

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

Listing 3: Exemplu de funcție în Octave.

Această funcţie trebuie salvată într-un fișier cu numele `suma.m`. Un exemplu de apel al funcției este: `> suma(3,2)` (testați și apelul `suma(2:5,3:6)`).

Putem avea și funcții void, de tipul `function funcție(parametri)`. Se pot returna și mai multe rezultate (un vector de rezultate), în felul următor:

```
function [x y z] = functie(parametri).
```

Funcții de citire/scriere de tipul C

Deschiderea fișierelor

Înaintea citirii și scrierii dintr-un fișier (text sau binar), acesta trebuie deschis folosind comanda `fopen` ce are 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, respectiv adăugare;
r+         % atât pentru citire cât și pentru scriere.
```

Dacă operația de deschidere fişier 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);
    endif
endwhile
```

Listing 4: Exemplu de program care deschide un fișier.

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 la consolă.

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/Octave:

```
%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 date până la sfârșitul fișierului. Cea de-a doua formă citește date de o dimensiune dată ca parametru de intrare. Pentru a specifica dimensiunea datelor se utilizează una din variantele:

```
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.

O linie din fișier se citeşte cu funcția `linie = fgetl(fid, LEN)`. Dacă se întâlnește `eof`, funcția `fgetl` întoarce `-1`. Parametrul `LEN` indică numărul de caractere de citit. Dacă acest parametru 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
```

```
    disp(linie);
endwhile
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)
```

Parametrul `origine` poate lua una din valorile:

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

Parametrul `deplasare` 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 memorarea variabilelor 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 memorarea variabilelor `A, B, x, y` în fișierul `date.mat`. Pentru obţinerea variabilelor păstrate într-un fișier `.mat` se folosește comanda `load`.

Vectorizări

Operaţiile cu vectori şi matrice sunt executate de 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 secvenţa:

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

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

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

Probleme rezolvate

Vom prezenta, în continuare, alte exemple de vectorizări în care am folosit puterea operatorilor logici şi a funcţiilor Octave.

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;
    endif
endfor
```

Listing 6: Exemplu de utilizare a buclei for.

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

Listing 7: Exemplu de cod vectorizat.

În acest exemplu, ne-am bazat pe faptul că funcţia `sqrt` primeşte ca parametri şi numere negative, având ca rezultat un număr complex. Am folosit operatorul < care pentru vectori are ca rezultat un alt vector cu 1 pe poziţiile ce satisfac condiţia. Mai mult, 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);
       endif
   endfor
endfor
```

Listing 8: Exemplu de utilizare a buclei for.

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

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

Listing 9: Exemplu de cod vectorizat.

Exemplul 3

```
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);
        endfor
        V(len) = 0; len = len-1;
    else
        i = i+1;
    endif
endwhile
V = char(V)
```

Listing 10: Exemplu de utilizare a buclei for.

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

```
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 altă metodă care elimină toate spaţiile:

```
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
```

Listing 13: Exemplu de funcție în Octave.

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

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

Listing 14: Exemplu de utilizare al operatorilor Hadamard.

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

Probleme propuse

Problema 1

Scrieți fișierul `valori.txt` cu valorile funcției $f(x) = 2x + 1$ pe intervalul `[0, 1]` şi pasul 0.1.

Problema 2

Scrieți o funcție care să calculeze suma numerelor impare mai mici ca `n` utilizând bucla `for`. Aceeași problemă utilizând bucla `while`. Citirea unui numar de la tastatură se face utilizând comanda:

```
var = input('Introduceţi variabila:').
```

Problema 3

Scrieți o funcție care citește o matrice pătratică din fișier și verifică dacă matricea are proprietățile unui pătrat magic (suma elementelor pe linii, coloane și diagonale este aceeași).

Problema 4

Scrieți o funcție pentru determinarea dimensiunii unui fișier. Funcția are ca parametru numele fișierului.

Problema 5

Scrieți 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. La final, se va afisa numărul total de apariții. Funcția are semnătura:

```
function y = NumarAparitii(numefis, sir)
```