Introducere în MATLAB. Funcții și instrucțiuni. Vizualizarea datelor.

Colaboratori: Andrei STAN, Bogdan Țigănoaia

February 15, 2025

Cuprins

1 Obiective laborator

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

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

2 Noțiuni teoretice

2.1 Ce este MATLAB?

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 limbaj de programare, cât și o suită software, special construit pentru calcul numeric, vizualizare de date, procesarea semnalelor, etc. Utilitatea limbajului constă că este bazat pe folosirea matricilor: toate datele sunt reprezentate sub forma unor matrici. Asta înseamnă că putem lucra mult mai ușor cu numerele, dar și mai rapid, toate operațiile fiind optimizate. Pentru vizualizare ne sunt puse la dispoziție numeroase funcții pentru crearea graficelor atât 2D, cât și 3D.

2.2 Software necesar

Recomandăm folosirea GNU Octave pentru parcurgera laboratoarelor, cât și a temelor.

Mult mai puternică este suita de la MathWorks: `https://matlab.mathworks.com/`. Pentru autentificare, folosiți mail-ul instituțional. Universitatea oferă deja licență tuturor. Mediul online este suficient pentru acest curs.

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

2.3 Interfața Octave

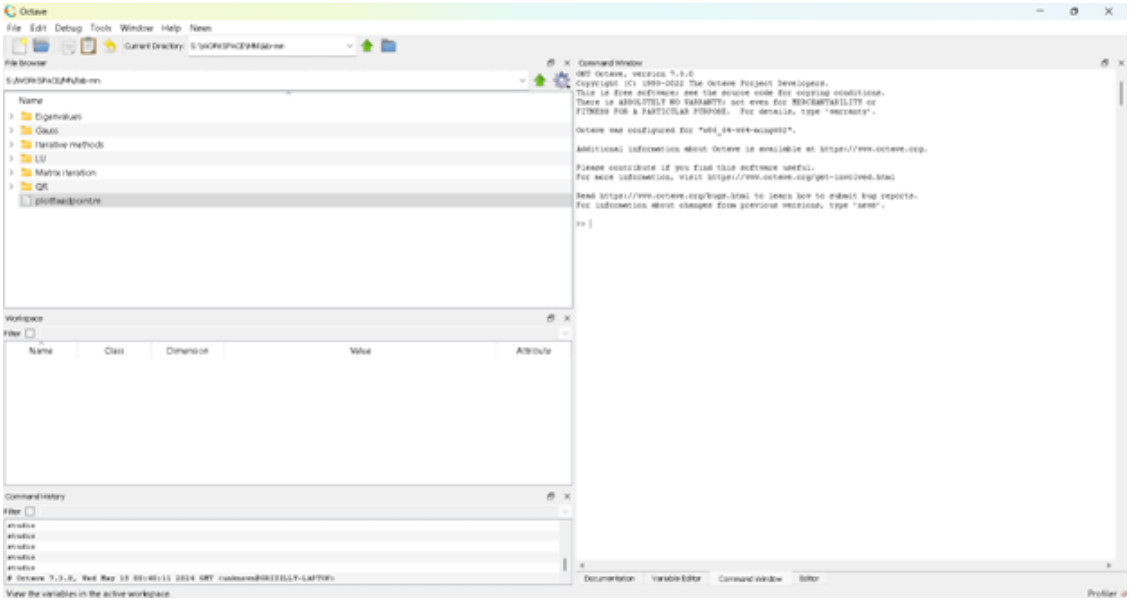

Figura 1: Interfața Octave

Limbajul este similar cu Python, fiind unul interpretat. În partea de sus vedem directorul curent (cel în care se vor rula comenzile).

În partea stângă avem 3 panouri:

- **File Browser** - unde putem naviga prin fișiere;
- **Workspace** - unde sunt stocate variabilele;
- **Command History** - istoricul comenzilor.

În partea dreaptă avem 4 panouri:

- **Cocumentation** - unde putem căuta ajutor;
- **Variable Editor** - unde putem modifica variabilele;
- **Command Window** - unde putem rula comenzile;
- **Editor** - unde putem scrie codul.

2.4 Introducere în MATLAB

De acum, pentru rularea comenzilor vom folosi *Command Window*.

2.4.1 Comenzi

Toate comenzile se execută în dircetorul curent. Putem folosi comanda `pwd` pentru a vedea directorul curent.

Rezultatul unui calcul se va stoca mereu în variabila `ans`:

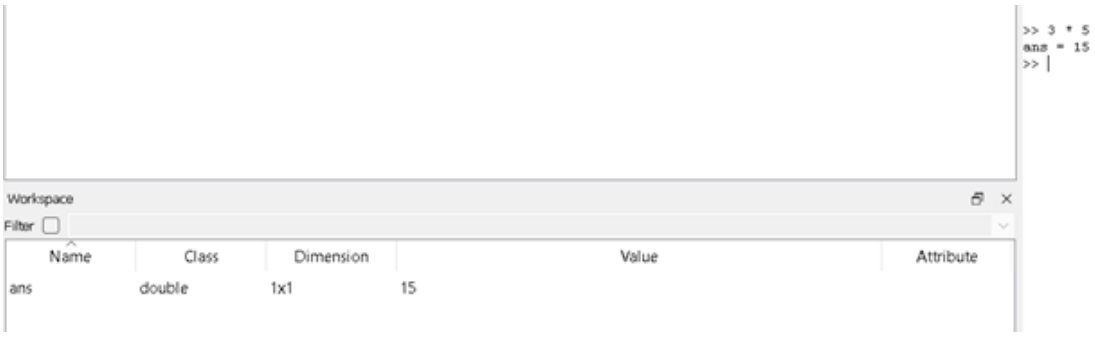

Figura 2: Variabila ans

Variabilele se crează prin atribuire:

```
>> m = 3 * 5
>> y = m / 2
```

După rularea comenzilor, obervați că mereu se afișează rezultatul. Pentru a nu afișa rezultatul, folosiți `;` la sfârșitul comenzii.

```
>> m = 3 * 5;
>> y = m / 2;
```

2.4.2 Manipularea matricelor

MATLAB este un limbaj orientat pe matrici. Toate operațiile sunt făcute pe matrici. Scalarii sunt considerați matrici de dimensiune 1x1. Pentru a crea o matrice sau un vector putem folosi construcțiile de mai jos:

```
>> A = [1 2 3; 4 5 6; 7 8 9] % matrice 3x3
>> B = [1 2 3] % vector linie
>> C = [1; 2; 3] % vector coloana
```

Pentru crearea vectorilor cu elemente egal depărtate, putem folosi construcția `start : step : end`:

```
>> D = 1 : 2 : 10 % vector cu elemente de la 1 la 10, cu pasul 2
>> E = 1 : 5 % vector cu elemente de la 1 la 5, pasul a fost omis (implicit 1)
```

Operatorii care acționează pe matrici sunt:

- `+` - adunare;
- `-` - scădere;
- `*` - înmulțire;

- `/` - împărțire;
- `'` - transpusa (hermitica);
- `.` - operatorul Hadamard (execută operația element cu element).

Prin împărțirea unei matrici la alta, se înțelege înmulțirea primei cu inversa celei de-a doua. De exemplu, pentru rezolvarea sistemului $Ax = b$, se poate folosi formula $x = A\backslash b$.

Exemplul de mai jos folosește operatorul Hadamard, rezultatul fiind matricea `A = [1 4; 9 16]`:

```
>> A = [1 2; 3 4];
>> B = A .^ 2
```

Pentru a afla dimensiunile unei matrici, folosim funcția `size`. Aceasta va returna un vector cu mai multe elemente, fiecare reprezentând lungimea unei dimensiuni.

Accesarea elementelor unei matrici se face folosind indicii (începând de la 1):

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(2, 3) % elementul de pe linia 2, coloana 3
>> A(2, :) % linia 2
>> A(:, 2) % coloana 2
>> A(1 : 2, 1 : 2) % submatricea formata din primele 2 linii si primele 2 coloane
>> A([1 3], :) % liniile 1 si 3
```

2.4.3 Salvarea și încărcarea datelor

Pentru a salva spațiul de lucru curent, adică toate variabilele și valorile lor, putem folosi comanda `save`:

```
>> save nume_fisier.mat
```

Pentru a încărca datele salvate, folosim comanda `load`:

```
>> load nume_fisier.mat
```

Pentru a șterge o variabilă din spațiul de lucru, folosim comanda `clear`:

```
>> clear nume_variabila
```

2.4.4 Funcții și variabile predefinite

MATLAB vine cu o mulțime de funcții și variabile predefinite. De exemplu, constanta `pi`, funcțiile `sin`, `cos`, `tan`, `exp`, `log`, `sqrt`, `abs`, `round`, `floor`, `ceil`.

```
>> pi / 2
% output: 1.5708
```

Chiar dacă sunt afișate doar 4 zecimale, MATLAB folosește o precizie de 16 zecimale.

O particularitate foarte frumoasă a funcțiilor este că acestea pot fi aplicate pe vectori, rezultatul fiind un vector cu aceeași dimensiune. Mai târziu vom prezenta conceptul de *vectorizare* și de ce e bine să scriem cod așa și nu cu bucle.

```
>> A = [0 pi/2 pi 3*pi/2 2*pi];
>> sin(A)
% output: [0 1 0 -1 0]
```

2.4.5 Script-uri

Un script este un fișier care conține o secvență de comenzi MATLAB. Acesta are extensia `.m`. Pentru a rula un script, putem folosi GUI-ul Octave apăsând pe butonul *Run* sau putem folosi linia de comandă scriind numele fișierului fără extensie.

Într-un script putem folosi toate comenzile pe care le-am folosit în *Command Window*. De exemplu, scriptul de mai jos va afișa pe ecran numerele de la 1 la 10:

```
for i = 1 : 10
  disp(i);
endfor % sau end
```

2.4.6 Instrucțiuni de control

MATLAB suportă instrucțiuni de control precum `if`, `for`, `while`, `switch`. Sintaxa este similară cu cea din alte limbaje de programare.

```
if conditie
  % cod
elseif conditie
  % cod
else
  % cod
endif % sau end
```

```
for i = 1 : 10 % for poate opera pe orice vector, variabila i va lua valorile pe rand
  % cod
endfor % sau end
```

```
while conditie
  % cod
endwhile % sau end
```

```
switch variabila
  case valoare1
    % cod
  case valoare2
    % cod
  otherwise
    % cod
endswitch % sau end
```

2.4.7 Funcții definite de utilizator

Funcțiile sunt secvențe de comenzi care primesc unul sau mai mulți parametri și întorc un rezultat. Pentru modularizrea codului, este bine să scriem fiecare funcție într-un fișier separat. Acest fișier va avea aceeași denumire cu funcția, dar cu extensia `.m`.

Creați fișierul `suma.m` cu următorul conținut:

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

Testați apelurile funcției `suma(1, 2)` și `suma(1 : 2, 3 : 4)`.

Putem avea și funcții void (care nu întorc nimic) folosind `function [] = nume_functie()` sau funcții care întorc mai multe valori folosind `[a, b, c, d] = nume_functie()`.

2.4.8 Vectorizări

MATLAB este optimizat pentru operațiile pe matrici. De aceea, este bine să folosim cât mai puține bucle în codul nostru. Acest lucru se numește *vectorizare*. De exemplu, în loc să scriem:

```
for i = 1 : 10
  y(i) = x(i) ^ 2;
endfor
```

Putem scrie:

```
y = x .^ 2;
```

În continuare, vom prezenta câteva exemple de vectorizare dar și de alte funcții și construcții utile.

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
```

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

În acest exemplu, ne-am bazat pe faptul că funcția `sqrt` primește ca parametrii și numere negative, având ca rezultat un numar complex. Am folosit operatorul `<` care aplicat vectorilor are ca rezultat un alt vector cu 1 pe pozițiile ce satisfac condiția. 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
```

```
M = magic(3);
ind = find(M > 4); % returneaza indicii elementelor care satisfac conditia
M(ind) = -M(ind)
```

În acest exemplu, secvența care folosește bucla for se execută în 23.4956 unități de timp iar secvența vectorizată, prin intermediul funcției find, se execută în 2.1153 unități de timp, deci de 11 ori mai rapid.

Exemplul 3:

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

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

În acest exemplu, am folosit operatorul Hadamard pentru eliminarea buclelor.

2.4.9 Vizualizarea datelor

MATLAB vine cu o mulțime de funcții pentru vizualizarea datelor. În laboratoarele următoare vom folosi aceste funcții pentru a vizualiza rezultatele obținute.

Funcția `plot` este folosită pentru a crea grafice 2D. Aceasta primește ca parametrii doi vectori, unul pentru axa x și unul pentru axa y. De asemenea, putem adăuga și alți parametrii pentru a personaliza graficul. Pentru mai multe detalii privind parametrii, consultați documentația.

```
x = 0 : 0.1 : 2 * pi;
y = sin(x);
plot(x, y, 'r--o')
```

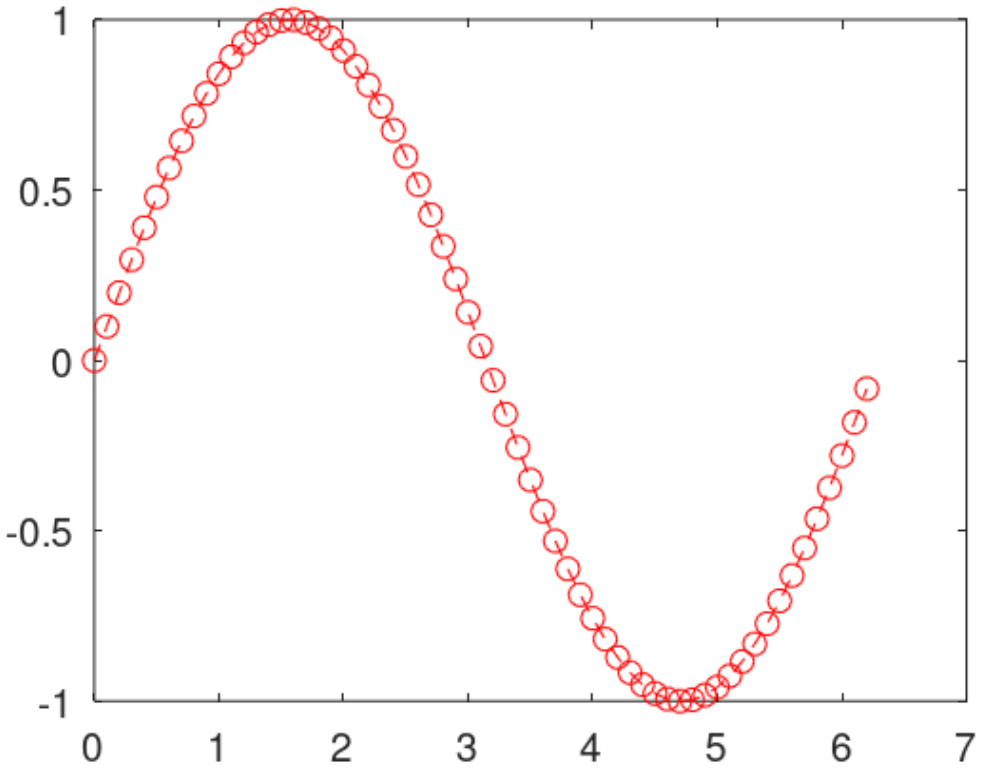

Figura 3: Funcția $sin(x)$ evaluată pe intervalul $[0, 2 * \pi]$, cu pas 0.1.

Pentru a trasa graficele funcțiilor multiple, putem folosi aceeași funcție `plot`, dar cu mai mulți parametrii.

```
x = 0 : 0.1 : 2 * pi;
y1 = sin(x);
y2 = cos(x);
plot(x, y1, '-', x, y2, 'o')
```

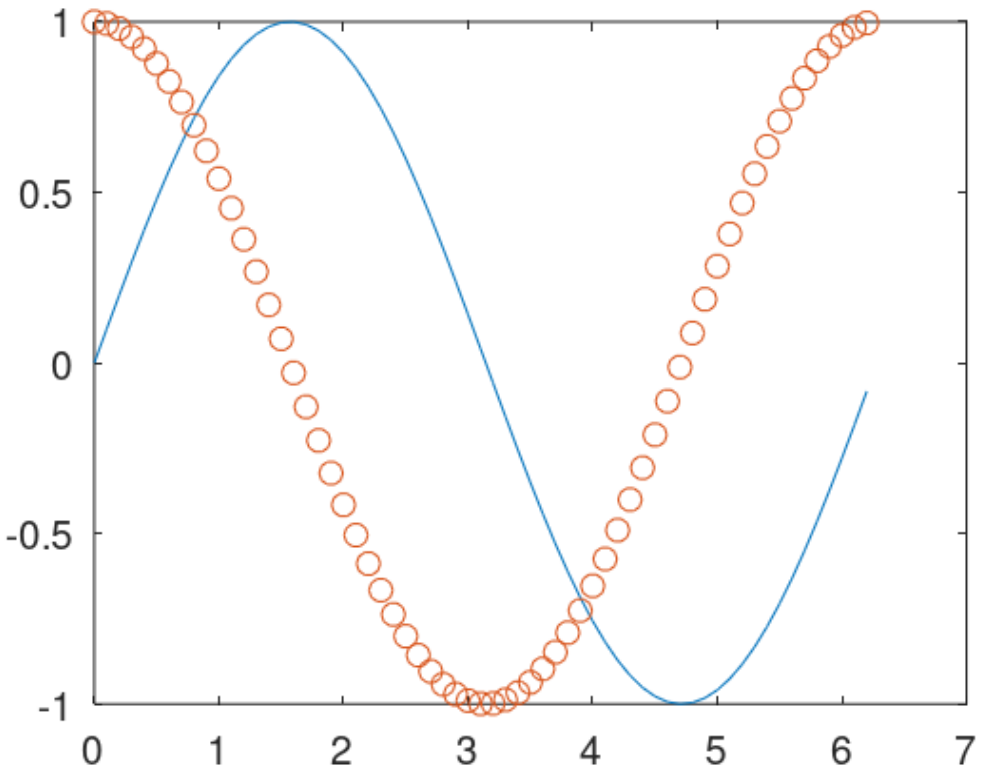

Figura 4: Funcția $sin(x)$ trasată prin interpolare liniară și funcția $cos(x)$ trasată prin puncte.

3 Probleme

1. Creați câte un script pentru funcțiile $f(x) = x^2$, $g(x) = x^3$ și $h(x) = x^4$. Trasați graficele acestora pe intervalul $[-10, 10]$.
2. Construiți vectorul x cu elemente de la 0 la 10 cu pasul 3. Construiți vectorul $y = sin(i), i \in x$. Trasați graficul rezultat din cei doi vectori. Rulați script-ul cu pași din ce în ce mai mici pentru x. Ce observați?