Dana-Elena GHIȚĂ Ioan ȘERBAN

MATLAB PENTRU INGINERIE ELECTRICĂ

Suport de curs

Editura Universității Transilvania din Brașov

2026

EDITURA UNIVERSITĂȚII TRANSILVANIA DIN BRAȘOV

Adresa: Str. Iuliu Maniu nr. 41A
500091 Brașov
Tel.: 0268 476 050
Fax: 0268 476 051
E-mail: editura@unitbv.ro

Editură acreditată de CNCSIS, cod 81.

ISBN 978-606-19-1706-8 (ebook)

Lucrarea a fost avizată de Consiliul Departamentului de Inginerie electrică și fizică aplicată, Facultatea de Inginerie electrică și știința calculatoarelor a Universității Transilvania din Brașov.

Cuprins

Introducere

În contextul dezvoltării accelerate a tehnologiilor digitale și a metodelor moderne de analiză inginerească, utilizarea instrumentelor software specializate a devenit esențială în activitatea inginerilor. În domeniul ingineriei electrice, analiza, modelarea și simularea fenomenelor electrice și electronice necesită instrumente performante care să permită realizarea rapidă a calculelor numerice, vizualizarea rezultatelor și dezvoltarea algoritmilor de analiză.

Unul dintre cele mai utilizate software environments în acest scop este MATLAB [1], un instrument puternic destinat calculului numeric, analizei datelor, modelării matematice și simulării sistemelor tehnice. Datorită capacităților sale extinse și a bibliotecilor specializate, MATLAB este utilizat pe scară largă atât în mediul academic, cât și în industrie, fiind un instrument esențial în formarea viitorilor ingineri.

Lucrarea **„MATLAB pentru Inginerie Electrică”** își propune să ofere studenților o introducere sistematică în utilizarea mediului MATLAB, punând accent pe conceptele fundamentale necesare pentru rezolvarea problemelor specifice domeniului ingineriei electrice. Manualul este conceput astfel încât să îmbine elementele teoretice cu aplicații practice și exemple relevante, facilitând înțelegerea modului în care MATLAB poate fi utilizat în analiza și simularea sistemelor electrice.

Exemplele furnizate sunt alcătuite din cazuri practice întâlnite de autori în activitatea profesională, precum și din alte publicații specifice utilizării MATLAB pentru rezolvarea problemelor din ingineria

electrică [2], [3]. Autorii au adaptat conținutul cursului la cele mai recente versiuni ale mediului de programare MATLAB, care are o istorie începută în anii 1980 și care a înregistrat progrese semnificative de-a lungul timpului. Referințele [4], [5] și [6] oferă o corelare a conținutului acestui curs cu programe similare din cadrul altor universități internaționale, în care MATLAB reprezintă o unealtă de bază în inginerie și în alte domenii.

Componenta Simulink a MATLAB, detaliată în ultimul capitol al cursului, are la bază experiența extensivă a autorilor în modelarea sistemelor electrice, precum și aplicații practice prezentate în [7].

Structura lucrării este organizată în capitole care urmăresc trecerea graduală de la conceptele de bază ale mediului MATLAB către aplicații mai complexe utilizate în analiza și modelarea circuitelor electrice.

Capitolul 1 prezintă noțiunile generale referitoare la mediul MATLAB, domeniile sale de utilizare și principalele componente ale interfeței grafice. Sunt descrise elementele de bază ale mediului de lucru, tipurile de fișiere utilizate în MATLAB, precum și modul de organizare a scripturilor și funcțiilor.

Capitolul 2 introduce conceptele de bază necesare pentru lucrul în MATLAB, precum definirea variabilelor, lucrul cu vectori și matrice, generarea structurilor de date și utilizarea operatorilor aritmetici, relaționali și logici. Aceste elemente constituie fundamentul tuturor calculelor realizate în MATLAB.

Capitolul 3 prezintă principalele mecanisme de control pentru execuția programelor, precum instrucțiunile condiționale și structurile repetitive.

Sunt descrise instrucțiunile if, switch, for și while, precum și modul de organizare a programelor MATLAB prin utilizarea funcțiilor.

Capitolul 4 abordează reprezentarea și manipularea numerelor complexe în MATLAB, aspect deosebit de important în analiza circuitelor electrice. Sunt prezentate funcțiile MATLAB dedicate numerelor complexe și aplicații practice utilizate în analiza sistemelor electrice.

Capitolul 5 tratează operațiile fundamentale cu matrice, element central în MATLAB. Sunt prezentate operațiile de adunare și scădere a matricilor, înmulțirea și împărțirea matricilor, ridicarea acestora la putere, precum și operația de transpunere. De asemenea, sunt discutate operațiile aritmetice cu tablouri, tehnici de indexare avansată și metode de rezolvare a sistemelor de ecuații liniare. Capitolul conține exemple și probleme rezolvate pentru o mai bună înțelegere a noțiunilor.

Capitolul 6 prezintă operațiile matriceale utilizate frecvent în analiza numerică, inclusiv manipularea matricilor, accesarea elementelor acestora și realizarea operațiilor matematice specifice. Sunt incluse exemple relevante pentru aplicații în analiza rețelelor electrice.

Capitolul 7 este dedicat lucrului cu polinoame în MATLAB, iar Capitolul 8 prezintă instrumentele grafice oferite de MATLAB.

Capitolul 9 este dedicat aplicațiilor MATLAB în domeniul metodelor numerice. Sunt prezentate funcții utilizate pentru interpolarea și aproximarea datelor, integrarea și derivarea numerică, precum și metode de integrare numerică a ecuațiilor diferențiale. De asemenea, sunt abordate metodele de rezolvare numerică a ecuațiilor diferențiale de ordin superior și a sistemelor de ecuații diferențiale.

Capitolul 10 introduce mediul Simulink, utilizat pentru modelarea și simularea sistemelor dinamice. În acest capitol sunt prezentate etapele realizării unui model în Simulink, metodele de exportare a rezultatelor simulării, crearea subsistemelor și inițializarea parametrilor unei simulări. De asemenea, sunt prezentate exemple de modelare și simulare a circuitelor electrice utilizând MATLAB/Simulink.

Prin structura sa, această lucrare își propune să ofere studenților o bază solidă pentru utilizarea mediului MATLAB în rezolvarea problemelor inginerești, contribuind la dezvoltarea competențelor necesare în domeniul analizei și simulării sistemelor electrice.

Capitolul 1

Introducere în MATLAB

Mediul de programare MATLAB este unul dintre cele mai utilizate instrumente software în ingineria electrică și în domeniile conexe. Acest prim capitol oferă studenților primele instrumente necesare pentru a lucra eficient în MATLAB: ce reprezinta, de ce îl folosim în inginerie electrică, cum este organizat mediul de lucru, cum definim variabile, matrici și scripturi/funcții elementare și cum verificăm/afișăm rezultate. De asemenea conținutul oferă explicații pas cu pas, exemple rezolvate și probleme propuse cu indicații. Se vor prezenta pe scurt istoria, domeniile de utilizare, avantajele și structura mediului Matlab, precum și tipurile de licențe disponibile.

1.1 Ce este MATLAB?

MATLAB este un mediu de calcul numeric și programare de nivel înalt, dezvoltat de firma **MathWorks Inc.** [1]. Denumirea provine de la **MATrix LABoratory**, ceea ce subliniază orientarea sa inițială către operații cu matrice.

În pezent, MATLAB este folosit pe scară largă în:

- matematică și calcul numeric,
- dezvoltarea algoritmilor,

- modelare, simulare și testarea prototipurilor,
- analiza și vizualizarea datelor,
- grafică inginerească și științe aplicate,
- dezvoltarea de aplicații (inclusiv interfețe grafice – GUI).

MATLAB reprezintă una dintre cele mai utilizate platforme software în domeniul ingineriei electrice, fiind recunoscut ca un mediu integrat de programare, analiză numerică și vizualizare grafică. Prin combinația sa de putere de calcul, flexibilitate și ușurință în utilizare, MATLAB permite: implementarea rapidă a modelelor matematice complexe, simularea proceselor fizice și a fenomenelor electrice reale, analiza datelor experimentale și verificarea ipotezelor teoretice, precum si proiectarea și testarea sistemelor de control și automatizare.

Platforma MATLAB, împreună cu mediul Simulink, oferă o suită completă de instrumente pentru proiectarea, analiza și optimizarea sistemelor electrice, electronice și de control automat. Cele mai frecvente aplicații practice includ:

1. Analiza circuitelor electrice liniare și neliniare: determinarea curenților și tensiunilor în regim staționar sau tranzitoriu; calculul impedanței și al puterii complexe; analiza răspunsului în frecvență al rețelelor RLC.
2. Procesarea semnalelor electrice și a armonicilor: analiza spectrală folosind transformata Fourier discretă (DFT), implementată prin algoritmul FFT, filtrarea și reconstrucția semnalelor, detecția componentelor armonice și analiza distorsiunilor.

3. Modelarea și simularea sistemelor dinamice: reprezentarea matematică a motoarelor electrice, convertoarelor și rețelelor, analiza stabilității și a răspunsului în timp, simularea interacțiunii dintre componente într-un sistem complex.
4. Proiectarea și testarea filtrelor și sistemelor de control: proiectarea filtrelor digitale, filtre digitale de tip FIR (Finite Impulse Response) și IIR (Infinite Impulse Response), sinteza compensatoarelor și controlerelor PID, simularea buclelor de reglare automată.
5. Dezvoltarea de interfețe și aplicații dedicate: realizarea interfețelor grafice interactive (GUI) pentru vizualizarea datelor, implementarea și testarea algoritmilor de control, conectarea MATLAB la dispozitive hardware și sisteme de achiziție de date (DAQ, Arduino și Programmable Logic Controller – PLC).

Structura pachetetelor software de lucru in MATLAB este afișata in Figura 1.

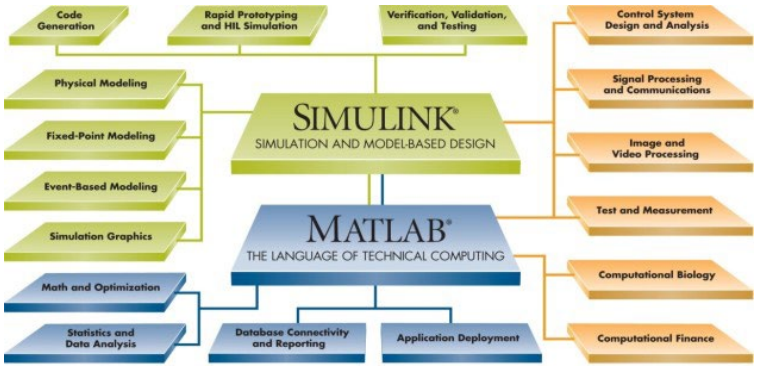


Figura 1 Familia de produse Matlab [1]

Licențe și alternative gratuite ale mediului MATLAB

MATLAB este un software comercial, însă Universitatea Transilvania din Brașov oferă acces campus gratuit pentru studenți prin adresa de e-mail instituțională (@student.unitbv.ro).

MATLAB campus-wide license (Matlab, Simulink și toolbox-urile conexe)

Instalarea pe calculatoarele studenților Universitații Transilvania se poate face prin portalul: https://www.mathworks.com/academia/tah-portal/universitatea-transilvania-din-brasov-31585031.html.

Este necesară crearea unui cont utilizând adresa de email instituțională.

Licența pentru calculatoarele individuale se numește MATLAB (Individual).

Accesul la Matlab Online se poate face la adresa: https://matlab.mathworks.com/

Există mai multe tipuri de licențe:

- Standard (comercială)
- Academică/Education
- Student
- Home

Matlab cloud online & mobile app:

- Online: https://matlab.mathworks.com/
- Mobile: https://www.mathworks.com/products/matlab-mobile.html

Programele alternative gratuite, dar cu funcționalitate similară sunt următoarele: **Scilab** (www.scilab.org) și **- GNU Octave** (https://octave.org). Majoritatea programelor simple realizate în

MATLAB pot fi rulate în aceste medii cu modificări minime, însă MATLAB oferă un suport mai bogat și toolbox-uri specializate.

Mediul MATLAB — ferestre și concepte cheie

La deschidere, interfața tipică conține:

- **Command Window** — spațiul de comenzi unde se introduc instrucțiuni și se obțin rezultatele imediat.
- **Workspace** – zona în care se afișează variabilele existente în memorie.
- **Editor (M-file)** – fereastra pentru scrierea și editarea programelor MATLAB sub formă de scripturi sau funcții.
- **Figure Window** – fereastra pentru afișarea graficelor 2D și 3D.
- **Current Folder** — locația fișierelor `.m`, `.mat`, etc.
- **Help / Documentation** — `help`, `doc`, `lookfor`.

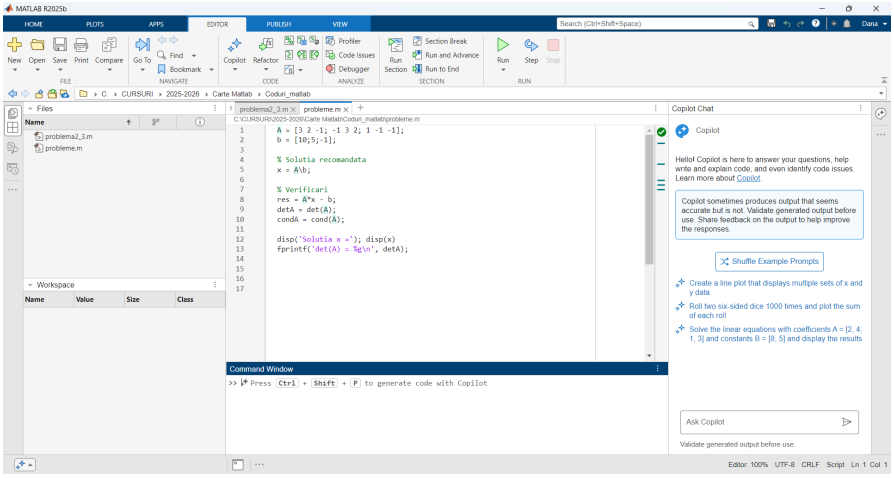


Figura 2 Interfața MATLAB

Sfat: folosiți `clc` pentru a curăța Command Window, `clear` pentru variabile, `close all` pentru inchiderea graficelor.

1.2 Fișiere MATLAB: scripturi vs funcții

Script (.m): reprezintă o secvență de comenzi executate în spațiul de lucru curent și nu are argumente de intrare/ieșire. Variabilele create rămân în **Workspace**.

Exemplu in MATLAB: `script1.m`

```
x = 1:0.1:2;
y = sin(x);
plot(x,y)
```

Funcție (.m): reprezintă un fișier care are argumente de intrare și ieșire, fiind mai ușor de reutilizat, iar variabilele sunt locale funcției. Prima linie are template-ul următor:

```
function [param_iesire] = nume_functie(param_intrare)
```

Exemplu in MATLAB (funcția `medie.m`):

```
function m = medie(v)
% calculeaza media aritmetica a vectorului v
  m = sum(v)/length(v);
end
```

Explicație linie cu linie a codului din fișierul funcție **medie.m**:

1. `function m = medie(v)`

   - Declară o funcție MATLAB numită `medie`.

   - `m` este variabila de ieșire (valoarea returnată).

   - `v` este argumentul de intrare (un vector).

   - Semnificația: atunci când apelăm `m = medie(v)`, MATLAB execută codul funcției și în `m` plaseaza rezultatul.

2. comentariu: `% calculeaza media aritmetica a vectorului v`

   - Linia începe cu `%` → comentariu. Comentariul este ignorat la executare.

   -Explică scopul funcției: returnează media aritmetică a elementelor vectorului `v`.

3. (indentare) `m = sum(v)/length(v);`

   -linia unde se efectuează calculul mediei:

   - `sum(v)` calculează suma tuturor elementelor din `v`. Dacă `v` este un vector rând sau coloană, `sum(v)` este suma scalară a elementelor.
   - `length(v)` returnează lungimea vectorului `v` (numărul de elemente al vectorului).
   - Împarte suma la numărul de elemente → obține media aritmetică.
   - Rezultatul este atribuit variabilei de ieșire `m`.

4. `end`

   - Marchează sfârșitul funcției. Pentru funcții scurte într-un fișier `.m`, `end` este opțional în versiunile mai vechi, dar recomandat pentru claritate.

Sfat: folosiți funcții pentru reutilizarea codului și scripturi pentru experimente rapide.

Exemplu

Creați un fișier script `test1.m` cu următorul conținut:

```
A = [1 2 3; 4 5 6];
B = [1 5 8 7 6 5];
disp('Matricea A este:'); disp(A)
disp('Vectorul B este:'); disp(B)
```

Rulați scriptul în MATLAB și observați rezultatul afișat în Command Window.

1.3 Probleme propuse

1.3.1. Instalați MATLAB și identificați ferestrele principale din interfață.

1.3.2. Creați un fișier script care definește două matrice, le adună și afișează rezultatul.

1.3.3. Scrieți o funcție MATLAB care calculează valoarea efectivă a unui semnal sinusoidal definit prin amplitudine și frecvență.

1.3.4 Enumerați domeniile de aplicare a MATLAB-ului în ingineria electrică.

Se recomandă

- folosirea comentariilor la începutul fiecărui fișier: scop, date test.
- folosirea numelor de variabile sugestive (`I_max`, `t`, `V_in`).

- evitarea `clear all` în scripturi care rulează continuu (poate elimina funcții compilate).
- folosirea `;` la finalul liniilor pentru a evita afișările inutile.
- documentarea funcțiilor cu comentarii care apar in *help*.

Capitolul 2

Noțiuni de bază în MATLAB: variabile, matrice, operatori

Acest capitol introduce conceptele fundamentale ale limbajului MATLAB: variabile, tipuri de date, matrice și vectori, precum și operatorii aritmetici și logici. Aceste noțiuni stau la baza oricărui program MATLAB și trebuie însușite înainte de a aborda aplicații mai complexe.

2.1 Variabile în MATLAB

O variabilă este un nume simbolic asociat unei valori stocate în memorie. În MATLAB nu este necesară declararea tipului variabilei înainte de utilizare, ci acesta se stabilește automat în funcție de valoarea atribuită. Nume permise de variabile pot conține litere, cifre (nu începe cu o cifră), underscore (_). Nu se folosesc cuvintele rezervate (`for`, `if`, `end`, `function` etc.).

Exemple variabile:

```
x = 5;          % variabilă scalar
nume = 'Ion';   % șir de caractere
z = 2 + 3i;     % număr complex
```

Variabila speciala	**Descriere**
ans	Numele implicit al unei variabile in cazul in care nu s-a asignat un nume.
pi	Variabila care are alocata valoarea $\pi = 3.14159...$
eps	Variabila in care este memorata eroarea relativa pentru calculele efectuate in virgula mobila. Valoarea implicita este: 2.2204e-16.
inf	Variabila folosita pentru reprezentarea lui $+\infty$ (rezultatul lui 1/0)
NaN sau *nan*	Variabila folosita pentru reprezentarea lui Not-a-Number (NaN) (rezultatul lui 0/0)
i sau *j*	Variabila folosita pentru reprezentarea lui $\sqrt{-1}$

Comenzi utile:

`who` – afișează variabilele existente în Workspace.

`whos` – afișează variabilele și tipul/dimensiunea acestora.

`clear x` — șterge variabila `x`.

`clear all` – șterge toate variabilele.

`save('file.mat')` / `load('file.mat')` — salvare/încărcare Workspace.

MATLAB folosește reprezentarea cu virgulă mobilă, dar se folosește des și notația științifică:

```
a = 2.35e-15;   % 2.35 × 10^-15
b = 5.23587e3;  % 5.23587 × 10^3
```

Comenzi Input/Output

În mediul de programare MATLAB comenzile pentru introducerea de informații în fereastra de comandă și pentru afișarea datelor (comenzi de intrare/ieșire) sunt: `echo`, `input`, `pause`, `keyboard`, `break`, `error`, `display`, `format` și `fprintf`.

O scurtă descriere a acestor comenzi este prezentată mai jos.

Comandă	Descriere
echo	Afișează în Command Window fiecare linie de cod executată dintr-un script.
input	Solicită utilizatorului să introducă o valoare sau un șir de caractere.
pause	Oprește temporar execuția programului până când utilizatorul apasă o tastă sau după un anumit timp.
keyboard	Oprește execuția și intră în modul interactiv pentru depanare
break	Oprește imediat execuția unei bucle (`for` sau `while`).
error	Afișează un mesaj de eroare și oprește execuția programului.
display	Afișează valoarea unei variabile sau expresii în Command Window.
format	Modifică modul de afișare a numerelor (de exemplu, zecimale, științific).
fprintf	Afișează text și/sau valori formatate în Command Window (similar cu `printf` din C).

Comanda **echo** poate fi folosită în scopuri de depanare (debugging). Comanda **echo** permite vizualizarea comenzilor pe măsură ce sunt executate. **Echo** poate fi activat sau dezactivat.

`echo on` – activează afișarea comenzilor

`echo off` – dezactivează afișarea comenzilor

`echo` – singură, comută starea de afișare a comenzilor

Funcția **input** permite programului să primească valori de la utilizator prin fereastra de comandă (Command Window).

```
variabila = input('Mesaj pentru utilizator: ');
```

Textul plasat între apostrofuri (`'...'`) este afișat ca mesaj pentru utilizator. Programul așteaptă ca utilizatorul să introducă o valoare și să apese tasta Enter. Valoarea introdusă este stocată în variabila specificată.

Comanda `break` este folosită pentru a ieși imediat dintr-o buclă (`for` sau `while`), indiferent dacă condiția buclei mai este adevărată sau nu. După executarea comenzii `break`, MATLAB continuă execuția codului imediat după buclă.

Functia **`fprintf`** este folosită pentru a afișa text și/sau valori numerice formatate în Command Window sau într-un fișier și este similară cu funcția `printf` din limbajul C.

```
fprintf('Mesaj text cu specificatori de format', variabile);
```

Textul plasat între apostrofuri este afișat exact cum este scris character cu caracter. Specificatorii de format (`%d`, `%f`, `%s`, etc.) indică cum sunt formatate variabilele pentru afișare astfel: `%d` → număr întreg, `%f` → număr real (zecimale), `%s` → șir de caractere. Variabilele se scriu ca argumente după mesajul sir de caractere, în ordinea în care apar specificatorii.

2.2 Matrice, vectori și scalari

Elementul fundamental de lucru in MATLAB-ul este matricea. Chiar și un scalar este considerat o matrice de dimensiune 1×1. Mai jos sunt exemplificate definițiile a trei matrice**:**

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

Indexare în matrice**:**

`A(i,j)` — elementul matricei A situate pe randul `i` coloana `j`.

`A(:,j)` — toată coloana `j`.

`A(i,:)` — toată linia `i`.

`A(1:3,2:5)` — submatrice (exemplu).

Indici vectori: `A([1 3 5],2:4)`.

Accesarea unui element: `A(i,j)` (i = linie, j = coloană).

`A(1,3)` % elementul de pe linia 1, coloana 3

Pentru a afla dimensiunea unei matrice, se poate utiliza funcția `size(numele_matricei)`, ce va returna un vector cu două elemente si anume numarul de linii (m) si numarul de coloane (n) al matricei:

```
[m,n] = size(A);
```

Pentru a afla numărul de elemente al unui vector se folosește funcția `length(v)`.

Transpunerea matricilor:

`A.'` — transpunere (fără conjugare)

`A'` — conjugata transpusă (pentru complexe)

Funcții de ajutor și documentație:

- `help funcname` — scurt ajutor în Command Window.
- `doc funcname` — pagină completă de documentație.
- `lookfor keyword` — caută în descrierile funcțiilor.

Exemplu: `help plot`, `doc fft`.

Funcții pentru reprezentare grafică simplă:

- `plot(x,y)` — grafic 2D, x, y sunt vectori.
- `xlabel`, `ylabel`, `title`, `legend`.
- `subplot(m,n,p)` pentru mai multe ploturi într-o figură.
- `figure` deschide o fereastră nouă, `saveas(gcf,'fig.png')` salvează.

Exemplu in MATLAB (`grafic.m`):

```
t = 0:0.001:0.1;
y = sin(2*pi*50*t);
plot(t,y); grid on;
xlabel('t [s]'); ylabel('v(t)'); title('Semnal sinus 50 Hz');
```

Explicație linie cu linie a codului din fisierul script `grafic.m`:

Linia 1. `t = 0:0.001:0.1;`

- Creează un vector rând `t` care începe de la valoarea `0`, crește cu pas de `0.001` și se oprește la `0.1` (inclusiv).

- `0:step:stop` este notația MATLAB pentru un vector cu pas constant.

- Lungimea lui `t` este `(0.1 - 0)/0.001 + 1 = 101` elemente.

-Semnul `;` la final oprește afișarea vectorului în Command Window.

Linia 2. `y = sin(2*pi*50*t);`

- Calculează funcția sinus pentru fiecare element din vectorul `t`. Operația este vectorizată: `sin(...)` primește un vector ca argument și returnează un vector de aceeași dimensiune.

- `2*pi*50` este frecvența unghiulară $\omega = 2\pi f$ cu `f = 50 Hz`. Deci `omega = 100*pi` rad/s.

- Rezultatul `y` este semnalul sinusoidal $y(t)=\sin(2\pi \cdot 50 \cdot t)$.

- Perioada semnalului este T=1/50=0.02 s. În intervalul `t = 0..0.1` se vor observa `0.1 / 0.02 = 5` cicluri complete.

- Dacă `t` ar fi fost coloană, `y` ar fi fost coloană; dimensiunea rămâne compatibilă cu `t`.

Linia 3. `plot(t,y); grid on;`

- `plot(t,y)` desenează graficul `y` în funcție de `t` într-o figură (figure window): axa orizontală este `t`, axa verticală este `y`.

- `grid on` afișează rețeaua de grid pe grafic (linii orizontale și verticale) pentru a ajuta citirea valorilor.

- Ambele instrucțiuni pot fi plasate pe aceeași linie si separate prin "`;`" (prima parte nu afișează nimic în Command Window datorită semnului `;`).

Linia 4. `xlabel('t [s]'); ylabel('v(t)'); title('Semnal sinus 50 Hz');`

- `xlabel('t [s]')` eticheta pe axa x: t[s] — indică faptul că `t` se măsoară în secunde.

- `ylabel('v(t)')` plasează eticheta pe axa y: v(t) — denumirea semnalului.

- `title('Semnal sinus 50 Hz')` afișează titlul graficului.

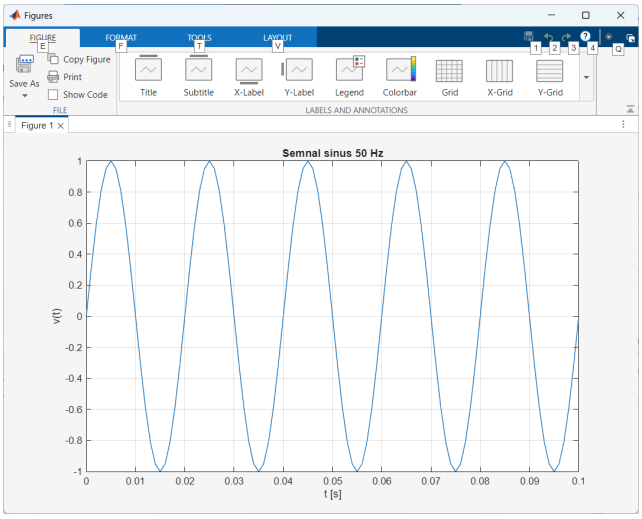


Figura 2.1 Fereastra generată in MATLAB dupa execuția scriptului grafic.m

2.3 Generarea vectorilor cu pas liniar

Pentru generarea vectorilor cu pas liniar se poate utiliza una din cele două metode de mai jos.

Metoda 1 - creăm un vector in intervalul [amin, amax] cu incrementul pas.

```
x = amin : pas : amax;
% exemplu MATLAB
x = -10:1:10; % generare vector de la -10 la 10 cu pas de 1
```

Metoda 2 - generăm un vector de **N valori** in intervalul [amin, amax], unde toate valorile sunt la **distanțe egale** (spațiere liniară).

```
x = linspace(amin, amax, N);
```

```
% exemplu MATLAB:
x = linspace(-10,10,20);
```

2.4 Matrice speciale

- **Matricea goală**: Pentru creșterea vitezei de lucru a anumitor secvențe de program se recomandă crearea unei matrice care nu conține niciun element (dar există in spațiul de variabile).

`X = []` (matrice de dimensiune 0×0)

- **Matricea unitate**: Reprezinta matricea cu toate elementele egale cu 1 si se apelează folosind una din sintaxele următoare:

```
U=ones(n)    % matrice patrata de dimeniune nxn
U=ones(m,n)  % va rezulta o matrice de dimensiune mxn, m
linii, n coloane
```

- **Matricea nulă**: Reprezinta matricea cu toate elementele egale cu 0 și se apelează cu una din sintaxele:

```
O=zeros(n)   % matrice pătrată de dimeniune nxn
O=zeros(m,n) % matrice de dimensiune mxn
```

- **Matricea identitate**: este matricea pătratică în care toate elementele de pe diagonala principală sunt egale cu 1, iar toate celelalte elemente sunt egale cu 0. Se poate apela cu urmatoarele sintaxe:

```
I=eye(n)     % matrice pătrată de dimensiune nxn
I=eye(m,n)   % matrice de dimensiune mxn
```

- **Matricea aleatoare:** Reprezintă matricea cu numere aleatoare care se pot obține utilizând sintaxele:

```
Ru=rand(n)   % matrice pătrată de dimensiune nxn
Ru=rand(m,n) % matrice de dimensiune mxn
```

2.5 Operatori aritmetici

MATLAB respectă ordinea operatorilor folosind aceleași reguli matematice (de exemplu paranteze → putere → înmulțire/împărțire → adunare/scădere).

adunare	+
scădere	-
înmulțire matricială	*
înmulțire element cu element	.*
împărțire la dreapta	/
împărtire la stânga	.\
ridicare la putere	^
ridicare la putere element cu element	.^

Exemplu 1:

```
x = [1 2 3];
y = x .* 2;   % [2 4 6]
```

Exemplu 2:

```
X = [1 2; 3 4];
Y = [5 6; 7 8];
Z1 = X*Y    % produs matriceal
Z2 = X.*Y   % produs element cu element
```

```
Va rezulta:
Z2 =
     5    12
    21    32
```

2.6 Operatori relaționali și logici

Operatorii relaționali sunt: < (mai mic), <= (mai mic sau egal), > (mai mare), >= (mai mare sau egal), == (identic), ~= (diferit).

Forma generală de utilizare a operatorilor operaționali este următoarea:

```
rezultat = expresie_1 op_rel expresie_2
```

unde:

`rezultat` – este 0 sau 1, reprezentând rezultatul comparației;

`expresie_1` și `expresie_2` –expresiile supuse comparației.

`op_rel` – unul dintre operatorii descriși anterior.

Rezultatul va fi egal cu 1 daca relația este adevarata si 0 in caz contrar.

Operatorii logici sunt: & (și), | (sau), ~ (negare)

Operatorii & și | se aplică asupra a doi operanzi (scalari sau matrice de dimensiuni egale). Aceștia efectuează operații logice sau pe biți. Rezultatul este 1 (ADEVĂRAT) dacă expresia este adevărată și 0 (FALS) în caz contrar. În evaluare, orice valoare diferită de 0 este considerată ADEVĂRAT, iar valoarea 0 este considerată FALS..

Exemplu:

```
a = 4; b = 3; c = 5;
r = (a>b & b<c) % 1 (adevărat)
```

În codul de mai sus `a > b` verifică dacă 4 > 3 → **adevărat (1)**; `b < c` verifică dacă 3 < 5 → **adevărat (1)**, iar operatorul "`&`" (AND logic) evaluează intreaga paranteză ca **1** doar dacă ambele expresii sunt adevărate: `1 & 1 = 1`, iar `r` devine **1**.

2.7 Probleme propuse

Problema 1

Creați o matrice de dimensiune 3×3 cu valori aleatorii între 0 și 1. Determinați suma fiecărei linii și reprezentați grafic valorile obținute.

Rezolvare propusă:

```
A = rand(3,3);
sume = sum(A,2);        % sumă pe linii
bar(sume)               % grafic tip bară
title('Suma pe linii a matricei A')
```

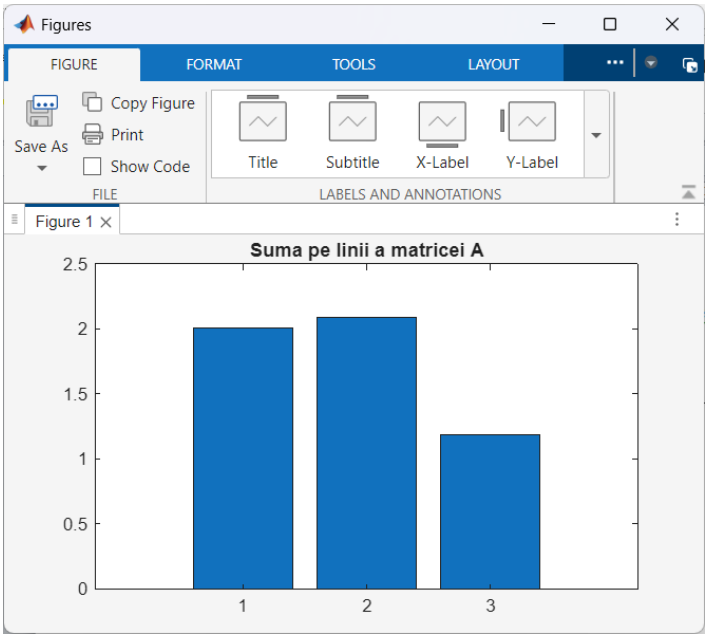


Figura 2.2 Fereastra generată in MATLAB dupa execuția Problemei 1.

Problema 2

Definiți o matrice aleatorie de dimensiune 4×4 și calculați media elementelor fiecărei coloane.

Rezolvare propusă:

```
% Problema2.m
% Creează o matrice 4x4 cu valori aleatorii în intervalul
(0,1)
A = rand(4,4);
% Calculează media pe coloane (dim=1 => operăm pe fiecare
coloană)
medii_col = mean(A, 1);
% Afișare
disp('Matricea A (4x4):')
disp(A)
fprintf('Mediile pe coloane sunt:\n')
fprintf('  %g  %g  %g  %g\n', medii_col)
```

In codul de mai sus `rand(4,4)` generează o matrice 4×4 cu elemente distribuite uniform în intervalul (0,1); `mean(A,1)` calculează media pentru fiecare coloană; rezultatul este un vector 1×4, iar `fprintf` afișează mediile într-o linie, formatate numeric.

După execuția programului obținem:

```
>> probleme
Matricea A (4x4):
    0.7431    0.7060    0.0971    0.9502
    0.3922    0.0318    0.8235    0.0344
    0.6555    0.2769    0.6948    0.4387
    0.1712    0.0462    0.3171    0.3816

Mediile pe coloane sunt:
  0.490506  0.265243  0.483129  0.451243
```

Problema 3

Generați un vector *x* de la -5 la 5 cu pas de 0.5 și calculați *y* = sin(x).*exp(-x.^2). Reprezentați grafic *y* în funcție de *x*.

Rezolvare propusă:

```
% Problema2.m
% Definire vector x
x = -5:0.5:5;    % de la -5 la 5 cu pas 0.5

% Calcul vector y
y = sin(x).* exp(-x.^2);   % operator .* și .^ pentru
operații element cu element
% Afișare valori (opțional)
disp('x = '); disp(x)
disp('y = '); disp(y)
% Reprezentare grafică
figure;                % deschide o fereastră grafică nouă
plot(x, y, '-o')       % linie cu marcaj
grid on
xlabel('x')
ylabel('y = sin(x) e^{-x^2}')
title('Grafic: y = sin(x) \cdot e^{-x^2}')
```

Graficul rezultat este următorul:

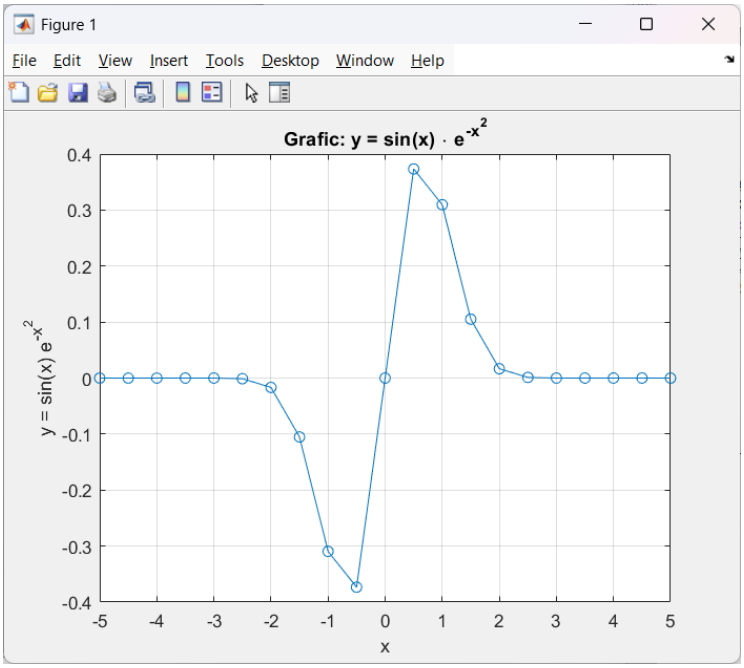


Figura 2.3 Fereastra generată in MATLAB dupa execuția Problemei 3

În programul de mai sus `sin(x)` și `exp(-x.^2)` produc vectori de aceeași dimensiune; folosim `.*` și `.^` pentru operații element cu element. Funcția `plot(x,y,'-o')` va afișa forma de undă atenuată de factorul `exp(-x^2)`: amplitudinea scade rapid pentru valori |x| mari. Graficul va avea o formă sinusoidală centrată în jurul lui 0, dar cu amplitudine foarte mică la marginile [-5,5] datorită factorului gaussian `exp(-x^2)`.

Problema 4

Scrieți un fișier script in care se definesc două matrice compatibile si efectuați înmulțirea acestora. Calculați produsul matriceal și produsul element cu element si afișați rezultatele cu funcțiile *disp* și *fprintf*.

Rezolvare propusă:

```
% Problema3.m
% Partea A: matrice compatibile pentru inmultire (2x3 * 3x2)
A = [1 2 3; 4 5 6];    % matrice 2x3
B = [7 8; 9 10; 11 12]; % matrice 3x2
% Produs matriceal A*B -> dimensiune matrice produs 2x2
Prod_mat = A * B;
disp('Partea A:');
disp('Matrice A (2x3):'); disp(A)
disp('Matrice B (3x2):'); disp(B)
disp('Produsul matriceal A * B (2x2):')
disp(Prod_mat)
fprintf('Element (1,1) al produsului = %g\n', Prod_mat(1,1));

% Partea B: matrice de aceeași dimensiune pentru a demonstra
* vs .*
C = [1 2; 3 4];
D = [5 6; 7 8];
% Produs matriceal
Prod_C_D = C * D;       % inmultire matriceala (algebrica)
% Produs element cu element
ElemProd = C .* D;      % inmultire element-wise
disp('Partea B:');
disp('Matrice C:'); disp(C)
```

```
disp('Matrice D:'); disp(D)
disp('Produsul matricial C * D:')
disp(Prod_C_D)
disp('Produsul element cu element C .* D:')
disp(ElemProd)
% Afișare formatată cu fprintf
fprintf('C * D = [ %g  %g ; %g  %g ]\n', Prod_C_D(1,1),
Prod_C_D(1,2), Prod_C_D(2,1), Prod_C_D(2,2));
fprintf('C .* D = [ %g  %g ; %g  %g ]\n', ElemProd(1,1),
ElemProd(1,2), ElemProd(2,1), ElemProd(2,2));
```

Dupa execuția programului obținem:

```
C * D = [ 19  22 ; 43  50 ]
C .* D = [ 5  12 ; 21  32 ]
```

Capitolul 3

Structuri de control și vectorizare în MATLAB

3.1 Scopul capitolului

Acest capitol prezintă instrucțiunile de control din cadrul fluxului unui program MATLAB (if, else, elseif, for, while and switch), funcțiile logice, precum și conceptul de vectorizare – modalitate de optimizare a calculelor prin înlocuirea buclelor cu operații pe vectori/matrice.

3.2 Instrucțiuni de control logic

if	Instrucțiune pentru execuția condiționată
else	Instrucțiune asociată cu *if*
elseif	Instrucțiune asociată cu *if*
for	Instrucțiune utilizată pentru crearea ciclurilor cu un număr specificat de pași
while	Instrucțiune pentru cicluri cu condiție logică
break	Instrucțiune pentru terminarea forțata într-un ciclu
return	Încheie execuția funcției și returnează controlul.
error	Instrucțiune pentru afișare mesaj de eroare
end	Instrucțiune pentru finalizare cicluri *for*, *while* și *if*

3.2.1 Instructiunile if, else, elseif

Instrucțiunile *if* folosesc operatori relaționali sau logici pentru a determina ce pași trebuie efectuați în rezolvarea unei probleme.

```
if expresie_logica
    instructiuni1
```

```
else
    instructiuni2
end
```

Dacă rezultatul `expresie_logica` este adevărat, se execută instrucțiunile din interiorul blocului `if`.

Clauza `else` execută instrucțiuni dacă expresia logică este falsă.

Exemplu:

```
if a<0
    disp('Număr negativ');
else
    disp('Număr pozitiv sau zero');
end
```

Operatorii relaționali și operatorii logici din MATLAB sunt prezentați mai jos:

Operator relațional	**Semnificație**
<	mai mic decât
<=	mai mic sau egal
>	mai mare decât
>=	mai mare sau egal
==	egal
~=	diferit (nu este egal)

Operator logic	Semnificație
&	and
|	or
~	not

În MATLAB există mai multe variații ale instrucțiunii *if*, care permit luarea deciziei în funcție de condiții diferite [4].

Tip	Semnificație	Exemplu MATLAB
if	Execută codul doar dacă condiția este adevărată	`if a>b disp('a > b'); end`
nested if	*if* în interiorul altui *if*, pentru condiții mai complexe	`if a>0 if b>0 disp('a și b pozitive'); end end`
if-else	Execută un bloc de instrucțiuni dacă condiția este adevărată, blocul după else dacă condiția este falsă	`if a>b disp ('a > b'); else disp('a <= b'); end`

Forma generală a unei instrucțiuni simple *if:*

```
if expresie_logica 1
    instructiuni 1
end
```

Forma generala a unei instrucțiuni nested *if* este:

```
if expresie_logica 1
    instructiuni 1
if expresie_logica 2
    instructiuni 2
end
instructiuni 3
end
instructiuni 4
```

Forma generală a unei instrucțiuni *if…else* este:

```
if expresie_logica 1
    instructiuni_1
else
    instructiuni_2
end
```

Instrucțiunea *if-elseif* poate fi folosită pentru testarea mai multor condiții succesive înainte de a executa un set de instrucțiuni.

Forma generală a instrucțiunii *if-elseif else*:

```
if expresie_logica_1
        instructiuni_1
    elseif expresie_logica_2
        instructiuni_2
    elseif expresie_logica_3
        instructiuni_3
    elseif expresie_logica_4
        instructiuni_4
end
```

Instrucțiunea *if-elseif-else* constă dintr-un grup de instrucțiuni care se execută dacă celelalte expresii logice sunt false. Forma generală a instrucțiunii *if-elseif-else* este:

```
if expresie_logica1
    instructiuni1
    elseif expresie_logica2
        instructiuni2
    elseif expresie_logica3
        instructiuni3
    elseif expresie_logica4
        instructiuni4
end
```

Problema rezolvată – aplicație instrucțiunea *if-elseif-else*

Un convertor A/D pe 3 biți, cu intrare analogică x și ieșire digitală y, este descris de următoarele relații:

y	**Interval x**
0	$x < -3.0$
1	$-3.0 \leq x < -2.0$
2	$-2.0 \leq x < -1.0$
3	$-1.0 \leq x < 0.0$
4	$0.0 \leq x < 1.0$
5	$1.0 \leq x < 2.0$
6	$2.0 \leq x < 3.0$
7	$x \geq 3.0$

Scrieti un program in MATLAB pentru a converti semnalul analogic x în semnal digital y. Testati programul folosind semnalul analogic cu amplitudinile: -2.5, 0.8 și 3.5.

Rezolvare propusă:

```
% Vector de test
x = [-2.5, 0.8, 3.5];
y = zeros(size(x)); % inițializare vector ieșire digitală

for i = 1:length(x)
    if x(i) < -3
        y(i) = 0;
    elseif x(i) < -2
        y(i) = 1;
    elseif x(i) < -1
        y(i) = 2;
    elseif x(i) < 0
        y(i) = 3;
    elseif x(i) < 1
        y(i) = 4;
    elseif x(i) < 2
        y(i) = 5;
    elseif x(i) < 3
        y(i) = 6;
    else
        y(i) = 7;
    end
end
```

```
% Afișare rezultate
disp('Intrare analogică x:');
disp(x);
disp('Ieșire digitală y:');
disp(y);
```

Dupa execuție obținem:

```
Intrare analogică x:
   -2.5000    0.8000    3.5000

Ieșire digitală y:
     1     4     7
```

3.2.2 Instrucțiunea *for*

Bucla *for* permite repetarea instrucțiunilor din corpul buclei, de un număr determinat de ori și are următoarea formă generală:

```
for idx=expr
    set_instrucțiuni;
end
```

unde:

`idx` – este e variabila cu rol de contor al buclei;

`expr` – este o matrice, un vector (cel mai des) sau un scalar;

`set_instructiuni` – orice set de instrucțiuni MATLAB;

Exemplu:

```
for k=1:5
    fprintf('Iterația %d\n',k);
end
```

In exemplul de mai sus `for k = 1:5` → bucla se execută 5 ori, unde variablia contor `k` ia valorile 1, 2, 3, 4, 5. Funcția `fprintf('Iterația %d\n', k)` → afișează textul „Iterația X”, unde X este valoarea curentă a lui `k`, iar `end` → încheie bucla.

Probleme rezolvate

Problema 1

Sa se calculeze și sa se afișeze grafic funcția:

$$f(x)=\begin{cases}2x+3, & daca \ -10\le x\le 2\\ 2x^2-1, & daca \ 2<x\le 20\end{cases}$$

Rezolvare propusă:

```
x=-10:20; %un vector cu pas de 1
f=[];

for k=1:length(x) %initial=1; pas=1; final=30
    if x(k) <= 2
        f(k) = 2 * x(k) + 3;
    else
        f(k)=2 * x(k)^2 - 1;
    end
end
plot(x,f);      %reprezentare grafică;
```

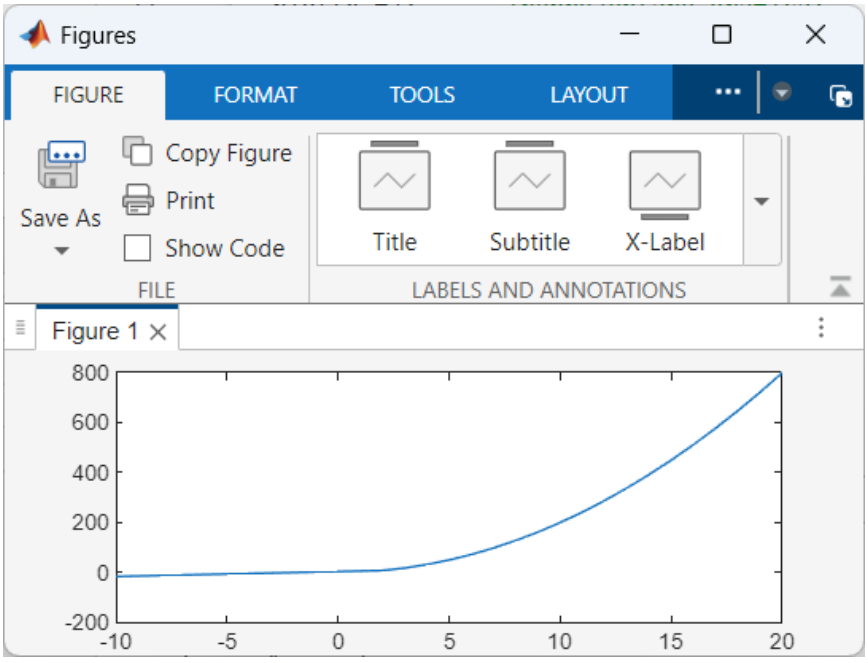


Figura 3.1 Fereastra generată in MATLAB dupa execuția Problemei 1 de mai sus

Problema 2

Calculați suma pătratelor primelor 100 de numere naturale:

Rezolvare propusă:

```
sum = 0;                % inițializăm suma
for i = 1:100           % bucla parcurge numerele de la 1 la 100
    sum = sum + i^2;    % adunăm pătratul fiecărui număr la valoarea variabilei sum
end
disp(sum);              % afișăm rezultatul
```

Problema 3

Afișați o matrice de 10x20 unde toate elementele au valoarea 1:

Rezolvare propusă:

```
n = 10;                 % numărul de rânduri
m = 20;                 % numărul de coloane
b = zeros(n,m);         % inițializăm matricea cu zerouri

for i = 1:n             % parcurgem fiecare rând
    for j = 1:m         % parcurgem fiecare coloană
        b(i,j) = 1;     % atribuim valoarea 1 fiecărui element
    end
end

disp(b);                % afișăm matricea completă
```

3.2.3 Bucla `while`

O buclă **`while`** permite repetarea unui set de instrucțiuni atât timp cât o condiție logică definită este adevărată. Structura generală a buclei *while* este:

```
while conditie_logica
    set_instructiuni;
end
```

Setul de instrucțiuni este executat atâta timp cât expresia logică `expresie` este adevărată.

În exemplul de mai jos codul începe prin inițializarea variabilei k cu valoarea 0. Bucla `while` verifică în mod repetat condiția k < 5, ceea ce înseamnă că instrucțiunile din interiorul buclei vor continua să fie executate atât timp cât k este mai mic decât 5. La fiecare iterație, k este incrementat cu 1 prin instrucțiunea k = k + 1, iar apoi este afișat pe ecran cu funcția disp(k). Bucla se oprește automat în momentul în care k ajunge la valoarea 5, deoarece condiția k < 5 nu mai este adevărată. Rezultatul afișat este succesiunea numerelor 1, 2, 3, 4 și 5.

```
k=0;
while k<5
    k=k+1;
    disp(k)
end
```

Problema 4

Determină numărul de numere întregi consecutive care, adunate, dau o valoare egală sau cât mai apropiată fără a depăși 160**.**

Rezolvare propusă:

```
sum = 0;    % inițializare sumă
n = 0;      % inițializare contor

while sum + (n+1) <= 160
    n = n + 1;
    sum = sum + n;
end
```

```
fprintf('Numărul de numere consecutive este %d\n', n);
fprintf('Suma acestora este %d\n', sum);
```

Dupa execute obținem:

Numărul de numere consecutive este 17

Suma acestora este 153

3.2.4 Instructiunea `switch`

Instrucțiunea **switch** în MATLAB se folosește atunci când utilizatorul are nevoie să aleaga între mai multe opțiuni posibile, în funcție de valoarea unei variabile sau expresii. Este utilă mai ales în situațiile în care avem mai multe condiții alternative și ar fi incomod sau neclar să folosim mai multe instrucțiuni if–elseif–else. Sintaxa instructiunii *switch* este urmatoarea:

```
switch expresie
    case test_expresie_1
        set_instructiuni_1;
    case {test_expresie_2, test_expresie_3, ... }
        set_instructiuni_2;
    case ....
        .
        .
        .
    otherwise
        set_instructiuni_n;
end
```

Instrucțiunea *switch* permite executarea diferitelor blocuri de cod în funcție de valoarea unei expresii. MATLAB compară valoarea expresiei cu fiecare `case`. Dacă găsește o potrivire, execută grupul de instrucțiuni corespunzător acelui `case`. Dacă expresia nu se potrivește cu niciun caz definit, se execută secțiunea `otherwise`. Instrucțiunea se încheie cu `end`.

In exemplul de mai jos se folosește instrucțiunea *switch* pentru a afișa un mesaj în funcție de valoarea variabilei zi. Dacă zi este 'luni', se afișează „Zi de început”. Dacă zi este 'sambata' sau 'duminica', se afișează „Weekend”. Pentru orice altă zi, instrucțiunea intră în secțiunea otherwise și afișează „Zi lucrătoare”. În cazul de față, zi este 'luni', deci programul afișează **„Zi de început”**.

```
zi='luni';
switch zi
    case 'luni'
        disp('Zi de început');
    case {'sambata','duminica'}
        disp('Weekend');
    otherwise
        disp('Zi lucrătoare');
end
```

3.3 Funcții de control logic

Funcțiile de control logic în MATLAB se folosesc pentru a lua decizii în program și pentru a controla fluxul execuției. Cu alte cuvinte, ele stabilesc ce instrucțiuni se execută în funcție de anumite condiții sau situații.

Funcție	Semnificație / Utilizare
exist	Verifică dacă o variabilă, fișier sau funcție există si sunt definite.
any	Verifică dacă **cel puțin un element** dintr-un vector/matrice este diferit de zero
all	Verifică dacă **toate elementele** dintr-un vector/matrice sunt diferite de zero (adevărat).
find	Returnează **indecșii elementelor** care îndeplinesc o anumită condiție.
isnan	Verifică dacă elementele sunt **NaN** (Not-a-Number).
isinf	Verifică dacă elementele sunt **infinit** (∞ sau -∞).
isfinite	Verifică dacă elementele sunt **finite** (nu sunt NaN sau Inf).

- `all(X<1)` returnează **adevărat (1)** dacă **toate elementele** din vectorul sau matricea X sunt **mai mici decât 1**. Dacă există măcar un element ≥ 1, rezultatul este **fals (0)**.

- `any(X>0)` returnează **adevărat (1)** dacă **cel puțin un element** din X este **mai mare decât 0**. Dacă niciun element nu este > 0, rezultatul este **fals (0)**.

- `exist('x')` – testează existența variabilei x.

3.4 Vectorizarea calculelor

În MATLAB, operațiile cu vectori și matrice sunt executate mult mai rapid decât operațiile interpretate (ciclurile for, while). De aceea, oriunde este posibil, se recomandă înlocuirea buclelor *for* și *while* cu operații vectorizate. De exemplu, programul care calculează funcția $\sin^2(x)$ de la 0 la 2π cu pasul $\pi/1000$, poate fi realizat în două moduri: utilizând un ciclu for (soluția 1) sau utilizând operații cu vectori (soluția 2).

```
%Solutia 1:înceată
%se foloseste bucla for
wt = 0:pi/1000:2*pi;    % creează un vector de valori de la 0
la 2π cu pasul π/1000
N = length(wt);          % determină numărul de elemente din
vector
for i = 1:N
  y(i) = sin(wt(i))^2; % calculează sin^2 pentru fiecare
element și îl stochează în y
end
plot(wt, y);           % afișează graficul funcției
```

Această metodă este mai lentă pentru că folosește o buclă, procesând element cu element. A doua solutie necesita mai putine linii de cod și are o execuție mai rapidă, mai ales pentru vectori mari.

```
%Solutia 2:recomandată
% (vectorizat)
wt = 0:pi/1000:2*pi;  % creează un vector de valori de la 0 la 2π cu pasul π/1000
y = sin(wt).^2;      % calculează sin^2 pentru toate elementele vectorului simultan
plot(wt, y);           % afișează graficul funcției
```

MATLAB este un limbaj optimizat pentru operații cu vectori/matrici. Bucla `for` poate fi adesea înlocuită cu o operație vectorială, care este mai rapidă. Pentru ca buclele `for` sunt mai lente, se recomandă folosirea operațiilor element-cu-element.

Observație: Operațiile de inmulțire, impartire si ridicare la putere cu vectori se realizeaza prin utilizarea operatorului punct plasat inaintea operatorului arithmetic. Utilizați .*, ./, .^ pentru a aplica operația element cu element.

3.5 Exemple rezolvate

Problema 3.5.1

Să se scrie un fisier script care citește o valoare a rezistenței *R* și decide dacă este în domeniul [100,200] Ω.

Rezolvare propusă:

```
R=input('Introduceți rezistența [Ω]: ');
if R>=100 && R<=200
    disp('Rezistența este în intervalul cerut.')
else
    disp('Rezistența este în afara intervalului.')
end
```

Problema 3.5.2

Calculați suma primelor n numere naturale folosing două metode:

Rezolvare propusă:

```
% Metoda 1: cu bucla for
n = 10;           % numărul de termeni (puteți schimba
valoarea)
suma = 0;         % inițializare sumă

for i = 1:n       % bucla parcurge toate numerele de la 1 la
n
    suma = suma + i;  % adaugă fiecare număr la sumă
end

fprintf('Suma primelor %d numere naturale este %d\n', n,
suma);
```

```
% Metoda 2: variantă vectorizată cu sum(1:n)
n = 10;       % numărul de termeni
s = sum(1:n); % calculează suma primelor n numere naturale
```

Problema 3.5.3

Calculați răspunsul $v(t) = e^{-t}sin(2\pi ft)$ pentru $t = 0 : 0.001 : 1, f = 50\ Hz$. Reprezentați grafic semnalul.

Rezolvare propusă:

```
%Vectorizare semnal
t = 0:0.001:1;                  % vector timp
f = 50;                         % frecvența în Hz
v = exp(-t) .* sin(2*pi*f*t);   % semnalul amortizat

% Reprezentare grafică
plot(t,v)
grid on
xlabel('t [s]')
ylabel('v(t)')
title('Semnal amortizat v(t)=e^{-t} sin(2π50t)')
```

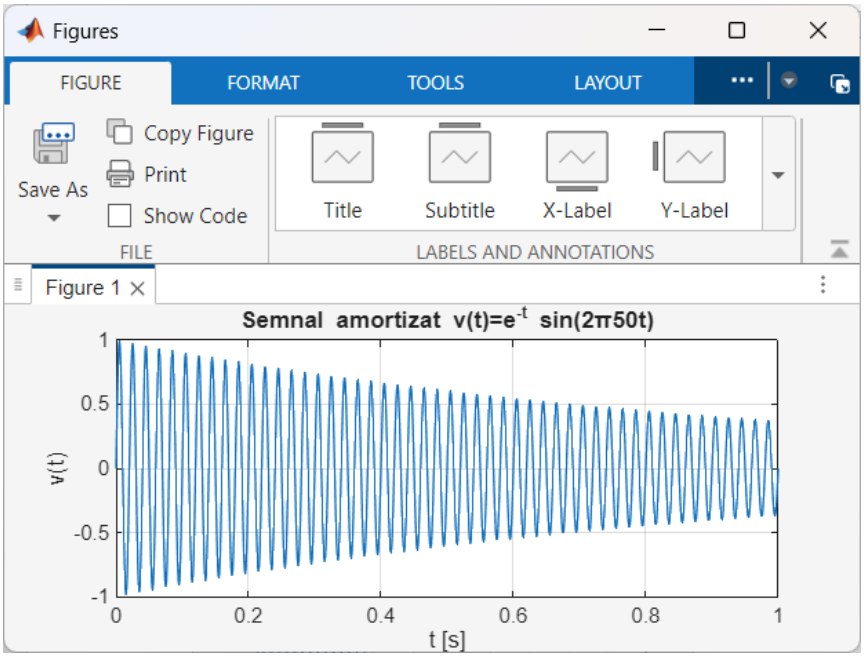


Figura 3.2 Fereastra generată in MATLAB dupa execuția problemei 3.5.3 de mai sus.

Problema 3.5.4

Scrieți un script care cere utilizatorului valoarea unei tensiuni U și decide dacă aceasta depășește tensiunea maximă admisă (250 V). Folosiți if/else pentru a lua o decizie. Observați tratarea cazului egal (==) și folosirea operatorilor logici.

Rezolvare propusă:

Pași și raționament

1. Citim valoarea folosind funcția *input*.
2. Comparăm U cu limita 250 V.
3. Afișăm mesaje diferite în funcție de rezultat.

4. Adăugăm verificări pentru intrări invalide.

```
% verifica_tensiune.m
% Verifică dacă tensiunea U depășește 250 V

U = input('Introduceți valoarea tensiunii U [V]: ');

% Verificare de bază (opțional)
if isempty(U) || ~isnumeric(U)
    error('Valoare invalidă. Introduceți un număr.');
end

if U > 250
    fprintf('Atenție: tensiunea U = %.2f V depășește limita
de 250 V.\n', U);
elseif U == 250
    fprintf('Tensiunea U = 250 V este egală cu limita maximă
admisă.\n');
else
    fprintf('Tensiunea U = %.2f V este în limite admise (<=
250 V).\n', U);
end
```

Problema 3.5.5

Calculați factorialul unui număr *n* atât cu o buclă *for*, cât și folosind o funcție *factorial (n)*.

Rezolvare propusă:

Pași și raționament

1. Citim un număr n întreg, n >= 0.
2. Calculăm factorialul iterativ (produs).
3. Calculăm folosind factorial(n) pentru a compara.
4. Afișăm rezultatele și tratăm cazurile invalide (n negativ sau non-integer).

```
% factorial_comparare.m
n = input('Introduceți un număr întreg n: ');
if ~isnumeric(n) || n<0 || floor(n)~=n
    error('n trebuie să fie un întreg pozitiv.');
end

% Variantă bucla for
fact_for = 1;
for k = 1:n
    fact_for = fact_for * k;
end

% Varianta built-in
fact_builtin = factorial(n);

fprintf('Factorialul calculat cu for: %g\n', fact_for);
fprintf('Factorialul calculat cu factorial(n): %g\n',
fact_builtin);
```

Problema 3.5.6

Scrieți un program care cere utilizatorului să introducă numere până când suma lor depășește 100; la final afișează suma și numărul de valori introduse.

Rezolvare propusă:

Pași și raționament

1. Inițializăm `suma = 0` și `count = 0`.
2. Folosim `while suma <= 100` pentru buclă.
3. În interior citim un număr, validăm, adăugăm la sumă și incrementăm contorul.
4. La ieșire afișăm rezultatele.

```
% suma_pana_100.m
suma = 0;
count = 0;

fprintf('Introduceți numere. Programul se oprește când suma depășește 100.\n');

while suma <= 100
    x = input('Introduceți un număr (sau Ctrl+C pentru a întrerupe): ');
    if isempty(x) || ~isnumeric(x)
        disp('Valoare invalidă. Introduceți un număr.');
        continue;
    end
```

```
    suma = suma + x;
    count = count + 1;
    fprintf('După %d numere, suma este = %.4f\n', count,
suma);
end

fprintf('S-a ajuns la suma = %.4f după %d valori.\n', suma,
count);
```

Problema 3.5.7

Folosind vectorizarea calculelor, calculați puterea instantanee $p(t) = u(t) * i(t)$ pentru semnalele $u(t) = 230\sqrt{2}\ sin(2\pi 50t)$, $i(t) = 10\sqrt{2}\ sin(2\pi 50t - \pi/4)$ pe intervalul $t \in [0, 0.04]$ s. Reprezentați grafic p(t).

Rezolvare propusă:

```
% putere_instantanee.m

% Parametri
f = 50;                     % frecvență [Hz]
t = 0:1e-4:0.04;            % vector timp, pas 0.0001 s
(rezoluție bună)
U_amp = 230*sqrt(2);        % amplitudinea tensiunii
I_amp = 10*sqrt(2);         % amplitudinea curentului
phi = -pi/4;                % decalaj faza curentului

% Semnale (vectorizat)
```

```
u = U_amp .* sin(2*pi*f.*t);
i = I_amp .* sin(2*pi*f.*t + phi);

% Puterea instantanee
p = u .* i;

% Reprezentare grafică
figure;
plot(t,p);
grid on;
xlabel('t [s]');
ylabel('p(t) [W]');
title('Puterea instantanee p(t) = u(t) \cdot i(t)');
% opțional: limita axelor pentru claritate
xlim([0, 0.04]);
```

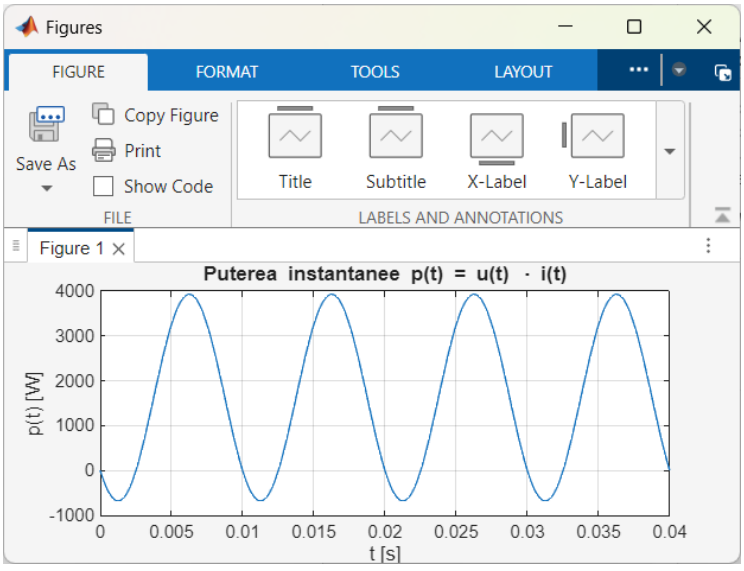


Figura 3.3 Fereastra generată in MATLAB dupa execuția problemei 3.5.7 de mai sus.

Capitolul 4

Numere Complexe in MATLAB

Un număr complex z poate fi scris sub forma algebrică $z = x + iy$, unde x reprezintă partea reală, iar y este partea imaginară a numărului complex. Sub formă trigonometrică, numărul z se exprimă prin modul si argument și fi scris astfel:

$$z = r\,(\cos\varphi + i\sin\varphi)$$

unde modulul se calculează folosind formula: $r=|z|=\sqrt{(x^2+y^2)}$, iar argumentul (faza, unghiul) $\varphi = \arg(z) = \arctan(y/x)$.

Forma exponențială (polară) se scrie folosind formula lui Euler ($e^{i\theta}=cos(\theta)+i*sin(\theta)$) astfel: $z = re^{i\varphi}$.

4.1 Funcții in MATLAB folosite cu numere complexe

Funcție	Descriere	Exemplu
`abs(z)`	Calculează **modulul** lui `z`	
`angle(z)`	Calculează **argumentul** lui `z` (unghiul în radiani)	`angle(1+i)` → `0.7854` (≈ π/4)
`conj(z)`	Returnează **conjugatul** lui `z` ((\bar{z} = x - iy))	`conj(3+4i)` → `3-4i`

Funcție	Descriere	Exemplu
`real(z)`	Returnează **partea reală** a lui `z`	`real(3+4i)` ⟶ `3`
`imag(z)`	Returnează **partea imaginară** a lui `z`	`imag(3+4i)` ⟶ `4`
`unwrap(theta)`	Corectează discontinuitățile de 2π din vectorul de unghiuri (utile pentru grafice sau FFT)	`theta = [0 pi -pi]; unwrap(theta)` ⟶ `[0 pi pi]`

Crearea numerelor complexe in MATLAB poate fi facută in mai multe moduri, de exemplu:

```
z1=1-2i;
z1=1-2*i;

z1=1-2j;
z1=1-2*j;
```

unde $i=j=\sqrt{-1}$ (in cazul in care *i* sau *j* nu au fost utilizate pentru alte variabile);

sau:

```
z1=complex(1,-2);
```

Extragerea modulului unui număr complex, *z*:

```
r=abs(z);
```

Extragerea argumentului (in radiani) unui număr complex, *z*:

```
fi=angle(z);
```

Extragerea părții reale a unui număr complex, *z*:

```
x=real(z);
```

Extragerea părții imaginare a unui număr complex, *z*:

```
y=imag(z);
```

Determinarea conjugatului unui număr complex, *z*:

```
z_conj=conj(z);
```

Operațiile matematice cu numere complexe se scriu la fel ca in cazul numerelor reale: `4=z1/z2-z2*sqrt(z3);`

De asemenea, definirea matricilor cu numere complexe se realizeaza la fel ca in cazul numerelor reale. Definirea matricei:

$$M = \begin{bmatrix} 2-3i & i \\ -2+1.5i & 6 \end{bmatrix}$$

se face astfel:

```
M=[2-3i i;-2+1.5i 6];
```

Observație: Se recomandă scrierea părții imaginare fără semnul de inmultire (*) intre număr si *i* sau *j*. Ex. 1.5*i -> 1.5i.

Adunare/scădere: se adună separat părțile reale și imaginare.

Înmulțire/împărțire: înmulțire trigonometrică sau distributivă în forma algebrică.

Conjugat: `conj(z)` $\bar{z}=x-iy$

Modul: `abs(z)`.

Argument (fază): `angle(z)`.

Funcții uzuale in lucrul cu numere complexe:

`real(z)` → partea reală.

`imag(z)` → partea imaginară.

`abs(z)` → modulul.

`angle(z)` → argumentul în radiani.

`conj(z)` → conjugatul.

`complex(a,b)`

`plot(real(z),imag(z),'o')` → reprezentare grafică.

Crearea numerelor complexe in MATLAB poate fi scrisă in mai multe moduri, de exemplu:

```
z1=1-2i;
z1=1-2*i;
z1=1-2j;
z1=1-2*j;
```

unde $i=j=\sqrt{-1}$ (in cazul in care i sau j nu au fost utilizate pentru alte variabile);

sau:

```
z1=complex(1,-2);
```

Problema 4.1.1

Scrieți un program ce convertește automat un număr complex introdus de utilizator în formă exponențială.

Rezolvare propusă:

```
% Număr complex

z = 1 + 1i*sqrt(3);

% Modul și fază
r = abs(z);      % modulul |z|
phi = angle(z);  % faza (argumentul) în radiani

% Afișare rezultate
fprintf('Forma carteziană: z = %.3f + i*%.3f\n', real(z),
imag(z));
fprintf('Forma exponențială: z = %.3f * exp(i*%.3f)\n', r,
phi);
```

Dupa execuție obținem:

```
Forma carteziană: z = 1.000 + i*1.732
Forma exponențială: z = 2.000 * exp(i*1.047)
```

Problema 4.1.2

Scrieți un program ce efectuează suma, produsul si raportul a două numere complexe.

Rezolvare propusă:

```
z1 = 3 + 4i;
z2 = 2 - 5i;

suma = z1 + z2
produs = z1 * z2
raport = z1 / z2

fprintf('z1 = %.2f%+.2fi\n', real(z1), imag(z1));
fprintf('z2 = %.2f%+.2fi\n', real(z2), imag(z2));
fprintf('suma = %.2f%+.2fi\n', real(suma), imag(suma));
fprintf('produs = %.2f%+.2fi\n', real(produs), imag(produs));
fprintf('raport = %.2f%+.2fi\n', real(raport), imag(raport));
```

Dupa execuție obținem:

```
suma =

   5.0000 - 1.0000i


produs =

  26.0000 - 7.0000i


raport =

  -0.4828 + 0.7931i
```

```
z1 = 3.00+4.00i
z2 = 2.00-5.00i
suma = 5.00-1.00i
produs = 26.00-7.00i
raport = -0.48+0.79i
```

Problema 4.1.3

Scrieți un program ce calculează modulul, argumentul si forma exponențială a unui număr complex.

Rezolvare propusă:

```
clear;clc
z = -3 + 4i;
r = abs(z);                    % modul
phi = angle(z);                % argument (radiani)
z_exp = r * exp(1i*phi);       % forma exponențială

fprintf('r=%.4f, phi=%.4f rad\n', r, phi);
disp(z_exp)                    % ar trebui să fie egal cu z
```

Dupa execuție obținem:

```
r=5.0000, phi=2.2143 rad
  -3.0000 + 4.0000i
```

Problema 4.1.4

Scrieți un program pentru reprezentarea grafică în plan complex a vectorului z.

```
z = [1+2i, 3+4i, -2+5i, -3-1i];

plot(real(z), imag(z),
'o','MarkerSize',8,'MarkerFaceColor','r');
grid on; xlabel('Re(z)'); ylabel('Im(z)');
title('Numere complexe pe planul complex');
```

Dupa execuție obținem:

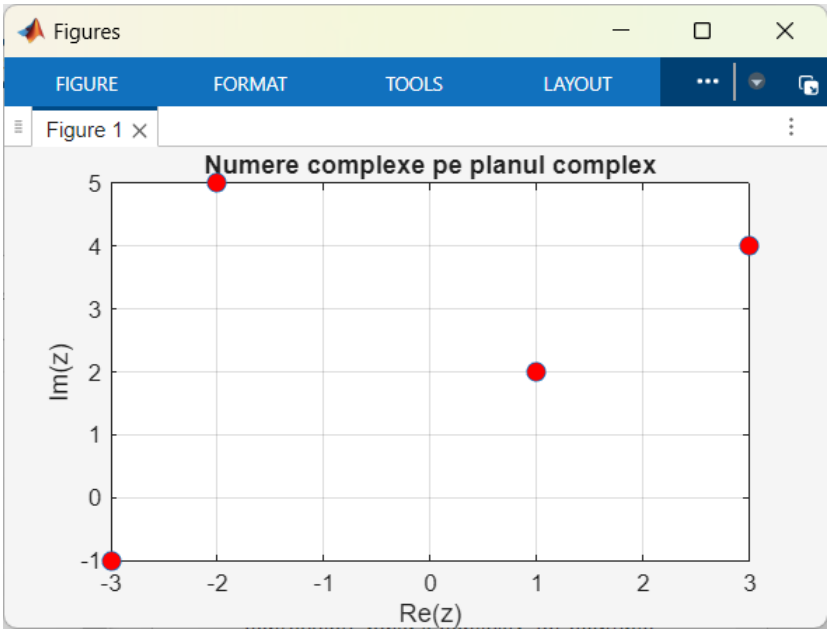


Figura 4.1 Fereastra generată in MATLAB dupa execuția problemei 4.1.4 de mai sus.

Problema 4.1.5

Fie z = −2 + 7i. Calculați modulul, argumentul (în radiani și grade) și forma exponențială.

Rezolvare propusă:

```
clear;clc
z = -2 + 7i;
r = abs(z);
phi = angle(z);
phi_deg = rad2deg(phi);
z_exp = r*exp(1i*phi);

fprintf('|z|=%.3f, arg(z)=%.3f rad (%.2f°)\n', r, phi,
phi_deg);
disp('Forma exponențială:'); disp(z_exp);
```

Dupa execuție obținem:

```
|z|=7.280, arg(z)=1.849 rad (105.95°)
Forma exponențială:
  -2.0000 + 7.0000i
```

Problema 4.1.6

Generați un vector de 10 numere complexe aleatoare cu partea reală și imaginară în intervalul [-5, 5]. Reprezentați-le în planul complex și adăugați linii de la origine.

Rezolvare propusă:

```
clear;clc
re = -5 + 10*rand(1,10);
im = -5 + 10*rand(1,10);
z = complex(re,im);
```

```
figure; hold on;
plot(real(z), imag(z), 'o','MarkerFaceColor','b');
for k=1:numel(z)
    plot([0 real(z(k))],[0 imag(z(k))],'k:');
end
grid on; axis equal;
xlabel('Re(z)'); ylabel('Im(z)');
title('Numere complexe random pe planul complex');
```

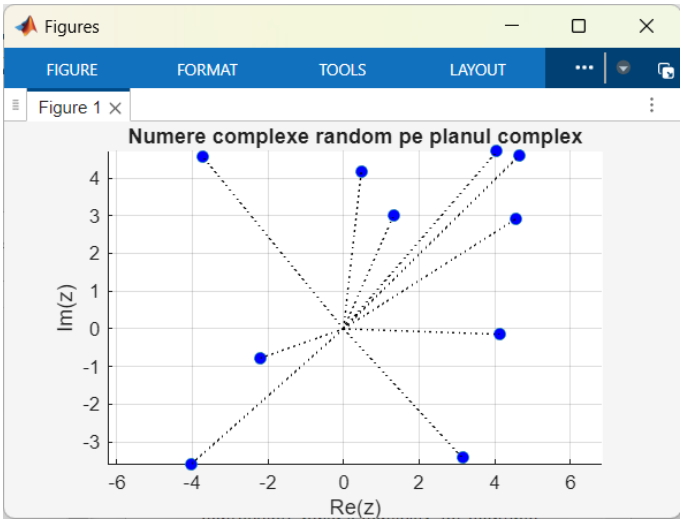


Figura 4.2 Fereastra generată in MATLAB dupa execuția problemei 4.1.6 de mai sus.

Problema 4.1.7

Fie un circuit serie R=10Ω, L=50mH, C=100μF alimentat cu o tensiune sinusoidală V=230∠0° la frecvența 50Hz. Calculați impedanța totală și curentul complex.

Rezolvare propusă:

```
R=10; L=50e-3; C=100e-6; f=50; w=2*pi*f;

Z_R = R;
Z_L = 1i*w*L;
Z_C = 1/(1i*w*C);
Z_total = Z_R + Z_L + Z_C;

V = 230*exp(1i*0);      % tensiune complexă
I = V/Z_total;          % curent complex
fprintf('Z_total=%.2f%+.2fi Ohm\n', real(Z_total),
imag(Z_total));
fprintf('I=%.2f%+.2fi A\n', real(I), imag(I));
fprintf('|I|=%.2f A, phi=%.2f°\n', abs(I),
rad2deg(angle(I)));
```

Dupa execuție obținem:

Z_total=10.00-16.12i Ohm
I=6.39+10.30i A
|I|=12.12 A, phi=58.19°

Problema 4.1.8

Fie z1=5∠30° si z2=3∠−45°. Calculați suma și produsul în MATLAB.

Rezolvare propusă:

```
clear;clc
```

```
z1 = 5*exp(1i*deg2rad(30));
z2 = 3*exp(1i*deg2rad(-45));

suma = z1 + z2;
produs = z1 * z2;

fprintf('Suma=%.2f%+.2fi\n', real(suma), imag(suma));
fprintf('Produs=%.2f%+.2fi\n', real(produs), imag(produs));

fprintf('Produs=%.2f%+.2fi\n', real(produs), imag(produs));
```

Dupa execuție obținem:

Suma=6.45+0.38i
Produs=14.49-3.88i

Observații

- Funcțiile `abs`, `angle`, `real`, `imag` sunt vectorizate. Se pot aplica direct pe vectori/matrici complexe.
- Pentru grafice se pot folosi functiile `compass(z)` sau `quiver(real(z),imag(z))`.
- Numerele complexe sunt fundamentale în analiza circuitelor AC și a semnalelor sinusoidale defazate.

Capitolul 5

Calcul numeric în MATLAB: operații numerice cu matrice

MATLAB are ca unitate fundamentală de lucru matricea și, ca urmare, a fost gândit pentru facilitarea calculelor matriceale. Operațiile numerice standard (adunare, scădere, înmulțire, împărțire, exponențiere) pot fi aplicate pe matrice fie în sens algebric (produs matriceal), fie element cu element (folosind operatorul punct astfel: `.*`, `./`, `.^`).

În MATLAB, calculele aritmetice asupra tablourilor de date pot fi realizate astfel:
– operații conform regulilor calculului matriceal – operații cu matrice;
– operații conform regulilor calculului scalar (element cu element) – operații cu tablouri.

Operație	**Matricial (algebric)**	**Element cu element**
Adunare	`A + B`	—
Scădere	`A - B`	—
Înmulțire	`A * B`	`A .* B`
Împărțire dreapta	`A / B`	`A ./ B`
Împărțire stânga	`A \ B`	`A .\ B`

Operație	Matricial (algebric)	Element cu element
Ridicarea la putere	`A ^ B`	`A .^ B`
Schimbarea semnului	`-A`	—
Adunare unară	`+A`	—
Transpunerea	`A'`	`A.'`

Observații:

- Operatorii fără punct efectuează operații matriciale (conform regulilor algebrei liniare).

- Operatorii cu punct efectuează operații element cu element, adică acționează independent asupra fiecărei poziții din matrice.

- Pentru operațiile element-cu-element matricele trebuie să aibă aceleași dimensiuni.

5.1 Adunarea și scăderea matricilor

Fie **X** și **Y** sunt două matrice de dimensiuni egale (in afara cazului in care **X** sau **Y** este un scalar). Adunarea și scăderea sunt definite astfel:

Z1=X+Y;

Z2=X-Y;

Operațiile de adunare și scădere se realizează element cu element. În cazul în care unul dintre operanzi este scalar, iar celălalt este o matrice, operația se aplică între scalar și fiecare element al matricei.

5.2 Înmulțirea matricilor

Definim **X** și **Y** două matrice și rezultă operatia **Z = X * Y** ca matricea produs, cu elementele egale cu:

$$Z(i,j) = \sum_k X(i,k)Y(k,j)$$

Produsul este condiționat de egalitatea dintre numărul coloanelor matricei **X** și numărul liniilor matricei **Y**. Matricea produs **Z** va avea dimensiunea *număr_linii_X* × *număr_coloane_Y*.

O variabilă scalară poate fi înmulțită cu orice matrice, realizându-se înmulțirea cu fiecare element al matricei.

Observație! Inmultirea matricilor nu este intotdeauna comutativă, X*Y ≠ Y*X.

5.3 Împărțirea matricilor

Dacă **X** și **Y** sunt două matrice, atunci au loc următoarele:

Z_1 = X / Y reprezintă împărțirea la dreapta a matricilor **X** și **Y**, operația de mai sus fiind identică cu:

Z_1 = X * Y^{-1}, unde **Y^{-1}** este inversa matricii **Y**.

Z_2 = X \ Y reprezintă operația de împărțire la stânga a matricilor **X** și **Y**, operația de mai sus fiind identică cu:

Z_2 = X^{-1} * Y, unde **X^{-1}** este inversa matricii **X**.

Exemplu:

```
%Sa se calculeze X/Y si X\Y;
clear;clc
X=[1 2;3 4];
Y=[5 6;7 8];
```

```
c=2;
Z1=X/Y;
Z2=X\Y;
```

5.4 Ridicarea la putere

O matrice *Z = X^p* reprezintă ridicarea la puterea **p** a matricii **X**, unde **X** este o matrice pătrată, iar **p** este un scalar, pozitiv sau negativ.

Exemplu:
Fie matricea **X** și **p = 2**. Să se calculeze X^p.

```
X=[1 2;3 4];
p=2;
Z=X^p
```

După execuție vom obține:

```
Z =

     7    10
    15    22
```

Ordinea operațiilor in MATLAB este aceeași ca în aritmetica standard, respectiv: parantezele, ridicarea la putere, inmulțirea si impartirea, adunarea si scăderea.

5.5 Transpunerea matricilor

Fie **Z = X'**: dacă **X** este de dimensiune $m \times n$, în urma acestei operații rezultă matricea **Z** de dimensiune $n \times m$. Dacă elementele matricii **X** sunt numere reale, atunci operația de transpunere constă în:

$$Z(i,j) = X(j,i);$$

iar dacă matricea **X** conține elemente complexe, atunci operația de transpunere constă în:

$$Z(i,j) = \text{conj}(X(j,i));$$

5.6 Operații aritmetice cu tablouri

Operațiile cu tablouri sunt operații aritmetice efectuate între elementele aflate pe aceeași poziție în tablouri, cunoscute sub numele de **operații element cu element**. Operațiile de adunare, scădere, înmulțire, împărțire a două tablouri sunt posibile doar dacă cele două tablouri au aceleași dimensiuni.

Pentru a preciza că operația aritmetică între două tablouri se realizează element cu element, se utilizează operatorul aritmetic precedat de punct, astfel:

```
Z1=X.*Y;
Z2=X./Y;
Z3=X.^4.
```

Operațiile de adunare și scădere **nu necesită** utilizarea punctului.

Operația de înmulțire a tablourilor este **comutativă**, adică X.*Y=Y.*X.

Atenție!

Operațiile X * Y și X .* Y vor furniza rezultate diferite!

5.7 Probleme rezolvate

Problema 5.7.1

Implementați un script numit `rms_calc.m` care: citește de la tastatură 5 valori ale unui semnal `v1, v2,…, v5`, calculează valoarea RMS și afișează rezultatul folosind `fprintf` formatat la 3 zecimale. Se vor folosi funcții de intrare/ieșire, vectorizare și afișare.

Rezolvare propusă:

```
% rms_calc.m
% Citire valori
vals = zeros(1,5);
for k=1:5
    vals(k) = input(sprintf('Introdu valoarea v%d: ', k));
end

% Calcul RMS
rms_val = sqrt(mean(vals.^2));

% Afișare
fprintf('Valoarea RMS a semnalului = %.3f\n', rms_val);
```

Observație: `vals.^2` — operație element-cu-element; funcția `mean` se execută pe vectorul rezultat.

Problema 5.7.2

Definiți următoarea matrice: A = $\begin{matrix} 2 & 1 \\ 3 & 4 \end{matrix}$

1. Calculați det(A) și inv(A).
2. Rezolvați sistemul A*x = b pentru b = [5; 6] cu A\b.
3. Verificați A * inv(A) și discutați diferența numerică față de eye(2).

Rezolvare propusă:

```
A = [2 1; 3 4];
detA = det(A);
invA = inv(A);
b = [5;6];
x = A\b;   % recomandat

% Verificare
res = A*invA - eye(2);

fprintf('det(A)=%.3f\n', detA);
disp('inv(A)='); disp(invA);
disp('Solutia x='); disp(x);
disp('A*inv(A)-I = (numerical residual)'); disp(res)
```

Dupa execuție obținem:

```
det(A)=5.000
inv(A)=
    0.8000   -0.2000
   -0.6000    0.4000
```

```
Solutia x=
    2.8000
   -0.6000

A*inv(A)-I = (numerical residual)
     0     0
     0     0
```

Problema 5.7.3

Reprezentați `u(t) = 230*sqrt(2)*sin(2*pi*50*t)` pe intervalul `t = 0:1e-4:0.02` (o perioadă). Afișați RMS numeric (folosind `rms` sau `sqrt(mean(u.^2))`).

Rezolvare propusă:

```
clear;clc
t = 0:1e-4:0.02;
U = 230*sqrt(2);
u = U*sin(2*pi*50*t);

figure;
plot(t,u); grid on;
xlabel('t [s]'); ylabel('u(t) [V]');
title('Tensiune sinusoidala 50 Hz');
% RMS numeric
U_rms_num = sqrt(mean(u.^2));
fprintf('U_rms numeric = %.3f V\n', U_rms_num);
```

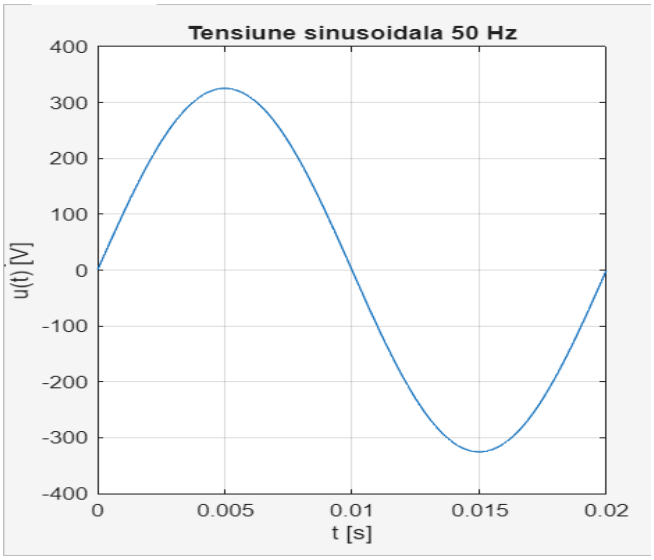


Figura 5.1 Fereastra generată in MATLAB după execuția problemei 5.7.3 de mai sus.

Problema 5.7.4

Implementați o funcție numită `medie = mymean(v)` care returnează media aritmetică a vectorului `v`. Demonstrati folosirea atât în script cât și direct in Command Window.

Rezolvare propusă:

```
%Functia mymean.m
function m = mymean(v)
% MYMEAN  calculeaza media aritmetica a vectorului v
  if isempty(v)
      m = NaN;
      return;
  end
```

```
  m = sum(v)/length(v);
end
```

```
Apel in Command Window:

>> a = [2 4 6 8];
>> m = mymean(a)
m =
     5
```

Problema 5.7.5

Fie un circuit serie R–C alimentat de `u(t) = Uhat*sin(omega*t)` cu `Uhat=230*sqrt(2)`, `f=50 Hz`, `R=100 ohm`, `C=47e-6 F`. Determinați:

1. Impedanța totală `Z = R - j/(omega*C)`, unde omega = ω=2πf
2. Curentul.
3. Tensiunea pe condensator în timp real `vC(t) = |Vc|*sin(omega*t + phi_vc)` și reprezentați pentru `t = 0:1e-4:0.1`.

Rezolvare propusă:

```
% Problema_RC.m
R = 100;
C = 47e-6;
f = 50;
```

```
omega = 2*pi*f;
Uhat = 230*sqrt(2);

% Impedanta
Zc = -1i/(omega*C);   % impedanta condensator (sau
1/(j*omega*C) = -j/(omega*C))
Z = R + Zc;

% Phasori
Iphas = Uhat / Z;
Vch_ = Iphas * Zc;    % tensiunea pe C in forma phasor
% amplitudine si faza
Vc_amp = abs(Vch_);
Vc_phi = angle(Vch_);

% reprezentare in timp
t = 0:1e-4:0.1;
vC = Vc_amp * sin(omega*t + Vc_phi);

figure;
plot(t, vC);
xlabel('t [s]'); ylabel('v_C(t) [V]');
title('Tensiunea pe condensator v_C(t)');

% afisari
fprintf('Z = %.3f %+.3fj ohm\n', real(Z), imag(Z));
fprintf('|I| = %.6f A, phi_I = %.3f rad\n', abs(Iphas),
angle(Iphas));
```

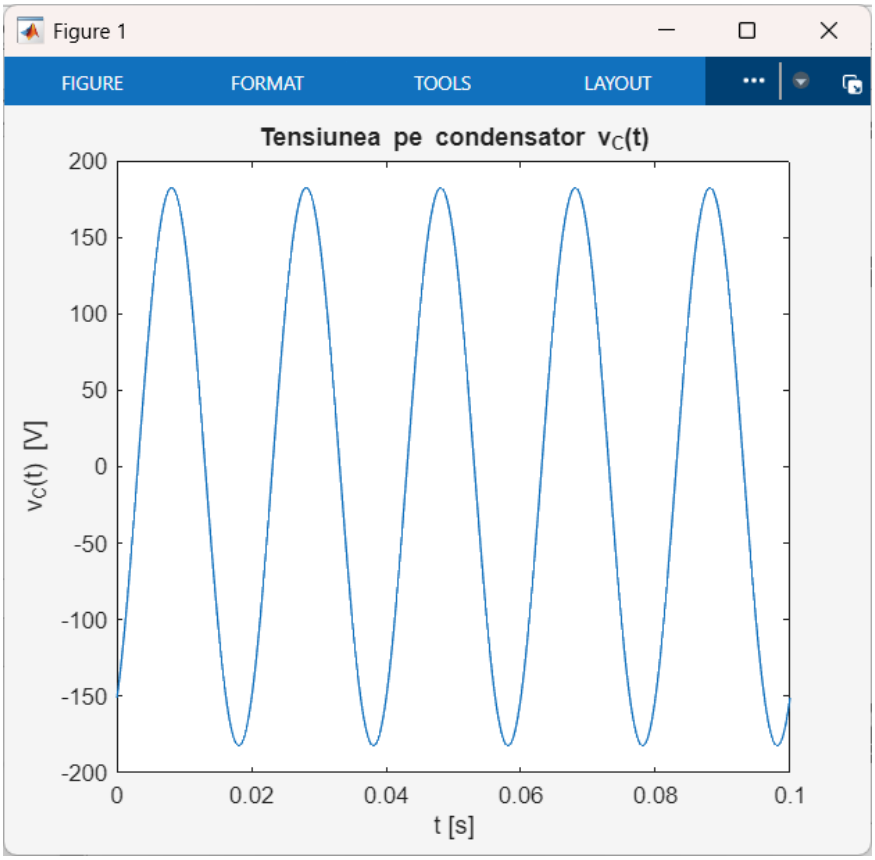


Figura 5.2 Fereastra generată în MATLAB dupa execuția problemei 5.5.3 de mai sus

Erori frecvente:

„Index exceeds matrix dimensions” → indexul este mai mare decât dimensiunea; verifică size(A).

Operații matriceale vs element-wise → folosește .*, ./, .^ dacă se lucrează cu vectori.

Variabile shadowing → sa nu se folosească nume de funcții ca nume de variabile.

În continuare vor fi prezentate urmatoarele topic-uri din calculul cu matrice: indexare avansată, funcțiile *diag*, *reshape*, și rezolvarea sistemelor liniare — explicații detaliate, exemple rezolvate și exerciții.

5.8 Indexare avansată

- **Indexare clasică**: `A(i,j)`.
- **Intervale**: `A(1:3,2:5)` → primele 3 linii, coloanele 2..5.
- **Tot pe o dimensiune**: `A(:,3)` sau `A(2,:)`.
- **Vector de indici**: `A([1 3 5], 2:4)` → linii 1,3,5; coloanele 2..4.
- **Indexare logică** (mask): `mask = A>5; A(mask)` → toate elementele >5.
- **Indexare liniară**: `A(:)` → pune toate elementele într-un vector coloana; folosit cu `sub2ind`/`ind2sub`.
- **`end`** în indici: `A(2:end, :)` — util la dimensiuni variabile.

Problema 5.8.1

Scopul acestui exercițiu este să înlocuim toate elementele negative cu zero și sa extragem elementele pare.

Rezolvare propusă:

```
% indexare_avansata.m
A = [-2 3 5; 10 -7 0; 4 9 -1];

% înlocuim negativele cu 0
A(A < 0) = 0;
```

```
% extragem elementele pare (vector)
pare = A(mod(A,2)==0);

disp('Matricea A dupa eliminarea negativelor:'); disp(A)
disp('Elemente pare gasite:'); disp(pare')
```

După execuție obținem:

```
Matricea A dupa eliminarea negativelor:
     0     3     5
    10     0     0
     4     9     0

Elemente pare gasite:
     0    10     4     0     0     0
```

Problema 5.8.2

Se consideră matricea pătrată 4×4 generată prin comanda `magic(4)`.

1. Determinați care este elementul situat la **indicele liniar** k = 7.
2. Folosiți funcția **ind2sub** pentru a afla coordonatele (linia și coloana) corespunzătoare acestui index.
3. Verificați operația inversă utilizând funcția **sub2ind**, adică să obțineți din coordonatele (i, j) valoarea lui k.

Rezolvare propusă:

```
A = magic(4);          % matrice 4x4
k = 7;                 % indice liniar
```

```
s% Conversie din indice liniar in (linie, coloana)
[i,j] = ind2sub(size(A), k);

% Verificare inversa: din (i,j) in indice liniar
k_back = sub2ind(size(A), i, j);

fprintf('La indicele liniar %d se afla elementul A(%d,%d) =
%d\n', ...
         k, i, j, A(i,j));
fprintf('Conversia inversa (sub2ind) intoarce k = %d\n',
k_back);
```

După execuție obținem**:**

```
La indicele liniar 7 se afla elementul A(3,2) = 7
Conversia inversa (sub2ind) intoarce k = 7
```

Comanda `A = magic(4);` creează un **pătrat magic 4×4**: o matrice care conține numerele 1 până la 16, fiecare o singură dată, astfel încât suma pe fiecare linie, coloană și diagonală este aceeași. MATLAB stochează elementele matricei **coloană cu coloană**. Asta înseamnă:

k=1 → elementul (1,1),

k=2 → elementul (2,1),

…,

k=4 → elementul (4,1),

k=5 → elementul (1,2),

k=6 → elementul (2,2),

k=7 → elementul (3,2),

ș.a.m.d.

Un singur index liniar parcurge matricea coloană cu coloană.

```
[i,j] = ind2sub(size(A), k);
```

Funcția `ind2sub` convertește un **indice liniar** (k) în coordonate **(linie i, coloană j)**.

`size(A) = [4,4]` %(matrice 4×4),

`k = 7`

k=7 corespunde poziției (3,2).

Astfel: i = 3 si j = 2.

Dacă se cunosc deja coordonatele `(i,j)` și vrem să aflam **indicele liniar k**, se foloseste funcția `sub2ind`:

```
k_back = sub2ind(size(A), i, j);
```

- `ind2sub` transformă **indice liniar → (linie, coloană)**.

- `sub2ind` transformă **(linie, coloană) → indice liniar**.

Este foarte util când funcții ca `max(A(:))` sau `find` returnează indici liniari și avem nevoie de poziția in format (linia și coloana) în matrice.

5.8.1 Comenzile `diag`, `tril`, `triu`, `fliplr`, `flipud`

`d = diag(A)` → extrage diagonala principală (vector).

`D = diag(d)` → creează matrice diagonală din vector `d`.

`diag(A, k)` → diagonala cu offset k (k>0 = deasupra, k<0 = sub diagonală).

Exemplu

```
A = [4 7 2 5;
     1 9 8 3;
     6 0 5 1;
     2 4 7 3];

d = diag(A);      % [4;9;5;3]
D = diag(d);      % matrice diagonala
d_above = diag(A,1); % diagonala imediat deasupra: [7;8;1]
```

După execuție obținem:

```
d =

     4
     9
     5
     3

D =

     4     0     0     0
     0     9     0     0
     0     0     5     0
     0     0     0     3

d_above =

     7
     8
     1
```

`L = tril(A)` păstrează elementele de pe și de sub diagonală; restul devine 0.

`U = triu(A)` păstrează elementele de pe și deasupra diagonalei.

Parametrizare: `triu(A,1)` exclude diagonala principală (păstrează ce este strict deasupra).

`fliplr(A)` — inversează ordinea coloanelor.

`flipud(A)` — inversează ordinea liniilor.

`rot90(A,k)` — rotește cu 90° de `k` ori (k implicit 1).

5.8.2 Comenzile `reshape`, `transpose`, `permute`, `repmat`, `kron`

`reshape` - Transformă forma matricei respectând ordering-ul coloanelor.

Exemplu

```
v = 1:12;          % 1..12
M = reshape(v, 3, 4);  % 3 linii, 4 coloane
% M este completată pe coloane: [1 4 7 10; 2 5 8 11; 3 6 9
12]'
```

```
M =
```

```
1     4     7    10
2     5     8    11
3     6     9    12
```

Pentru o transformare pe linii, `reshape` se poate combina cu `transpose` sau se poate folosi comanda `reshape(v,4,3).'`.

5.8.3 Comanda `permute`

`permute(A, order)` schimbă ordinea dimensiunilor (util in lucrul cu tensori).

5.8.4 Comenzile `repmat` și `kron`

`repmat(A,m,n)` crează o replica a matricei `A` m×n.

`kron(A,B)` este produsul Kronecker (util în modelări matriciale).

5.8.5 Analiză matriceală: `det`, `inv`, `rank`, `eig`, `svd`

Determinant și inversă

```
d = det(A)

invA = inv(A)
```

Recomandare: se folosește `A\b` pentru rezolvarea `Ax=b` (mai stabil/numeric eficient).

Rădăcini proprii, SVD

`eig(A)` returnează vectorii și valorile proprii (folosit pentru stabilitate, dinamica sistemelor).

`svd(A)` dezmembrează matricea; util pentru condiționare și compresie.

5.9 Rezolvarea sistemelor de ecuatii liniare

Considerăm un sistem de trei ecuații liniare cu trei necunoscute:

$$\begin{cases} 2x + 3y - z = 10 \\ -2x + y + 2z = 5 \\ x - y - z = -1 \end{cases}$$

Acesta poate fi scris matriceal astfel:

$AX = B$, unde:

A – reprezintă matricea coeficientilor necunoscutelor, de dimensiune 3x3;

X – reprezintă matricea necunoscutelor, de dimensiune 3x1;

B – reprezintă matricea termenilor liberi, de dimensiune 3x1.

$$A = \begin{bmatrix} 2 & 3 & -1 \\ -2 & 1 & 2 \\ 1 & -1 & -1 \end{bmatrix}, \qquad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad B = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Rezolvarea sistemelor de ecuații liniare se poate face prin două metode:

A) rezolvarea sistemelor prin împărțirea matricilor;

B) rezolvarea sistemelor prin folosirea matricei inverse.

5.9.1 Rezolvarea sistemelor prin împărțirea matricilor

Rezolvarea ecuației matriceale AX=B presupune împărțirea la stanga a matricilor:
AX=B => X=A\B.

Pentru exemplul anterior, se utilizează secvența MATLAB:

```
A=[2 3 -1;-2 1 2;1 -1 -1];
B=[10;5;-1];
X=A\B
```

și se obține rezultatul:

```
X=[23 -3 27]
```

Adică: x=23; y=-3; z=27.

5.9.2 Rezolvarea sistemelor de ecuații liniare prin folosirea matricei inverse

Sistemul descris prin ecuația matriceală:
A X = B,
poate fi rezolvat astfel:

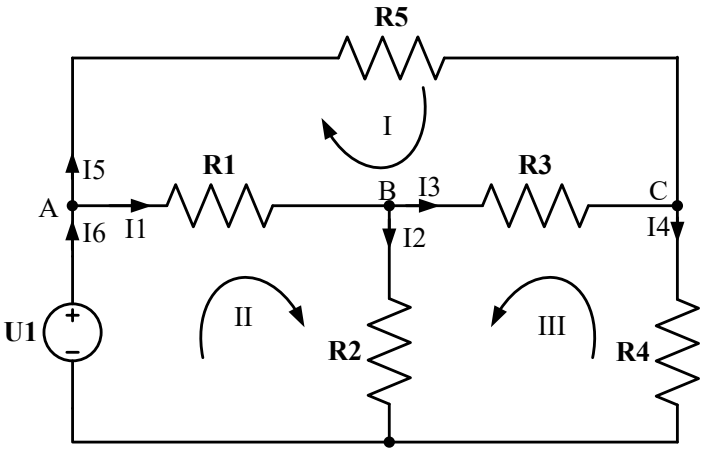
```
X=inv(A)*B
```

si se obține acelasi rezultat ca in cazul precedent.

Chiar dacă, în general, cele două soluții oferă același rezultat, este de preferat utilizarea primei metode de rezolvare prin împărțirea matricilor, deoarece se obține un timp de execuție mai rapid față de cea de a doua soluție.

5.9.3 Aplicație la rezolvarea sistemelor de ecuații liniare – calculul curenților dintr-un circuit electric de c.c.

Fie circuitul din figura de mai jos. Sa se determine curenții prin fiecare latura a circuitului.



Problema se rezolvă prin aplicarea Teoremelor lui Kirchhoff (I si II):

- TKI: Suma curenților care converg intr-un nod este nulă;
- TK2: Suma căderilor de tensiune pe laturile unui ochi este egală cu suma tensiunilor electromotoare existente in laturile ochiului.

Folosind prima teorema se pot scrie (N-1) ecuații de circuit, iar cu cea de a doua pot fi scrise L-N+1 ecuații de circuit. In total vom avea L ecuații.

$$\begin{cases} A: & -I1 & +0\cdot I2 & +0\cdot I3 & +0\cdot I4 & -I5 & +I6 & =0 \\ B: & I1 & -I2 & -I3 & +0\cdot I4 & +0\cdot I5 & +0\cdot I6 & =0 \\ C: & +0\cdot I1 & +0\cdot I2 & I3 & -I4 & +I5 & +0\cdot I6 & =0 \\ I: & -R1\cdot I1 & +0\cdot I2 & -R3\cdot I3 & +0\cdot I4 & R5\cdot I5 & +0\cdot I6 & =0 \\ II: & R1\cdot I1 & +R2\cdot I2 & +0\cdot I3 & +0\cdot I4 & +0\cdot I5 & +0\cdot I6 & =U1 \\ III: & +0\cdot I1 & +R2\cdot I2 & -R3\cdot I3 & -R4\cdot I4 & +0\cdot I5 & +0\cdot I6 & =0 \end{cases}$$

Acesta reprezintă un sistem de 6 ecuații liniare cu 6 necunoscute (I1...I6) si se poate scrie sub forma matriceală:

– $R*I=U$, unde R este matricea coeficienților, I este matricea curenților prin laturi, iar U este matricea termenilor liberi.
– Rezolvarea circuitului constă in determinarea matricei curenților din laturi, I, astfel: $I=R\backslash V$.

$$R=\begin{bmatrix} -1 & 0 & 0 & 0 & -1 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ -R1 & 0 & -R3 & 0 & R5 & 0 \\ R1 & R2 & 0 & 0 & 0 & 0 \\ 0 & R2 & -R3 & -R4 & 0 & 0 \end{bmatrix} \qquad I=\begin{bmatrix} I1 \\ I2 \\ I3 \\ I4 \\ I5 \\ I6 \end{bmatrix} \qquad V=\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ U1 \\ 0 \end{bmatrix}$$

Rezolvare propusă:

```
clear;clc
Rm=[2 4 6 8 10];
U=10;
R=[-1 0 0 0 -1 1;
    1 -1 -1 0 0 0;
    0 0 1 -1 1 0;
    -Rm(1) 0 -Rm(3) 0 Rm(5) 0;
    Rm(1) Rm(2) 0 0 0 0;
    0 Rm(2) -Rm(3) -Rm(4) 0 0];

V=[0;0;0;0;U;0];
I=R\V;
disp('I='); disp(I');


I=
    1.7925    1.6038    0.1887    0.6604    0.4717    2.2642
```

Capitolul 6

Calcule cu matrice

Mediul de programare MATLAB reprezintă un „laborator de matrice” – unitatea fundamentala de lucru fiind matricea. Un scalar este o matrice 1×1, un vector este o matrice cu o singură linie sau coloană, iar matricele generale sunt colecții bidimensionale de elemente numerice. În ingineria electrică:

- tensiunile și curenții din rețele se pot nota sub formă de vectori;
- parametrii pot fi reprezentați ca matrice;
- rezolvarea sistemelor de ecuații (Kirchhoff) se face eficient cu operații matriceale.

6.1 Extragerea submatricilor prin indici

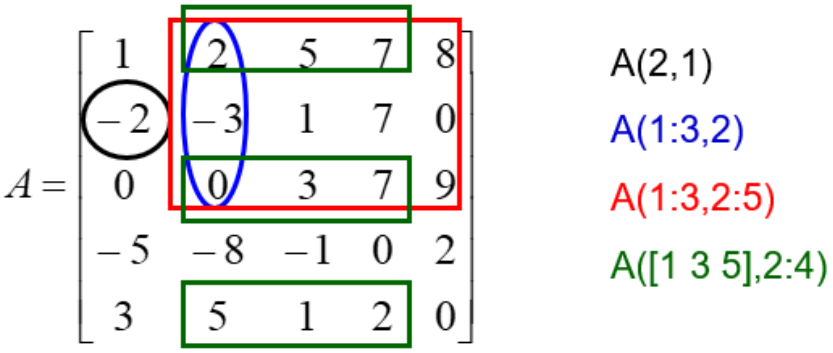
Fie matricea M de dimensiune 3x3: M = [1 2 3; 4 5 6; 7 8 9].

a) Pentru accesarea unui element: M (i,j) → elementul de pe linia i, coloana j, se folosește instrucțiunea M (2,3) – linia 2 coloana 3, elementul cu valoarea 6. Extragerea unei linii întregi se realizează folosind instructiunea M (2,:) ce va extrage toata linia a doua a matricei M: [4 5 6]. O coloană intregă se poate extrage folosind M (:,3) ce va rezulta in coloana 3: [3;6;9].

b) Pentru extragerea unei submatrici B din M cu elementele liniilor 1-2 si coloanelor 2-3: B = M(1:2,2:3), B=[2 3;5 6].

Observație: Sintaxa (start:pas:end) este extrem de puternică în MATLAB pentru selectarea de intervale din vectori/matrici.

Fie matricea A de mai jos:



Instrucțiunea **A(2,1)** extrage elementul de pe linia a doua și prima coloană din matrice A. Indicii pot fi **scalari** (ca în exemplul anterior) sau **vectori**; indicii vectori permit definirea unor submatrici, astfel instrucțiunea **A(1:3,2)** va extrage din matricea **A** o submatrice de dimensiune **3x1**, constand din primele 3 elemente din coloana a 2-a.
De asemenea, instrucțiunea: A(1:3,2:5) va extrage din matricea A o submatrice de dimensiune 3×4, construită din elementele primelor 3 linii (1–3) și elementele coloanelor 2, 3, 4 și 5.
Utilizarea semnului „:” în locul unui indice presupune considerarea tuturor elementelor de pe liniile sau coloanele respective. Instrucțiunea **A(:,3)** extrage din matricea **A** o submatrice care conține toate liniile din coloana a 3-a.

De asemenea, se pot extrage submatrici cu elemente disparate. Astfel, instrucțiunea **A([1 3 5], 2:4)** va extrage o submatrice din matricea **A**, cu liniile 1, 3 și 5 și coloanele 2, 3 și 4.

6.2 Funcții numerice uzuale

`sum(A)`, `sum(A,2)` – sumă pe coloane / pe linii.

`mean(A)` – medii pe coloane.

`max(A)`, `min(A)` – elementul maxim/minim.

`det(A)` – determinantul.

`inv(A)` – inversa (folosit cu precauție, de preferat `A\b`).

`rank(A)` – rangul.

`trace(A)` – suma elementelor de pe diagonala principală.

`rand()` – generează numere aleatoare (pseudoaleatoare) distribuite uniform în intervalul *(0, 1)*. Valorile sunt de tip real (nu complexe) și sunt foarte folosite în simulări, testări și generarea de matrice inițiale.

`randi()`– generează numere întregi, spre deosebire de `rand` (care generează numere reale).

`magic(n)` - funcția generează o matrice magică de ordinul n × n. O matrice magică este o matrice pătrată în care toate elementele sunt

numere întregi pozitive, distincte, numerele sunt aranjate astfel încât suma elementelor de pe fiecare linie, fiecare coloană și cele două diagonale principale este aceeași.

Exemplu:

```
% Definim matricea A (matrice 3x3)
A = [1 2 3; 4 5 6; 7 8 9];

% Calculăm și afișăm determinantul matricei A.
fprintf('det(A)=%.2f\n', det(A));

% Calculăm și afișăm rangul (rank) matricii A.
fprintf('rank(A)=%d\n', rank(A));

% Calculăm și afișăm urma (trace) matriciei A.
% Urma este suma elementelor de pe diagonala principală: 1 +
5 + 9 = 15.
fprintf('trace(A)=%.2f\n', trace(A));
```

6.3 Asamblarea și combinarea matricilor

Tehnicile de asamblare și combinare a matricilor sunt utile în prelucrarea datelor, în metode numerice, în grafică, în programare științifică și în multe aplicații inginerești. Fie matricele:

A=[1 2;3 4];

B=[5 6; 7 8];

Asamblare pe orizontală: `[A B]`

Asamblare pe verticală: `[A;B]`

```
%Codul Matlab - obtinerea unei matrice C compusă din cele
doua matrice A si B:
>> A=[1 2;3 4]; B=[5 6; 7 8];
>> C=[A B]
C =
     1     2     5     6
     3     4     7     8
>> D=[A;B]

D =
     1     2
     3     4
     5     6
     7     8
```

Asamblarea unei matrice bloc: `M = [A zeros(2); zeros(2) B]`

6.3.1 Funcții utile pentru manipularea matricilor

`diag` - extrage sau creează o diagonală a unei matrici

`fliplr` - (flip left-right) inversează ordinea coloanelor unei matrici (întoarce matricea în oglindă pe orizontală).

`flipud` (flip up-down) - inversează ordinea liniilor (oglindire verticală).

`rot90` - rotește o matrice cu 90° în sens trigonometric.

`tril` - (triangular lower) extrage partea triunghiulară inferioară a unei matrici.

`triu` - (triangular upper) extrage partea triunghiulară superioară a unei matrici.

`inv` - inversa unei matrici calculează matricea inversă a unei matrici pătratice (dacă aceasta este inversabilă).

6.4 Probleme propuse și rezolvate

Problema 6.4.1

Fie `A=randi(10,4,5)`. Calculați:

- suma fiecărei linii;
- media fiecărei coloane;
- indexul elementului maxim din matrice.

Rezolvare propusă:

```
A = randi(10,4,5);
sume_linii = sum(A,2);          % sumă pe linii
medii_coloane = mean(A,1);      % medii pe coloane
[val_max, idx_max] = max(A(:)); % max pe tot A
[i_max,j_max] = ind2sub(size(A), idx_max);
```

Problema 6.4.2

Fie `A=rand(3,3)`. Calculați determinantul, rangul și inversa lui `A` și verificați `A*inv(A)=I`.

Rezolvare propusă:

```
A = rand(3,3);
detA = det(A);
rA = rank(A);
invA = inv(A);
I_check = A*invA;  %verificare - matricea identitate
```

Problema 6.4.3

Fie `A=magic(3)`. Calculați:

`A*A` (produs matricial).

`A.*A` (produs element cu element).

Rezolvare propusa:

```
A = magic(3);
prod_mat = A*A;
prod_elem = A.*A;
```

Problema 6.4.4

Generați o matrice aleatoare 6×6 și calculați suma elementelor de sub diagonala principală.

Rezolvare propusă:

```
% Generarea unei matrice aleatoare 6x6
A = rand(6);
```

```
% Afișarea matricei
disp('Matricea A:');
disp(A);

% Calculul sumei elementelor de sub diagonala principală
suma_sub_diag = sum(sum(tril(A, -1)));

% Afișarea rezultatului
disp(['Suma elementelor de sub diagonala principală este: ',
num2str(suma_sub_diag)]);
```

După execuție obținem

```
Matricea A:
    0.8147    0.2785    0.9572    0.7922    0.6787    0.7060
    0.9058    0.5469    0.4854    0.9595    0.7577    0.0318
    0.1270    0.9575    0.8003    0.6557    0.7431    0.2769
    0.9134    0.9649    0.1419    0.0357    0.3922    0.0462
    0.6324    0.1576    0.4218    0.8491    0.6555    0.0971
    0.0975    0.9706    0.9157    0.9340    0.1712    0.8235

Suma elementelor de sub diagonala principală este: 9.1603
```

Problema 6.4.5

Scrieți o funcție MATLAB care primește o matrice și returnează vectorul cu mediile fiecărei linii și vectorul cu mediile fiecărei coloane.

Rezolvare propusă:

Funcție MATLAB: `medii_linii_coloane.m`

```
function [medii_linii, medii_coloane] =
medii_linii_coloane(A)
    % Verifică dacă A este matrice numerică
    if ~isnumeric(A)
        error('Inputul trebuie să fie o matrice numerică.');
    end

    % Calculează mediile fiecărei linii
    medii_linii = mean(A, 2);   % mean pe a doua dimensiune -
> linii

    % Calculează mediile fiecărei coloane
    medii_coloane = mean(A, 1); % mean pe prima dimensiune ->
coloane
end
```

Problema 6.4.6

Folosind rand(1000) comparați timpul de execuție pentru sum(A,1) versus un for explicit.

Rezolvare propusă:

```
% Generare matrice
A = rand(1000);
```

```
% --- Metoda vectorizată: sum(A,1) ---
tic;%porneste cronometru
s1 = sum(A, 1);
t1 = toc;% toc oprește cronometrul și ofera timpul scurs

% --- Metoda cu for explicit ---
tic;
s2 = zeros(1, size(A,2));
for j = 1:size(A,2)
    for i = 1:size(A,1)
        s2(j) = s2(j) + A(i,j);
    end
end
t2 = toc;

% Afișare rezultate
fprintf('Timp sum(A,1): %f secunde\n', t1);
fprintf('Timp for explicit: %f secunde\n', t2);

% Verificare diferență numerică
fprintf('Norma diferenței: %e\n', norm(s1 - s2));
```

După execuție obținem

```
Timp sum(A,1): 0.000812 secunde
Timp for explicit: 0.001331 secunde
Norma diferenței: 1.319416e-11
```

Functiile **tic** si **toc**: **tic** și **toc** sunt funcții MATLAB folosite pentru măsurarea timpului de execuție al unui segment de cod. "**tic"** marchează începutul unui cronometru intern si nu are argumente, iar "**toc"** marchează sfârșitul cronometrului pornit cu "tic" si returnează timpul scurs în secunde.

6.5 Rezolvarea sistemelor de ecuații liniare

Fie următorul sistem de trei ecuații liniare cu trei necunoscute:

$$\begin{cases} 2x + 3y - z = 10 \\ -2x + y + 2z = 5 \\ x - y - z = -1 \end{cases}$$

Acesta poate fi scris matriceal astfel: $AX = B$

unde:

A – reprezintă matricea coeficienților necunoscutelor, de dimensiune 3x3;

X – reprezintă matricea necunoscutelor, de dimensiune 3x1;

B – reprezintă matricea termenilor liberi, de dimensiune 3x1.

$$A = \begin{bmatrix} 2 & 3 & -1 \\ -2 & 1 & 2 \\ 1 & -1 & -1 \end{bmatrix}, \qquad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad B = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Rezolvarea sistemelor de ecuatii liniare se poate face prin două metode:

1) rezolvarea sistemelor prin împarțirea matricilor;

2) rezolvarea sistemelor prin folosirea matricei inverse.

6.5.1 Rezolvarea sistemelor prin împărțirea matricilor

Rezolvarea ecuației matriceale AX=B presupune impărțirea la stânga a matricilor:

AX=B => X=A\B.

Pentru exemplul anterior, se utilizează secvența MATLAB:

```
A=[2 3 -1;-2 1 2;1 -1 -1];
B=[10;5;-1];
X=A\B
```

Dupa executie obtinem:

```
X =

    23
    -3
    27
```

Necunoscutele sunt x=23; y=-3; z=27.

6.5.2 Rezolvarea sistemelor prin împarțirea matricilor

Sistemul descris de ecuația matriceală AX=B poate fi rezolvat astfel:

X=inv(A)*B

Se obține același rezultat ca și în cazul precedent.

Chiar dacă, în general, cele două soluții oferă același rezultat, este de preferat utilizarea primei metode de rezolvare prin împărțirea matricilor, deoarece se obține un timp de execuție mai rapid față de cea de a doua soluție.

Observații

- Folosiți A\b pentru a rezolva sisteme liniare, nu inv(A)*b. Este mai rapid și mai stabil numeric.
- Operatorii cu punct (.*, ./, .^) sunt esențiali pentru vectorizare.

6.6 Probleme rezolvate

Problema 6.6.1

Arătați numeric diferența dintre `inv(A)*b` și `A\b` pentru un sistem aleator.

Rezolvare propusă:

```
% Generăm un sistem aleator
A = rand(5);       % matrice 5x5 aleatoare
b = rand(5,1);     % vectorul termenilor liberi

% Soluția cu inversă
x_inv = inv(A) * b;

% Soluția cu operatorul backslash (metoda recomandată)
x_backslash = A \ b;
```

```
% Diferența numerică
diferenta = norm(x_inv - x_backslash);

% Afișăm rezultatele
disp('Soluția cu inv(A)*b:');
disp(x_inv);

disp('Soluția cu A\\b:');
disp(x_backslash);

disp('Norma diferenței între soluții:');
disp(diferenta);
```

Dupa execuție obținem:

```
Soluția cu inv(A)*b:
    0.5496
    0.7249
   -1.8141
   -2.2834
    3.3837

Soluția cu A\\b:
    0.5496
    0.7249
   -1.8141
   -2.2834
    3.3837

Norma diferenței între soluții:
   1.8377e-15
```

Problema 6.6.2

Fie sistemul de ecuații liniare:

$$\begin{cases} 2x - y + z = 2 \\ 3x + 3y + 9z = -1 \\ 3x + 3y + 5z = 4 \end{cases}$$

Rezolvare propusă:

%Pentru rezolvare se scrie sistemul sub formă matricială AX=B:

```
%cod Matlab
% Matricea coeficienților
A = [2 -1 1; 3 3 9; 3 3 5];

% Vectorul termenilor liberi
b = [2; -1; 4];

% Rezolvare sistem
x = A \ b;

% Afișare rezultat
disp('Soluția sistemului:');
disp(x);

% Verificare residuu
disp('Residuu:');
disp(norm(A*x - b));
```

După execuție:

```
Soluția sistemului:
    2.2222
    1.1944
   -1.2500

Residuu:
     0
```

În codul de mai sus: A \ b reprezintă metoda recomandată pentru rezolvarea sistemelor liniare, x va conține vectorul soluțiilor $[x, y, z]^T$. Funcția `norm(A*x - b)` verifică cât de aproape este soluția de sistemul original; valoarea rezultată ar trebui să fie foarte mică (aproximativ 0).

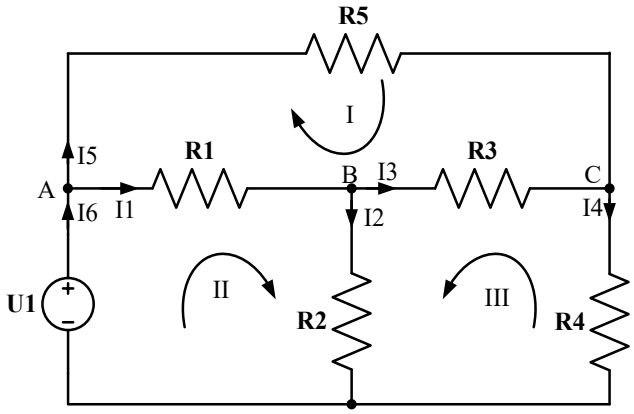
Rezolvare generalizată a unui sistem de ecuații liniare

```
clear;clc
disp('Rezolvarea sistemelor de ecuatii liniare');
disp(' ');
A=input('Introduceti matricea coeficientilor A[n][n]:\n');
B=input('Introduceti matricea termenilor liberi B[n][1]:\n');

if size(B)~=[length(A) 1]
    disp('Matrici introduse incorect');
else
    disp('Solutia este:'); disp(A\B);
end
```

In programul de mai sus utilizatorul este rugat să introducă matricea coeficienților A și vectorul termenilor liberi B folosind functia `input`. Codul verifică dacă dimensiunile lui B sunt compatibile cu A (adică B trebuie să fie un vector coloană cu același număr de linii ca matricea A) și, în cazul în care nu sunt, afișează un mesaj de eroare. Dacă dimensiunile sunt corecte, programul calculează soluția sistemului liniar folosind operatorul \ (metoda recomandată de MATLAB pentru stabilitate numerică) și afișează rezultatul folosind funcția `disp`.

6.6.3 Aplicație la rezolvarea sistemelor de ecuații liniare – calculul curenților dintr-un circuit electric de c.c.

Fie circuitul din figura de mai jos. Să se determine curenții prin fiecare latură a circuitului.



Problema se rezolvă prin aplicarea Teoremelor lui Kirchhoff (I și II):

- TKI: Suma curenților care converg într-un nod este nulă;
- TK2: Suma căderilor de tensiune pe laturile unui ochi este egală cu suma tensiunilor electromotoare existente în laturile ochiului.

Folosind prima teoremă se pot scrie (N-1) ecuații de circuit, iar cu cea de a doua pot fi scrise L-N+1 ecuații de circuit. In total vom avea L ecuații.

$$\begin{array}{l} A: \\ B: \\ C: \\ I: \\ II: \\ III: \end{array} \left\{ \begin{array}{cccccccc} -I1 & +0\cdot I2 & +0\cdot I3 & +0\cdot I4 & -I5 & +I6 & = & 0 \\ I1 & -I2 & -I3 & +0\cdot I4 & +0\cdot I5 & +0\cdot I6 & = & 0 \\ +0\cdot I1 & +0\cdot I2 & I3 & -I4 & +I5 & +0\cdot I6 & = & 0 \\ -R1\cdot I1 & +0\cdot I2 & -R3\cdot I3 & +0\cdot I4 & R5\cdot I5 & +0\cdot I6 & = & 0 \\ R1\cdot I1 & +R2\cdot I2 & +0\cdot I3 & +0\cdot I4 & +0\cdot I5 & +0\cdot I6 & = & U1 \\ +0\cdot I1 & +R2\cdot I2 & -R3\cdot I3 & -R4\cdot I4 & +0\cdot I5 & +0\cdot I6 & = & 0 \end{array} \right.$$

Acesta reprezintă un sistem de 6 ecuații liniare cu 6 necunoscute (I1...I6) și se poate scrie sub forma matriceală: *R*I=U*, unde *R* este matricea coeficientilor, *I* este matricea curenților prin laturi, iar *U* este matricea termenilor liberi.

Rezolvarea circuitului constă in determinarea matricei curenților din laturi, *I*, astfel: *I=R\V.*

$$R = \begin{bmatrix} -1 & 0 & 0 & 0 & -1 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 1 & 0 \\ -R1 & 0 & -R3 & 0 & R5 & 0 \\ R1 & R2 & 0 & 0 & 0 & 0 \\ 0 & R2 & -R3 & -R4 & 0 & 0 \end{bmatrix} \quad I = \begin{bmatrix} I1 \\ I2 \\ I3 \\ I4 \\ I5 \\ I6 \end{bmatrix} \quad V = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ U1 \\ 0 \end{bmatrix}$$

```
% cod Matlab
clear;
Rm=[2 4 6 8 10];
U=10;
```

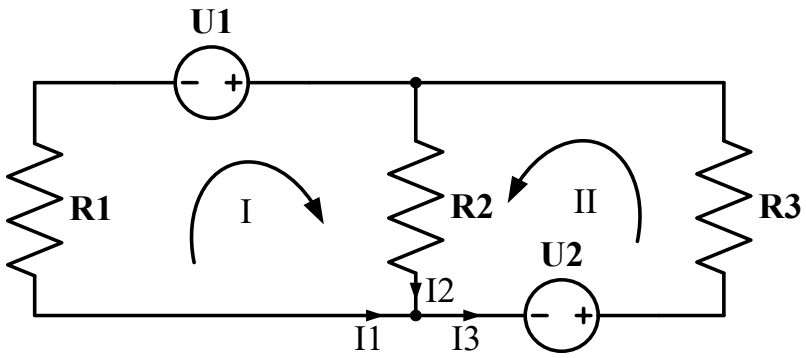
```
R=[-1 0 0 0 -1 1;
    1 -1 -1 0 0 0;
    0 0 1 -1 1 0;
    -Rm(1) 0 -Rm(3) 0 Rm(5) 0;
    Rm(1) Rm(2) 0 0 0 0;
    0 Rm(2) -Rm(3) -Rm(4) 0 0];

V=[0;0;0;0;U;0];
I=R\V;
disp('I='); disp(I');
```

Dupa execuție obținem:

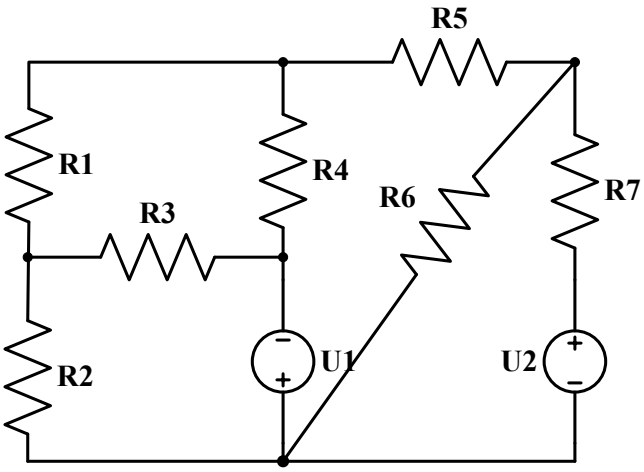
```
I=
  1.7925   1.6038   0.1887   0.6604   0.4717   2.2642
```

6.6.4 Exerciții propuse

Sa se calculeze curenții din următoarele circuite:



R1=1Ω, R2=2 Ω, R3=3 Ω , U1=7V, U2=12V



R1=R2=R3=R4=R5=R6=R7=1k Ω , U1=100V, U2=45V

6.7 Funcții matematice uzuale

Funcții pentru aproximarea numerelor

Funcția	Descriere	Exemplu
ceil	Returnează cel mai mic număr întreg mai mare sau egal cu valoarea dată (rotunjire în sus, spre +∞)	`ceil(4.25)= 5` `ceil(-4.25)= -4`
fix	returnează partea întreagă a valorii, prin rotunjire spre zero.	`fix(4.25)= 4` `fix(-4.25)= -4`
floor	returnează cel mai mare număr întreg mai mic sau egal cu valoarea dată (rotunjire în jos, spre −∞)	`floor(4.25)=4` `floor(-4.25)=-5`
round	Returnează cel mai apropiat număr întreg de valoarea dată (rotunjire la cel mai apropiat întreg)	`round(4.25)=4` `round(5.78)=6`

rem	Returnează restul împărțirii dintre două valori.	`rem(5,2)=1`

sign	returnează semnul argumentului: −1 , 0 sau 1	`sign(-4.25)=-1`
		`sign(4.25)=1`
		`sign(0)=0`

Funcțiile putere, radical, logaritm si exponențială

Funcția	**Descriere**	**Exemplu**
^	Ridicare la putere	`2^3=8`
exp	Exponențiala (e^x)	`exp(2)=7.3891`
log	Logaritmul natural (ln)	`log(7.3891)=2`
log2	Logaritmul in baza 2 ($\log_2$)	`log2(4)=2`
log10	Logaritmul zecimal ($\log_{10}$)	`log10(100)=2`
pow2	Calcul puteri ale lui 2	`pow2(5)=32`
sqrt	Radical de ordinul 2	`sqrt(16)=4`
nthroot	Radical de ordinul n	`nthroot(125,3)=5`

Funcțiile trigonometrice

Functii trigonometrice directe

Functia	**Descriere**	**Exemplu**
sin	Calcul sinusul argumentului **in radiani**	`sin(pi/6)=0.5000`
cos	Calculeaza cosinusul argumentului **in radiani**	`cos(pi/6)=0.8660`
tan	Calculeaza tangenta argumentului **in radiani**	`tan(pi/6)=0.5774`
cot	Calculeaza cotangenta argumentului **in radiani**	`cot(pi/6)=1.7321`
sind	Calculeaza sinusul argumentului **in grade***	`sind(30)=0.5000`
cosd	Calculeaza cosinusul argumentului **in grade***	`cosd(30)=0.8660`

tand	Calculeaza tangenta argumentului **in grade***	`tand(30)=0.5774`
cotd	Calculeaza cotangenta argumentului **in grade***	`cotd(30)=1.7321`

Functii trigonometrice inverse

Functia	**Descriere**	**Exemplu**
asin	Calculeaza **in radiani** arcsinusul argumentului	`asin(0.5)=0.5236`
acos	Calculeaza **in radiani** arccosinusul argumentului	`acos(0.866)=0.5236`
atan	Calculeaza **in radiani** arctangenta argumentului	`atan(0.5774)=0.5236`
acot	Calculeaza **in radiani** arccotangenta argumentului	`acot(1.7321)=0.5236`
asind	Calculeaza **in grade** arcsinusul argumentului	`asind(0.5)=30`
acosd	Calculeaza **in grade** arccosinusul argumentului	`acosd(0.8660)=30`
atand	Calculeaza **in grade** arctangenta argumentului	`atand(0.5774)=30`
acotd	Calculeaza **in grade** arccotangenta argumentului	`acotd(1.7321)= 30`

Funcții de prelucrare a datelor

min	Determină cel mai mic element al unui vector/matrice
max	Determină cel mai mare element al unui vector/matrice
mean	Calculează valoarea medie
sum	Calculează suma tuturor elementelor unui vector/matrice
prod	Calculează produsul tuturor elementelor unui vector/matrice
sort	Sortează crescator elementele unui vector/matrice

Funcții de minim și maxim

```
M = max(X)
M = min(X)
```

Daca *X* este un vector, aceste funcții returnează un scalar egal cu cel mai mare (respectiv cel mai mic) element. Dacă *X* este o matrice, funcțiile returnează un vector linie care conține valorile maxime (respectiv minime) din fiecare coloană. Daca se dorește determinarea maximului (minimului) precum si a indicelui acestora, se poate utiliza sintaxa:

```
[M,I] = max(X)
[m,I] = min(X)
```

Pentru a compara doi scalari sau matrice, se pot utiliza sintaxele:

```
s = max(s1,s2); s = min(s1,s2);

C = max(A,B); C = min(A,B)
```

Media

Pentru a calcula media aritmetică a unui vector de date se utilizează funcția:

```
media=mean(X)
```

Daca *X* este un vector, funcția returnează un scalar egal cu valoarea medie a elementelor, iar dacă *X* este o matrice, funcția returnează un vector linie care conține valoarea medie a fiecărei coloane.

Sume și produse

Calculul sumei și a produsului elementelor unui vector se pot face cu funcțiile::

```
Y=sum(X)
Y=prod(X)
```

Daca *X* este un vector, funcția returneaza un scalar, iar daca *X* este o matrice funcția returnează un vector linie care contine suma (respectiv produsul) elementelor de pe fiecare coloană.

Sortarea

Sortarea elementelor unui vector sau ale unei matrice, în ordine crescătoare sau descrescătoare, se poate realiza cu funcția *sort*, folosind următoarea sintaxă:

```
Y=sort(X,'mode'); sau [Y,I]=sort(X,'mode');
```

unde: *mode* reprezintă unul dintre cuvintele cheie:

- *'ascend'* in ordine crescătoare (implicit)
- *'descend'* in ordine descrescătoare

Daca *X* este o matrice funcția sorteaza elementele de pe coloanele acesteia. Cea de-a doua variantă de apelare a funcției returnează pe lângă matricea elementelor sortate *Y* și matricea *I* a indicilor elementelor sortate.

Problema 6.7.1

Se dă vectorul:

$$v = [8, 3, 5, 12, 7]$$

Determinați elementul minim și maxim, calculați media și suma elementelor, calculați produsul tuturor elementelor si sortați vectorul crescător.

Rezolvare propusă:

```
clear;clc
v = [8 3 5 12 7];

min_v = min(v);       % 3
max_v = max(v);       % 12
mean_v = mean(v);     % 7
sum_v = sum(v);       % 35
prod_v = prod(v);     % 10080
v_sort = sort(v);     % [3 5 7 8 12]
disp(min_v); disp(max_v); disp(mean_v); disp(sum_v);
disp(prod_v); disp(v_sort);
```

După execuție obținem:

```
     3

    12

     7

    35

       10080

     3     5     7     8    12
```

Problema 6.7.2

Se dă matricea: $A = \begin{matrix} 2 & 5 & 1 \\ 7 & 3 & 4 \\ 6 & 8 & 9 \end{matrix}$

Determinați cel mai mic element din întreaga matrice, determinați cel mai mare element din fiecare coloană, calculați suma fiecărei linii, calculați media globală a matricii si sortati fiecare linie crescător.

Rezolvare propusă:

```
clear
clc
A = [2 5 1; 7 3 4; 6 8 9];

min_A = min(A(:));       % 1
max_A_col = max(A);      % [7 8 9]
sum_A_row = sum(A, 2);   % [8; 14; 23]
mean_A_tot = mean(A(:)); % 5
A_sort_row = sort(A, 2); % [1 2 5; 3 4 7; 6 8 9]

disp(min_A); disp(max_A_col); disp(sum_A_row);
disp(mean_A_tot); disp(A_sort_row);
```

După execuție obținem:

```
1

   7     8     9

   8
  14
  23

   5

   1     2     5
   3     4     7
   6     8     9
```

Problema 6.7.3

Se dau vectorul *v* și matricea *B*:

$$v = [4,7,2,9],\ \ B = \begin{matrix} 3 & 1 \\ 6 & 5 \end{matrix}$$

Calculați produsul tuturor elementelor din v, determinați suma elementelor fiecărei coloane din *B*, aflați media vectorului v și media globală a lui *B* si sortati v crescător și liniile lui *B* crescător.

Rezolvare propusă:

```
clear
clc

v = [4 7 2 9];
B = [3 1; 6 5];
```

```
prod_v = prod(v);       % 504
sum_B_col = sum(B);     % [9 6]
mean_v = mean(v);       % 5.5
mean_B_tot = mean(B(:));% 3.75
v_sort = sort(v);       % [2 4 7 9]
B_sort_row = sort(B, 2);% [1 3; 5 6]

disp(prod_v); disp(sum_B_col); disp(mean_v);
disp(mean_B_tot); disp(v_sort); disp(B_sort_row);
```

După execuție obținem:

```
504

     9     6

    5.5000

    3.7500

     2     4     7     9

     1     3
     5     6
```

Problema 6.7.4

O linie de producție măsoară temperaturile (°C) in 6 puncte diferite de control într-o zi, pentru 5 zile consecutive. Datele sunt stocate într-o matrice T (linii = zile, coloane = puncte de măsură).

Determinați temperatura minimă și maximă din fiecare zi, calculați temperatura medie zilnică și globală, afișați coloanele sortate crescător pentru fiecare zi (pentru monitorizarea uniformității temperaturii).

Rezolvare propusă:

```
clear
clc
T = [350 355 360 358 352 354;
     348 356 361 357 351 353;
     349 354 359 356 350 355;
     351 357 362 359 353 356;
     350 355 361 358 352 354];

T_min = min(T, [], 2);        % minim pe linii (fiecare zi)
T_max = max(T, [], 2);        % maxim pe linii (fiecare zi)
T_mean_day = mean(T, 2);      % medie zilnică
T_mean_global = mean(T(:));   % medie globală
T_sorted_day = sort(T, 2);    % sortare crescătoare pe linii

disp(T_min);
disp(T_max);
disp(T_mean_day);
disp(T_mean_global);
disp(T_sorted_day);
```

După execuție obținem:

```
 350
 348
 349
 351
 350

 360
 361
 359
 362
 361

354.8333
354.3333
353.8333
356.3333
355.0000

354.8667

 350   352   354   355   358   360
 348   351   353   356   357   361
 349   350   354   355   356   359
 351   353   356   357   359   362
 350   352   354   355   358   361
```

Capitolul 7

Calcule numerice cu polinoame

Fie polinomul:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0$$

Acesta poate fi reprezentat in MATLAB printr-un vector linie care conține coeficienții in ordinea descrescătoare a puterii:

F=[a_n a_{n-1} ... a_1 a_0]

Exemplu: f(x)=x^4+x^2+2x+3 => F=[1 0 1 2 3]

Funcțiile MATLAB aferente calculelor cu polinoame

Funcția	**Descriere**
conv	Calculează produsul a două polinoame
deconv	Calculează catul si restul impartirii a două polinoame
poly	Calculează coeficientii unui polinom cu radacinile date
polyval	Evaluează un polinom la valorile precizate ale variabilei
polyder	Calculează derivata polinoamelor
polyfit	Aproximează un set de date cu un polinom de grad *n*
residue	Descompune in fracții simple raportul a două polinoame
roots	Calculează radacinile unui polinom

7.1 Evaluarea polinoamelor (polyval)

Fie polinomul $f(x)=x^4+x^2+2x+3$. Pentru a calcula *f* in funcție de una sau mai multe valori ale lui *x*, se poate utiliza funcția *polyval*, cu sintaxa:

```
p=polyval(f,x)
```

unde: *f* reprezintă vectorul linie al coeficienților polinomului;

x reprezintă un scalar, vector sau matrice cu valori pentru calculul polinomului;

p reprezintă un scalar, vector sau matrice cu rezultatul.

Exemplu

```
F=[1 0 1 2 3];
X=[-1 0 1 2];
P=polyval(F,X)
```

va rezulta:

```
P=[3 3 7 27]
```

7.2 Operații aritmetice cu polinoame

Operații:

- derivată: `polyder(p)`
- integrată: `polyint(p)`
- rădăcini: `roots(p)`
- înmulțire: `conv(p,q)`
- împărțire: `[q,r]=deconv(p,q)`

7.2.1 Adunarea si scăderea

Fie polinoamele *f(x)* si *g(x)* de mai jos:

$f(x)=x^4+x^2+2x+3$

$g(x)=x^2-2$

Se cere să se calculeze:

$s(x)=f(x)+g(x)$

$d(x)=f(x)-g(x)$

Pentru rezolvarea adunării mai intai trebuie sa ne asigurăm ca dimensiunea vectorilor *f* si *g* va fi egală, în acest exemplu având valoarea 5 astfel:

```
F=[1 0 1 2 3]       %f(x)= x4+0x3+x2+2x+3
G=[0 0 1 0 -2]             %f(x)=0x4+0x3+ x2+0x-2
S=F+G
D=F-G
```

rezultă:

```
S=[1  0  2  2  1]
```

```
D=[1  0  0  2  5]
```

ceea ce corespunde polinoanelor: s*(x)=x^4+2x^2+2x+1* si *d(x)=x^4+2x+5*

7.2.2 Inmulțirea si impărțirea

Pentru a înmulți două polinoame folosim sintaxa:

```
p=conv(f,g)
```

unde:

p reprezintă vectorul coeficienților polinoamelor care se înmulțesc: $p(x) = f(x) \cdot g(x)$;

f și *g* reprezintă doi vectori linie ce conțin coeficienții celor două polinoame.

Pentru a împărți două polinoame folosim sintaxa:

```
[c r]=deconv(f,g)
```

unde:

c și r reprezintă vectorii coeficienților polinoamelor cât și rest:

$$f(x) = c(x) \cdot g(x) + r(x);$$

f și *g* reprezintă doi vectori linie ce conțin coeficienții celor două polinoame.

Exemplu:

Se dau două polinoame: $f(x) = x^2 - 2x - 3$ și $g(x) = x + 1$. Se vor calcula polinoamele obținute prin înmulțirea și împărțirea celor două.

Codul MATLAB:

```
F=[1 -2 -3];
G=[1 1];
P=conv(F,G)
[C R]=deconv(F,G)
```

rezultă:

```
P =

     1    -1    -5    -3


C =

     1    -3


R =

     0     0     0
```

ceea ce corespunde polinoamelor: $p(x) = x^3-x^2-5x-3$; $c(x) = x-3$ și $r(x) = 0$.

7.2.3 Calculul rădăcinilor unui polinom

Funcția MATLAB *roots* determină rădăcinile polinoamelor:

```
r=roots(f)
```

Funcția MATLAB *poly* determină coeficienții unui polinom ale cărui rădăcini sunt cunoscute și se apelează cu sintaxa:

```
f=poly(r)
```

f reprezintă un vector linie care conține coeficienții polinomului;

r reprezintă un vector coloană care conține rădăcinile polinomului.

Exemplu:

Fie polinomul $f(x)=x^3-2x^2-3x+10$, sa se determine rădăcinile acestuia.

```
F=[1 -2 -3 10];
R=roots(f)
```

va rezulta:

```
R =
  -2.0000
   2.0000 + 1.0000i
   2.0000 - 1.0000i
```

Pentru a determina coeficienții polinomului cunoscând rădăcinile acestuia, de exemplu r=2, 0, -1, în MATLAB:

```
R=[2;0;-1]
F=poly(R)
```

va rezulta:

```
F = [ 1    -1    -2     0]
```

ceea ce analitic reprezintă polinomul $f(x)=x^3-x^2-2x$.

7.2.4 Calculul derivatei unui polinom

Derivata unui polinom se calculează în MATLAB cu funcția:

```
D = polyder(F)
```

unde: *F* este un vector ce conține coeficienții polinomului (în ordine descrescătoare);

D este vectorul asociat coeficienților polinomului derivat.

Pentru a calcula $(F*G)'$ folosim sintaxa:

```
D=polyder(F,G)
```

Funcția va returna în vectorul *D* coeficienții derivatei polinomului produs *F*G*, conform bine cunoscutei relații:

$$D(x) = [F(x)G(x)]' = F(x)'G(x) + F(x)G(x)'$$

Pentru a calcula $(F/G)'$ folosim sintaxa:

```
[D,E]=polyder(F,G)
```

ce va returna (in vectorii *D* si *E*) coeficienții numărătorului si numitorului rezultatului derivatei raportului *D/E*, conform relației matematice:

$$D(x) / E(x) = [F(x)/G(x)]' = [F(x)'G(x) - F(x)G(x)'] / G(x)^2$$

Exemplu:

Fie polinoamele: $f(x)=2x^3+x+1$ si $g(x)=x^2-1$. Sa se calculeze derivatele f', g', (f*g)' si (f/g)'.

Cod MATLAB:

```
F=[2 0 1 1];
G=[1 0 -1];
Fd=polyder(F)
Gd=polyder(G)
FGd=polyder(F,G)
[D E]=polyder(F,G)
```

va rezulta:

```
Fd=[6 0 1]
Gd=[2 0]
FGd=[10 0 -3 2 -1]
D=[ 2 0 -7 -2 -1]
E=[1 0 -2 0 1]
```

7.3 Probleme rezolvate

Problema 7.3.1

Creați o matrice A de dimensiune 5×5 cu numere aleatoare între 0 și 10. Extrageți submatricea B formată din liniile 2–4 și coloanele 3–5. Calculați pentru noua matrice extrasă valorile minime și maxime, mediile pe coloane și pe linii.

Rezolvare propusă:

```
%Cod MATLAB:
clear;clc
% randi([0,10],5,5) generează o matrice 5×5. rng(42) fixează
"seed-ul"
% generatorului pentru a obține aceleași valori la fiecare
rulare
A = randi([0,10],5,5) % matrice 5x5 cu valori intregi 0..10
disp('Matricea A ='); disp(A)

% extragem liniile 2..4 si coloanele 3..5
B = A(2:4, 3:5);
disp('Submatricea B = A(2:4,3:5):');
disp(B)
%B(:) transformă matricea într-un vector coloana, util pentru
min/max globale.
% mean(B,1) calculează media pe fiecare coloană; mean(B,2) pe
fiecare linie.
min_B = min(B(:));      % minimul global
max_B = max(B(:));      % maximul global
```

```
mean_col = mean(B,1);   % medii pe coloane (1x3)
mean_row = mean(B,2);   % medii pe linii (3x1)

fprintf('Min(B)= %d, Max(B)= %d\n', min_B, max_B);
fprintf('Medii pe coloane: %g %g %g\n', mean_col);
fprintf('Medii pe linii:\n'); disp(mean_row);
```

Dupa execuție obținem:

```
A =

     8     9     3     1     5
     8     6     8     7     5
     0     3     7     8     4
     3     0     9     6     0
     1     3     5     8     1

Matricea A =
     8     9     3     1     5
     8     6     8     7     5
     0     3     7     8     4
     3     0     9     6     0
     1     3     5     8     1

Submatricea B = A(2:4,3:5):
     8     7     5
     7     8     4
     9     6     0
```

```
Min(B)= 0, Max(B)= 9
Medii pe coloane: 8 7 3
Medii pe linii:
    6.6667
    6.3333
    5.0000
```

Problema 7.3.2

Construiți două matrice A și B de dimensiune 2×3 și concatenați-le pe verticală într-o matrice C de dimensiune 4×3. Calculați media fiecărei coloane a lui C și reprezentați-o grafic cu un bar chart. Rotiți matricea C cu 90° și afișați rezultatul.

Rezolvare propusă:

```
clear;clc
A = [1 2 3; 4 5 6];
B = [7 8 9; 10 11 12];
%Concatenarea pe verticală
C = [A; B];
disp('Matricea C (4x3) = [A;B]:');
disp(C)
%Calculul mediilor pe coloane și reprezentarea grafică
mean_col = mean(C,1);  % media pe coloane
fprintf('Medii pe coloanele lui C: %g %g %g\n', mean_col);
% mean(C,1) returnează vectorul cu media fiecărei coloane.
% bar reprezintă grafic valorile.
figure;
bar(1:3, mean_col);
```

```
xlabel('Coloana');
ylabel('Media');
title('Media pe coloane a matricei C');
%Rotirea matricei C cu 90°
C90 = rot90(C);
disp('Matricea C rotita 90°:'); disp(C90)
```

Dupa execuție obținem:

```
Matricea C (4x3) = [A;B]:
     1     2     3
     4     5     6
     7     8     9
    10    11    12

Medii pe coloanele lui C: 5.5 6.5 7.5
Matricea C rotita 90°:
     3     6     9    12
     2     5     8    11
     1     4     7    10
```

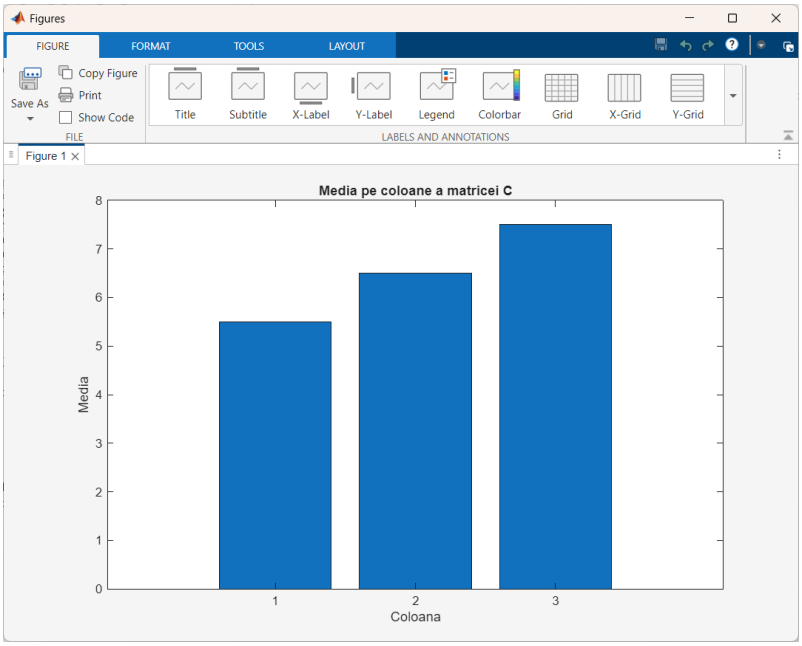


Figura 7.1 Reprezentare grafică corespunzătoare problemei 7.3.2

Problema 7.3.3

Se consideră o matrice A de ordinul 4.

1. Extrageți diagonala principală a lui A într-un vector d.
2. Creați o matrice diagonală cu elementele lui d.
3. Extrageți partea inferioară triunghiulară (sub diagonală) și partea superioară triunghiulară (deasupra diagonalei).
4. Întoarceți matricea A stânga–dreapta (`fliplr`) și sus–jos (`flipud`).
5. Reprezentați grafic (heatmap) rezultatele pentru a vizualiza efectul fiecărei funcții.

Rezolvare propusă:

```
% clear;clc
A = [4 7 2 5;
     1 9 8 3;
     6 0 5 1;
     2 4 7 3];
disp('Matricea A:');
disp(A)

d = diag(A);           % diagonala
disp('Diagonala principala a lui A:'); disp(d')

D = diag(d);           % matrice diagonala
disp('Matricea diagonala D:'); disp(D)

L = tril(A);           % partea inferioara
disp('Partea inferioara a lui A (L):'); disp(L)

U = triu(A);           % partea superioara
disp('Partea superioara a lui A (U):'); disp(U)

A_lr = fliplr(A);      % inversare stanga-dreapta
disp('A întoarsă stânga–dreapta:'); disp(A_lr)

A_ud = flipud(A);      % inversare sus-jos
disp('A întoarsă sus–jos:'); disp(A_ud)
```

După execuție obținem:

Matricea A:

4 7 2 5

1 9 8 3

6 0 5 1

2 4 7 3

Diagonala principala a lui A:

4 9 5 3

Matricea diagonala D:

4 0 0 0

0 9 0 0

0 0 5 0

0 0 0 3

Partea inferioara a lui A (L):

4 0 0 0

1 9 0 0

6 0 5 0

2 4 7 3

Partea superioara a lui A (U):

4 7 2 5

0 9 8 3

0 0 5 1

0 0 0 3

A întoarsă stânga–dreapta:

5	2	7	4
3	8	9	1
1	5	0	6
3	7	4	2

A întoarsă sus–jos:

2	4	7	3
6	0	5	1
1	9	8	3
4	7	2	5

Problema 7.3.4

Fie polinomul:

$f(x)=x^4 + x^2 + 2*x + 3$

1. Reprezentati grafic f(x) pe intervalul [−3,3].
2. Determinați rădăcinile (reale și complexe) ale polinomului și reprezentați-le în planul complex.
3. Calculați derivata f'(x) folosind *polyder* și găsiți extremele (punctele critice) numeric. Reprezentați-le pe grafic.
4. Verificați identitatea numeric: evaluarea polinomului cu *polyval* vs. calcul direct.

5. Exemplu aplicativ: generați date zgomotoase obținute din f(x) pe [-2,2] și aproximați-le cu un polinom de grad 3 folosind polyfit. Comparați graficele (date, polinom adevărat, aproximație).
6. (Opțional) Descompuneți polinomul în factori (folosind roots pentru a obține factorii linari) și reconstruiți-l cu *poly*.

```
% problema_polinoame.m
% Definire polinom f(x) = x^4 + x^2 + 2x + 3
p = [1 0 1 2 3]; % coeficienti descrescatori

% domeniu si evaluare
x = linspace(-3,3,401);
y = polyval(p, x);

% plot
figure;
plot(x,y,'-','LineWidth',1.2); hold on;
xlabel('x'); ylabel('f(x)');
title('f(x) = x^4 + x^2 + 2x + 3');
grid on;

r = roots(p);
disp('Radacinile polinomului:'); disp(r);

% reprezentare in plan complex
figure;
plot(real(r), imag(r), 'o', 'MarkerFaceColor','b');
xlabel('Re'); ylabel('Im'); grid on; axis equal;
title('Radacinile polinomului in planul complex');
```

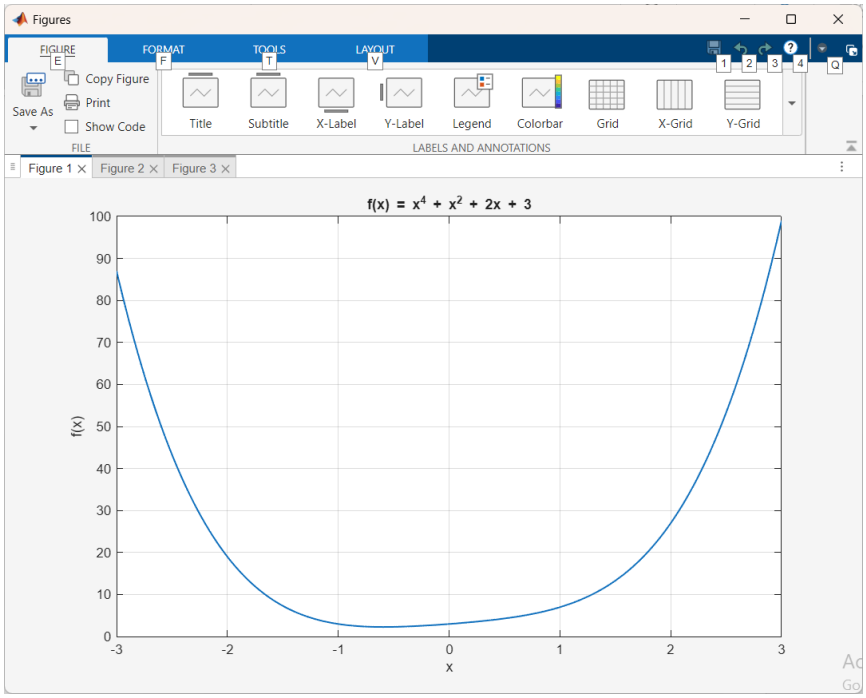
```
pd = polyder(p);           % coeficientii lui f'(x)
crit = roots(pd);          % puncte critice (pot fi complexe)
crit_real = crit(imag(crit)==0); % păstrăm doar reale pentru
plot
ycrit = polyval(p, crit_real);

% afisare
disp('Coeficientii derivatii:'); disp(pd);
disp('Punctele critice (reale):'); disp(crit_real);
disp('Valori f(crit):'); disp(ycrit);

% onto the plot from step 1, add critical points
plot(crit_real, ycrit, 'ro', 'MarkerFaceColor','r');
legend('f(x)','Puncte critice reale');



x_test = [-2, 0, 1.5];
y_polyval = polyval(p, x_test);
y_direct = x_test.^4 + x_test.^2 + 2*x_test + 3;
disp(table(x_test', y_polyval', y_direct', 'VariableNames',
{'x','polyval','direct'}));

% date zgomotoase
xdata = linspace(-2,2,20);
ytrue = polyval(p, xdata);
ydata = ytrue + 2*randn(size(xdata)); % zgomot gaussian
sigma=2

% fit grad 3
```

```
deg = 3;
coeff_fit = polyfit(xdata, ydata, deg);
yfit = polyval(coeff_fit, xdata);

% plot
figure;
scatter(xdata, ydata, 'filled'); hold on;
plot(xdata, ytrue, 'k--','LineWidth',1);
plot(xdata, yfit, 'r-','LineWidth',1.2);
legend('Date zgomotoase','Polinom adevarat','Aproximare
polyfit grad 3');
xlabel('x'); ylabel('y');
title('Ajustare polinomiala - exemplu');
grid on;

D%r=roots(p), putem reconstrui polinomul cu poly(r) și
compara cu vectorul inițial p.
r = roots(p);
p_recon = poly(r);  % va returna coeficientii (posibil up to
numeric rounding)
disp('Coeficienti originali:'); disp(p);
disp('Coeficienti reconstructie (din radacini):');
disp(p_recon);
%poly(roots(p)) trebuie să revină la același vector
```

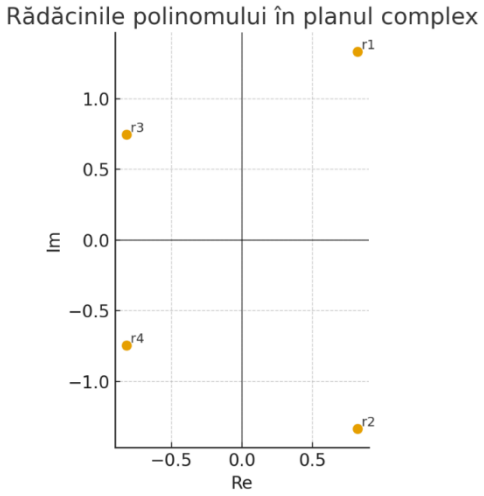
Dupa execuție obținem:

```
Radacinile polinomului:
   0.8176 + 1.3342i
   0.8176 - 1.3342i
```

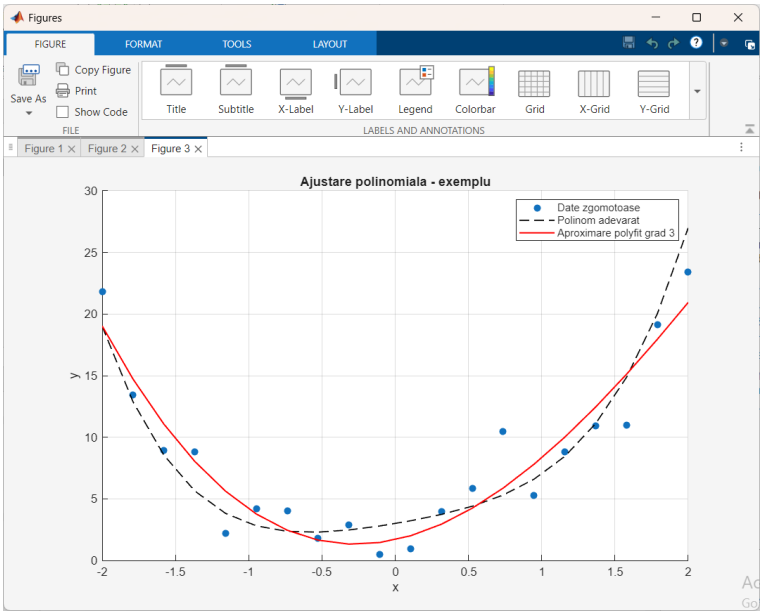
```
  -0.8176 + 0.7462i
  -0.8176 - 0.7462i

Coeficientii derivatii:
     4     0     2     2

Punctele critice (reale):
   -0.5898

Valori f(crit):
    2.2893
```



(a)



(b)



(c)

Figura 7.2 (a, b, c) Reprezentări grafice corespunzătoare problemei 7.3.4

Capitolul 8

Grafică în MATLAB

Mediul de lucru MATLAB conține funcții integrate care permit generarea de diagrame tip bară, grafice x-y, polare, de contur și 3D. MATLAB permite, de asemenea, adăugarea de titluri la grafice, etichetarea axelor x și y și adăugarea unei grile pe grafice. În plus, există comenzi pentru controlul ecranului și al scalării. In tabelul de mai jos este reprezentată o listă a funcțiilor grafice integrate în MATLAB. Se poate folosi funcția de ajutor (*help*), pentru a obține mai multe informații despre funcțiile grafice.

Funcție	**Descriere**
axis	fixează limitele axelor
bar	creează un grafic tip bară
contour	realizează diagrame de contur
grid	adaugă o grilă pe grafic
gtext	inserează text poziționat cu ajutorul mouse-ului
hist	generează un grafic tip histogramă
fplot	Reprezentare 2D a unei functii

Funcție	Descriere
hold	menține graficul curent (pentru suprapunerea altor grafice)
loglog	creează un grafic pe scala logaritmica pe ambele axe Ox si Oy
mesh	realizează un grafic 3D tip plasă (mesh)
meshgrid	creează domeniul pentru un grafic 3D tip plasă
pause	introduce o pauză între grafice
plot	realizează un grafic liniar x-y
polar	realizează un grafic polar
pie	reprezentare 2D de tip pie
semilogx	creează un grafic semilog x-y (axa x logaritmică)
semilogy	creează un grafic semilog x-y (axa y logaritmică)
stairs	creează un grafic cu trepte
stem	reprentare 2D pentru seturi de date discrete
text	plasează text într-o poziție specificată pe grafic
title	adaugă un titlu graficului
xlabel	etichetează axa x
ylabel	etichetează axa y

8.1 Componentele unei ferestre grafice

În MATLAB, o fereastră grafică (figure window) conține mai multe componente importante care permit afișarea și manipularea graficelor. Principalele componente sunt:

1. Figure – fereastra grafică propriu-zisă.
2. Axes – zona în care se desenează graficul.
3. Menu bar – bara de meniu (File, Edit, View…).
4. Toolbar – butoane pentru zoom, pan, rotate etc.
5. Title, labels, legend – titlu, etichete de axe și legendă (dacă sunt adăugate).

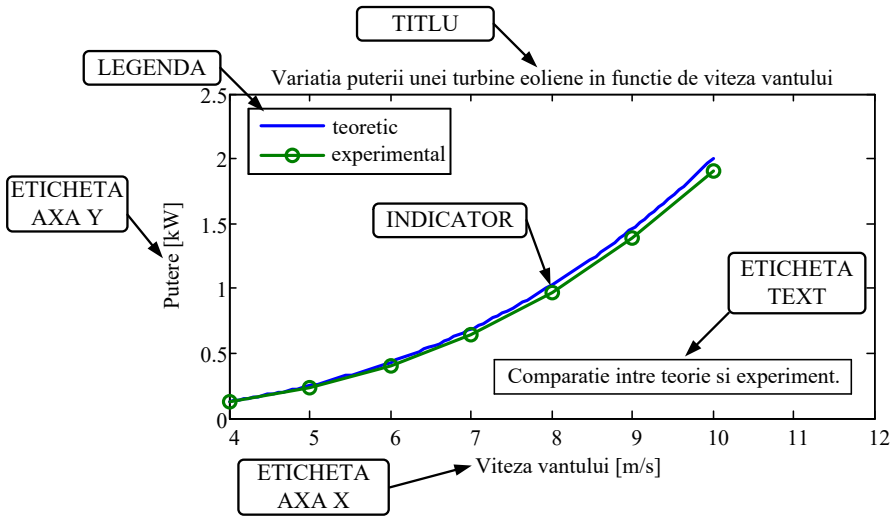


Figura 8.1 Principalele componente ale unei ferestre grafice MATLAB

8.2 Funcția *plot*

Funcția *plot* permite trasarea graficelor 2D și are diferite forme, in funcție de argumentele de intrare. Cea mai simplă formă este următoarea:

plot(x,y)

unde *x* și *y* sunt doi vectori de aceeași dimensiune.

Exemplu:

```
clear
clc
x=linspace(0,2*pi);
y=sin(x);
plot(x,y);
```

Va rezulta următorul grafic:

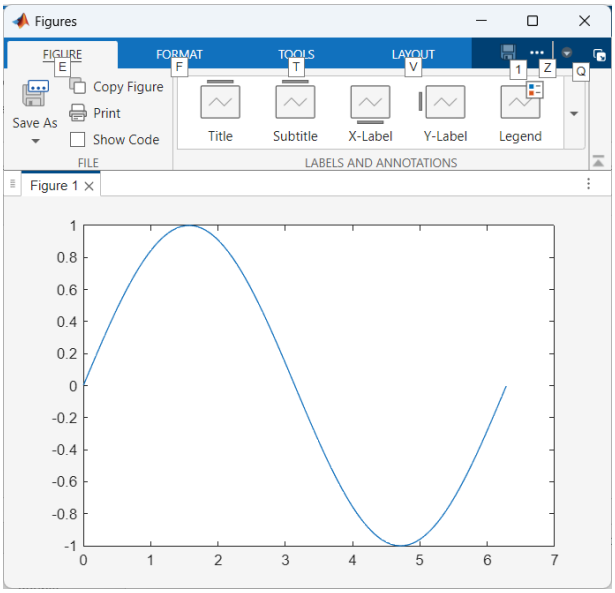


Figura 8.2 Graficul 2D al funcției $y = \sin(x)$generat cu comanda plot pentru exemplul de mai sus

Comanda **plot** generează un grafic liniar x-y. Există trei variante ale comenzii **plot**:

(a) plot(x)
(b) plot(x, y)
(c) plot(x1, y1, x2, y2, x3, y3, ..., xn, yn)

plot(x) va produce un grafic liniar al elementelor vectorului **x** în funcție de indicele elementelor din **x**. MATLAB va conecta punctele prin linii drepte.

Dacă **x** este un vector, fiecare coloană va fi trasată ca o curbă separată pe același grafic. De exemplu, dacă:

```
x = [0 3 6.5 6.4 1.8 7.9];
plot(x)
```

atunci plot(x) va genera graficul:

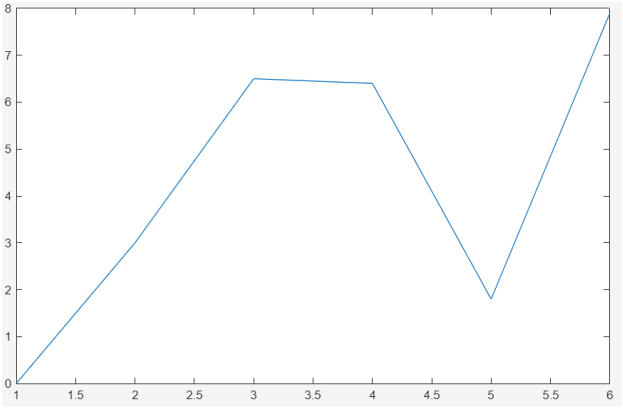


Figura 8.3 Graficul elementelor vectorului x în funcție de indicele lor după execuția exemplului de mai sus

Vectorii **x** și **y** trebuie să fie de aceeași lungime, iar comanda plot(x, y) reprezintă grafic, în plan cartezian, elementele lui x (axa orizontală) în raport cu elementele lui y (axa verticală). De exemplu, comenzile MATLAB:

```
t = 0:1.5:16;
y = 3*exp(t);
plot(t, y)
```

generează graficul din figura de mai jos.

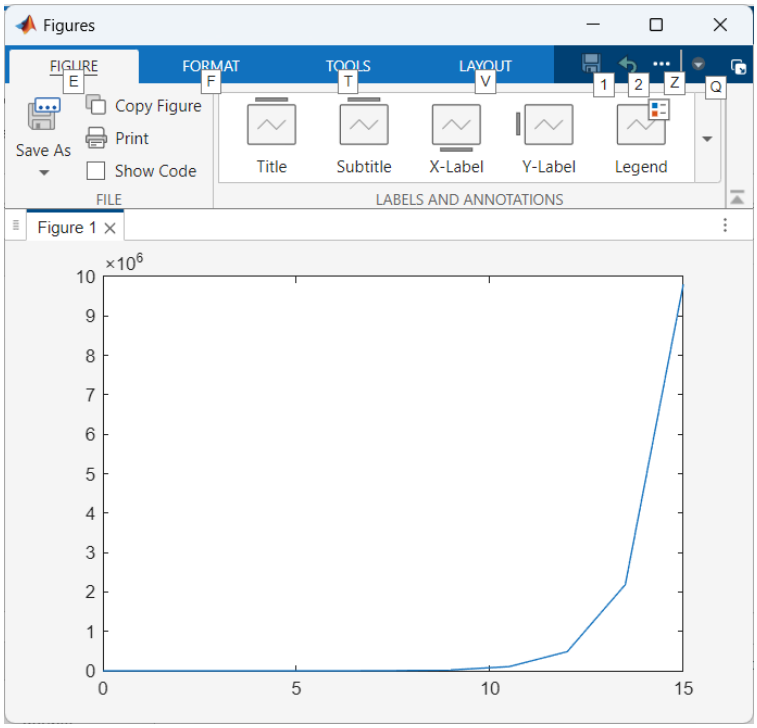


Figura 8.4 Graficul funcției $y = 3e^t$ reprezentat cu comanda plot(t,y)

Pentru a reprezenta mai multe curbe pe același grafic, se poate folosi comanda *plot* cu mai multe argumente, de exemplu:

```
plot(x1, y1, x2, y2, x3, y3, ..., xn, yn)
```

Variabilele x1, y1, x2, y2, x3, y3, ..., xn, yn sunt perechi de vectori. Fiecare pereche (x, y) este reprezentată grafic, generând mai multe linii pe același grafic. Comanda *plot* de mai sus permite afișarea vectorilor de lungimi diferite pe același grafic. MATLAB scalează automat graficele. De asemenea, graficul rămâne graficul curent până când se generează un alt grafic; în acest caz, graficul vechi este șters. Comanda *hold* menține graficul curent pe ecran și împiedică ștergerea și rescalarea acestuia. Comenzile *plot* ulterioare vor suprapune graficele noi peste curbele originale. Comanda *hold* rămâne activă până când este emisă din nou.

8.2.1 Argumente opționale ale funcției *plot*

```
plot(x,y,'line specifiers','PropertyName', PropertyValue)
```

În această comandă, *line specifiers'* furnizează tipul si culoarea liniei de reprezentare grafică, astfel:

Tipul liniei	**Specificator**
Continua (implicit)	‚-'
Intrerupta	‚--'
puncte	‚:'
Linie-punct	‚-.'

Culoare linie	**Specificator**	**Culoare linie**	**Specificator**
Rosu (**r**ed)	‚r'	**M**agenta	‚m'
Verde (**g**reen)	‚g'	Galben (**y**ellow)	‚y'
Albastru (**b**lue)	‚b'	Negru (blac**k**)	‚k'
Cyan	‚c'	Alb (**w**hite)	‚w'

Tip indicator	**Specificator**	**Tip indicator**	**Specificator**
Plus	‚+'	patrat	‚s'
Cerc	‚o'	romb	‚d'
Steluta	‚*'	Stea cu 5 colturi	‚p'
punct	‚.'	Stea cu 6 colturi	‚h'
x	‚x'		

Exemple:

`plot(x,y)`	Linie albastra continua (implicit) fara indicator
`plot(x,y,'r')`	Linie rosie continua fara indicator
`plot(x,y,'--g')`	Linie verde intrerupta fara indicatori
`plot(x,y,'*')`	Punctele *x* si *y* sunt marcate cu indicatorul ‚*', fara nicio linie
`plot(x,y,'g:d')`	Linie verde punctata, iar punctele sunt marcate cu romburi.

Argumentele *PropertyName* si *PropertyValue* pot fi utilizate pentru a specifica grosimea liniei, dimensiunea indicatorului, culoarea liniei si interiorului indicatorului.

PropertyName	**Descriere**	***PropertyValue***
‚LineWidth' (‚linewidth')	Specifică grosimea liniei de reprezentare	Un numar reprezentand grosimea in puncte (implicit 0.5)
‚MarkerSize' (‚markersize')	Specifică dimensiunea indicatorului	Un numar reprezentand dimensiunea in puncte
‚MarkerEdgeColor' (‚markeredgecolor')	Specifică culoarea indicatorului, sau culoarea marginii indicatorului	Una din optiunile de culoare din tabelul anterior
‚MarkerFaceColor' (‚markerfacecolor')	Specifică culoarea de umplere a indicatorului	Una din optiunile de culoare din tabelul anterior

Exemplu:

```
plot(x,y,'-ro',
'linewidth',2,'markersize',12,'MarkerEdgeColor','g','Mar
kerFaceColor','y')
```

Comanda va reprezenta punctele corespunzatoare vectorilor *x* si *y* cu linie continuă de culoare roșie si grosime 12, indicator tip cerc cu marginea de culoare verde si culoarea de umplere galbenă.

8.3 Reprezentarea unei funcții utilizand comanda „fplot”

O altă metodă pentru reprezentarea grafică a funcțiilor matematice este utilizarea comenzii *fplot*, cu sintaxa următoare:

```
fplot(functia, limite, ' line specifiers')
```

In comanda de mai sus `functia` poate fi scrisă direct in format text, de exemplu:

```
'3.5^(-0.5*x)*cos(6*x)'
```

sau sub forma unei funcții (de preferat):

```
@(x)3.5^(-0.5*x)*cos(6*x)
```

Argumentul *„limite”* reprezintă un vector cu două elemente [xmin, xmax] care specifică intervalul de pe axa Ox de reprezentare a funcției sau un vector cu patru elemente [*xmin*,*xmax*,*ymin*,*ymax*] care specifică intervalele de reprezentare a funcției atat pe axa Ox cat și pe axa Oy.

Exemplu:

Afișați grafic funcția $f(x) = 3.5^{-0.5x}\cos(6x)$, definită in intervalul $-2 \le x \le 4$.

Pentru rezolvare, folosim instrucțiunea:

```
fplot('3.5^(-0.5*x)*cos(6*x)',[-2 4])
```

sau (de preferat):

```
fplot(@(x)3.5^(-0.5*x)*cos(6*x),[-2 4])
```

Va rezulta următorul grafic:

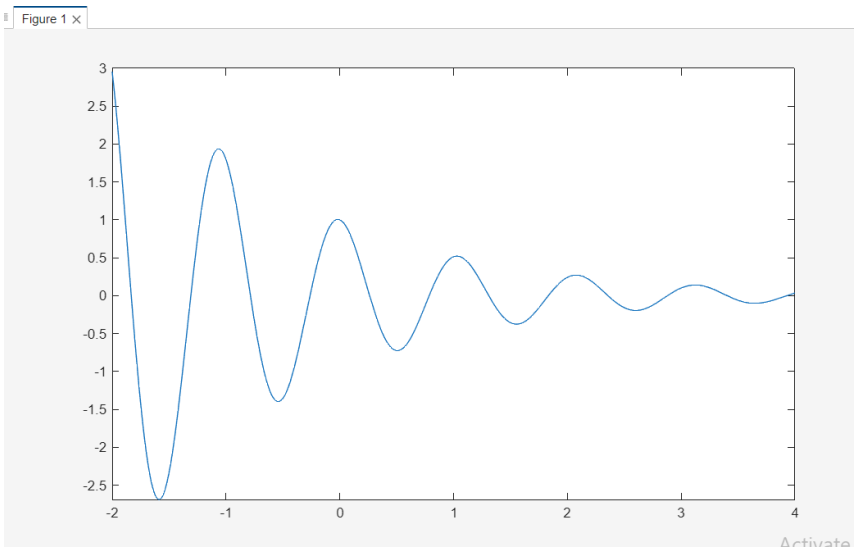


Figura 8.5 Reprezentarea grafică a funcției $3.5^{-0.5x}\cos(6x)$cu comanda `fplot`

8.4 Reprezentarea mai multor grafice în aceeași fereastră

Funcția plot poate fi apelată cu mai multe perechi de vectori (x, y) astfel încât să fie reprezentate mai multe funcții pe același grafic.

```
plot(x,y,u,v,t,h)
```

Vectorii pereche trebuie să aibă aceleași dimensiuni.

Exemplu:

```
clear;clc
x=linspace(-2,4);
y1=3*x.^2-10*x;
y2=x-5;
plot(x,y1,x,y2);
```

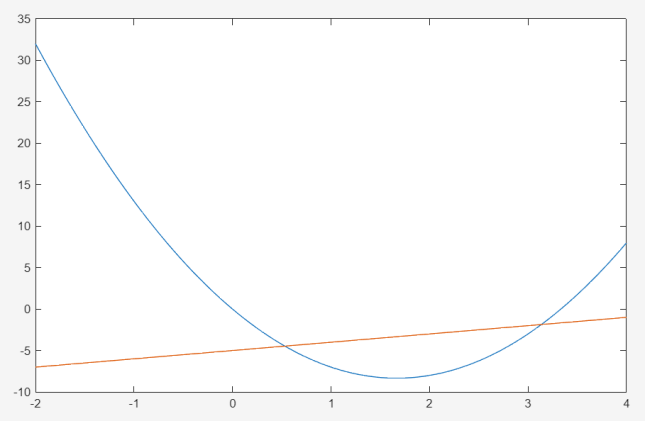


Figura 8.6 Reprezentarea pe același grafic a funcțiilor $y_1 = 3x^2 - 10x$ și $y_2 = x - 5$

Comenzile *hold on / off / all*

Comanda **hold on** menține graficul curent, permițând suprapunerea altor grafice peste acesta. Comanda **hold all** menține ciclul culorilor și stilul liniei de reprezentare grafică; este de preferat în multe situații în loc de hold on.

Comanda **hold off** resetează sistemul de axe. Din acest punct, orice apelare a funcției plot conduce la ștergerea graficelor precedente.

```
x=linspace(-2,4);
y1=3*x.^2-10*x;
y2=x-5;
plot(x,y1);
hold on; %mentine graficul curent
%hold all; % mentine ciclul culorilor si stilului liniei
%de reprezentare grafica; de preferat in multe situatii
%in loc de hold on
plot(x,y2);
%pot urma si alte apelari ale functiei plot
hold off %reseteaza sistemul de axe
%din acest punct orice apelare a functiei plot
%conduce la stergerea graficelor precedente
```

8.5 Funcția *line*

O metodă alternativă pentru reprezentarea graficelor multiple constă în utilizarea funcției line, cu sintaxă aproape identică cu cea a funcției plot. Spre deosebire de plot, apelarea funcției line succesiv nu va șterge graficele precedente din fereastra activă de reprezentări grafice.

Sintaxa:

```
line(x,y,'PropertyName',PropertyValue)
```

8.6 Reprezentarea mai multor subferestre grafice într-o fereastră grafică

Funcția folosită pentru reprezentarea mai multor grafice într-o singură fereastră grafică, organizate în grilă (rânduri × coloane), se numește **subplot** și are o sintaxă de bază.

```
subplot(m, n, p);
```

unde: **m** – numărul de rânduri în grilă
n – numărul de coloane în grilă
p – poziția curentă (numărul subgraficului, numărat de la stânga la dreapta, de sus în jos)

După apelarea funcției *subplot*, orice comandă de plotare (plot, bar, mesh etc.) va desena graficul în subgraficul indicat de p. Utilizarea funcției subplot permite compararea mai multor grafice într-o singură fereastră și fiecare subgrafic poate avea titlu, etichete, legendă și grilă independent.. De exemplu, *subplot(3,2,1)* crează 6 subferestre aranjate ca in doua trei linii si două coloane si activeaza prima fereastra.
Exemplu:

```
x=linspace(0,2*pi);
y1=sin(x); y2=cos(x); y3=sin(x).^2;
subplot(3,1,1);
plot(x,y1);
subplot(3,1,2);
plot(x,y2);
subplot(3,1,3);
plot(x,y3);
```

După execuție va rezulta graficul:

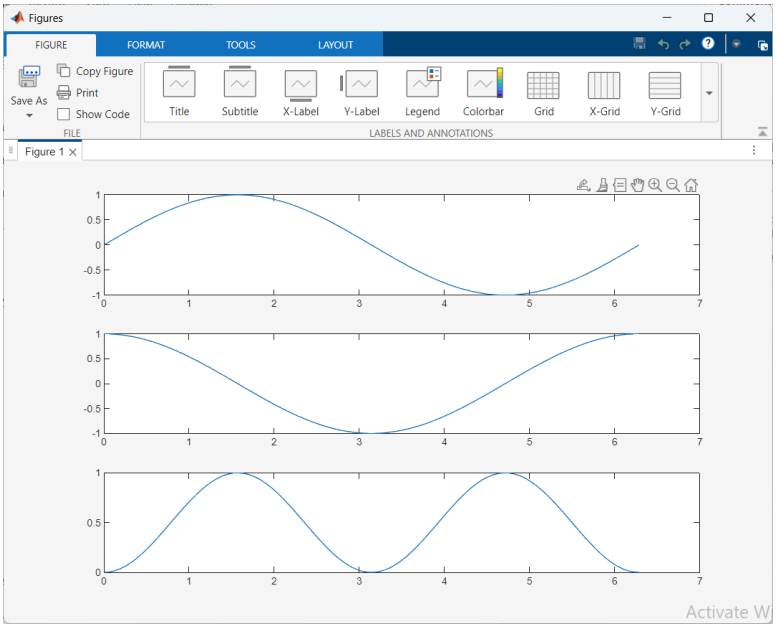


Figura 8.7 Reprezentarea în subgrafice a funcțiilor $\sin(x)$, $\cos(x)$și $\sin^2(x)$folosind `subplot`

8.7 Editarea unui grafic

Când un grafic este trasat, putem adăuga o grilă, un titlu, o etichetă și axe x și y astfel: inserarea etichetelor axelor Ox și Oy se poate face cu funcțiile dedicate pentru etichetarea axelor.

```
xlabel('eticheta axa Ox');
ylabel('eticheta axa Oy');
```

- Inserarea unui titlu se poate face cu comanda:

```
title('titlul graficului');
```

- Inserarea unui anumit text într-o zonă a graficului se poate face specificând coordonatele punctului unde va fi plasat textul.

```
text(x,y,'text');
```

unde *x* si *y* reprezinta coordonatele punctului din grafic unde va fi plasat textul.

- Specificarea unui anumit interval pentru reprezentarea grafică pe axele Ox și Oy poate fi făcută prin definirea limitelor pentru fiecare axă.

```
axis([xmin,xmax,ymin,ymax]);
```

Pentru inserarea sau eliminarea unui caroiaj (rețea de linii orizontale și verticale) în grafic se utilizează comenzile pentru activarea sau dezactivarea grilei, `grid on` si `grid off`.

Problemă rezolvată

Pentru un circuit R-L, tensiunea **v(t)** și curentul **i(t)** sunt definite astfel:

$$v(t) = 20\cos(500t)$$

$$i(t) = 8\cos(500t - 45^\circ)$$

Schițați **v(t)** și **i(t)** pentru intervalul $t = 0$ până la 20 milisecunde.

Rezolvare propusă

Cunoaștem următoarele:

Amplitudinea tensiunii: 20 V

Amplitudinea curentului: 8 A

Pulsația: 500 rad/s

Defazajul curentului: −45° (curentul rămâne în urmă — situație tipică în circuite R-L)

Conversia în radiani:

$-45° = -\pi/4$

Intervalul de timp:

$0\ s \leq t \leq 0.02\ s$

După execuție se obține graficul corespunzător.

```
clear;clc
% cod MATLAB
% Intervalul de timp
t = linspace(0, 0.02, 1000);

% Semnalele
v = 20 * cos(500 * t);
i = 8 * cos(500 * t - pi/4);   % -45° în radiani

% Reprezentare grafică
plot(t, v, 'b', 'LineWidth', 1.5); hold on;
plot(t, i, 'r', 'LineWidth', 1.5);

grid on;
xlabel('Timp (s)');
ylabel('Amplitudine');
title('Semnalele v(t) și i(t) pentru t = 0 ... 20 ms');
legend('v(t)', 'i(t)');
```

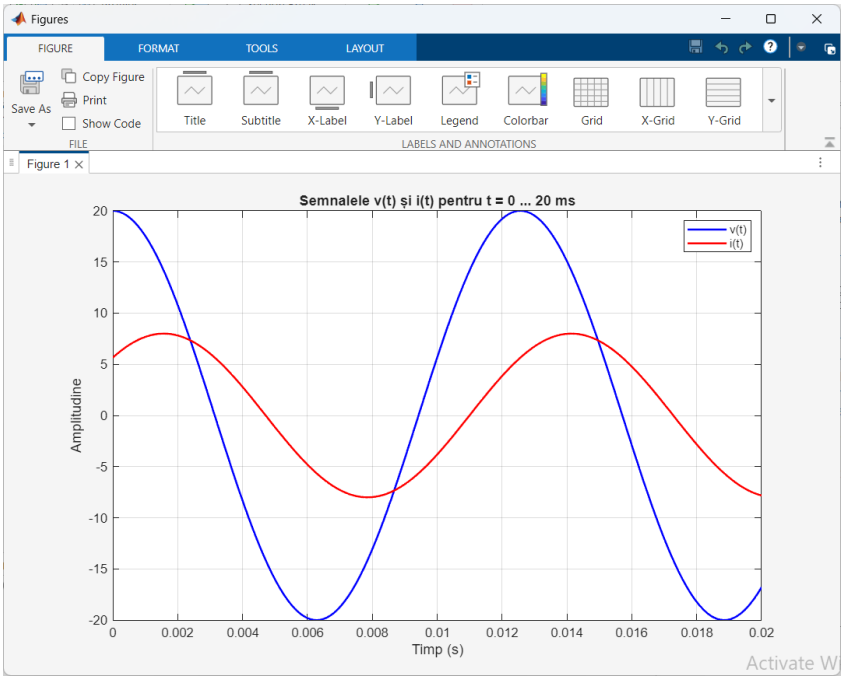


Figura 8.8 Reprezentarea tensiunii $v(t)$și a curentului $i(t)$într-un circuit R-L

8.8 Grafice logaritmice și polare

Graficele logaritmice și semi-logaritmice pot fi generate folosind comenzile *loglog*, *semilogx* și *semilogy*. Apelarea acestor funcții este asemănătoare cu cea a comenzii *plot*, discutată în secțiunea anterioară. Descrierea lor este următoarea:

loglog(x, y) – reprezintă un grafic în care pe ambele axe se reprezintă logaritmul în baza 10, adică $\log_{10}(x)$ versus $\log_{10}(y)$

semilogx(x, y) – generează un grafic cu axa x în scară logaritmică ($\log_{10}(x)$) și axa y în scară liniară

semilogy(x, y) – reprezintă un grafic cu axa x liniară și axa y în scară logaritmică ($\log_{10}(y)$)

Este important de menționat că logaritmul numerelor negative sau al valorii zero nu există, astfel încât datele reprezentate pe aceste tipuri de grafice nu trebuie să conțină valori negative sau zero.
O reprezentare polară care ilustrează dependența dintre un unghi și o mărime poate fi realizată cu ajutorul comenzii:

```
polar(theta, rho)
```

În această comandă, *theta* și *rho* sunt vectori: *theta* conține valorile unghiurilor în radiani, iar *rho* reprezintă valorile corespunzătoare ale magnitudinii.

Problemă rezolvată:

Un număr complex **z** este exprimat în forma polară:

$$z = r\, e^{j\theta}$$

Puterea **n** a acestui număr este dată de relația:

$$z^n = r^n e^{jn\theta}$$

Considerând **r = 0.8** și **θ = 45 grade**, realizați un **grafic polar** care să reprezinte valorile lui z^n în funcție de unghiul **nθ**, pentru **n = 1** până la **n = 24**.

Rezolvare propusă:

```
clear;clc
```

```
% Datele problemei
r = 0.8;
theta = 45 * pi/180;    % conversie in radiani

n = 1:24;               % valori pentru n

% Calculul magnitudinii și al unghiului
rho = r.^n;             % r^n
angles = n * theta;     % n*theta

% Reprezentarea grafică polară folosind comanda polar
polar(angles, rho, 'o-');
title('Reprezentarea polara a lui z^n folosind comanda
polar');
```

După execuție va rezulta graficul din figura următoare.

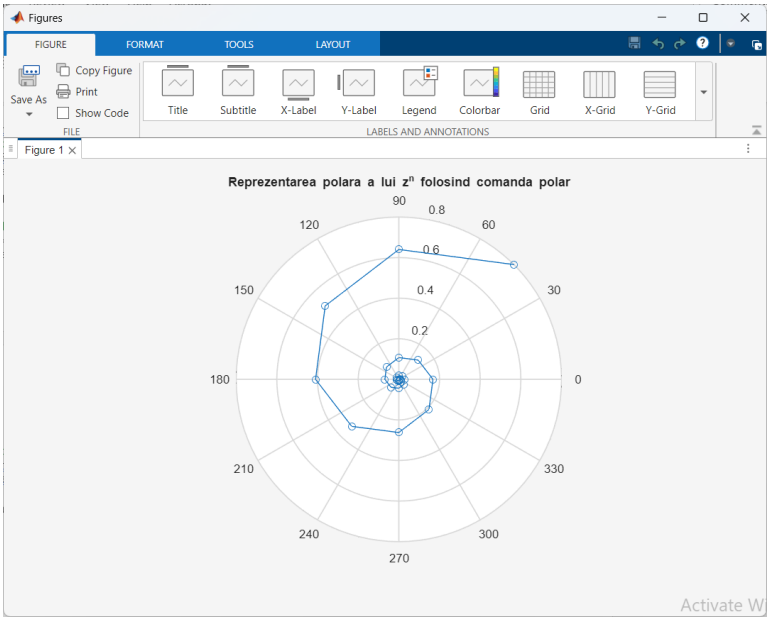
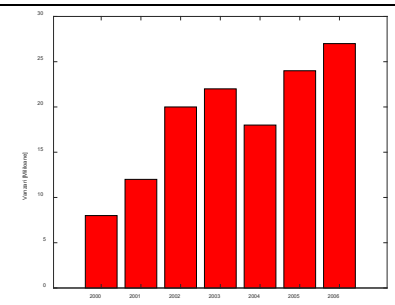
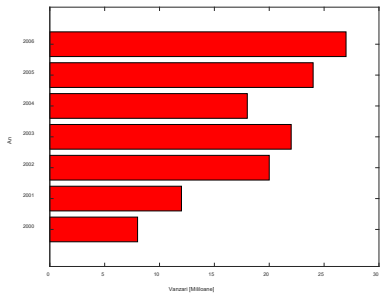


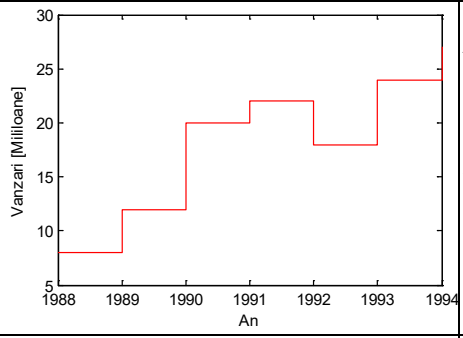
Figura 8.9 Reprezentarea polară a valorilor lui z^npentru $n = 1..24$

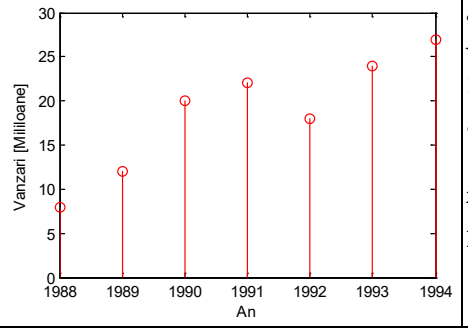
8.9 Reprezentări speciale de grafice

Functia MATLAB	Exemplu grafic	Exemplu cod MATLAB

`bar(x,y)`		`an=[2000:2006];`
		`vz=[8 12 20 22 18 24 27];`
		`bar(an,vz,'r');`
		`xlabel('An');`
		`ylabel('Vanzari [Mililoane]');`

`barh(x,y)`		`an=[2000:2006];`
		`vz=[8 12 20 22 18 24 27];`
		`barh(an,vz,'r');`
		`ylabel('An');`
		`xlabel('Vanzari [Mililoane]');`

`stairs(x,y)`		`an=[1988:1994];`
		`vz=[8 12 20 22 18 24 27];`
		`stairs(an,vz);`
		`xlabel('An');`
		`ylabel('Vanzari [Mililoane]');`

`stem(x,y)`		`an=[1988:1994];`
		`vz=[8 12 20 22 18 24 27];`
		`stem(an,vz,'r');`
		`xlabel('An');`
		`ylabel('Vanzari [Mililoane]');`

Funcția *pie*

Funcția **pie** este folosită pentru a crea diagrame circulare (pie charts)**,** adică grafice care reprezintă proporții ale unui întreg sub formă de felii ale unui cerc. Sintaxa de baza este *pie(X)* unde X este un vector numeric, fiecare element din vector reprezintă mărimea unei felii din diagram iar MATLAB calculează automat proporțiile și le afișează în diagramă.

Exemplu:

O companie produce patru tipuri de produse: A, B, C și D. Vânzările în luna curentă, exprimate în unități, sunt următoarele:

Produs	Vânzări (unități)
A	40
B	20
C	15
D	25

-Reprezentați grafic vânzările folosind o diagramă circulară (pie chart) în MATLAB.

-Adăugați etichete corespunzătoare fiecărui produs pe diagramă.

-Adăugați un titlu sugestiv pentru diagramă.

Rezolvare propusă

```
clear;clc
%cod Matlab
```

```
data = [40 20 15 25];
labels = {'Produs A', 'Produs B', 'Produs C', 'Produs D'};

pie(data, labels)
title('Distribuția vânzărilor pe produse')
```

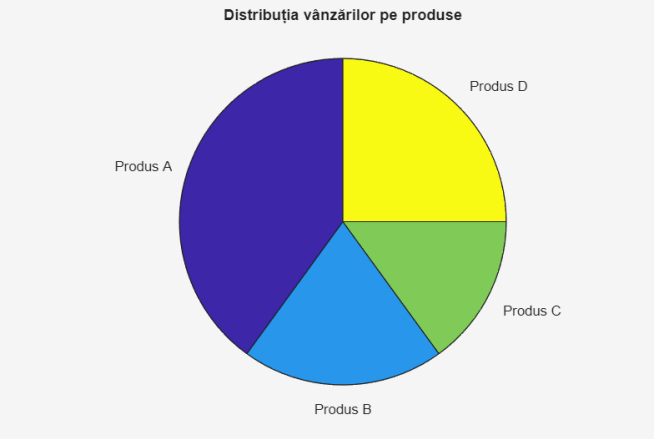


Figura 8.10 Diagramă circulară (pie chart) a distribuției vânzărilor pe produse

8.10 Histograme

Histograma este o reprezentare grafică a distribuției valorilor dintr-un set de date. Setul de date este împărțit în subdomenii în care sunt grupate valorile din setul respectiv. Graficul rezultat reprezintă numărul de valori din fiecare subdomeniu. Numărul de subdomenii poate fi specificat. Sintaxa de bază este: histogram(X), unde X este vectorul de date. Mai există versiunea histogram(X, n), unde n este numărul de intervale (bins) dorit.

Sintaxa:

```
histogram(X, 'BinWidth', w)
```

implică un argument nou, w, ce reprezintă lățimea fiecărui interval.

Problemă rezolvată:

O stație meteorologică a înregistrat temperaturile zilnice (în °C) pe parcursul unei luni de 30 de zile. Valorile temperaturilor sunt următoarele (exemplu):
T = [15, 17, 16, 18, 19, 21, 20, 18, 17, 16, 15, 16, 17, 18, 19, 20, 21, 22, 23, 21, 20, 19, 18, 17, 16, 15, 16, 17, 18, 19];
- Reprezentați aceste temperaturi folosind o **histogramă** în MATLAB.
- Configurați histograma astfel încât să afișeze intervalele (bins) de 1°C.
- Adăugați **titlu** și **etichete pentru axe** corespunzătoare: „Temperatura (°C)” pe axa X și „Număr de zile” pe axa Y.

Rezolvare propusă:

```
clear;clc
% Vectorul temperaturilor
T = [15, 17, 16, 18, 19, 21, 20, 18, 17, 16, 15, 16, 17, 18,
19, 20, 21, 22, 23, 21, 20, 19, 18, 17, 16, 15, 16, 17, 18,
19];

% Crearea histogramei
histogram(T, 'BinWidth', 1);  % Setează lățimea intervalelor
la 1°C
```

```
% Adăugarea etichetelor și titlului
xlabel('Temperatura (°C)');
ylabel('Număr de zile');
title('Distribuția temperaturilor zilnice pe o lună');

% Activarea grilei pentru vizibilitate
grid on;
```

După execuție, obținem graficul:

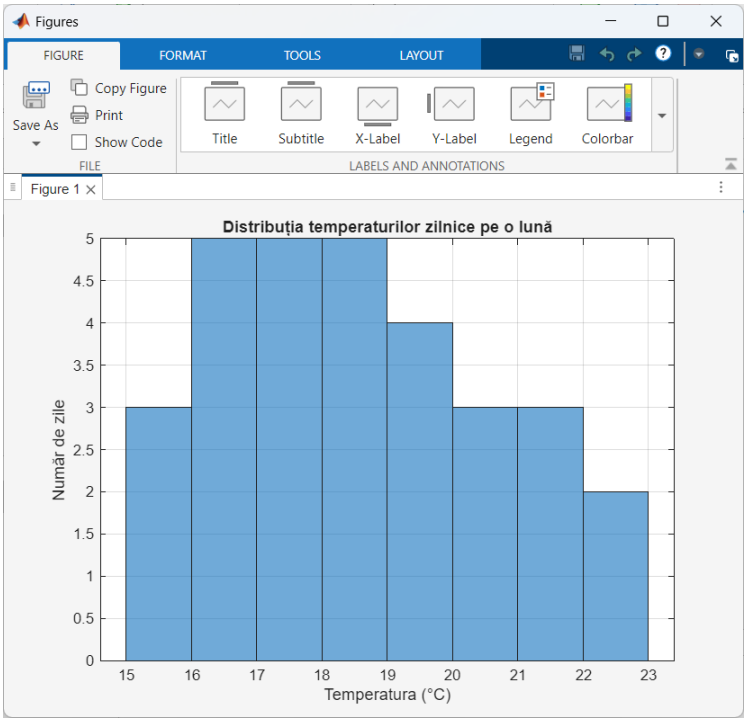


Figura 8.11 Histograma distribuției temperaturilor zilnice într-o lună

8.11 Reprezentarea grafică a numerelor complexe

Atunci când funcția *plot* este apelată cu numere complexe, partea imaginară va fi ignorată, cu excepția situației în care funcției i se transmite un *singur argument complex*.

Fie *x* și *y* doi vectori conținând numere complexe. Funcția *plot(x, y)* va extrage din vectorii x și y doar părțile reale și le va reprezenta grafic. Fie *Z = a + i*b* un vector conținând numere complexe. Funcția: *plot(Z)* va extrage partea reală și partea imaginară din vectorul Z și le va reprezenta grafic. Apelarea este echivalentă cu următoarea: *plot(real(Z), imag(Z))*.

Exemplu:

```
%programul urmator va reprezenta cercul trigonometric:
t=linspace(0,2*pi);
z=exp(i*t);
plot(z);
```

După execuție va rezulta graficul:

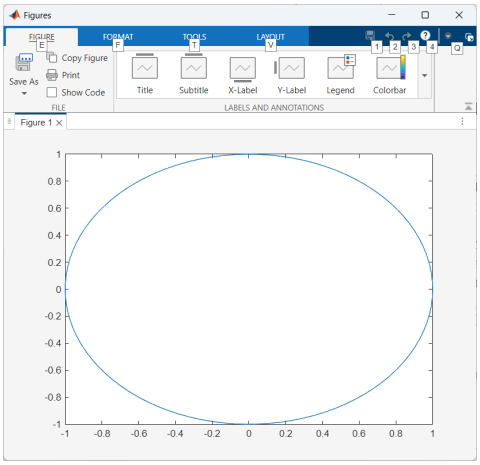


Figura 8.12 Reprezentarea cercului trigonometric folosind numere complexe

8.12 Reprezentarea grafică a vectorilor

Pentru reprezentarea grafică a vectorilor putem utiliza funcția MATLAB **compass.** Vectorii vor fi reprezentați cu punctul de pornire în punctul zero al sistemului de axe de coordonate. Sintaxa este:

compass(z) sau *compass(a,b)*

în care: z este un număr complex, z = a + ib.

Problemă rezolvată:

Să se grafic vectorii (pe același grafic):

z1=2-5i; *z2=-2+i*; *z3=3+2i*

Rezolvare propusă:

```
%in Matlab
clear;clc
z=[2-5i -2+i 3+2i];
compass(z); grid on;
```

După execuție rezultă graficul:

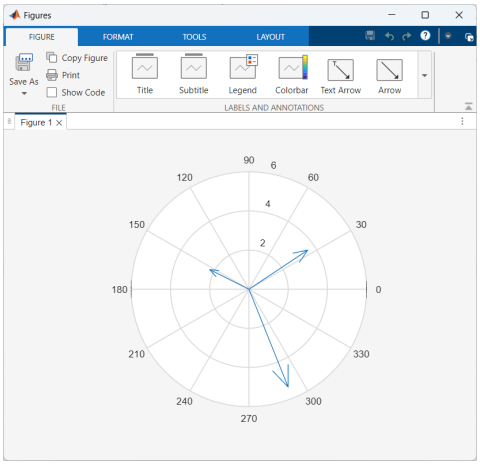


Figura 8.13 Reprezentarea vectorilor complecși folosind funcția compass

8.13 Reprezentari grafice 3D

MATLAB permite crearea și vizualizarea graficelor tridimensionale, care sunt foarte utile pentru analiza funcțiilor de două variabile sau pentru date care au trei dimensiuni (x, y, z).

Funcțiile de bază pentru grafice 3D sunt:

- `plot3(x, y, z)` – trasează un graf 3D al punctelor sau al curbelor definite de vectorii x, y și z. Exemplu: o curbă în spațiu.
- `mesh(X, Y, Z)` – creează o suprafață tridimensională sub formă de plasă.
    - X și Y sunt matrici care definesc coordonatele punctelor.
    - Z este matricea valorilor funcției f(X,Y).
- `surf(X, Y, Z)` – generează o suprafață 3D colorată, utilă pentru vizualizarea funcțiilor continue.
- `contour3(X, Y, Z)` – realizează linii de contur 3D, pentru a vizualiza topologia unei suprafețe.

Funcțiile folosite și o scurta descriere a lor este afișată in tabelul de mai jos.

Functia	**Descriere**
plot3	Creaza un grafic 3D tip linie
meshgrid	Creeaza o retea de puncte intr-un plan *x-y*
mesh, surf	Reprezentare grafica 3D tip suprafata
meshz	Reprezentare grafica 3D cu perdea verticala
meshc, surfc	Reprezentare grafica 3D si contur dedesubt
surfl	Trasează suprafețe luminate dintr-o anumită direcție
contour	Realizeaza mai multe contururi in plan 2D a unei suprafete

Reprezentări 3D de tip linie

Se folosește comanda:

```
plot3(x,y,z,'line specifiers', 'PropertyName',property value);
```

în care *x*, *y*, si *z* reprezintă 3 vectori de lungimi egale, iar *'line specifiers', 'PropertyName', property value* sunt opționali, reprezentand specificatorii prezentați in cazul funcției *plot*.

Problemă rezolvată:

```
%in Matlab
clear;clc
t=0:0.1:6*pi;
x=sqrt(t).*sin(2*t);
y=sqrt(t).*cos(2*t);
z=0.5*t;
plot3(x,y,z); grid on;
xlabel('x'); ylabel('y');zlabel('z');
```

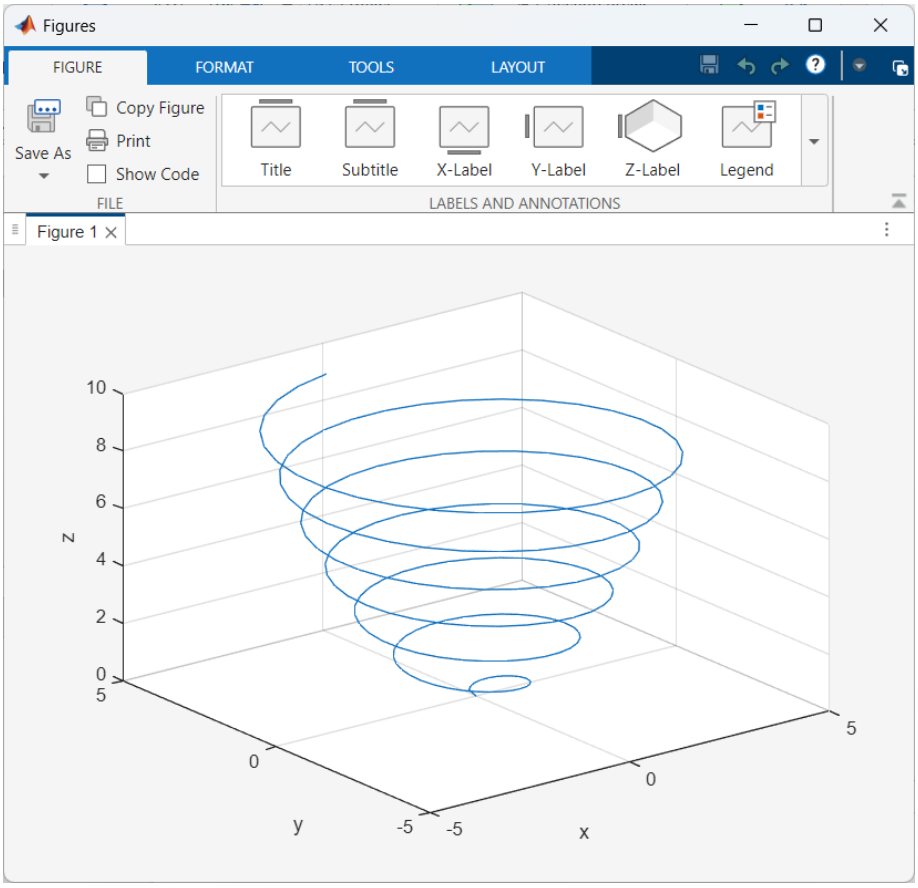


Figura 8.14 Reprezentarea 3D a unei curbe spațiale folosind plot3

Reprezentări 3D de tip suprafață

Graficele 3D de tip suprafață sunt utilizate pentru reprezentarea funcțiilor de două variabile, de forma $z = f(x, y)$. Rezultatul va fi o suprafață definită prin coordonata z a punctelor situate pe o rețea de puncte în planul $x - y$.

Reprezentările grafice 3D sunt realizate în 3 pași:

1. Primul pas constă în realizarea unei rețele în planul $x - y$ care acoperă domeniul de definiție al funcției f.
2. Urmează determinarea coordonatelor $z$pentru toate punctele (x, y)din planul definit la pasul anterior.
3. A treia etapă constă în reprezentarea 3D a coordonatelor (x, y, z), iar punctele alăturate vor fi conectate prin linii.

Fiecare pas va fi discutat in detaliu mai jos.

Pasul 1. Crearea rețelei in planul x-y

Pentru a reprezenta grafic o funcție 3D de forma $z = f(x, y)$, trebuie mai întâi să construim o rețea de puncte în planul $x - y$. Această rețea se creează folosind funcția `meshgrid` cu comanda:

```
[X,Y]=meshgrid(x,y);
```

unde:

X este o matrice de dimensiuni: *length(x)* x *length(y)*, conținând pe fiecare linie vectorul *x*;

Y este o matrice de dimensiuni: *length(x)* x *length(y)*, conținând pe fiecare linie vectorul *y*;

x și *y* sunt doi vectori cu pas liniar care împart axele Ox și Oy în puncte discrete.

Dacă *x* și *y* sunt identici, atunci funcția *meshgrid* poate fi apelată cu un singur argument conținând vectorul respectiv.

Exemplu:

```
clear;clc
x=-1:3;
y=1:4;
[X,Y]=meshgrid(x,y)
```

va rezulta:

```
X =

    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3

Y =

     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
     4     4     4     4     4
```

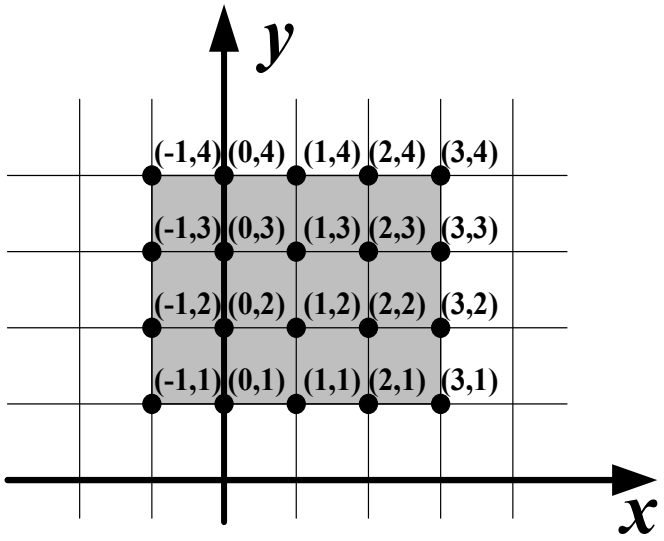


Figura 8.15 Rețeaua de puncte în planul $x - y$generată cu funcția meshgrid (după pasul 1)

În acest pas:

- Matricea **X** conține toate valorile lui x repetate pe fiecare linie.
- Matricea **Y** conține toate valorile lui y repetate pe fiecare coloană.

Astfel, fiecare element $(X(i,j), Y(i,j))$reprezintă un punct din rețea.

Pasul 2. Calcularea valorilor z pentru fiecare punct al retelei x-y

Pentru determinarea valorilor *z=f(x,y)* pentru fiecare punct al rețelei *x-y* create, se utilizează calculul element cu element.

De exemplu, dacă *z* este de forma:

$$z = \frac{xy^2}{x^2 + y^2}$$

Valorile lui *z* se calculează astfel:

```
Z=X.*Y.^2./(X.^2+Y.^2)
```

rezultând o matrice de dimensiuni *length(x)* **x** *length(y)*:

```
Z =

   -0.5000         0    0.5000    0.4000    0.3000
   -0.8000         0    0.8000    1.0000    0.9231
   -0.9000         0    0.9000    1.3846    1.5000
   -0.9412         0    0.9412    1.6000    1.9200
```

Pasul 3. Reprezentarea 3D a setului de coordonate (x,y,z)

MATLAB-ul pune la dispoziție două funcții pentru reprezentări 3D, *mesh* si *surf*. Reprezentarea prin *mesh* este realizată prin conectarea punctelor prin linii, iar funcția *surf* adauga culoare suprafetelor rezultate prin unirea punctelor.

Exemplu:

```
clear;clc
% pas 1
x=linspace(-1,3,10);
```

```
y=linspace(1,4,10);
[X,Y]=meshgrid(x,y);
% pas 2
Z=X.*Y.^2./(X.^2+Y.^2);
% pas 3
subplot(1,2,1);
mesh(X,Y,Z);
subplot(1,2,2);
surf(X,Y,Z);
```

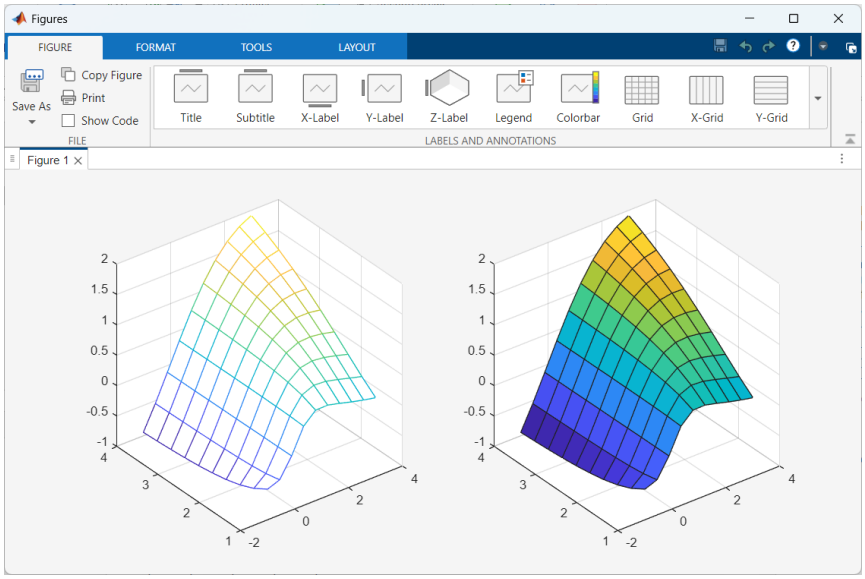
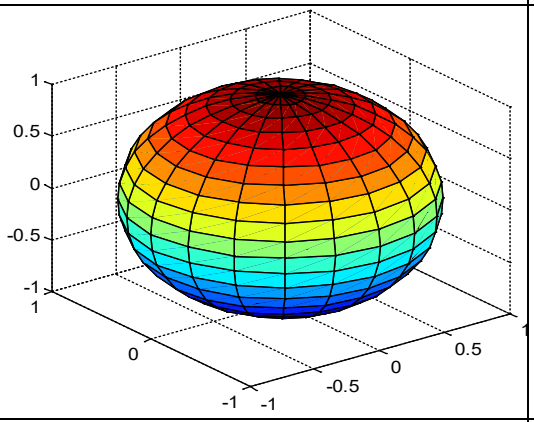
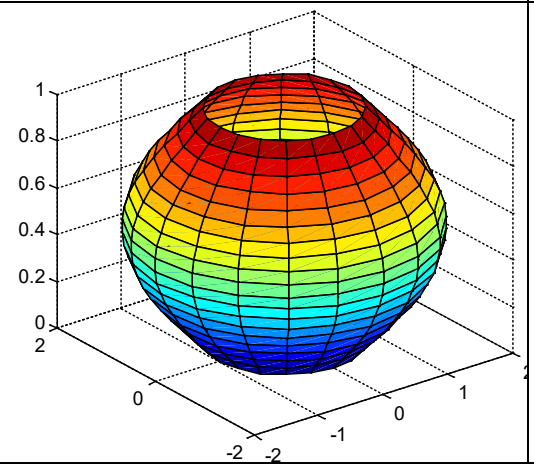


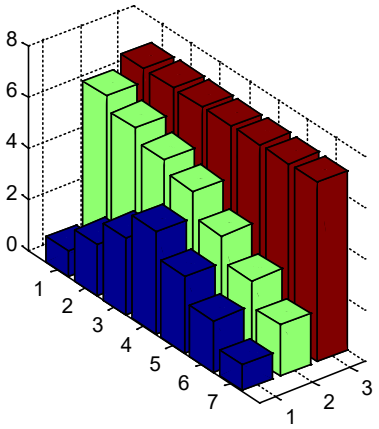
Figura 8.16 Reprezentarea 3D a funcției folosind mesh și surf

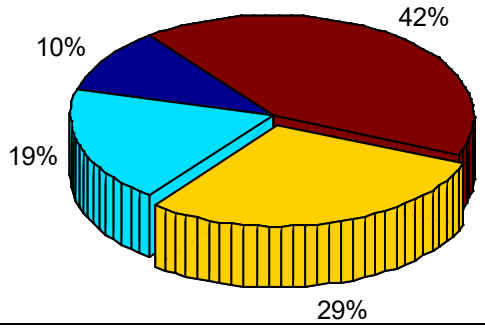
8.14 Reprezentări grafice 3D speciale

Funcția MATLAB	Exemplu de reprezentare grafică	Cod MATLAB
Reprezentarea unei sfere unitate (r=1) cu *n* fatete : `sphere(n)`		`[X,Y,Z]=sphere(20);` `surf(X,Y,Z);`

Reprezentarea unui cilindru cu un profil exterior *r:*
 	
`Cylinder (r)`
		`t=linspace(0,pi,20);`
		`r=1+sin(t);`
		`[X,Y,Z]=cylinder(r);`
		`surf(X,Y,Z);`

Funcția MATLAB	Exemplu de reprezentare grafică	Cod MATLAB

Reprezentare 3D tip bara:
Bar3(Y)
		`Y=[1 6.5 7;2 6 7;`
		`3 5.5 7;4 5 7;`
		`3 4 7;2 3 7;`
		`1 2 7];`
		`bar3(Y);`

Reprezentare 3D te tip *pie*:
Pie3(X, explode);
		`X=[5 9 14 20];`
		`explode=[0 0 1 0];`
		`pie3(X,explode);`

8.15 Probleme rezovate

Problema 1

Reprezentați funcția:

$$y(t) = e^{-0.2t}\cos(4t)$$

pentru $t \in [0,20]$. Adăugați grilă, titlu și etichete pentru axe.

Rezolvare propusă:

```
t = 0:0.01:20;                  % Vector timp
y = exp(-0.2*t) .* cos(4*t);    % Oscilație amortizată

plot(t, y, 'b', 'LineWidth', 1.5);
grid on;
xlabel('Timp (s)');
ylabel('Amplitudine');
title('Oscilație amortizată: y(t) = e^{-0.2t} cos(4t)');
```

După execuție rezulta graficul:

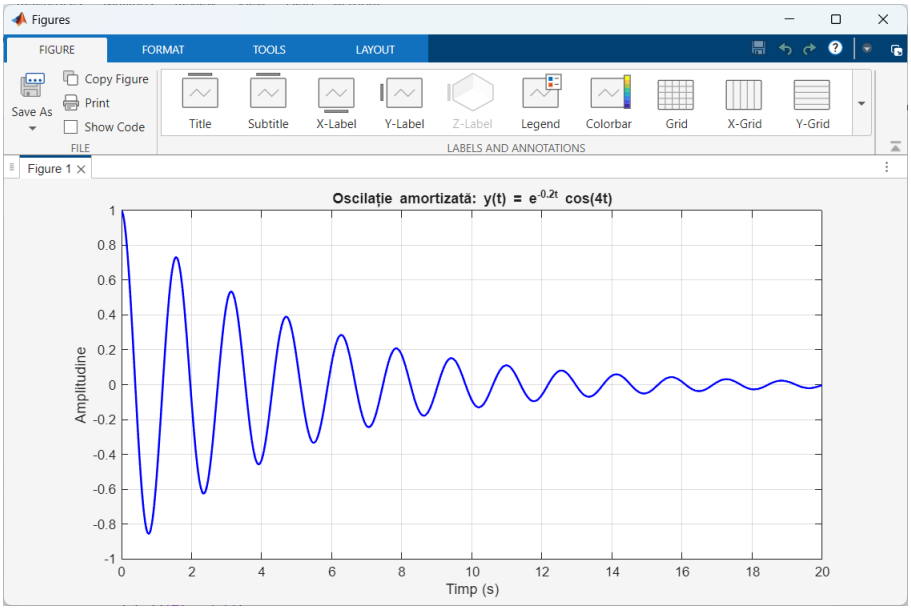


Figura 8.17 Oscilație amortizată $y(t) = e^{-0.2t}\cos(4t)$

Problema 2

Reprezentați pe același grafic funcțiile:

$$y_1 = \sin(t), y_2 = \sin(2t), y_3 = \sin(3t)$$

pentru $t \in [0,2\pi]$. Adăugați legendă.

Rezolvare propusă:

```
clear;clc
t = linspace(0,2*pi);

y1 = sin(t);
y2 = sin(2*t);
y3 = sin(3*t);

plot(t, y1, 'r', t, y2, 'g', t, y3, 'b', 'LineWidth', 1.4);
grid on;
xlabel('t');
ylabel('Amplitudine');
title('Reprezentarea funcțiilor sinusoidale');
legend('sin(t)', 'sin(2t)', 'sin(3t)');
```

După execuție obținem următorul grafic:

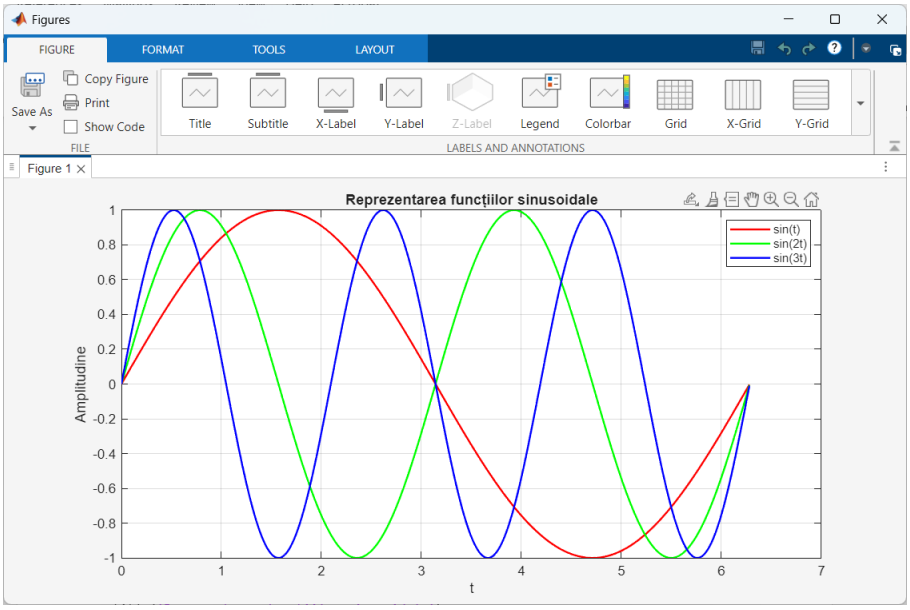


Figura 8.18 Reprezentarea funcțiilor $\sin(t)$, $\sin(2t)$și $\sin(3t)$

Problema 3

Reprezentați spirala (lui Arhimede) definită de $\rho = 0.1\theta$ pentru $\theta =$ 0până la 10π.

Rezolvare propusă:

```
clear;clc
theta = linspace(0, 10*pi, 300);
rho = 0.1 * theta;

polar(theta, rho);    % sau polarplot(theta, rho)
title('Spirala lui Arhimede');
```

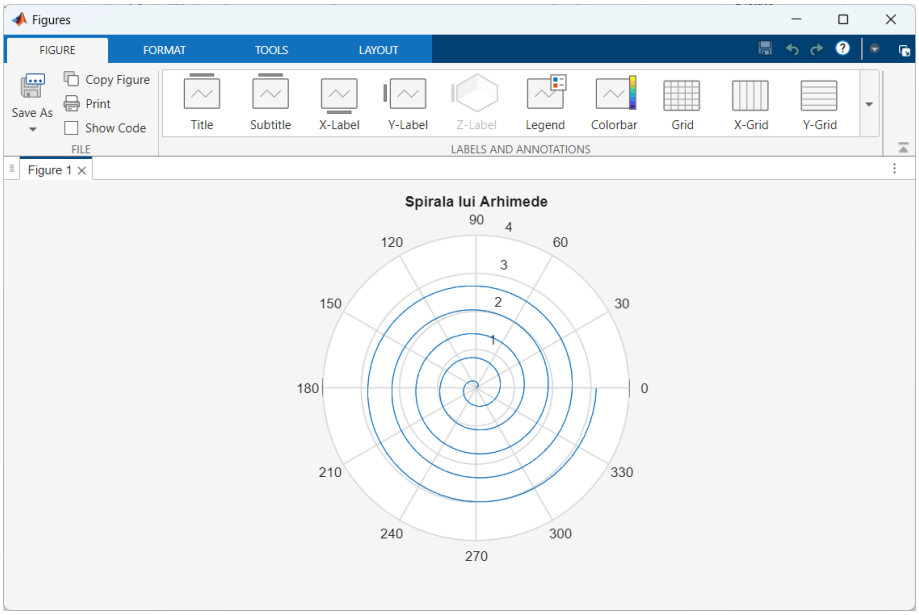


Figura 8.19 Spirala lui Arhimede reprezentată în coordonate polare

Problema 4

Reprezentați suprafața $z = x^2 + y^2$, pentru $x, y \in [-3,3]$.

Rezolvare propusă:

```
clear;clc
x = -3:0.1:3;
y = -3:0.1:3;

[X, Y] = meshgrid(x, y);
Z = X.^2 + Y.^2;

surf(X, Y, Z);
```

```
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Suprafața 3D: z = x^2 + y^2');
```

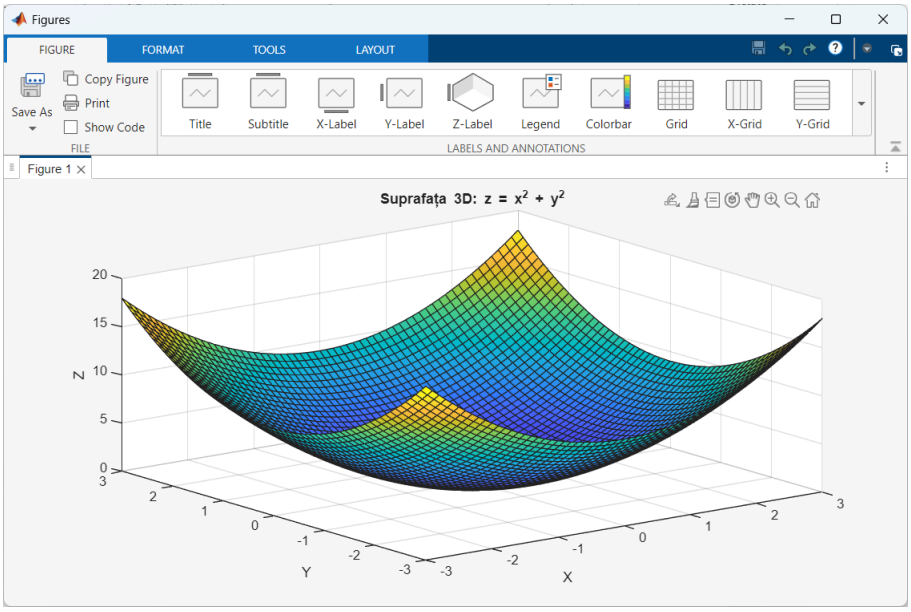


Figura 8.20 Suprafața 3D $z = x^2 + y^2$

Problema 5

Reprezentați suprafața definită prin $z = \sin(x)\cos(y)$, pentru $x, y \in [-\pi, \pi]$.

Rezolvare propusă:

```
clear;clc
x = -pi:0.1:pi;
```

```
y = -pi:0.1:pi;

[X, Y] = meshgrid(x, y);
Z = sin(X) .* cos(Y);

mesh(X, Y, Z);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Suprafață mesh: z = sin(x) cos(y)');
```

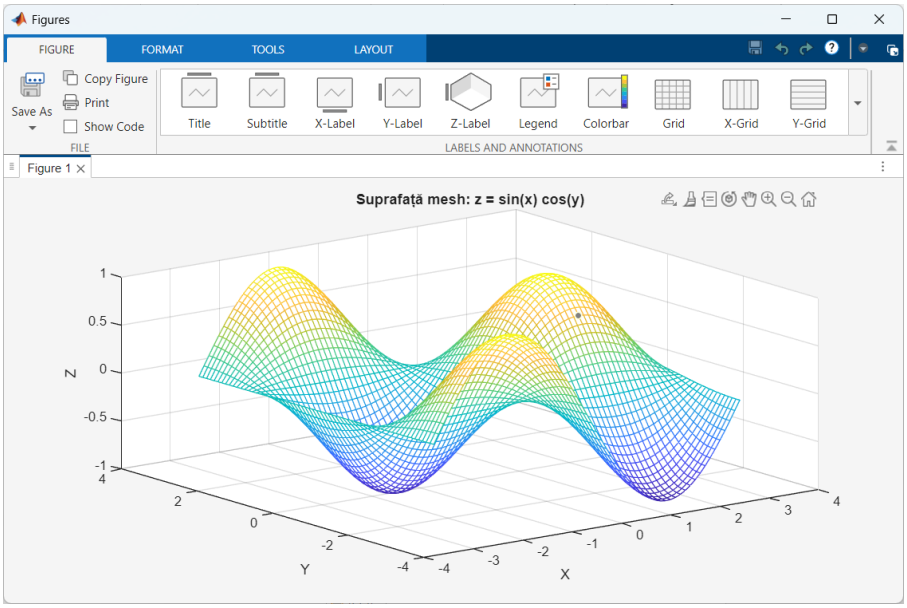


Figura 8.21 Reprezentarea suprafeței $z = \sin(x)\cos(y)$cu mesh

Problema 6

Reprezentați liniile de contur 3D ale funcției $z = e^{-(x^2+y^2)}$, pentru $x, y \in [-2,2]$.

Rezolvare propusă:

```
clear;clc
x = -2:0.05:2;
y = -2:0.05:2;

[X, Y] = meshgrid(x, y);
Z = exp(-(X.^2 + Y.^2));

contour3(X, Y, Z, 30);  % 30 niveluri de contur
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Contur 3D al funcției Gaussiene');
grid on;
```

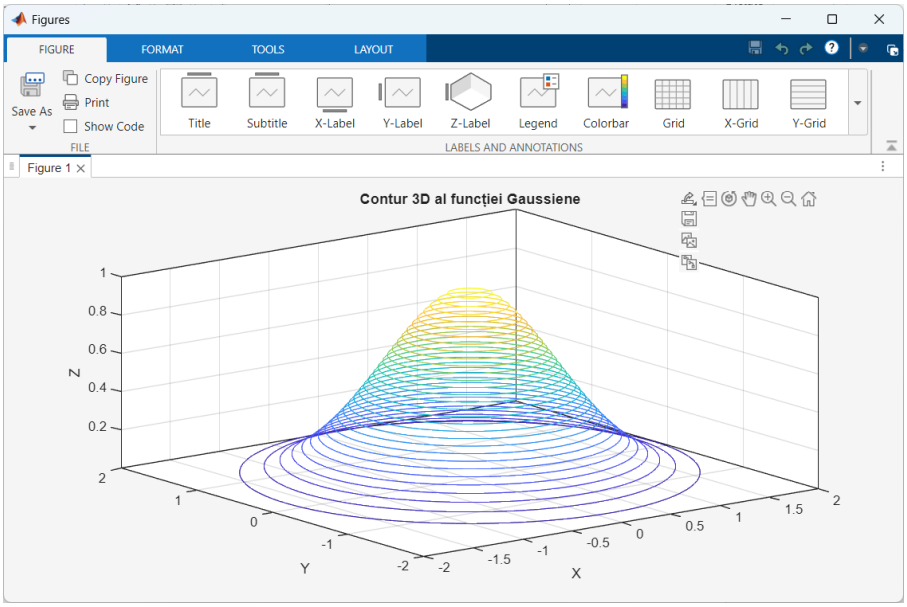


Figura 8.22 Liniile de contur 3D ale funcției gaussiene

Problema 7

Reprezentați în aceeași fereastră, dar în trei subferestre diferite, funcțiile:

$y_1 = \sin(t), y_2 = \cos(t), y_3 = \sin(t) + \cos(t)$, pentru $t \in [0,2\pi]$.

Rezolvare propusă:

```
clear;clc
t = 0:0.01:2*pi;

y1 = sin(t);
y2 = cos(t);
y3 = sin(t) + cos(t);
```

```
subplot(3,1,1);
plot(t, y1, 'r');
title('y_1 = sin(t)');
grid on;

subplot(3,1,2);
plot(t, y2, 'g');
title('y_2 = cos(t)');
grid on;

subplot(3,1,3);
plot(t, y3, 'b');
title('y_3 = sin(t) + cos(t)');
grid on;
```

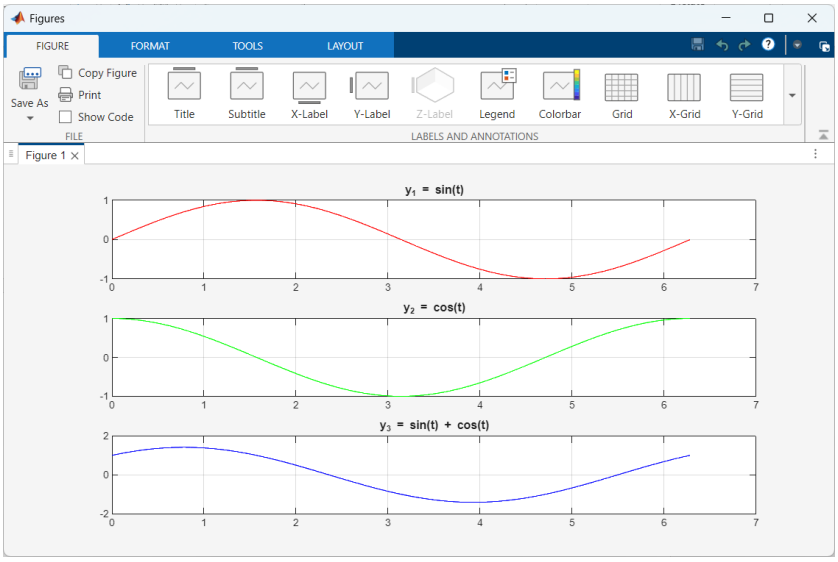


Figura 8.23 Reprezentarea funcțiilor $\sin(t)$, $\cos(t)$și $\sin(t) + \cos(t)$ în subploturi

Problema 8

Pentru funcția $f(t) = e^{-0.2t}\sin(3t)$ reprezentați în două subgrafice:

1. funcția $f(t)$,
2. derivata sa numerică.

Rezolvare propusă:

```
clear;clc
t = 0:0.01:10;
f = exp(-0.2*t) .* sin(3*t);

df = gradient(f, t);   % derivata numerică

subplot(2,1,1);
plot(t, f, 'b');
title('Funcția f(t)');
grid on;

subplot(2,1,2);
plot(t, df, 'r');
title('Derivata numerică f''(t)');
grid on;
```

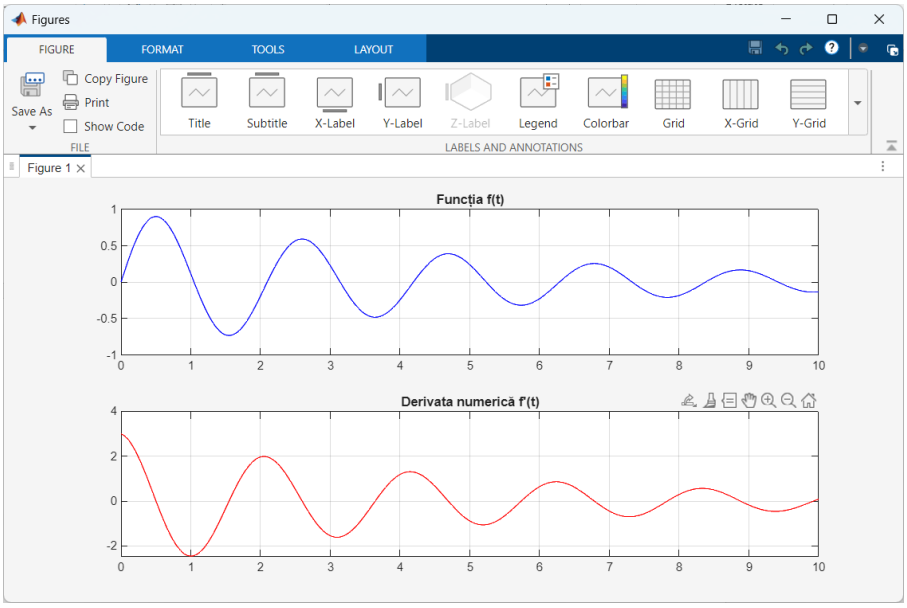


Figura 8.24 Funcția $f(t)$ și derivata numerică $f'(t)$

Problema 9

O stație meteo înregistrează temperatura și umiditatea pe parcursul unei zile (24 ore). Temperatura (°C):

```
T = [12 13 14 16 18 20 22 24 25 26 26 27 27 26 25 23 21
19 17 15 14 13 12 12];
```

Umiditatea (%):

```
U = [80 82 85 87 85 82 78 75 73 70 68 66 65 67 70 74 78
80 82 84 85 86 87 88];
```

Reprezentați temperatura și umiditatea în două subploturi (2×1).

Rezolvare propusa:

```
clear;clc
T = [12 13 14 16 18 20 22 24 25 26 26 27 27 26 25 23 21 19 17
15 14 13 12 12];
U = [80 82 85 87 85 82 78 75 73 70 68 66 65 67 70 74 78 80 82
84 85 86 87 88];

time = 0:23;

subplot(2,1,1);
plot(time, T, 'r', 'LineWidth', 1.3);
title('Temperatura zilnică');
xlabel('Ora');
ylabel('Temperatura (°C)');
grid on;

subplot(2,1,2);
plot(time, U, 'b', 'LineWidth', 1.3);
title('Umiditatea zilnică');
xlabel('Ora');
ylabel('Umiditate (%)');
grid on;
```

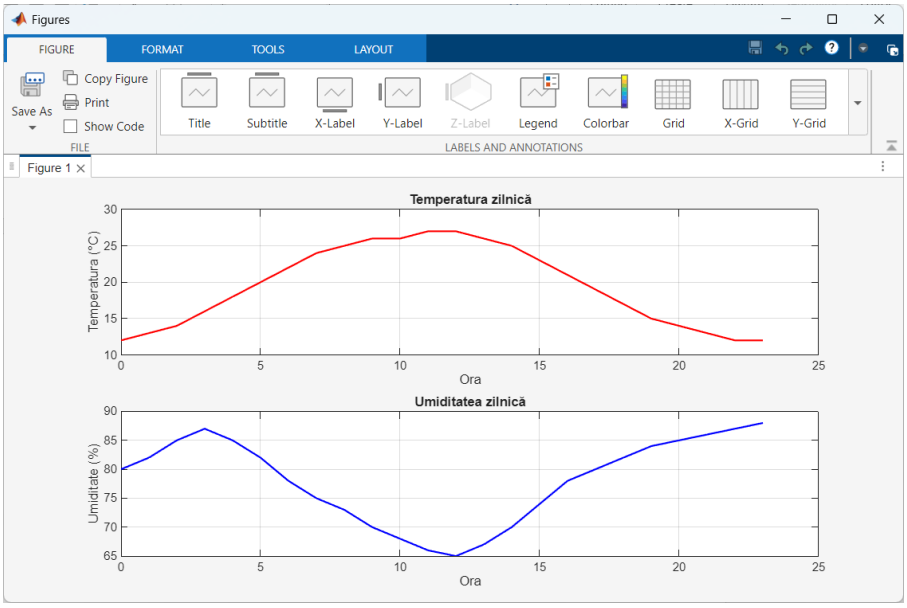


Figura 8.25 Reprezentarea temperaturii și umidității zilnice

Problema 10

Reprezentați într-o fereastră cu 2 subploturi:

- sus: spirala $\rho = \theta/5$,

- jos: funcția 2D $y = \theta\sin(\theta)$,

pentru $\theta \in [0,6\pi]$.

Rezolvare propusă:

```
clear;clc
theta = linspace(0, 6*pi, 300);
rho = theta/5;
```

```
subplot(2,1,1);
polar(theta, rho);
title('Spirală polară');

subplot(2,1,2);
plot(theta, theta .* sin(theta), 'm');
title('Grafic 2D: y = \theta sin(\theta)');
xlabel('\theta');
ylabel('y');
grid on;
```

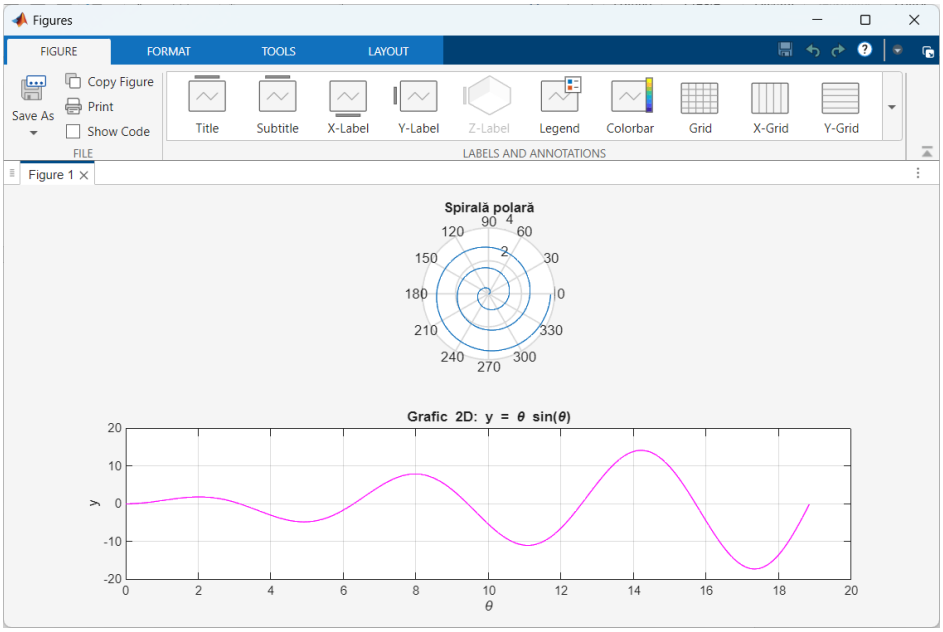


Figura 8.26 Spirală polară și graficul $y = \theta \sin(\theta)$

Capitolul 9

Metode Numerice in MATLAB

9.1 Funcții pentru interpolarea si aproximarea datelor

Atunci când lucrăm cu date cunoscute doar în anumite puncte, apare necesitatea estimării valorilor intermediare sau a descrierii comportării generale a acestora. În acest scop se folosesc metodele de interpolare și aproximare. Interpolarea are ca scop determinarea unei funcții care trece exact prin punctele date, pentru a calcula valori intermediare, iar aproximarea are ca scop determinarea unei funcții care descrie cât mai bine tendința datelor, minimizând eroarea globală, fără a trece neapărat prin toate punctele.

9.1.1 Interpolarea in Matlab

Se folosește când:

- avem valori măsurate corect
- avem nevoie de valori intermediare
- funcția trebuie să treacă prin toate punctele

Pentru un set discret de date (xi,yi), interpolarea presupune determinarea unei funcții *f(x)* astfel încât *f(xi)=yi*, în vederea estimării valorii funcției în orice alt punct x0≠xi.

Interpolare => curba de interpolare trece prin toate punctele și estimează valori necunoscute între puncte cunoscute, presupunând că datele sunt exacte. Vom discuta trei tipuri de interpolare liniară, spline cubică, polinomială.

Interpolare liniara:

In cazul funcțiilor de o singură variabilă interpolarea liniară poate fi realizată astfel*:*

```
y=interp1(xi,yi,x)
```

unde $\mathbf{x_i}$ și $\mathbf{y_i}$ reprezintă vectorii punctelor de coordonate cunoscute, iar **x** reprezintă vectorul care conține punctele de pe abscisă pentru care se dorește determinarea valorilor interpolate ale lui **y**.

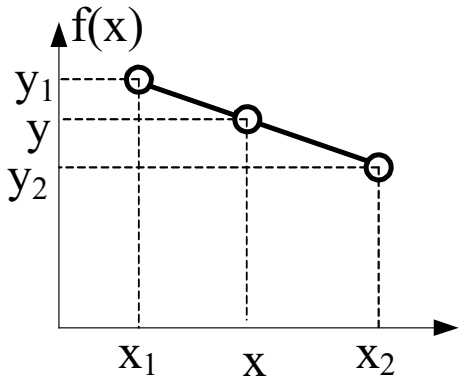


Figura 9.1 Interpolare liniară - Curba de interpolare liniară trece exact prin toate punctele date și estimează valorile necunoscute între punctele cunoscute, presupunând că datele sunt exacte.

Exemplu:

Sa se estimeze valorile temperaturii la momentele de timp 2.5, 3.5 și 4.5 secunde, cu datele din tabelul urmator:

Timp [s]	Temperatura [ºC]
0	0
1	10
2	40
3	75
4	80
5	95

Un program care poate rezolva această problemă este:

```
%Ex. interp. Liniara:
timp=0:1:5;
temp=[0 10 40 75 80 95];
x=[2.5 3.5 4.5];
y=interp1(timp,temp,x)
```

va returna după execuție:

```
y=[ 57.5000   77.5000   87.5000]
```

Interpolare de tip *spline* cubică:

Spline-ul cubic generează o curbă netedă, alcătuită din polinoame de gradul trei. Fiecare interval dintre două puncte este descris de un astfel de polinom, determinat astfel încât trecerea între segmente să fie continuă și lină. Interpolarea spline se realizeaza in MATLAB cu funcția *spline*, având sintaxa:

```
y=spline(xi, yi, x)
```

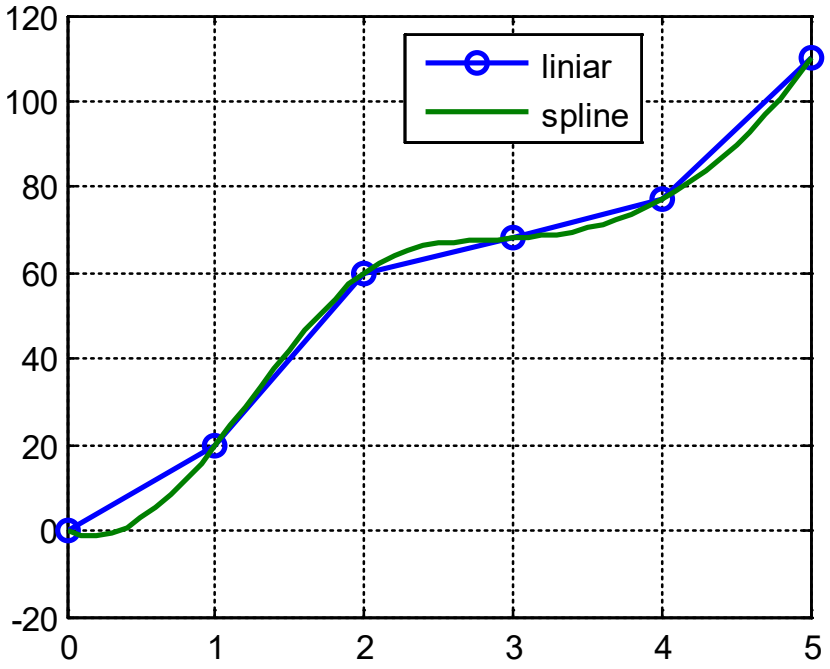


Figura 9.2 Interpolare spline cubică

Interpolarea multiplă:

Putem folosi sintaxa generală:

```
y = interp1(xi,yi,x, 'metoda')
```

unde *'metoda'* reprezintă unul dintre următoarele cuvinte cheie:

- *linear* pentru interpolare liniară (implicit)
- *spline* pentru interpolare spline cubică;
- *nearest* pentru interpolare de tip *cel mai apropiat punct*;

Rezultatul expresiei $y=interp1(x_i,y_i,x,$ *'spline'*$)$ este identic cu $y=spline(x_i,y_i,x)$.

Interpolarea funcțiilor de 2 variabile

Putem folosi sintaxa generală:

```
zi = interp2(x,y,z,xi,yi, 'metoda');
```

unde:

- x, y, z – punctele cunoscute ale funcției 2D (x ,y trebuie sa fie ordonate monoton);
- xi, yi – punctele pentru care se dorește determinarea valorilor zi prin interpolare;
- `'metoda'` reprezintă metoda de interpolare: ‘linear’ (implicit), ‘spline’, ‘nearest’;

Exemplu

Fie setul de date definit în tabelul următor:

y | x	**1**	**2**	**3**	**4**
1	2	7	8	5
2	3	6	1	4
3	5	6	2	3

Să se estimeze z_i pentru x=2.5, y=2.5.

Rezolvare propusă:

```
clear;clc
x=1:4;
y=1:3;
z=[2 7 8 5;3 6 1 4;5 6 2 3];
zi=interp2(x,y,z,2.5,2.5) %interpolare liniară
```

Dupa execuție obținem:

zi =

3.7500

9.1.2 Aproximarea (fitting-ul)

Prin aproximare descriem tendința generală a datelor printr-un model matematic, căutand o funcție care se potrivește cel mai bine datelor, dar al cărei grafic nu trece neapărat prin toate punctele.
Se folosește când:

- datele sunt experimentale și au zgomot
- lucrăm cu un model simplu (polinom, exponențial etc.)
- dorim să obținem predicții sau interpretare fizică

Aproximarea datelor prin metoda celor mai mici pătrate

Aproximarea unui set de date (x_i, y_i) prin metoda celor mai mici pătrate presupune determinarea unui polinom astfel încât suma pătratelor distanțelor de la punctele xi, yi la curba de aproximare sa fie minimă. Astfel se obține o curbă de aproximare care nu trece neapărat prin punctele date, acestea putând să nu se afle pe curba de aproximare.
Aproximarea unui set de date *(xi, yi)* presupune indentificarea coeficientilor a_i ai unui polinom de gradul *n* de forma:

$$p(x) = \sum_{i=0}^{n} a_i x^{n-i} = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n$$

Aproximarea unui set de date *(xi, yi)* cu un polinom de grad *n* se realizează în MATLAB cu funcția:

```
p=polyfit(xi,yi,n)
```

unde:

- *p* reprezintă vectorul coeficientilor polinomului de gradul *n*
- dacă gradul polinomului *n* este egal cu 1, rezulta o aproximare liniară (regresie liniară), iar pentru n≥2 avem o aproximare polinomială (regresie polinomială).

Exemplu:
Se consideră urmatorul set de date:
xi=[0.9 1.5 3 4 6 8 9.5] si
yi=[0.9 1.5 2.5 5.1 4.5 4.9 6.3].
Aproximați aceste date cu un polinom de ordinul n, unde n=1...6.

Rezolvare propusă:

```
clear;clc
x=[0.9 1.5 3 4 6 8 9.5];
y=[0.9 1.5 2.5 5.1 4.5 4.9 6.3];

x1=linspace(x(1),x(end));

for i=1:6
    p=polyfit(x,y,i);
    y1(i,:)=polyval(p,x1);
    subplot(2,3,i);
    plot(x1,y1(i,:),x,y,'o');grid
    xlabel('x'); ylabel('y');
    title(['n=' num2str(i)]);
end
```

Dupa execuție obținem graficele de mai jos.

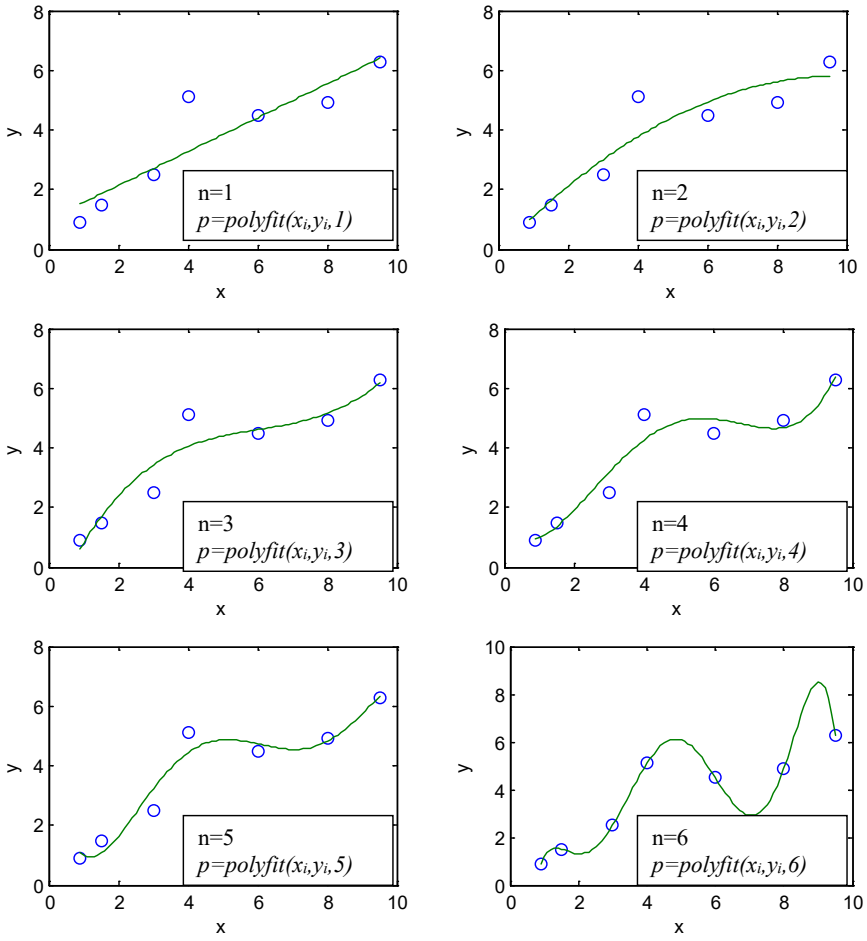


Figura 9.3 Subgraficele arată aproximarea datelor experimentale cu polinoame de ordin n=1…6, evidențiind creșterea preciziei aproximării odată cu creșterea ordinului polinomului.

9.1.3 Interfața „BASIC FITTING” (MATLAB)

Basic Fitting este o interfață grafică (GUI) din MATLAB destinată pentru ajustarea simplă (fitting) a datelor experimentale cu modele

uzuale (polinomiale, exponențiale, logaritmice etc.), fără a fi nevoie să scriem cod. In fereastra de reprezentare grafica se selecteaza: Figure-> Tools->Basic Fitting:

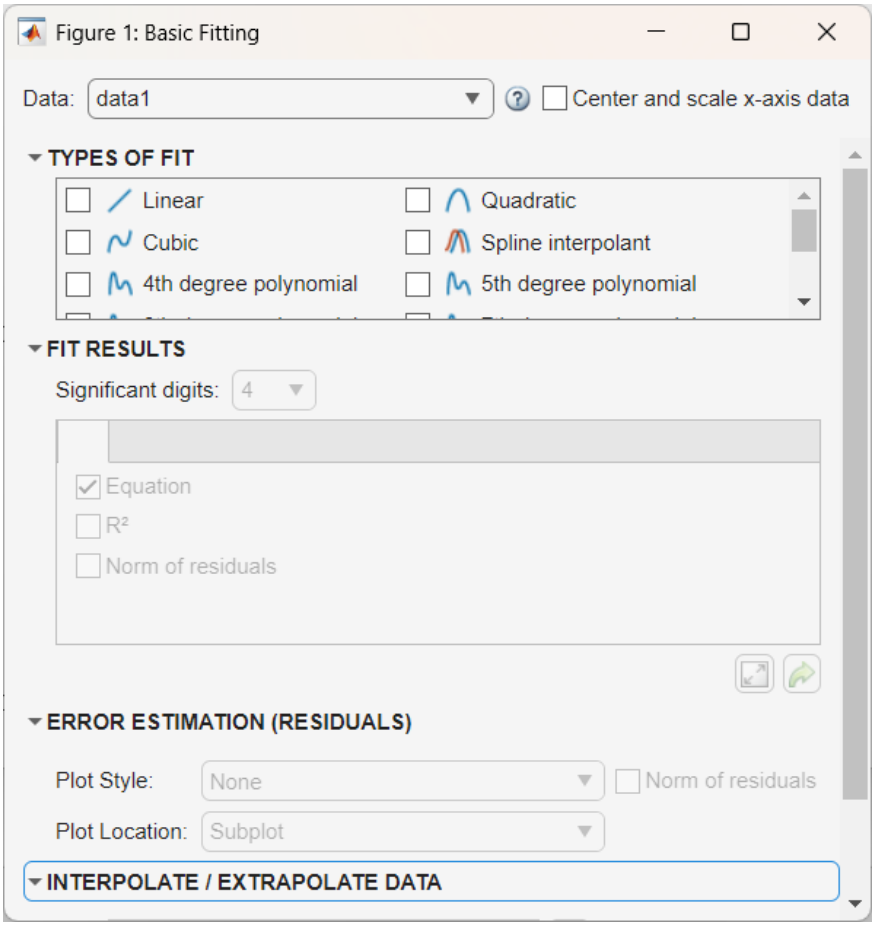


Figura 9.4 Interfața Basic Fitting; Fereastra GUI „Basic Fitting” din MATLAB permite ajustarea rapidă a datelor experimentale cu modele polinomiale, exponențiale, logaritmice și alte funcții, fără a scrie cod.

9.1.4 Aproximarea datelor cu alte funcții diferite de cele polinomiale

În multe situaţii, în ştiinţă şi inginerie, este necesară aproximarea datelor cu diferite funcţii, altele decât cele polinomiale. In tabelul de mai jos sunt descrise moduri diferite de aproximare a funcțiilor neliniare în MATLAB folosind metode de linearizare și polinomiale. In prima coloanal a tabelului avem forma funcției neliniare de aproximat (de exemplu $y = bx\text{^}m$, $y = be\text{^}mx$ etc.); în a doua coloană scrierea echivalentă sub o formă liniară prin transformări logaritmice, pentru a putea aplica metoda polinomială; cea de-a treia coloană conține comanda MATLAB pentru a efectua aproximarea polinomială după linearizarea datelor (folosind polyfit sau polyval), iar a patra coloană ne afișează formulele ce extrag parametrii funcției originale (de exemplu m, b) din coeficienții polinomului obținut.

In tabelul de mai jos este prezentată transformarea funcțiilor neliniare în forme liniare, astfel încât să le putem aproxima cu polinoame în MATLAB, și interpretarea coeficienților pentru a obține parametrii funcției neliniare originale.

Functia	Scrierea sub forma: $y=mx+b$	Sintaxa MATLAB pentru aproximare	Parametrii functiei
$y = bx^m$	$\ln(y) = m\ln(x) + \ln(b)$	`p=polyfit(log(x), log(y),1)`	`m=p(1)` `b=e^p(2)`
$y = be^{mx}$	$\ln(y) = mx + \ln(b)$	`p=polyfit(x,log(y),1)`	`m=p(1)` `b= e^p(2)`
$y = b10^{mx}$	$\log(y) = mx + \log(b)$	`p=polyfit(x,log10(y),1)`	`m=p(1)` `b= 10^p(2)`
$y = m\ln(x) + b$ $y = m\log(x) + b$	este deja scrisa in forma	`p=polyfit(log(x), y,1)` `p=polyfit(log10(x),y,1)`	`m=p(1)` `b=p(2)`

O altă unealtă utilă oferita de MATLAB pentru aproximarea datelor cu diferite funcții este „Open Curve Fitting Tool”, care se pornește prin tastarea comenzii: *cftool* ce necesită libraria ‘Curve Fitting Toolbox”

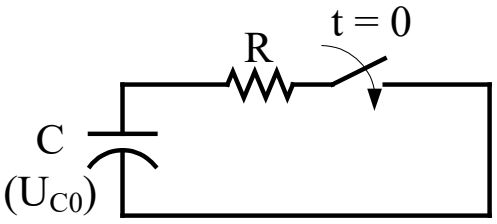
Exemplu

Un condensator cu tensiunea inițială U_{c0} se descarcă prin intermediul unei rezistențe fixe de valoare R=2kΩ. Tensiunea pe condensator este măsurată la intervale fixe de timp de 1s, timp de 10 s, valorile fiind prezentate în tabelul următor. Să se determine valoarea condensatorului (in μF) și tensiunea inițială a condensatorului.

t(s)	1	2	3	4	5	6	7	8	9	10
$U_c(s)$	12.7	7.4	4.5	2.7	1.6	1	0.6	0.4	0.2	0.1

Tensiunea pe condensator în funcție de timp este dată de relația exponențială:

$$U_C = U_{C0} e^{-t/(RC)}$$



Rezolvare propusă:

```
clear;clc
R=2e3;
t=1:10;
Uc=[12.7 7.4 4.5 2.7 1.6 1 0.6 0.4 0.2 0.1];
p=polyfit(t,log(Uc),1);
C=-1/R/p(1);
Uc0=exp(p(2));
```

```
tplot=0:0.1:10;
Uplot=Uc0*exp(-tplot./(R*C));
plot(tplot,Uplot,t,Uc,'o'); grid on;
```

După execuție va rezulta:

```
C=9.5977e-004
Uc0=21.8
```

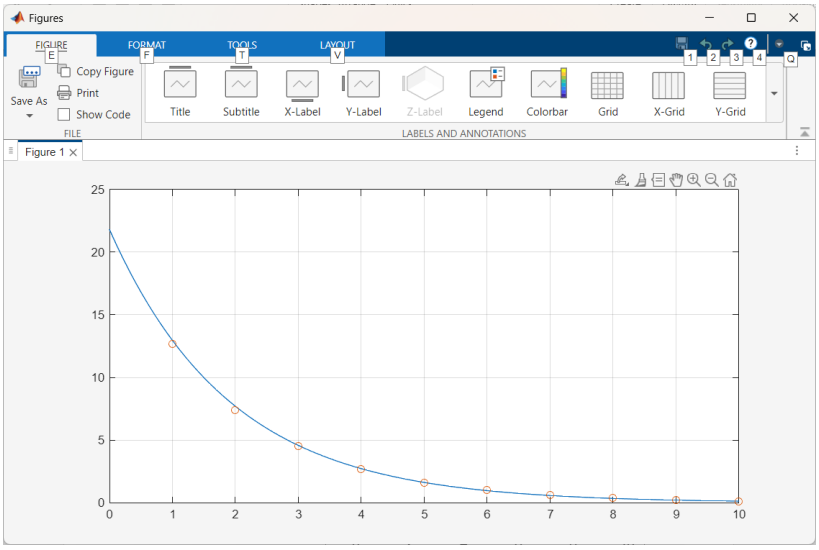


Figura 9.5 Aproximarea descărcării unui condensator. Graficul arată tensiunea Uc(t) pe condensator în timp, comparând valorile experimentale (puncte) cu curba exponențială aproximată.

9.2 Aplicații în metode numerice

Metodele numerice reprezintă un instrument esențial în analiza și rezolvarea problemelor din domeniul ingineriei, atunci când soluțiile analitice nu pot fi determinate sau sunt dificil de obținut. MATLAB oferă un mediu puternic și flexibil pentru implementarea acestor metode, permițând rezolvarea numerică a ecuațiilor, interpolarea și aproximarea datelor, integrarea și derivarea numerică, precum și simularea și analiza sistemelor matematice. Datorită capacităților sale de calcul și vizualizare, MATLAB este larg utilizat în aplicații de metode numerice.

9.2.1 Rezolvarea ecuațiilor cu o singură variabilă

Rezolvarea unei ecuații *f(x)=0*, in MATLAB se realizează folosind funcția *fzero*, cu sintaxa:

```
x=fzero('f', x0) % metoda 1
```

sau

```
x=fzero(@(x)f(x), x0) % metoda 2 recomandata
```

unde:

- **f** reprezintă ecuația care trebuie rezolvată, exprimată într-una dintre cele două forme: ca o expresie matematică sub forma unui string (metoda 1) sau ca o funcție (metoda 2);
- *x0* reprezintă o valoare în jurul căreia se va căuta soluția ecuației;
- **x** este soluția ecuației.

Exemple:

a) Sa se calculeze zeroul funcției $f(x)=sin(x)$, în jurul valorii inițiale x0=3.

Rezolvare propusă:

```
x=fzero('sin(x)',3) ) % metoda 1
x=fzero(@(x)sin(x),3) % metoda 2 recomandata
```

b) Sa se calculeze zeroul funcției $f(x)=(x-3)^2-1$, în jurul valorii inițiale x0=1.5.

```
x=fzero('(x-3)^2-1',1.5) % metoda 1
x=fzero(@(x)(x-3)^2-1,1.5) % metoda 2 recomandată
```

In cazul in care nu se cunoaște valoarea x0, se poate realiza o reprezentare grafică a funcției intr-un anumit interval pentru a vedea cu aproximație locul in care functia intersectează axa Ox.

c) Sa se rezolve ecuația $xe^{-x}=0.2$. Se va reprezenta grafic funcția f(x)= $xe^{-x}-0.2$, in intervalul [0 8].

```
fplot('x*exp(-x)-0.2', [0 8])
```

Se observa ca funcția intersectează axa OX de doua ori, între 0-1 și 2-3.

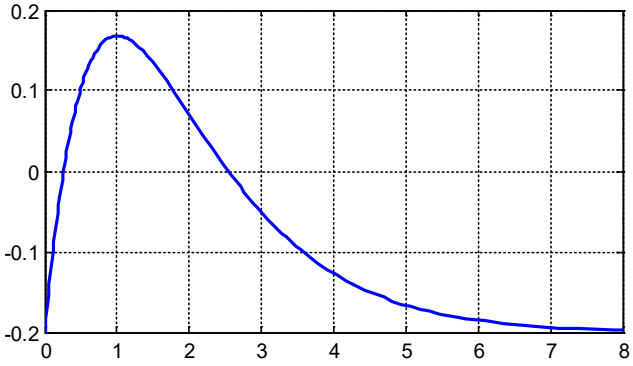


Figura 9.6 Reprezentarea grafică a funcției f(x)=xe^(-x)-0.2 pe intervalul [0,8] arată aproximativ punctele în care funcția intersectează axa Ox, indicând intervalele pentru căutarea rădăcinilor.

Pentru aflarea rădăcinilor exacte:

```
x1=fzero('x*exp(-x)-0.2',0) % metoda 1
x2=fzero('x*exp(-x)-0.2',2.5) % metoda 1
```

sau

```
x1=fzero(@(x)x*exp(-x)-0.2,0) % metoda 2 recomandata
x2=fzero(@(x)x*exp(-x)-0.2,2.5) % metoda 2 recomandata
```

9.2.2 Calculul minimului funcțiilor de o singură variabilă

Pentru a calcula minimul unei funcții de o singură variabilă folosim urmatoarea sintaxă:

```
[x fval]=fminbnd('f',x1,x2) % Matlab
```

sau

```
[x fval]=fminbnd(@(x)f(x),x1,x2) %Matlab/Octave recomandata
```

unde: *f* reprezintă funcția

x1 si *x2* - reprezintă intervalul de căutare $x1 \leq x \leq x2$

x - valoarea pentru care funcția *f(x)* este minimă in intervalul specificat.

fval - reprezintă valoarea funcției in punctul minim

Exemple:

a) Sa se calculeze coordonatele minimului funcției *f(x)=sin(x)* in intervalul $[0,2\pi]$.

```
[xmin ymin]=fminbnd('sin(x)',0,2*pi) %metoda 1
[xmin ymin]=fminbnd(@(x)sin(x),0,2*pi) %metoda 2
```

După execuție obținem:

```
xmin =

    4.7124


ymin =

   -1.0000
```

b) Sa se determine minimul funcției $f(x)=(x-3)^2-1$, in intervalul [0 5]:

```
[xmin ymin]=fminbnd('(x-3)^2-1',0,5) %metoda 1
[xmin ymin]=fminbnd(@(x)(x-3)^2-1,0,5) %metoda 2
```

După execuție:

```
xmin =
     3
ymin =
    -1
```

c) Să se determine **maximul** funcției $f(x)= xe^{-x}-0.2$, in intervalul [0 8]; Cum putem rezolva cerința utilizand *fminbnd* ?

```
clear;clc
%[x_max, fmin] = fminbnd(@(x) x.*exp(-x) - 0.2, 0, 8);

% Definirea functiei
f = @(x) x.*exp(-x) - 0.2;
% Functia de minimizat (pentru determinarea maximului)
g = @(x) -f(x);
% Determinarea maximului pe intervalul [0, 8]
[x_max, fmin] = fminbnd(g, 0, 8);
% Valoarea maxima a functiei
f_max = -fmin;
% Afisarea rezultatelor
```

```
fprintf('Maximul functiei este f(%.4f) = %.4f\n', x_max,
f_max);
% Reprezentare grafica
x = linspace(0, 8, 1000);
plot(x, f(x), 'b', x_max, f_max, 'ro')
grid on
xlabel('x')
ylabel('f(x)')
legend('f(x)', 'Maxim')
```

După execuție:

```
Maximul functiei este f(1.0000) = 0.1679
```

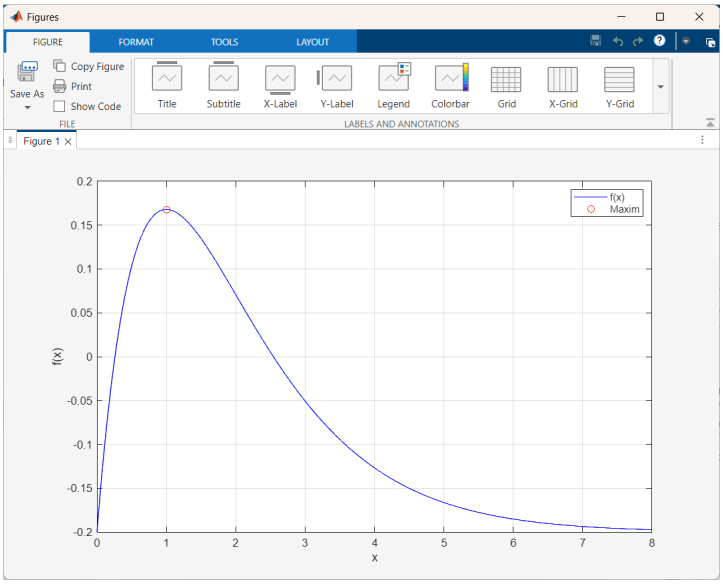


Figura 9.7 Determinarea maximului funcției f(x) = x·exp(-x)-0.2. Graficul ne arată funcția in intervalul [0,8] și punctul în care funcția atinge valoarea maximă, marcat cu cerc roșu. Este exemplul utilizării funcției *fminbnd* pentru determinarea maximului prin minimizarea funcției inverse.

9.3 Integrare numerică

Integrarea numerică este utilizată pentru aproximarea integralelor definite atunci când soluția analitică este dificil de obținut sau nu poate fi determinată. MATLAB oferă funcții eficiente care implementează metode numerice adaptive, cu control automat al erorii.

Integrala unei funcții *f(x)*, definită pe intervalul [a,b], are semnificația ariei delimitată de axa Ox, curba *f(x)* și dreptele *x=a* și *x=b*:

$$S = \int_a^b f(x)dx$$

Funcții MATLAB de integrare:

- *quad*;
- *quadl*;
- *trapz;*

Funcția *quad* integrează funcții folosind metoda adaptiv-recursivă Simpson, avand sintaxa:

`S=quad('f',a,b);` sau `S=quad(@(x)f,a,b)`

unde: *f* reprezintă funcția ce urmează a fi integrată (o expresie matematica sau un fișier-M)

a si *b* reprezintă limitele de integrare;

Funcția quadl integrează funcții folosind metoda adaptiv-recursivă Newton Cotes de ordinul 8, fiind mai eficientă in unele situații decat *quad*. Sintaxa funcției *quadl* este:

`S=quadl('f',a,b);` sau `S=quadl(@(x)f,a,b)`

Funcția **trapz** calculează integrala prin metoda trapezelor, pentru o funcție furnizată sub formă numerică (puncte de coordonate). Sintaxa funcției este:

S = trapz(x, y)

unde: **x** și **y** reprezintă doi vectori cu coordonatele punctelor prin care este definită funcția.

Exemplu:

Să se calculeze următoarea integrală: $\int_0^8 (xe^{-x^{0.8}} + 0.2)dx$.

```
S=quad('x.*exp(-x.^0.8)+0.2',0,8)
```

sau

```
S=quad(@(x) x.*exp(-x.^0.8)+0.2,0,8)
```

După execuție obținem:

```
S =

    3.1604
```

9.4 Derivarea numerică

Derivata funcției $f(x)$ exprimă rata de variație a acesteia în raport cu variabila x. Derivata se definește ca raportul dintre variația funcției$f(x)$, notată cu $df(x)$, și variația variabile x, notată cu dx:

$$f'(x) = {df(x)}/{dx}$$

Derivarea numerică prezintă unele dificultăți și riscuri față de derivarea analitică. Derivarea numerică poate amplifica micile erori sau zgomotul dintr-un semnal măsurat experimental, conducând astfel la o depărtare de soluția analitică. Erorile și zgomotul influențează negativ rezultatul derivării. Într-o aplicație practică, acolo unde este posibil, se recomandă evitarea utilizării operației de derivare.

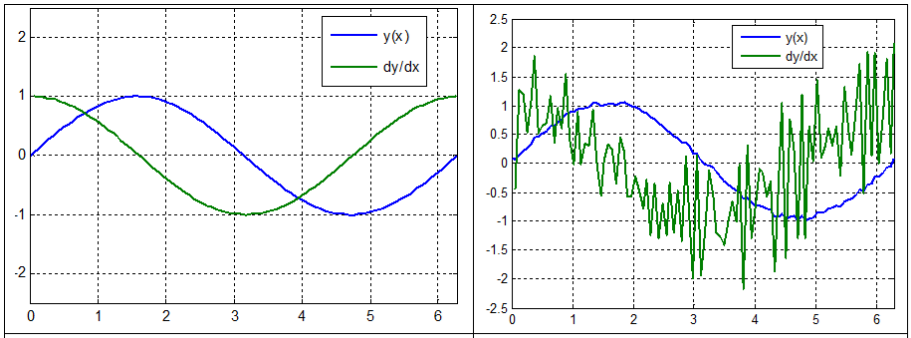


Figura 9.8 – Derivarea numerică, primul grafic: derivata numerică a unui semnal sinusoidal ideal, fără zgomot, netedă și bine definită. Al doilea grafic: derivata numerică a aceluiași semnal afectat de zgomot, cu amplificarea semnificativă a oscilațiilor.

Sintaxa folosită este:

`DX = diff(X)` sau `DX = diff(X,N)`

unde: *X* reprezintă un vector linie sau coloana

DX reprezintă un vector linie ce conține diferențele elementelor succesive ale lui *X*, avand dimensiunea egală cu *length(X)-1*;

N ordinul derivatei (daca nu se specifică, implicit N=1)

Funcția *diff* permite calcularea derivatei numerice a unei funcții *y(x)* , definite numeric prin vectorii y și x, cu expresia:

`dy=diff(y)./diff(x)`

Exemple:

1) Să se calculeze numeric derivata funcției $y = \sin(x)$ pe intervalul $[0,2\pi]$și să se compare grafic funcția inițială cu derivata numerică.

```
clear;clc
x = linspace(0, 2*pi, 100);
y = sin(x);
dy = diff(y) ./ diff(x);

% Pentru a avea același număr de puncte ca x, putem interpola
sau aproxima
x_mid = (x(1:end-1) + x(2:end)) / 2;

plot(x, y, 'b', x_mid, dy, 'r')
legend('y = sin(x)', 'Derivata numerică')
```

După execuție obținem:

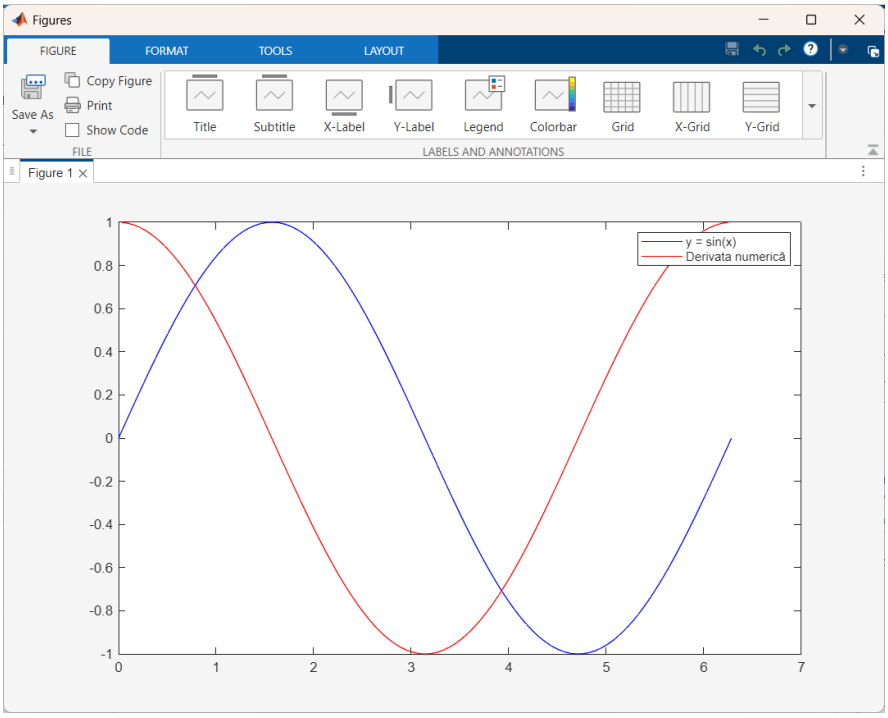


Figura 9.9 – Compararea funcției y = sin(x) cu derivata numerică calculată prin diferențe finite**.** Graficul prezintă funcția sin (curba albastră) și derivata sa numerică (curba roșie), obținută prin diferențiere numerică a vectorului de valori. Se observă cât de bine se potrivește derivata numerică cu forma teoretică a derivatei.

2) Sa se calculeze diferențele de ordinul 1 si 2 ale vectorului: X=[1 4 5 6].

```
X=[1 4 5 9];
Y=diff(X)
Z=diff(Y)   %sau Z=diff(X,2)
```

După execuție obținem:

```
Y =

     3     1     4

Z =

    -2     3
```

3) Fie x=[0 1 2 3 4 5] și y=[1 2 5 3 6 4 7], să se calculeze *dy/dx*.

```
clear;clc
x=[0 1 2 3 4 5 6]
y=[ 1 2 5 3 6 4 7]
dy=diff(y)./diff(x)
```

După execuție obținem:

```
x =

   0   1   2   3   4   5   6

y =
1   2   5   3   6   4   7

dy =

   1   3  -2   3  -2   3
```

9.5 Integrarea numerică a ecuațiilor diferențiale

O ecuație diferențială de ordinul întâi poate fi scrisă sub forma:

$$y' = \frac{dy}{dt} = g(t, y)$$

în care **t** este o variabilă independentă (un exemplu este variabila timp), iar **y** este funcția necunoscută.

MATLAB-ul oferă mai multe funcții pentru integrarea numerică a ecuațiilor diferențiale de ordinul întâi, bazate pe metoda Runge-Kutta, printre care: **ode45**, **ode23**, **ode113**, **ode15s**, **ode23s**, **ode23t**, **ode23tb**. Cele mai utilizate sunt **ode23**, care rezolvă ecuații diferențiale prin metoda Runge-Kutta de ordinul 2-3, și **ode45**, care rezolvă ecuații diferențiale prin metoda Runge-Kutta de ordinul 4-5.

Funcțiile menționate, **odeX** (unde X poate fi 45, 23, 113, 15s, 23s, 23t, 23tb) se apelează folosind sintaxele:

```
[t,y]=odeX('yprim',tspan,y0) sau
[t,y]=odeX('yprim',[t0 tf],y0)
```

unde: *yprim* reprezintă numele unui fișier-M, de tip funcție in care este definită ecuația diferențială;

tspan reprezintă un vector care indică intervalul de variație a variabilei *t,* de forma: *t0:pas:tf*;

y0, reprezintă valoarea inițiala a lui *y*;

t0 și *tf* reprezintă capetele intervalului de variație a variabilei *t*.

In cazul celei de-a doua sintaxă de apelare, pasul de integrare este ales automat in funcție de gradientul soluției.

9.5.1 Etapele rezolvării unei ecuații diferențiale de ordinul 1, in MATLAB

Etapele vor fi discutate in detaliu folosind următorul exemplu ce are ca cerință integrarea următoarei ecuații diferențiale:

$\frac{dy}{dt} = \frac{t^3-2y}{t}$, pentru $1 \leq t \leq 3$, cu $y0=1.2$, la $t=1$, utilizand un pas fix de 0.01.

Pasul 1:

Se creează un fișier M-funcție (intitulat `my_f`.m) care calculează derivata de ordinul intai *dy/dt*, pentru orice valoare a lui *t* și *y* astfel:

```
function DyDt=my_f(t,y)

DyDt=(t^3-2*y)/t;
```

Atenție: fișierul *m-funcție* se salvează cu același nume cu cel al funcției (my_f.m)!

Pasul 2:

Se selectează una dintre metodele de rezolvare menționate mai sus, *ode45*.

Pasul 3:

Se apelează funcția *ode45* cu următoarea sintaxă (dintr-un alt fișier .m decat cel funcție creat la pasul 1) astfel:

```
[t,y]=ode45('my_f',[1:0.01:3],1.2)
```

sau

```
[t,y]=ode45('my_f',[1 3],1.2) %cu pas automat
```

Exemplu rezolvat:

Să se integreze ecuația diferențială: $y' = 3y + e^{2x}$, pe intervalul x=[0, 2], cu condiția inițială *y(0)=3*, utilizand metoda (solverul) *ode23*, cu pas automat.

Se creează un fișier M-functie (numit *g.m*) astfel:

```
function Dy=g(x,y)

Dy=3*y+exp(2*x);
```

și se salvează cu numele „*g.m*”.

In MATLAB (fereastra de comenzi sau fișier M funcție) se apelează astfel:

```
[x,yn]=ode23('g',[0 2],3);

plot(x,yn); %afisare grafica a functiei y(x)
```

Pentru rezolvarea problemelor în genul celor de mai sus este disponibilă și varianta de utilizare a unei funcții anonime, scrisă direct în scriptul programului, astfel nemaifiind necesar un fișier m-function separat:

%utilizarea unei functii 'anonime' in script:

```
my_f=@(t,y)(t^3-2*y)/t; %definirea functiei de tip
'anonymous' [t,y]=ode45(my_f,[1 3],1.2); %pas automat
plot(t,y);
```

Funcții anonime în MATLAB

În MATLAB, o funcție anonimă este definită într-o singură linie, direct în script, fără necesitatea unui fișier .m separat.

Sintaxă generală

```
f = @(parametri) expresie;
```

`@` → indică o funcție anonimă

`parametri` → argumentele funcției

`expresie` → o singură expresie (nu bloc de cod)

Exemple:

```
1) %Funcție anonimă de un singur parametru
   f = @(x) x^2 + 3*x + 1;


2) %Funcție anonimă de doi parametri
```

```
   aria = @(r, h) pi * r^2 * h;

3) %Funcție anonimă ca argument (foarte frecvent!)
   f = @(x) sin(x);
   integral(f, 0, pi)
   %sau intr-o singura linie de cod
   integral(@(x) sin(x), 0, pi)

4) %Funcție anonimă de doi parametri
   aria = @(r, h) pi * r^2 * h;
```

9.6 Integrarea ecuațiilor diferențiale de ordin superior și a sistemelor de ecuații diferențiale

Ecuațiile diferențiale de ordinul doi sau mai mare pot fi transformate într-un sistem de ecuații diferențiale de ordinul întâi, cuplate, și integrate ca atare.

Fie ecuația diferențiala de ordinul doi:

$$y''+(y^2-1)y'+y=0,$$

cu condițiile inițiale: y(0) = 0.25 si y'(0) = 0.
Această ecuație este echivalentă cu sistemul:

$$\begin{cases} y_1' = -(y_2^2-1)y_1 - y_2 \\ y_2' = y_1 \end{cases}$$

În prima etapă a rezolvarii se realizează un fișier M-funcție (intitulat `ecdif.m`), în care se scriu cele două ecuații ale sistemului:

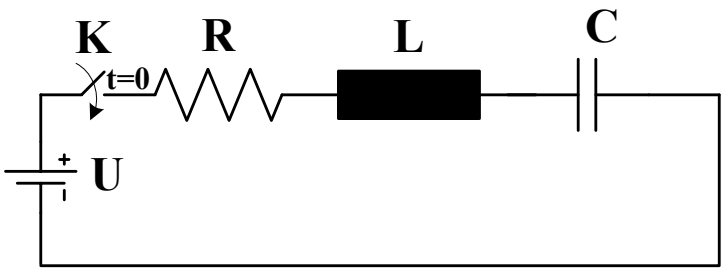
```
function yprim=ecdif(t,y)

yprim=zeros(2,1);

yprim(1)=y(1)*(1-y(2)^2)-y(2);

yprim(2)=y(1);
```

Pentru integrarea sistemului pe intervalul [0, 20], se utilizează instrucțiunile MATLAB:

```
t0=0; tf=20;

y0=[0 0.25];

[t,y]=ode23('ecdif',[t0 tf],y0);

plot(t,y(:,1),t,y(:,2));
```

Aplicația 1 – Rezolvarea unui circuit RLC serie in regim tranzitoriu cu sursa de c.c.

Se consideră un circuit RLC serie alimentat de la o sursa de tensiune constantă printr-un comutator *K*. Sa se determine variația curentului prin circuit si a tensiunii pe condensator, daca la t = 0 se inchide comutatorul *K*.



Se prezintă in continuare o variantă de program in MATLAB pentru rezolvarea acestei probleme. Pentru integrarea ecuației diferentiale se scrie fișierul M-funcție:

```
function Dy=RLC_serie(t,y)
Dy=zeros(2,1);

global R L C u
% rezolvare eq. diff: LCuc''+RCuc'+uc=u
Dy(1)=u/(L*C)+(-R/L)*y(1)-1/(L*C)*y(2);
Dy(2)=y(1);
```

Calculul variației curentului si tensiunii pe condensator se realizează cu secvența:

```
%regim tranzitoriu RLC-serie

clear;clc;

global R L C u  %var. globale pentru a fi accesate din
functia "RLC_serie"

R=3.5;L=10e-3;C=100e-6;u=100;

%determinarea perioadei de regim tranzitoriu:

omega0=1/sqrt(L*C);

omega=omega0*sqrt(1-(R/2/sqrt(L/C))^2);
```

```
T=2*pi/omega;

t0=0; tf=10*T;

%conditii initiale:

i0=0; uc0=0;

%rezolvarea ecuatiei diferentiale:

[t uc]=ode45('RLC_serie',[0 tf],[i0 uc0]);

%reprezentare grafica uc si ic:

subplot(2,1,1);

plot(t,uc(:,2)); title('uc'); grid on;

subplot(2,1,2);

ic=C*uc(:,1);        %ic=C*duc/dt

plot(t,ic); title('ic'); grid on;
```

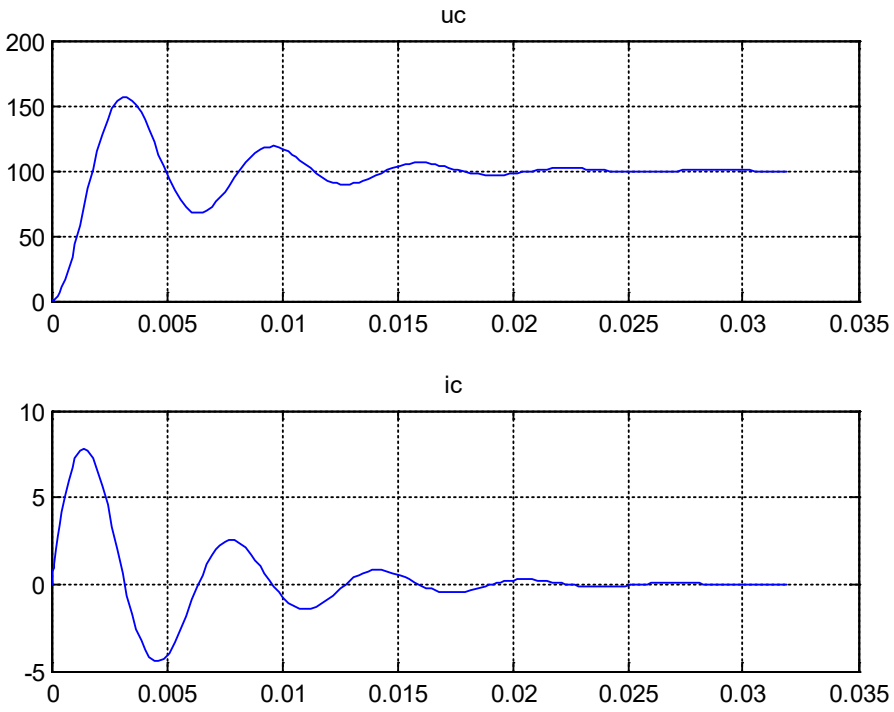


Figura 9.10 *Sus*: variația tensiunii pe condensatorul dintr-un circuit RLC serie în regim tranzitoriu. Graficul arată tensiunea uc(t) în timp, calculată numeric cu ode45 pentru un circuit RLC serie alimentat de o sursă de tensiune constantă, după închiderea comutatorului K la t=0. *Jos*: curentul prin condensatorul din circuitul RLC serie în regim tranzitoriu pe intervalul de timp analizat.

Capitolul 10

Aplicații Matlab/Simulink pentru Modelarea și Simularea Circuitelor Electrice

10.1. Introducere

Simulink reprezintă o componentă integrată a mediului software MATLAB, utilizată pentru modelarea, simularea și analiza sistemelor dinamice, fie ele liniare sau neliniare. Platforma pune la dispoziția utilizatorilor o interfață grafică intuitivă, bazată pe blocuri, care permite construirea schematică a modelelor fără a fi necesară programarea directă în cod [7].

Originea Simulink se află în teoria sistemelor de reglare automată, domeniu în care comportamentul sistemelor dinamice este descris prin funcții de transfer, ecuații diferențiale sau diagrame bloc. Datorită acestui fundament, Simulink este folosit pe scară largă în diferite domenii ca inginerie electrica, control automat, electronică, mecatronică, inginerie mecanică, robotică și multe altele.

Fereastra Simulink poate fi deschisă fie prin tastarea comenzii "Simulink" în workspace-ul MATLAB, fie prin selectarea icoanei dedicate din bara de instrumente, așa cum se prezinta în Figura 10.1.

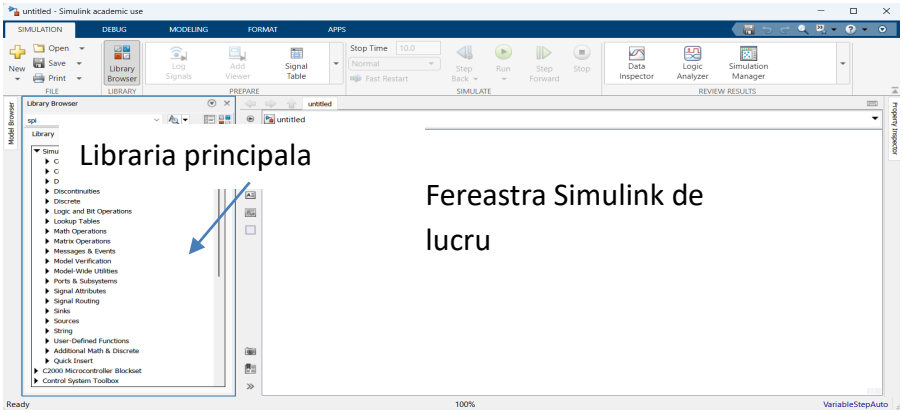


Figura 10.1 Fereastra principala Matlab/Simulink

Odată deschisa interfața Simulink, utilizatorul are acces la biblioteca standard, care include blocuri pentru surse, semnale, operații matematice, sisteme dinamice, instrumente de vizualizare etc. În plus, se pot integra librării suplimentare specializate, denumite *toolbox*-uri, precum *Simscape Electrical*, sau alte extensii axate pe diverse domenii de aplicație, ceea ce extinde semnificativ capabilitățile de modelare. De menționat că, la data publicării acestui material, Universitatea *Transilvania* din Brașov oferă studenților și cadrelor didactice acces complet la toate librăriile Matlab/Simulink prin intermediul licenței de tip *MATLAB campus-wide license.*

10.2. Etapele realizării unui model Simulink

În cele ce urmează vom utiliza termenul de *model* pentru a defini ansamblul de blocuri, conexiuni și setări care descriu comportamentul unui sistem în mediul Simulink. Procesul de realizare a unui model Simulink presupune parcurgerea mai multor etape, fiecare având rolul

de a transforma descrierea conceptuală a sistemului (circuitului în cazul nostru) într-o reprezentare ce poate fi simulată și analizată. Luând ca exemplu un circuit electric, realizarea și analizarea unui model în Simulink este analogă cu implementarea circuitului în laborator și observarea mărimilor de interes cu ajutorul osciloscopului sau al altor aparate de măsura.

Etapele realizării unui model pot fi sintetizate astfel:

- ✓ Se aduc în fereastra de lucru componentele necesare din bibliotecile Simulink și se realizează conexiunile corespunzătoare;
- ✓ Se parametrizează fiecare bloc, dacă este necesar;
- ✓ Se configurează parametrii simulării în fereastra *Configuration Parameters* (CTRL+E):
  - o Stabilirea timpului de simulare;
  - o Alegerea tipului *solverului*: pas variabil (implicit) sau pas discret;
  - o Selectarea solverului utilizat: ode45 (implicit), ode23 etc.;
  - o Setarea pasului maxim de simulare: auto (implicit);
- ✓ Se rulează simularea (butonul Start sau combinația de taste CTRL+T);
- ✓ Se vizualizează semnalele de interes.

10.2.1. Exemplu de realizare a unui model Simulink simplu

Așa cum am menționat anterior, analogia cu un circuit de laborator este foarte utilă pentru înțelegerea funcționării modelelor Simulink. Ca prim exemplu, vom realiza un circuit care conține un generator de semnal sinusoidal și un osciloscop. Modelul Simulink al acestui circuit va arăta

ca în Figura 10.2, în care semnalul de la generator (ex. tensiune) este transmis la osciloscop pentru vizualizare. Cele două blocuri, Signal Generator și Scope, se regăsesc în biblioteca Simulink, în categoriile *Sources* și respectiv *Sinks*. Conexiunea dintre blocuri se realizează prin trasarea unui *fir Simulink*, ținând apăsat click-stânga de la blocul sursă (Signal Generator) până la blocul receptor (Scope). O metodă mai rapidă de trasare automată a legăturilor constă în selectarea blocului sursă, menținerea tastei CTRL apăsată și selectarea blocului receptor cu mouse-ul.

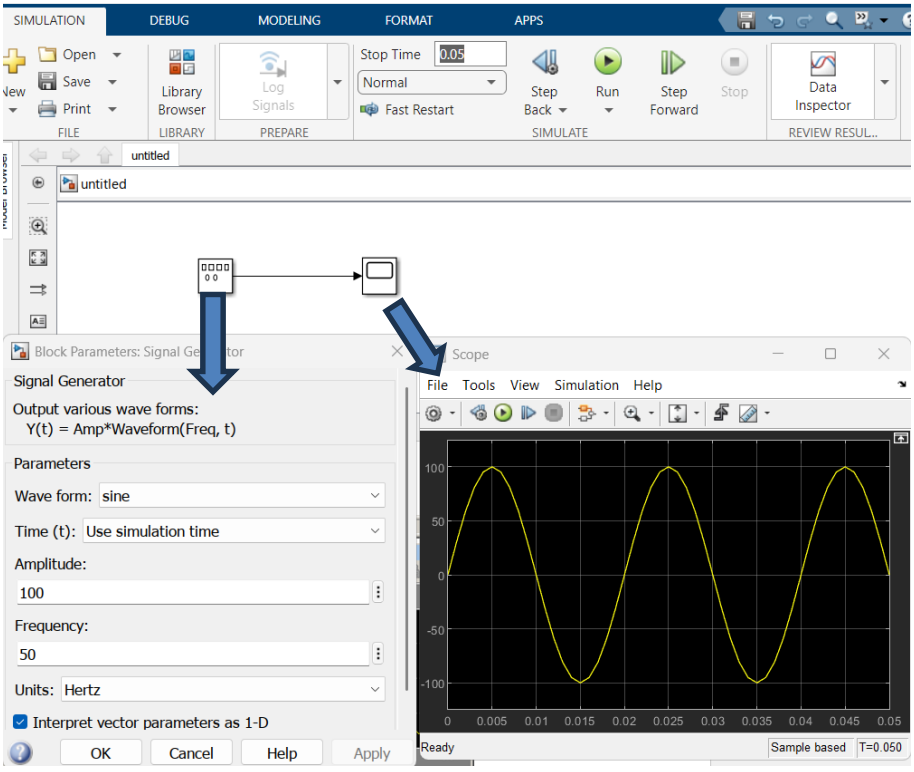


Figura 10.2 Exemplu model Simulink simplu

Majoritatea blocurilor din Simulink conțin parametri care pot fi modificați de utilizator. Aceștia se accesează printr-un dublu click pe blocul respectiv. În exemplul analizat, blocul *Signal Generator* poate fi configurat pentru diferite tipuri de semnale și parametrii corespunzători fiecăruia. Pentru un semnal sinusoidal, parametrii principali sunt amplitudinea și frecvența, ce vor putea fi modificate așa cum se prezintă în Figura 10.2. După finalizarea parametrizării tuturor blocurilor, se setează timpul de simulare, care va depinde de caracteristicile modelului analizat. Etapa legată de *solver* și parametrii acestuia poate fi ignorată deocamdată, lăsându-se valorile implicite. Modelul poate fi apoi salvat cu un nume sugestiv și rulată simularea prin apăsarea butonului *Run*, sau prin combinația de taste CTRL+T. Dacă simularea se desfășoară fără erori, rezultatele pot fi vizualizate. În exemplul nostru, semnalul sinusoidal poate fi observat printr-un dublu click pe blocul *Scope*.

Semnalele din Simulink pot fi agregate (multiplexate) într-un singur fir, analog unui cablu electric multifilar care transmite mai multe semnale simultan. În exemplul nostru, putem combina două semnale pentru a le vizualiza suprapuse pe osciloscop, așa cum se prezintă în Figura 10.3. Aceasta se realizează cu ajutorul blocului *Mux* (multiplexor), iar operația inversă, de separare a semnalelor, se efectuează cu blocul *Demux*.

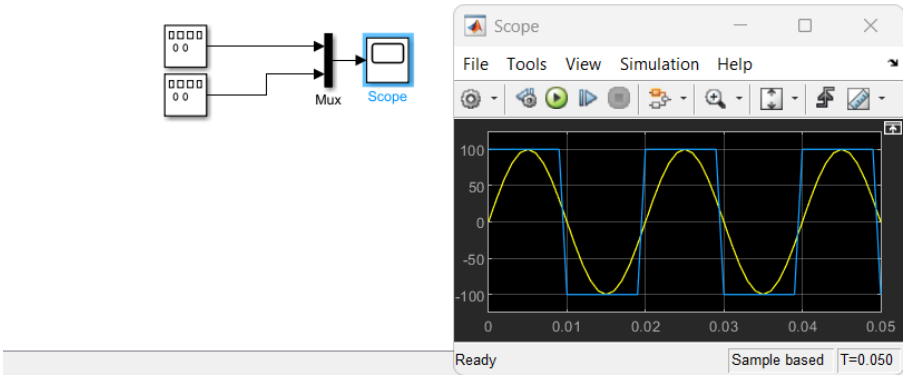


Figura 10.3 Exemplu model Simulink cu semnale multiplexate

10.2.2. Configurarea *solverului* parametrilor de simulare

Așa cum am discutat anterior, pe lângă realizarea modelului și setarea parametrilor blocurilor constituente, o etapă importantă este reprezentată de configurarea parametrilor de simulare. Această etapă este similară cu cea pe care o efectuăm atunci când folosim un osciloscop digital și dorim să parametrizăm baza de timp, frecvența de eșantionare sau modul de achiziție.

Pagina Simulink dedicată parametrilor de simulare se poate accesa din meniul Modeling → Model Settings, sau mai simplu prin combinația de taste CTRL+E. Fereastra *Configuration Parameters* care se deschide (Figura 10.4) prezintă mai multe secțiuni, prima dintre ele, denumită *Solver*, fiind cea de interes pentru parametrizarea simulării. Aici putem seta timpul de simulare (Stop time), același parametru fiind accesibil și în pagina principală a modelului (Figura 10.2). Următoarea secțiune, referitoare la *Solver*, conține parametrii care definesc pasul de simulare

(fix/variabil), metoda de rezolvare a ecuațiilor modelului (solver), precizia simulării și alte elemente specifice, asigurând astfel o simulare corectă și stabilă a modelului. Opțiunea implicită este solver-ul automat cu pas variabil, ceea ce înseamnă că programul alege varianta optimă de solver și pasul de simulare. În majoritatea modelelor simple, această setare implicită este suficientă, însă trebuie să avem în vedere că anumite modele pot necesita configurarea manuală a acestor parametri. Astfel de situații apar atunci când utilizatorul se confruntă cu erori, cu rularea greoaie a simulării sau cu rezultate care nu reflectă corect comportamentul sistemului modelat.

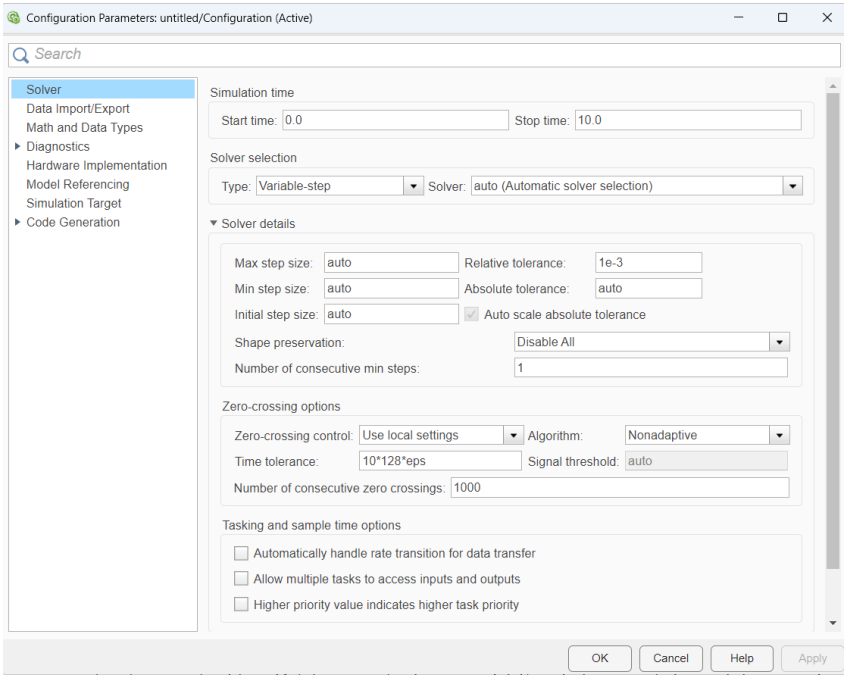
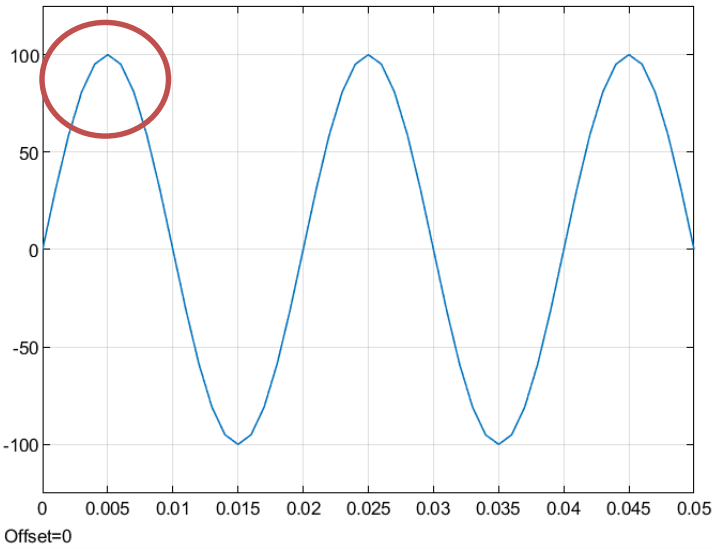
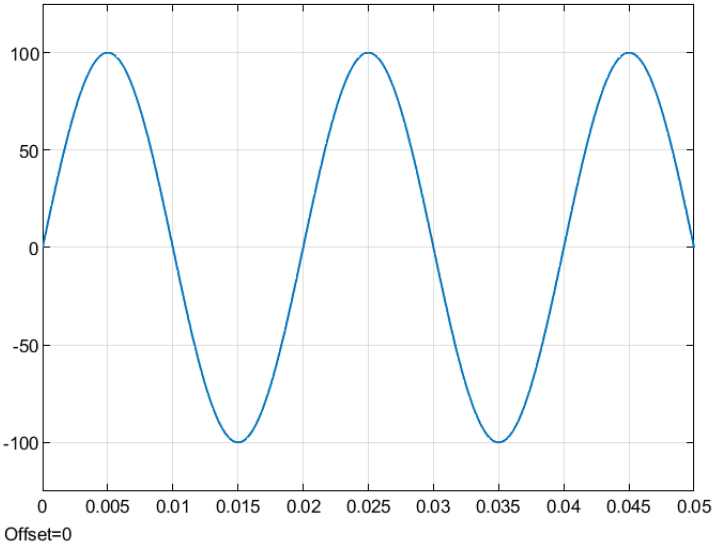


Figura 10.4 Fereastra de configurare a parametrilor modelului (Configuration Parameters)

O altă setare care necesită atenție în pagina din Figura 10.4 este legată de pasul maxim de simulare și toleranța relativă. Primul este implicit setat pe auto, iar al doilea la valoarea 1e-3. Aceste setări implicite pot fi utilizate în majoritatea cazurilor, însă atunci când rezultatele simulărilor nu reflectă fidel procesul pe care îl modelăm, va fi necesară ajustarea lor. Luând ca exemplu modelul din Figura 10.3, o analiză mai detaliată a semnalelor afișate pe ecranul blocului Scope arată că semnalul sinusoidal este ușor deformat la vârfuri, așa cum se evidențiază în Figura 10.6a. Această problemă apare din cauza preciziei reduse de simulare, datorată, în exemplul nostru, utilizării unui pas de simulare variabil calculat automat de program. Pentru a remedia astfel de situații, putem modifica parametrul *Max step size* din fereastra Configuration Parameters, alegând o valoare suficient de mică astfel încât semnalele vizualizate pe osciloscop să fie redate fidel, în funcție de cerințele aplicației. De exemplu, în cazul semnalului sinusoidal cu frecvență de 50 Hz, dacă setăm pasul maxim de simulare la 100e-6, vom obține o reprezentare suficient de precisă a semnalului, așa cum se observă în Figura 10.6b.



a)



b)

Figura 10.5 Efectul setării *Max step size* pentru îmbunătățirea rezultatelor unei simulări: a) Max step size: auto; b) Max step size: 100e-6

10.3. Exportarea rezultatelor unei simulări

10.3.1 Exportarea semnalelor Scope în spațiul de lucru Matlab

După rularea unui model, semnalele atașate unui bloc Scope pot fi vizualizate în fereastra caracteristică acestui bloc. Datele generate de un model Simulink nu sunt disponibile în mod implicit în workspace-ul MATLAB. Pentru a exporta semnale Simulink în spațiul de lucru al MATLAB-ului, există mai multe metode, iar în continuare, se prezintă una dintre acestea.

Blocurile Scope pot fi configurate să salveze semnalele în workspace-ul MATLAB accesând fereastra *Parameters*, în tabul *Data History* (Figura 10.6). Acolo se bifează opțiunea *Save data to workspace*, se atașează un nume variabilei și se alege formatul de salvare. Este recomandat să se folosească formatele *Structure* sau *Structure with time*, deoarece acestea grupează semnalul (axa Oy) și timpul (axa Ox) într-o structură ușor de accesat în MATLAB. De exemplu, dacă variabila asociată blocului Scope se numește „tensiune1” și s-a ales formatul *Structure with time*, semnalul poate fi accesat și reprezentat grafic în MATLAB astfel:

```
plot(tensiune1.time, tensiune1.signals.values)
```

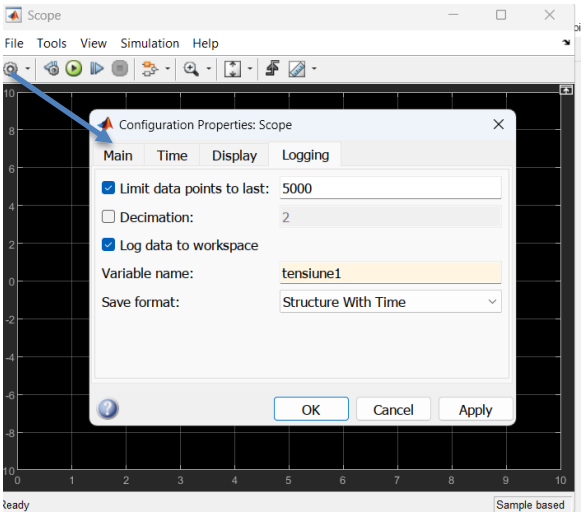


Figura 10.6 Setarea salvării datelor unui bloc Scope

De menționat că, în versiunile recente de MATLAB, datele salvate dintr-un model Simulink în *workspace* sunt împachetate într-o singură structură, care trebuie despachetată înainte de a putea fi utilizată. Pentru a simplifica folosirea acestei opțiuni, se recomandă dezactivarea împachetării datelor într-o singură structură. Acest lucru se realizează prin debifarea opțiunii *Single simulation output* din setările modelului Simulink (CTRL+E), în fereastra *Data Import/Export* (Figura 10.7).

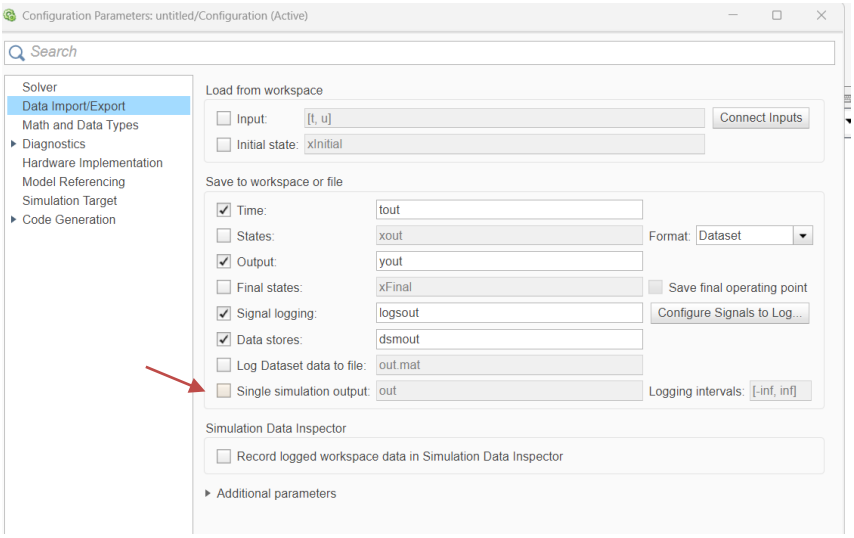


Figura 10.7 Setarea exportării datelor din Simulink în workspace-ul Matlab

10.3.2 Salvarea ecranelor blocurilor *Scope*

Varianta prezentată anterior de exportare a semnalelor Simulink în spațiul de lucru MATLAB permite prelucrarea acestora cu ajutorul funcțiilor de reprezentare grafică. O metodă mai simplă de salvare a rezultatelor unei simulări din blocurile Scope, în diferite formate (grafic sau numeric), constă în utilizarea opțiunilor *Copy to Clipboard* sau *Print to Figure*, disponibile în meniul *File* al fiecărui bloc Scope. De menționat că prima opțiune, *Copy to Clipboard*, generează o imagine folosind setările implicite de rezoluție și dimensiune, care pot fi configurate în fereastra principală MATLAB, din meniul *Preferences*, secțiunea *Figure Copy*. Cea de-a doua opțiune, *Print to Figure*, exportă conținutul blocului Scope într-o fereastră de tip MATLAB *Figure*, care poate fi ulterior editată și salvată la fel ca orice altă figură MATLAB.

10.4. Crearea subsistemelor

Pentru organizarea unui model, în special în cazul celor complexe, se pot utiliza subsisteme care încorporează componente ale modelului. Simulink oferă mai multe tipuri de blocuri de tip subsistem, disponibile în librăria „Ports & Subsystems", din care utilizatorul poate alege în funcție de necesități. Un subsistem poate avea una sau mai multe intrări și ieșiri, permițând interacțiunea cu restul modelului. Crearea unui subsistem se poate face fie manual, prin plasarea blocurilor componente direct în fereastra subsistemului, fie automat, prin selectarea unui grup de blocuri din fereastra principală, urmată de clic dreapta și alegerea opțiunii „Create Subsystem". După cum se prezinta în Figura 10.8, subsistemul poate fi de tip Normal, sau poate fi controlat de un semnal exterior de tip *Trigger* sau *Enable*, oferind astfel flexibilitate în simularea și organizarea modelului. În ultimele două situații, conținutul subsistemului va fi rulat în simulare doar când este îndeplinită condiția semnalului de control extern.

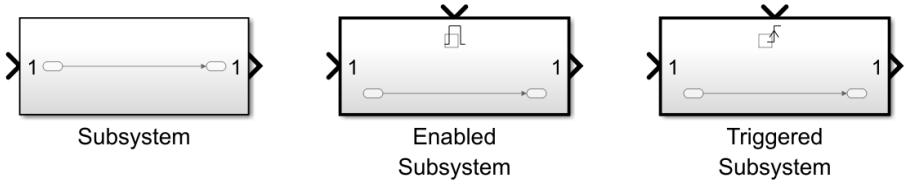


Figura 10.8 Exemple subsisteme Simulink.

10.5. Inițializarea parametrilor unei simulări

După cum vom vedea în cele ce urmează, modelele Simulink ale circuitelor electrice analizate necesită definirea unor parametri precum

rezistența, inductanța, tensiunea etc. Aceste valori trebuie stabilite înainte de rularea simulării. Blocurile asociate acestor parametri (de exemplu, blocurile *Gain* sau *Constant*) pot fi configurate fie prin introducerea directă a unei valori numerice, fie prin utilizarea unor variabile. În cazul utilizării variabilelor, acestea trebuie să fie deja definite în spațiul de lucru MATLAB în momentul pornirii simulării; în caz contrar, Simulink va genera o eroare. Definirea variabilelor se poate realiza în mai multe moduri:

- ✓ Direct în Workspace-ul MATLAB;
- ✓ Într-un fișier script (*.m), care trebuie rulat înainte de simulare;
- ✓ În secțiunea de inițializare a modelului Simulink (variantă recomandată).

Această ultimă metodă poate fi accesată prin click dreapta în fereastra modelului Simulink, selectarea opțiunii *Model Properties*, apoi accesarea ferestrei *Callbacks*, unde se alege opțiunea *InitFcn*. În această secțiune se pot introduce comenzi MATLAB care vor fi executate automat la inițializarea simulării, inclusiv definirea parametrilor necesari modelelor circuitelor electrice.

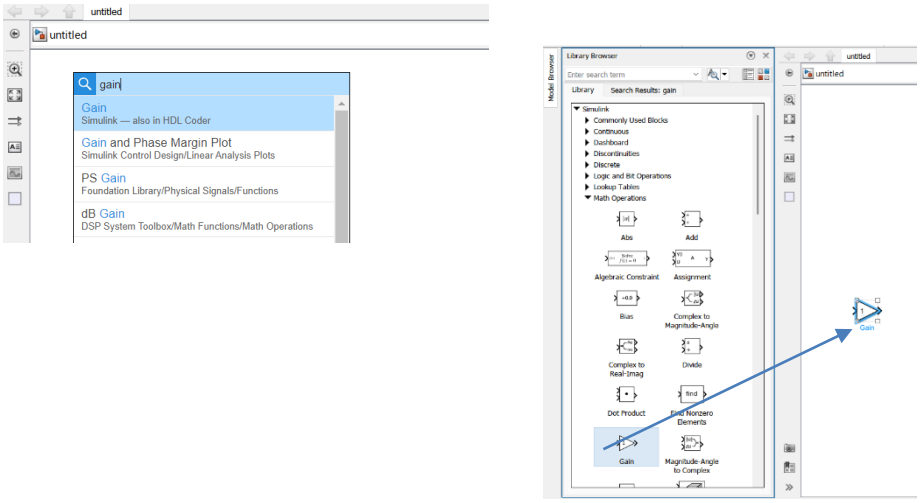
10.6. Modelarea și simularea circuitelor electrice în Matlab/Simulink

În continuare, pentru exemplificarea modului în care pot fi realizate modele Simulink ale unor procese sau sisteme fizice, vor fi luate ca exemplu circuite electrice cu care studenții de la domeniul de inginerie electrică sunt deja familiarizați. Astfel, vom modela cu ajutorul Simulink circuite electrice în regim tranzitoriu și vom vizualiza tensiuni și curenți pe osciloscop, într-un mod similar realizării și utilizării unui montaj de laborator.

Ca elemente principale din librăria Simulink vom utiliza următoarele blocuri:

Gain	
Add	
Integrator	
Step	

Toate blocurile utilizate pot fi aduse în fereastra de lucru Simulink fie prin tastarea denumirii acestora oriunde în spațiul liber al foii de lucru, fie prin căutarea lor în librăria Simulink, așa cum se prezintă în Figura 10.9.



a) b)

Figura 10.9 – Exemplificarea încărcării blocurilor în fereastra de lucru Simulink: a) prin tastarea denumirii blocului; b) prin căutarea blocului în librăria Simulink și încărcarea acestuia prin *drag & drop*.

Modelarea va porni de la relațiile matematice diferențiale care descriu comportarea în instantaneu a circuitului electric. Aceste relații sunt studiate în cadrul cursurilor de circuite electrice, motiv pentru care nu vom insista asupra detaliilor teoretice, ci ne vom concentra pe modul în care aceste ecuații pot fi transpuse în modele Simulink.

10.6.1. Circuit RL serie

Unul dintre primele circuite electrice studiate este circuitul RL serie alimentat de la o sursă de tensiune oarecare, conform schemei din Figura 10.10. Pentru a exemplifica modul în care un astfel de circuit poate fi modelat și simulat în Simulink, este necesar să definim numeric

valorile parametrilor circuitului și ale sursei de alimentare. Astfel, se consideră un circuit RL serie alimentat de la o sursă de tensiune *u*, care poate avea diferite forme de variație în timp (treaptă, sinusoidală, dreptunghiulară etc.). Scopul este realizarea unui model al circuitului care să permită determinarea curentului prin circuit atât în regim tranzitoriu, cât și în regim staționar.

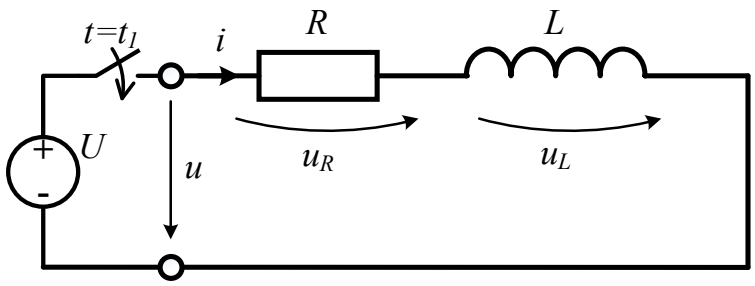


Figura 10.10 Schema circuitului RL serie

Circuitul cu parametrii R (rezistența) și L (inductanța) poate fi descris în instantaneu prin ecuația diferențială:

$$u = u_R + u_C = Ri + L\frac{di}{dt} \tag{10.1}$$

Pentru a extrage curentul prin circuit integrăm ecuația anterioară, rezultând:

$$i(t) = \int \frac{1}{L} u_L dt = \int \frac{1}{L}(u - u_R)dt = \int \frac{1}{L}(u - Ri)dt \tag{10.2}$$

Aceasta ultimă formă a ecuației ce definește circuitul RL serie în instantaneu va fi utilizată pentru dezvoltarea modelului Simulink. Din

aceasta ecuație rezultă operațiile care vor trebui implementate cu blocuri în modelul Simulink, astfel:

- căderea de tensiune pe rezistenta: $u_R = R \cdot i$

- căderea de tensiune pe inductanță: $u_L = u - u_R$

- curentul prin inductanța, respectiv prin circuit: $i = \int \frac{1}{L} u_L dt + i_L(0)$, unde $i_L(0)$ reprezintă valoarea inițială (la t = 0) a curentului prin inductanță (implicit zero).

Pentru modelare, este necesar să definim numeric parametrii circuitului, iar în continuare vom lua ca exemplu următoarele valori:

- Tensiunea de alimentare se consideră de tip treapta, ce variază de la 0 la 100V la momentul t_1=0.1s;

- Rezistența și inductanța: *R=1Ω, L=0.1H.* Curentul inițial prin inductanță: $i_L(0)=0$.

Modelul Simulink rezultat din ecuația (10.2) a circuitului analizat este prezentat în Figura 10.11. După cum se poate observa, modelul include blocurile descrise în secțiunea anterioară, adică *Step*, *Add*, *Gain*, *Integrator* și *Scope*. Blocurile constituente vor fi parametrizate după cum urmează:

- Blocul Step, ce modelează sursa de alimentare tip treaptă, va fi parametrizat cu: *Step time=0.1*; *Initial value=0*; *Final value=100*.

- Blocul Add se parametrizeaza cu: *List of signs:*+−

- Cele două blocuri *Gain* se parametrizează astfel: *Gain=1/L*; respectiv *Gain=R*. Valorile pentru R și L pot fi introduse direct sau pot fi definite sub formă de variabile, urmând ca înaintea rulării simulării aceste variabile să fie declarate în workspace cu valorile corespunzătoare (conform celor prezentate în secțiunea 10.5).

- Blocul Integrator va fi menținut cu parametrii impliciți. Dacă există o valoare inițială a curentului prin bobină diferită de zero, aceasta va fi setată în parametrul *Initial condition* al integratorului.

Timpul de simulare se alege astfel încât să includă întregul regim tranzitoriu. În exemplul ales, s-a considerat un timp de simulare de 0.6 s.

Pentru a simula regimul tranzitoriu al circuitului RL cu semnal sinusoidal la intrare, în locul blocului Step se aduce din librăria *Sources*, blocul *Signal Generator* sau *SineWave*. Pe lângă curentul prin circuit, pot fi vizualizate pe osciloscop și alte mărimi ale circuitului, cum ar fi tensiunea sursei (*u*), precum și căderile de tensiune pe bobină (uL) și pe rezistență (uR).

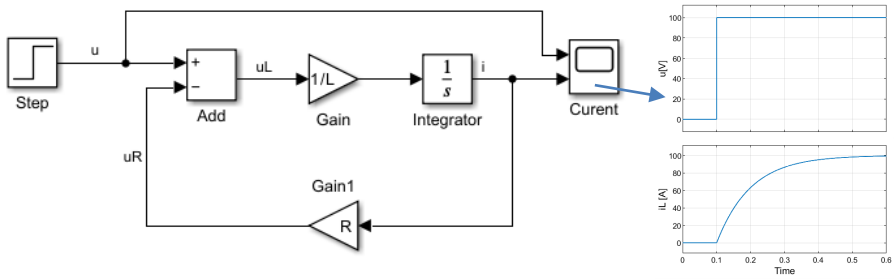


Figura 10.11 Modelul Simulink al circuitului RL serie

10.6.2. Circuit RC serie

Circuitul RC serie având schema din Figura 10.12 poate fi reprezentat prin ecuația diferențială:

$$u = u_R + u_C = Ri + \frac{1}{C}\int idt \qquad (10.3)$$

Pentru a extrage curentul prin circuit vom rescrie ecuația astfel:

$$i = \frac{u_R}{R} = \frac{u - u_C}{R} = \frac{1}{R}\left(u - \frac{1}{C}\int idt\right) \qquad (10.4)$$

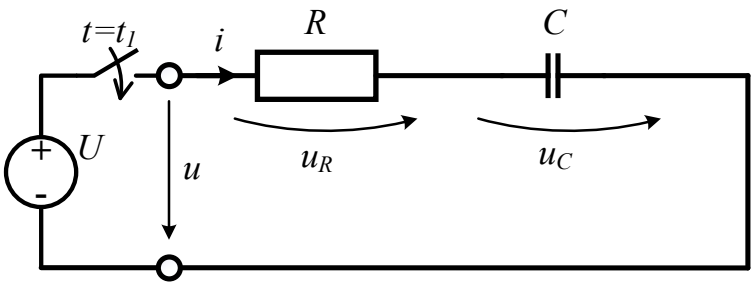


Figura 10.12 Schema circuitului RC serie

Așa cum am procedat și în cazul circuit RL serie, vom considera și pentru acest circuit valori numerice pentru elementele componente, după cum urmează: u=100V semnal tip treapta la t=0.1s; R=1Ω, C=0.1F și valoarea inițială a tensiunii pe condensator $u_C(0)=0$.

Modelul Simulink rezultat este prezentat în Figura 10.13. Se observă că sunt utilizate aceleași blocuri ca în exemplul anterior, însă acestea vor fi interconectate astfel încât să reprezinte de data aceasta ecuația (10.4). Așadar, descompunând ecuația în componentele ei de baza, vom avea

următoarele operații ce sunt implementate cu blocuri Simulink în modelul din Figura 10.13:

- căderea de tensiune pe condensator: $u_C = \frac{1}{C}\int idt + u_C(0)$, unde $u_C(0)$ este valoarea inițială (la t = 0) a tensiunii pe condensator (implicit zero).

- căderea de tensiune pe rezistență: $u_R = u - u_C$;

- curentul prin rezistență, respectiv prin circuit: $i = \frac{1}{R} \cdot u_R$.

Așa cum s-a precizat și la exemplul RL serie, pentru a simula regimul tranzitoriu al circuitului RC serie cu semnal sinusoidal la intrare, în locul blocului Step se aduce din librăria *Sources*, blocul *Signal Generator* sau *SineWave*. Pe lângă curentul prin circuit, pot fi vizualizate pe osciloscop și alte mărimi ale circuitului, cum ar fi tensiunea sursei (u), căderile de tensiune pe condensator (uC) și pe rezistență (uR).

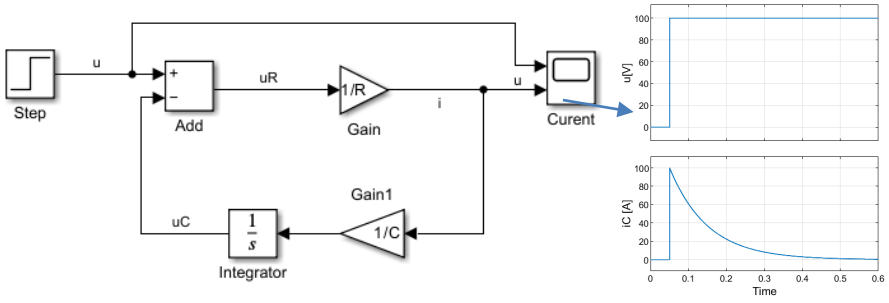


Figura 10.13 Modelul Simulink al circuitului RC serie

10.6.3. Circuit RLC serie

Circului RLC prezentat în Figura 10.14 poate fi modelat plecând de la circuitul RL serie prezentat anterior și adăugând ecuația tensiunii pe condensator prezentată la circuitul RC serie. Așadar, ecuația diferențială va rezulta sub următoarea forma:

$$u = u_R + u_L + u_C = Ri + L\frac{di}{dt} + \frac{1}{C}\int idt \qquad (10.5)$$

Similar circuitului RL serie, vom extrage curentul prin circuit astfel:

$$\begin{aligned} i(t) &= \int \frac{1}{L}(u - u_R - u_C)dt \\ &= \int \frac{1}{L}\left(u - Ri - \frac{1}{C}\int idt\right)dt \end{aligned} \qquad (10.6)$$

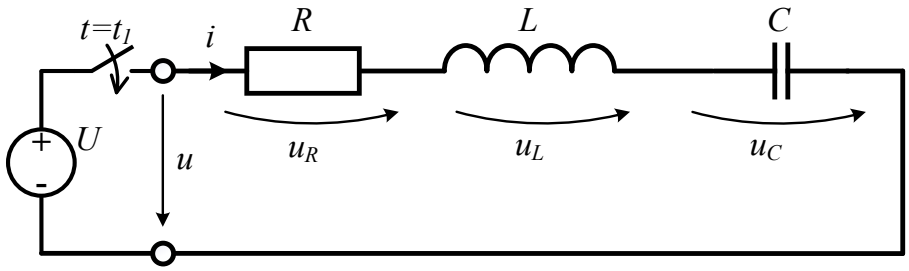


Figura 10.14 Schema circuitului RLC serie

Vom considera și pentru acest circuit valori numerice pentru elementele componente, după cum urmează: u=100V semnal tip treapta la t=0.1s; R=1Ω, L=0.1H, C=0.1F, valoarea inițială a curentului prin inductanța $i_L(0)=0$, iar valoarea inițială a tensiunii pe condensator $u_c(0)=0$.

Modelul Simulink rezultat este prezentat în Figura 10.15, în care observăm o combinație a modelelor dezvoltate anterior pentru circuitele RL și RC serie. Având în ecuația (10.6) trei tensiuni care se însumează (u, uR și uC), blocul Add va fi parametrizat *List of signs* +−− . Timpul de simulare va fi setat la o valoare care să cuprindă întreg regimul tranzitoriu (ex. 1.2s).

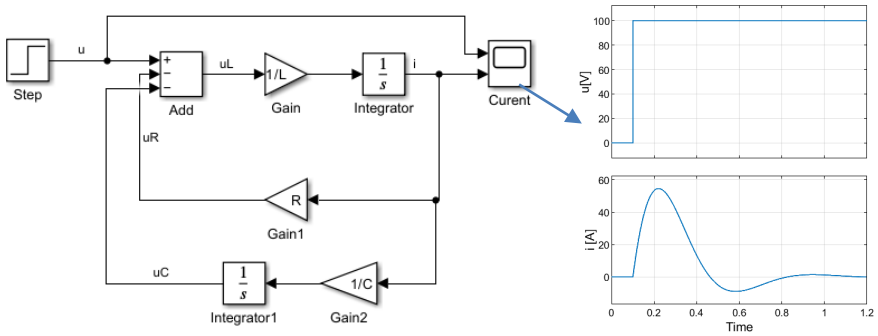


Figura 10.15 Modelul Simulink al circuitului RLC serie

10.6.4. Circuit C-RL serie cu tensiune inițială pe condensator

O variantă particulară a circuitului RLC serie este reprezentată în Figura 10.16, unde condensatorul este inițial încărcat cu o tensiune U_{C0}, iar la momentul t = t1 se închide comutatorul K, punând în serie condensatorul cu grupul RL serie. Se poate observa că, după închiderea comutatorului, circuitul rezultat este identic cu cel prezentat în exemplul anterior, cu deosebirea că nu mai există o tensiune de intrare, motiv pentru care se va considera u = 0. De asemenea, trebuie ținut cont de polaritatea tensiunii inițiale a condensatorului în raport cu sensul pozitiv al tensiunii pe acesta. Așa cum este reprezentat condensatorul

în schemă, polaritatea tensiunii U_{C0} este în sens invers curentului și sensului considerat pozitiv al căderii de tensiune pe condensator. Prin urmare, pentru o reprezentare corectă a circuitului, în modelul Simulink se va considera tensiunea inițială pe condensator egală cu $-U_{C0}$.

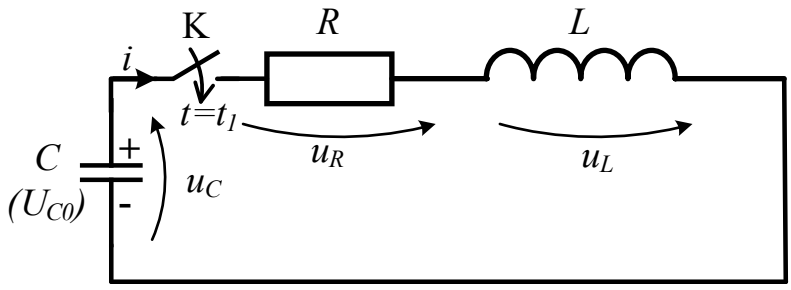


Figura 10.16 Schema circuitului C-RL serie, cu tensiune inițială pe condensator

Modelul Simulink al circuitului este prezentat în Figura 10.17. Pentru a seta valoarea inițială U_{C0}, se va utiliza integratorul corespunzător ecuației tensiunii pe condensator (Integrator1). În setările integratorului, parametrul *Initial Condition* va fi egal cu $-U_{C0}$. Valoarea inițială poate fi impusă intern sau extern, în funcție de opțiunea selectată pentru *Initial Condition Source*. În exemplul prezentat a fost aleasă opțiunea de condiție inițială furnizată extern. Odată selectată această opțiune, integratorul va avea o intrare separată, la care poate fi conectat un bloc de tip Constant, ce furnizează valoarea tensiunii inițiale. De asemenea, pentru a considera un moment inițial de conectare a comutatorului t1 diferit de zero, se utilizează un bloc de tip *Enabled Subsystem*, la care se conectează un bloc Step, prin intermediul căruia poate fi stabilit momentul t1 (parametrul *Step time*). Pe lângă curentului

prin circuit, pot fi vizualizate cu ușurință și căderile de tensiune pe fiecare element. În exemplul din Figura 10.17 sunt afișate pe același osciloscop atât curentul, cât și tensiunea pe condensator.

Parametrii circuitului în exemplul prezentat sunt: t1 = 0.1s, R = 1Ω, L = 0.1H, C = 0.1F, valoarea inițială a curentului prin inductanță iL(0) = 0, iar valoarea inițială a tensiunii pe condensator uc(0) = 10V.

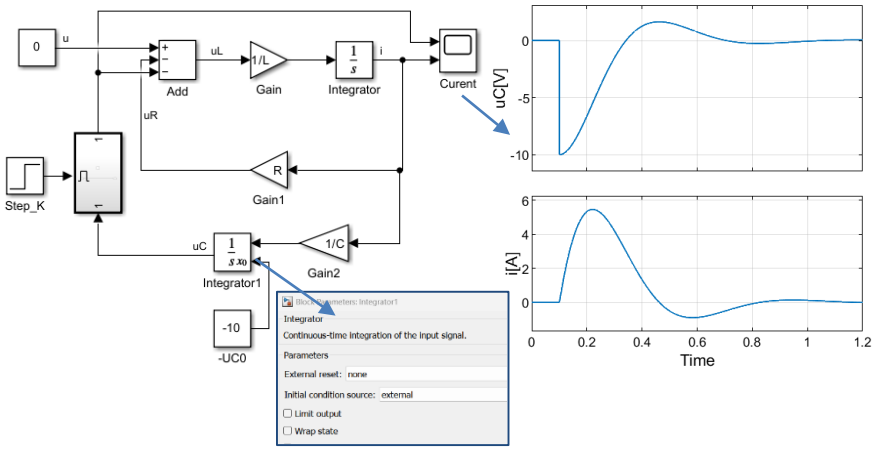


Figura 10.17 Modelul Simulink al circuitului C-RL serie, cu tensiune inițială pe condensator

10.6.5. Circuit R-LC

Schema circuitului analizat în continuare este prezentată în Figura 10.18 și este format dintr-o rezistenta R în serie cu un grup LC paralel. Circuitul poate fi alimentat cu o tensiune oarecare, ca și în exemplele anterioare, dar de data aceasta vom exemplifica cazul alimentării cu o sursă de tensiune alternativă sinusoidală.

Circuitul poate fi exprimat prin ecuațiile următoare:

$$u = u_R + u_C = Ri + \frac{1}{C}\int i_C dt \tag{10.7}$$

$$u_L = u_C$$

$$i = i_C + i_L \tag{10.8}$$

$$i_L = \frac{1}{L}\int u_L dt$$

Rescriind aceste ecuații într-o formă optimă, rezultă:

$$i = \frac{1}{R}\left(u - \frac{1}{C}\int i_C dt\right) \tag{10.9}$$

$$i_C = i - \frac{1}{L}\int u_L dt$$

Modelul Simulink ce reiese din ecuațiile mai sus este prezentat în Figura 10.19. Pentru exemplificare numerică, vom considera parametri circuitului: tensiunea de intrare sinusoidală cu amplitudinea de 100V și frecvența 50Hz (faza inițială zero), R=1Ω, L=0.1H, C=10μF, valoarea inițială a curentului prin inductanță $i_L(0)=0$, iar valoarea inițială a tensiunii pe condensator $u_C(0)=0$. Timpul de simulare se alege astfel încât să cuprindă regimul tranzitoriu și cel staționar (în exemplul nostru este de 0.3s). Similar cu exemplele anterioare, pe lângă curentul de intrare *i* putem vizualiza și căderile de tensiune pe fiecare element, sau curenții prin bobină și condensator.

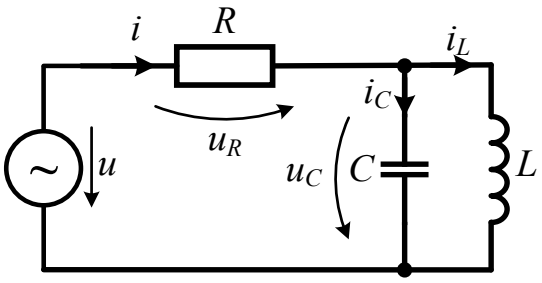


Figura 10.18 Schema circuitului R-LC analizat

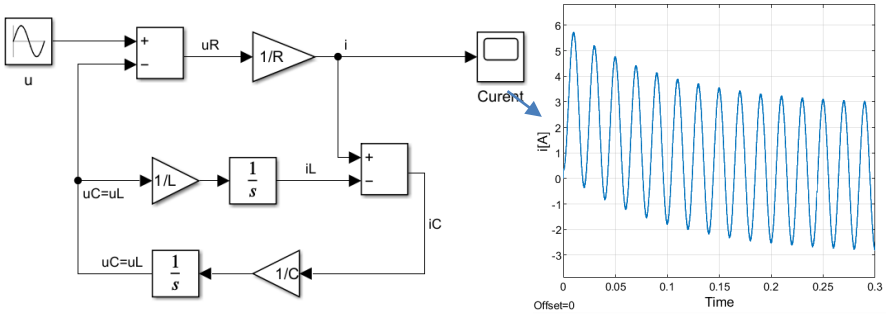


Figura 10.19 Modelul Simulink al circuitului R-LC

10.6.6. Redresor monofazat mono-alternanță cu sarcină RLE

Redresoarele au aplicații diverse în ingineria electrică și fac parte din categoria circuitelor neliniare. Cel mai simplu tip de redresor este redresorul monofazat mono-alternanță (Figura 10.20), care include o singură diodă. Studenții sunt deja familiarizați cu acest tip de dispozitiv semiconductor, dioda fiind studiată în cadrul cursului de electronică analogică. Noi vom considera dioda ca un simplu comutator, permițând trecerea curentului electric în sens pozitiv dinspre anod spre catod atunci când este polarizată direct (tensiune anod–catod pozitivă) și blocând curentul în caz contrar. Sarcina considerată în acest exemplu

este reprezentată de o rezistență în serie cu o inductanță și o sursă de tensiune continuă. În practică, de exemplu, sarcina poate fi un motor de curent continuu, unde tensiunea continuă E reprezintă tensiunea electromotoare, iar RL rezistența și inductanța circuitului statoric.

Ecuația cu ajutorul căreia modelăm circuitul pornește de la relația specifică unui circuit RL serie, la care se adaugă efectul tensiunii continue E. De asemenea, funcționarea diodei ca element de comutație unidirecțională, așa cum a fost prezentată mai sus, este reprezentată prin condiția de a avea doar valori pozitive ale curentului prin circuit (adică să circule de la anod la catod, conform poziționării diodei în circuit). De menționat că, pentru simplificare, căderea de tensiune pe diodă se consideră zero. Cu aceste detalii, ecuația redresorului analizat poate fi scrisă astfel:

$$i = \begin{cases} \frac{1}{L}\int (u - Ri - E)dt, \text{dacă } i > 0 \\ 0, \text{în rest} \end{cases} \tag{10.10}$$

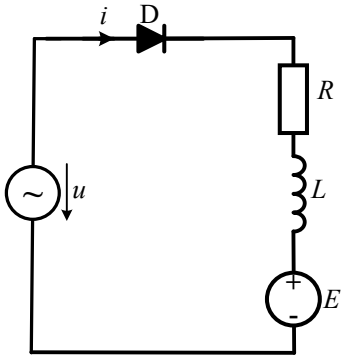


Figura 10.20 Schema redresorului monofazat mono-alternanță cu sarcină RLE

Reprezentarea condiției din ecuația de mai sus poate fi implementată simplu în modelul Simulink prin utilizarea opțiunii *Saturation* în blocul *Integrator*. Astfel, se selectează limita de jos egală cu 0 (nu se permit valori negative ale curentului), iar pentru dezactivarea limitei superioare introducem variabila implicită Matlab *inf*. Modelul Simulink al redresorului analizat va rezulta ca în Figura 10.21. Pentru exemplificare numerică, vom considera ca parametri ai circuitului: tensiunea de intrare sinusoidală cu amplitudinea de 20V și frecvența 50Hz (faza inițială zero), R = 0.1Ω, L=10mH, E=12V. Curentul inițial prin bobină se consideră nul.

Se observă că, odată setate limitele superioară și inferioară, simbolul blocului Integrator se modifică pentru a indica această funcționalitate. Dacă vizualizăm curentul prin circuit, observăm că acesta are valori pozitive atunci când se îndeplinește condiția de polarizare directă a diodei. De menționat că, spre deosebire de modelele circuitelor RL anterioare, în cazul de față căderea de tensiune pe bobina L nu mai poate fi vizualizată în mod direct din semnalul care intră în blocul Gain (1/L). Pentru o interpretare corectă din punct de vedere fizic al tensiunii uL, vom adaugă în model un bloc *Compare to zero* conectat la semnalul curentului *i*. Așadar, tensiunea uL va fi data de relația:

$$u_L = (u - E - u_R) * (i > 0) \qquad (10.11)$$

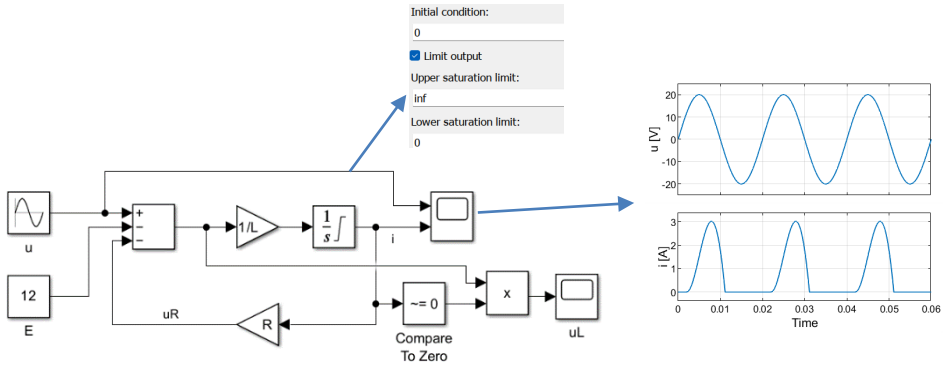


Figura 10.21 Modelul Simulink al redresorului monofazat mono-alternanță cu sarcină RLE

10.6.7. Redresor monofazat bialternanță cu sarcină RLE

Un alt redresor monofazat frecvent întâlnit în aplicațiile practice este redresorul bialternanță, care permite redresarea ambelor alternanțe (pozitivă și negativă) ale tensiunii de intrare prin utilizarea a patru diode montate în configurație punte. Figura 10.22 prezintă schema acestui redresor cu sarcină de tip RLE. Ne propunem realizarea unui model în Simulink care să permită vizualizarea curentului i_s furnizat de sursa alternativă și a curentului i_d prin sarcină.

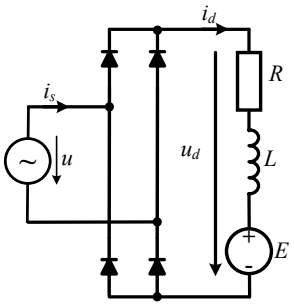


Figura 10.22 Schema redresorului monofazat bialternanță cu sarcină RLE

Pentru modelarea redresorului vom utiliza același principiu ca în cazul anterior, respectiv vom condiționa polaritatea curentului i_d doar pentru valori pozitive, folosind opțiunea *Saturation* a blocului *Integrator*. Ecuația curentului prin sarcină rămâne aceeași ca în relația (10.5). Diferența față de redresorul monoalternanță constă în faptul că tensiunea redresată u_d aplicată sarcinii este formată din alternanța pozitivă a tensiunii de intrare u și din alternanța negativă inversată de redresor. Și de această dată se neglijează căderile de tensiune pe diode.

Modelarea efectului redresorului asupra tensiunii de intrare se realizează prin aplicarea funcției matematice modul (valoare absolută), implementată în Simulink cu ajutorul blocului *abs*. În aceste condiții, modelul Simulink al redresorului analizat este prezentat în Figura 10.23. Pentru exemplificare numerică se consideră următorii parametri ai circuitului: tensiune de intrare sinusoidală cu amplitudinea de 20 V, frecvența de 50 Hz și faza inițială zero, rezistența R = 0.1Ω , inductanța L=1mH și sursa de tensiune continuă E = 12 V. Curentul inițial prin bobină se considera nul.

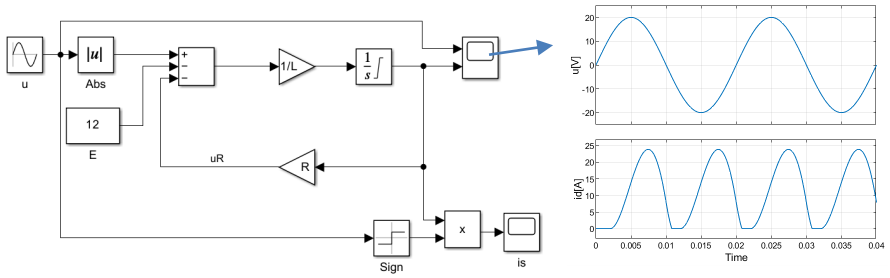


Figura 10.23 Modelul Simulink al redresorului monofazat bialternanță cu sarcină RLE

Așa cum s-a descris în cazul redresorului monoalternanță, tensiunea pe bobina nu poate fi interpretata direct, ci prin intermediul unui calcul suplimentar, așa cum a fost prezentat în Figura 10.21. În modelul redresorului bialternanță avem și curentul sursei de intrare (i_s) care, de data aceasta, este diferit de cel de sarcină. Astfel, în modelul din Figura 10.23 a fost introdusă o secțiune de calcul a acestui curent, bazată pe ecuația de mai jos:

$$i_s = i_d \cdot sign(u) \tag{10.12}$$

10.6.8. Redresor bialternanță cu sarcină RC

Un ultim circuit discutat în cadrul acestui capitol este redresorul bialternanță prezentat anterior, dar cu sarcină RC paralel. Acest circuit este frecvent întâlnit în etajul de intrare al surselor de alimentare utilizate în numeroase aparate electronice alimentate de la rețeaua monofazată de tensiune. Condensatorul are rol de filtrare a tensiunii redresate. Schema circuitului este prezentată în Figura 10.24. Având condensatorul montat în paralel cu sursa de tensiune, schema include și o rezistență serie (r_s), care permite limitarea curentului dintre sursa de tensiune u si condensator.

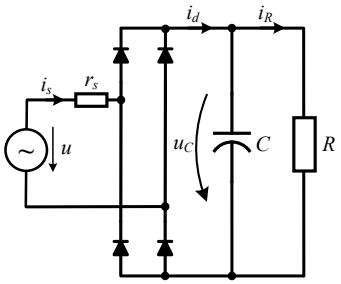


Figura 10.24 Schema redresorului monofazat bialternanță cu sarcină RC-paralel

Principalele ecuațiile care stau la baza modelului circuitului din Figura 10.24, pot fi descrise astfel:

$$i_d = \frac{1}{r_s}(|u| - u_c), pentru\ |u| \geq u_c \qquad (10.13)$$

$$u_c = \frac{1}{C}\int \left(i_d - \frac{u_c}{R}\right) dt \qquad (10.14)$$

Curentul de la sursa de intrare (i_s) va fi reprezentat de aceeași ecuație ca în (10.12).

Modelul Simulink al redresorului rezulta cel din Figura10.25. Reprezentarea condiției din ecuația (10.13) se realizează cu blocul *Saturatio*n, care va fi setat cu limita inferioară egală cu zero. Restul elementelor sunt utilizate în același mod ca în exemplele anterioare. Pentru exemplificare numerică se consideră următorii parametri ai circuitului: tensiune de intrare sinusoidală cu amplitudinea de 100V, frecvența de 50 Hz și faza inițială zero, rezistența R = 100Ω, capacitatea C=220μF, rezistența serie de intrare r_s=0.5 Ω. Tensiunea inițială pe condensator se consideră egală cu zero.

Se exemplifică, de asemenea, o variantă de integrare a mai multor forme de undă pe același osciloscop, pentru a putea fi vizualizate sincron. Astfel, putem vedea simultan: curentul de intrare (i_s), curentul redresat (i_d) și tensiunile u și u_C.

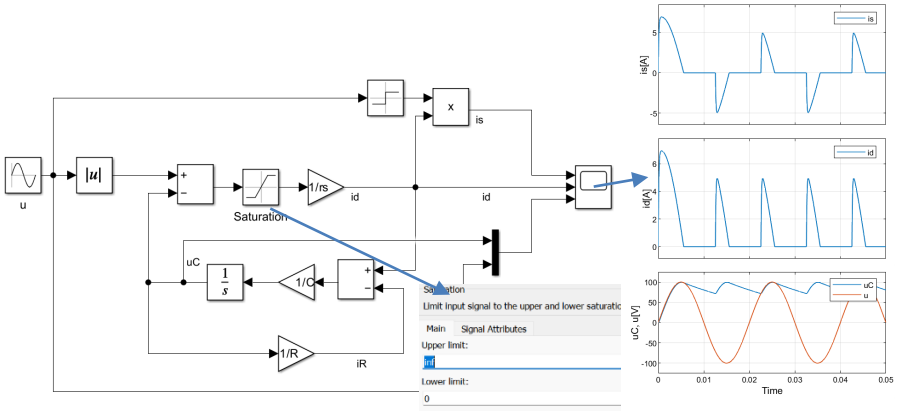


Figura 10.25 Modelul Simulink al redresorului monofazat bialternanță cu sarcină RC

Bibliografie

[1] Mathworks, Matlab, https://www.mathworks.com/

[2] Ghinea Marin, Fireteanu Virgiliu. (2003). *MATLAB – Calcul numeric, Grafica, Aplicații*. București: Editura Teora.

[3] Tudorache, Tiberiu. (2006). *Medii de calcul în inginerie electrică – MATLAB*. București: Editura Matrix Rom.

[4] Attia, John Okyere. (1999). *Electronics and Circuit Analysis using MATLAB.* Boca Raton: CRC Press LLC: John Okyere Attia Publishing.

[5] Karris, Steven T. (2006). *Introduction to Simulink with Engineering Applications*. USA: Orchad Publications.

[6] Manassah, Jamal T. (2001). *Elementary mathematical and computational tools for electrical and computer engineers using MATLAB*. CRC Press.

[7] Perelmuter, V.M. (2024). *Electrical Drive Simulation with MATLAB/Simulink: Selected Technologies*. Taylor & Francis Group.