

Radu Mircea Morariu-Gligor

**PROGRAMAREA CALCULATOARELOR ÎN
MATLAB / FREEMAT / OCTAVE**



**U.T. PRESS
CLUJ-NAPOCA, 2017
ISBN 978-606-737-249-6**



Editura U.T.PRESS
Str.Observatorului nr. 34
C.P.42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999 / Fax: 0264 - 430.408
e-mail: utpress@biblio.utcluj.ro
<http://biblioteca.utcluj.ro/editura>

Director: Ing. Călin D. Câmpean

Recenzia: Conf.dr.ing. Ovidiu Deteșan
Prof.dr.ing.mat. Nicolae Ursu-Fischer

Copyright © 2017 Editura U.T.PRESS
Reproducerea integrală sau parțială a textului sau ilustrațiilor din
această carte este posibilă numai cu acordul prealabil scris al editurii
U.T.PRESS.

ISBN 978-606-737-249-6

Prefață

Evoluția din ultimii ani a limbajelor și a mediilor de programare a condus la apariția unor medii de programare noi, bazate pe limbaje de programare deja consacrate.

În această lucrare sunt tratate aspecte legate de programarea calculatoarelor în **FreeMat** și **Octave**, care sunt medii de programare compatibile cu **MATLAB** însă sunt versiuni gratuite.

În lucrare sunt prezentate asemănările și diferențele dintre aceste medii de programare, precum și generalități ale acestor medii de programare.

În primul capitol este prezentată modalitatea de utilizare a mediilor de programare ca și calculator. Sunt prezentate tipurile de date, formatul datelor, caractere speciale, noțiunile fiind însoțite de o serie de exemple.

În al doilea capitol, sunt prezentate aspecte legate de prelucrarea șirurilor de date, generarea și manipularea vectorilor și a matricelor, respectiv extragerea elementelor individuale dintr-un vector sau matrice.

În al treilea capitol sunt prezentate modalitățile de rezolvare a ecuațiilor algebrice și transcendente, aspecte privind operații cu polinoame, rezolvarea sistemelor de ecuații liniare și neliniare, interpolarea și aproximarea datelor, derivarea și integrarea numerică, rezolvarea ecuațiilor diferențiale ordinare, noțiunile teoretice fiind completate de exemple.

Capitolul patru al lucrării tratează aspectele legate de grafica bidimensională și tridimensională.

Capitolul cinci al lucrării prezintă o introducere în programarea în FreeMat / Octave / MATLAB, ilustrând modul în care se scriu programele, utilizarea fișierelor script și a celor de tip funcție. În ultimul subcapitol sunt prezentate instrucțiunile de decizie.

Capitolul șase al lucrării tratează instrucțiunea de ciclare for, noțiunile teoretice fiind însoțite de exemple sugestive.

Capitolul al șaptelea al lucrării prezintă instrucțiunile de ciclare while...end, do ... until, precum și instrucțiunile break, continue, error și switch. Exemplele care însoțesc noțiunile teoretice au rolul de a facilita modul de înțelegere a noțiunilor prezentate.

Ultimul capitol prezintă câteva aplicații, al căror nivel de dificultate este ușor-mediu, utile pentru înțelegerea noțiunilor prezentate anterior.

Cuprins:

1. Prezentare generală a mediilor de programare FreeMat / OCTAVE / MATLAB. Expresii. Formatul datelor. Caractere speciale. Vectori și matrice.	9
1.1. Generalități	10
1.1.1. Asemănări și diferențe	10
1.1.2. Prezentare generală a mediului de programare OCTAVE.....	10
1.1.3. Prezentare generală a mediului de programare FreeMat	13
1.1.4. Prezentare generală a mediului de programare MATLAB.....	17
1.2. Utilizarea FreeMat / OCTAVE / MATLAB ca și calculator.....	20
1.2.1. Expresii.....	20
1.2.2. Funcții OCTAVE / MATLAB.....	20
1.3. Tipurile de date din FreeMat / OCTAVE / MATLAB	21
1.4. Formatul datelor	22
1.5. Caractere speciale în FreeMat / Octave / MATLAB	23
1.6. Exemple de utilizare a mediilor de programare FreeMat / OCTAVE / MATLAB ca și calculator	23
2. Prelucrarea șirurilor de date. Generarea și manipularea matricelor. Extragerea elementelor individuale dintr-un vector sau matrice	25
2.1. Prelucrarea șirurilor de date.....	26
2.2. Generarea și manipularea matricelor.....	27
2.3. Extragerea elementelor individuale dintr-un vector sau matrice	32
3. Rezolvarea ecuațiilor algebrice și transcendente. Polinoame. Rezolvarea sistemelor de ecuații liniare și neliniare. Interpolarea și aproximarea datelor. Derivarea numerică. Integrarea numerică. Rezolvarea ecuațiilor diferențiale ordinare	34
3.1. Rezolvarea ecuațiilor algebrice și transcendente.....	35
3.2. Polinoame în FreeMat / OCTAVE / MATLAB	37
3.3. Rezolvarea sistemelor liniare și neliniare	41
3.4. Interpolarea și aproximarea datelor	42
3.5. Derivarea numerică	43
3.6. Integrarea numerică	44
3.7. Rezolvarea ecuațiilor diferențiale ordinare.....	46
4. Grafică 2D și 3D.....	48
4.1. Grafică 2D	49
4.2. Grafică 3D	58
5. Declarații și variabile. Formatul datelor de ieșire. Operatori. Instrucțiuni de introducere și extragere de date. Fișiere script. Fișiere funcție. Instrucțiuni de decizie.	63
5.1. Etapele de rezolvare a unei probleme.....	64
5.2. Declarații și variabile.....	64
5.3. Formatul datelor de ieșire.....	66
5.4. Operatori.....	67

5.5. Instrucțiuni de introducere și extragere de date.....	68
5.6. Fișiere script	72
5.7. Exemple de fișiere script OCTAVE / MATLAB	72
5.8. Fișiere funcție	74
5.9. Exemple de fișiere funcție în OCTAVE / MATLAB.....	75
5.10. Instrucțiuni de decizie.....	76
5.10.1. Instrucțiunea if ... end	76
5.10.2. Instrucțiunea if ... else ... end.....	77
5.10.3. Instrucțiuni if ... else suprapuse	79
5.10.4. Instrucțiunea de decizie multiplă elseif	80
6. Instrucțiunea de ciclare for. Cicluri for suprapuse.....	82
6.1. Instrucțiunea de ciclare for.	83
6.2. Exemple de programe cu ciclul for.....	84
6.3. Cicluri for suprapuse	87
6.4. Exemple de utilizare a ciclurilor for suprapuse	88
7. Instrucțiunea de ciclare while...end. Instrucțiunile break, continue, error, switch.	90
7.1. Instrucțiunea de ciclare while ... end	90
7.2. Instrucțiunea de ciclare do-until	92
7.3. Exemple de programe cu instrucțiuni while ... end și do – until	93
7.4. Instrucțiunea break	101
7.5. Instrucțiunea continue	102
7.6. Instrucțiunea error	102
7.7. Instrucțiunea switch.....	103
8. Aplicații FreeMat / OCTAVE / MATLAB	106
8.1. Analiza mecanismului bielă-manivelă.....	106
8.2. Rezolvarea unei integrale prin metoda cuadratură adaptivă.....	108
8.3. Rezolvarea unei integrale prin metoda Gauss-Legendre	109
8.4. Program pentru verificarea validității Codului Numeric Personal	111
8.5. Program pentru calculul coordonatelor centrului de greutate al unei plăci plane	113
Bibliografie:.....	115

1. Prezentare generală a mediilor de programare FreeMat / OCTAVE / MATLAB. Expresii. Formatul datelor. Caractere speciale. Vectori și matrice.

Introducere:



În cadrul acestui curs se urmărește familiarizarea studenților cu mediile de programare **FreeMat**, **OCTAVE** și **MATLAB**. După parcurgerea acestui curs studenții vor fi capabili să utilizeze mediile de programare pentru efectuarea unor calcule matematice simple și pentru rezolvarea unor probleme de dificultate scăzută.

Cursul este structurat în șase părți, în fiecare dintre acestea fiind prezentate noțiuni teoretice și practice necesare înțelegerii modului de organizare și prezentare a mediilor de programare. În prima parte sunt prezentate noțiuni generale privind cele trei medii de programare, sunt prezentate interfețele celor trei medii, precum și asemănări și diferențe între cele trei medii de programare. A doua parte descrie modalitatea de utilizare a mediilor de programare **FreeMat**, **OCTAVE** și **MATLAB** ca și calculator, fiind prezentate câteva funcții uzuale, utile în scrierea și interpretarea unor expresii matematice. Următoarele două părți prezintă într-o formă succintă formatul și tipurile de date din **FreeMat** / **MATLAB**. În următoarea parte sunt prezentate câteva din caracterele speciale și modul de utilizare a acestora. Partea a șasea conține o serie de exemple de utilizare a mediilor de programare **FreeMat** / **MATLAB** ca și calculator.

Obiective:



Prin parcurgerea acestui material, studenții: vor ști să utilizeze mediile de programare **FreeMat** / **OCTAVE** / **MATLAB** ca și calculator; vor învăța să utilizeze câteva din funcțiile oferite de aceste medii de programare; se vor familiariza cu tipurile de date și cu formatul datelor;

1.1. Generalități

1.1.1. Asemănări și diferențe

Aceste limbaje de programare sunt limbaje de nivel înalt, interpretate și structurate, bazate pe calculul matriceal și sunt utilizate în special pentru calcule științifice și ingineresti. Programele în **FreeMat** / **OCTAVE** / **MATLAB** constau dintr-o listă de funcții sau un script. Comenzile se introduc prin tastare la prompter sau pot fi citite dintr-un script.

Caracterul punct-virgulă (;) separă comenzile în cadrul unei linii, rezultatul unei comenzi terminată cu punct-virgulă nu este afișat pe ecran. Caracterul virgulă (,) se utilizează pentru separarea comenzilor în cadrul unei linii, cu afișarea rezultatelor comenzilor pe ecran.

Pentru continuarea unei comenzi pe linia următoare se utilizează

Limbajele **FreeMat**, **OCTAVE** și **MATLAB** sunt *case-sensitive* adică fac diferența între literele mari și cele mici.

Limbajele **Freemat** și **OCTAVE** sunt clone cu distribuție gratuită a limbajului **MATLAB**, astfel că sintaxa celor trei limbaje este foarte asemănătoare.

Limbajele de programare **FreeMat** / **OCTAVE** / **MATLAB** nu permit transmiterea argumentelor prin referință și suportă diverse structuri de date.

Limbajele **MATLAB** și **OCTAVE** / **FreeMat** sunt asemănătoare în mare parte, existând însă și câteva deosebiri în ceea ce privește mediul de programare **OCTAVE**, cum ar fi: instrucțiunile **if**, **for**, **while** se pot termina cu **end** dar și cu **endif**, **endfor**, **endwhile**; funcțiile pot fi scrise direct în linia de comandă înainte de utilizare și se termină cu **endfunction**; începutul unei linii de program de comentariu poate fi marcată cu simbolul %, dar și cu simbolul # sau ##.

1.1.2. Prezentare generală a mediului de programare **OCTAVE**

GNU **OCTAVE** este un mediu de programare orientat în special pentru rezolvarea problemelor de matematică, de statistică, de prelucrare a datelor în diverse moduri (inclusiv grafic), de optimizare. Este un limbaj de programare interpretat, poate integra programe scrise în diverse alte limbaje de programare precum C++ sau Fortran dacă sunt compilate.

Programele **OCTAVE** constau dintr-o listă de funcții sau un script. Sintaxa este bazată pe matrice și oferă diverse funcții pentru operațiile cu matrice. **OCTAVE** nu este orientat - obiect, dar suportă diverse structuri de date.

Este compatibil cu **MATLAB**, sintaxa sa fiind foarte apropiată de a acestuia. Poate importa / exporta programe scrise în / pentru **MATLAB**. Există însă anumite diferențe între cele două medii de programare, însă cu o programare atentă a scriptului se poate obține rularea corectă atât în **OCTAVE** cât și în **MATLAB**.

QtOCTAVE este un front-end pentru **OCTAVE** și este un program interactiv pentru calcule numerice de înaltă performanță și vizualizări. **QtOCTAVE** integrează toate acestea într-un mediu ușor de învățat și folosit, în care enunțurile problemelor și rezolvările acestora sunt exprimate în modul cel mai natural posibil, așa cum sunt scrise matematic, fără a fi necesară programarea tradițională.

Pentru că **OCTAVE** este disponibil sub Licența Publică Generală GNU, poate fi copiat de pe internet de la adresa <http://www.octave.org> și utilizat în mod gratuit. Programul rulează pe mai multe sisteme de operare ca: Unix, Unix-like, precum și pe Microsoft Windows.

Programul **QtOCTAVE** lucrează cu cinci tipuri de ferestre (figura 1.1): o fereastră pentru afișarea variabilelor locale și a funcțiilor disponibile (2), o fereastră tip navigator (1), o fereastră tip editor (3), o fereastră de comenzi (terminal) (4) și o fereastră pentru afișarea comenzilor lansate în terminal, și a rezultatelor acestora (5).

Fiecare comandă din meniul principal furnizează un meniu specific, selecția comenzii dorite se realizează prin deplasarea zonei active cu ajutorul săgeților sau prin selecția directă cu ajutorul mouse-ului.

Meniurile aplicației **QtOctave** sunt prezentate succint în continuare:

Meniul File cuprinde opțiunile: **Projects** – permite gestionarea proiectelor; **Run an Octave Script** – permite rularea unui script (program); **Change Directory** – permite schimbarea directorului curent; **Quit** – determină părăsirea aplicației;

Meniul View cuprinde următoarele opțiuni: **Clear terminal** – se utilizează pentru curățarea ferestrei de comenzi (terminal); **Dock tools** – conține comenzi pentru activarea ferestrelor aplicației; **Windows Layout** – permite configurarea interfeței aplicației; **Show/Hide Objects** – permite afișarea / ascunderea ferestrelor sau a bărilor cu butoane.

Meniul Analysis cuprinde opțiunile: **Integrate function** – pentru integrarea funcțiilor; **Ordinary Differential Equations** – pentru rezolvarea ecuațiilor diferențiale.

Meniul Data conține următoarele opțiuni: **Table** – se utilizează pentru crearea tabelor de date (vectori sau matrice);

Format of the displayed output – se utilizează pentru stabilirea formatului de afișare; **Load matrix from file** – pentru încărcarea unei matrice dintr-un fișier; **Save matrix to file** – pentru salvarea unei matrice într-un fișier.

Meniul Equations conține comenzi pentru rezolvarea ecuațiilor: **Solve equation by bisection method** – rezolvarea ecuațiilor prin metoda bisecției; **Linear equation** – rezolvarea ecuațiilor liniare; **Nonlinear equation** – rezolvarea ecuațiilor neliniare.

Meniul Matrix cuprinde comenzi pentru efectuarea operațiilor cu matrice, cum ar fi: **A+B** - Adunarea; **A*B** - Înmulțirea; **A**n** - Ridicarea la putere; **Determinant** - Calculul determinantului; **Eigenvalues and eigenvectors** - calculul vectorilor și a valorilor proprii; **Inverse** - calculul inversei; **Transpose** - calculul transpusei; **Submatrix** - formarea unor submatrice.

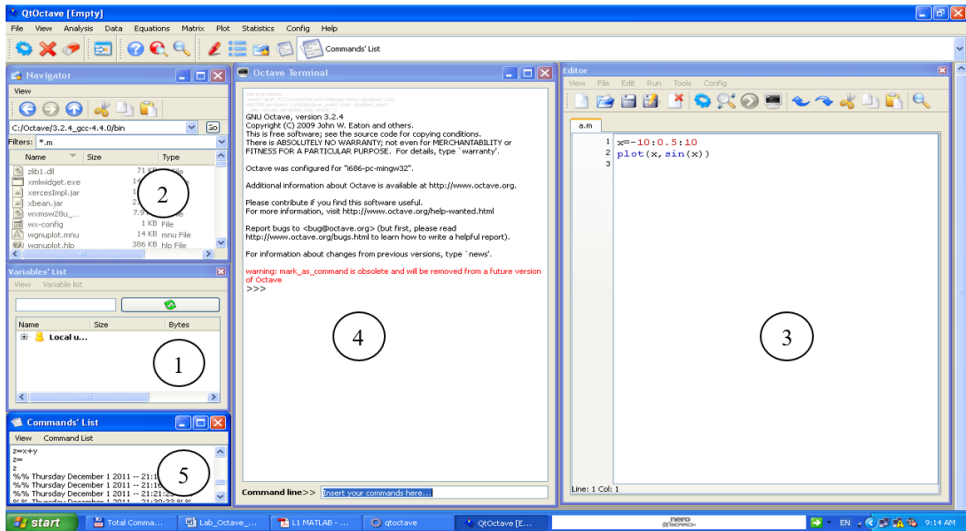


Figura 1.1. Interfața aplicației OCTAVE

Meniul Plot conține comenzi pentru generarea graficelor 2D (**2D**), respectiv 3D (**3D**), precum și: **Axis scale** - stabilirea modului de reprezentare a valorilor pe axe; **Title and labels** - stabilirea textului corespunzător titlului și etichetelor; **Export** - exportul graficelor;

Meniul Statistics conține comenzi pentru prelucrări statistice.

Meniul Config permite configurarea aplicației.

Meniul Help permite utilizarea sistemului de informații ajutătoare.

1.1.3. Prezentare generală a mediului de programare FreeMat

FreeMat este un program gratuit de matematică, asemănător cu **MATLAB**.

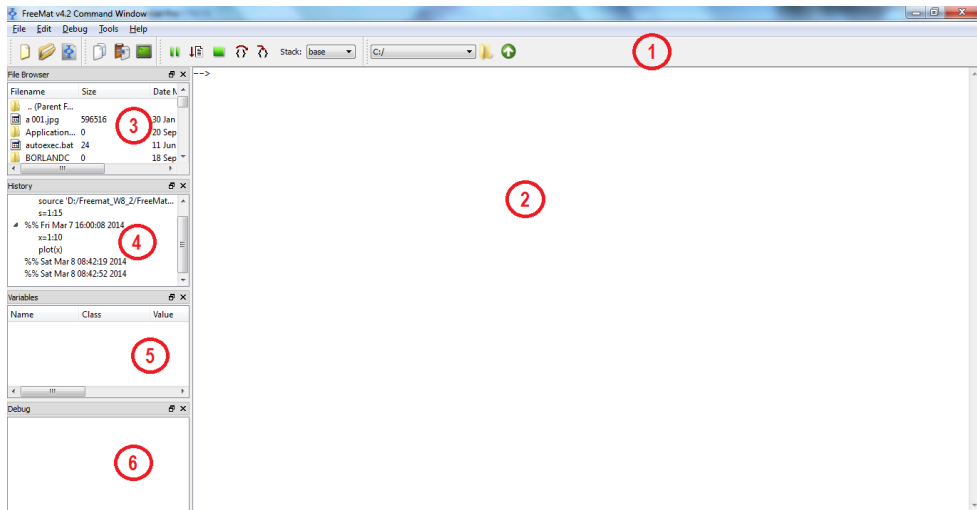


Figura 1.2. Interfața aplicației **FreeMat**

Interfața aplicației **FreeMat** conține următoarele elemente (figura 1.2): 1 – meniul principal și bara cu unelte; 2 – fereastra de comenzi: permite introducerea comenzilor, definirea variabilelor sau a constantelor, precum și definirea funcțiilor; 3 – browser-ul de fișiere: permite vizualizarea directoarelor și a fișierelor curente; 4 – fereastra de istoric: ilustrează în ordine cronologică toate comenzile introduse precum și rezultatele executării acestora. Cele mai noi comenzi sunt cele din partea de sus, iar cele mai vechi sunt cele din partea inferioară; 5 – fereastra “Variables”: în această fereastră apar toate variabilele utilizate; 6 – fereastra “Debug”: în această fereastră sunt prezentate acțiunile rezultate în urma procesului de depanare;

Meniul principal conține următoarele meniuri:

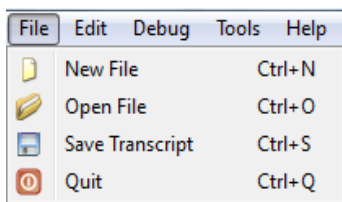


Fig. 1.3. Meniul **File**

Meniul **File** conține comenzi pentru operarea cu fișiere (figura 1.3):

New File (Ctrl + N): permite crearea unui fișier nou;

Open File (Ctrl + O): se utilizează pentru deschiderea unui fișier existent;

Save Transcript (Ctrl + S): Salvează conținutul ferestrei de comenzi într-un fișier text.

Quit (Ctrl + Q): se utilizează pentru închiderea aplicației **FreeMat**.

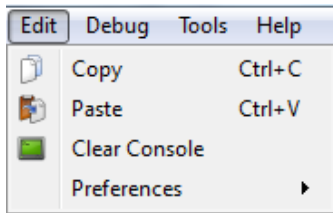


Fig. 1.4. Meniul **Edit**

Meniul **Edit** conține comenzi de editare (figura 1.4):

Copy (Ctrl + C): copiază în memoria tampon textul selectat;

Paste (Ctrl + V): inserează în poziția curentă a cursorului, conținutul memoriei tampon;

Clear console: curăță conținutul ferestrei de comenzi;

Preferences: permite stabilirea dimensiunilor fonturilor și a numărului de linii scrise anterior care pot fi apelate;

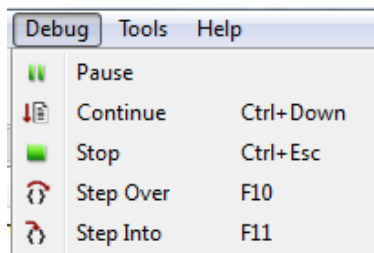


Fig. 1.5. Meniul **Debug**

Meniul **Debug** conține comenzi pentru depanarea programelor (figura 1.5):

Pause: se utilizează pentru a întrerupe temporar procesul de depanare;

Continue (Ctrl + Down): permite continuarea procesului de rulare;

Stop (Ctrl + Esc): determină oprirea procesului de depanare;

Step Over (F10): determină saltul peste o

linie în procesul de depanare;

Step Into (F11): se utilizează pentru saltul în interiorul apelului unei funcții.

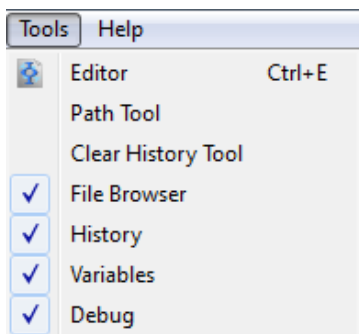


Fig. 1.6. Meniul **Tools**

Meniul **Tools** conține comenzi pentru gestionarea instrumentelor (figura 1.6), astfel:

Editor: deschide editorul de texte;

Path Tools: permite stabilirea directorului curent;

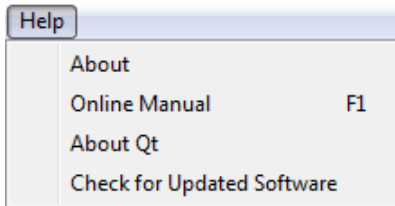
Clear History Tool: șterge conținutul istoricului;

File Browser: activează / dezactivează browser-ul de fișiere;

History: activează / dezactivează fereastra de istoric;

Variables: activează / dezactivează fereastra “Variables”;

Debug: activează / dezactivează fereastra “Debug”;



Meniul **Help**: permite utilizarea sistemului de informații ajutătoare (figura 1.7).

Fig. 1.7. Meniul Help

Prezentarea editorului de texte al mediului de programare **FreeMat**.

Mediul de programare **FreeMat** conține un editor de texte cu ajutorul căruia se pot scrie programe.

Interfața acestuia (figura 1.8) conține: un meniu principal (1), bări cu unelte (2), zonă de editare (3).

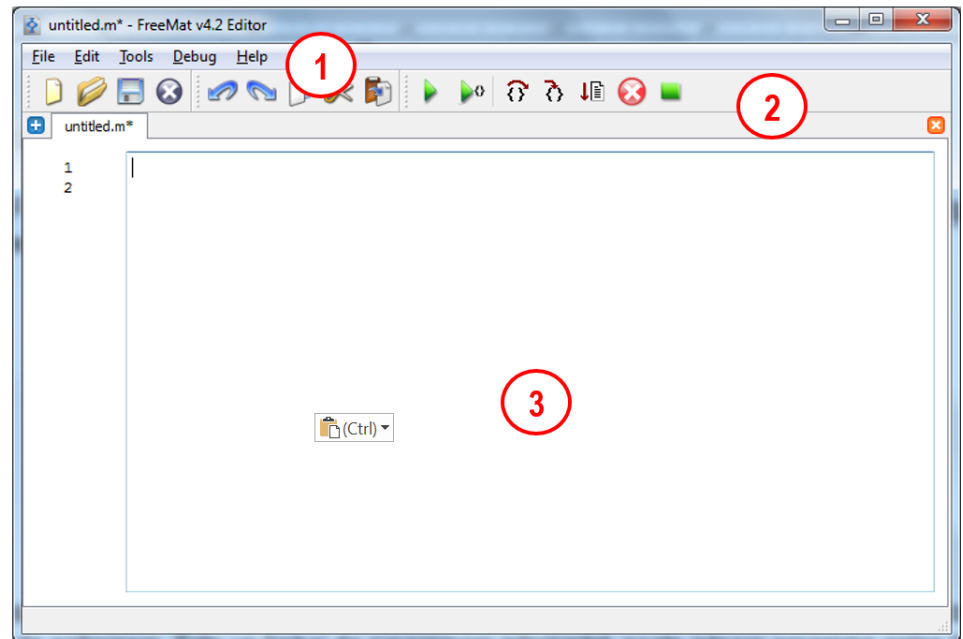


Figura 1.8. Interfața editorului de texte al mediului de programare **FreeMat**

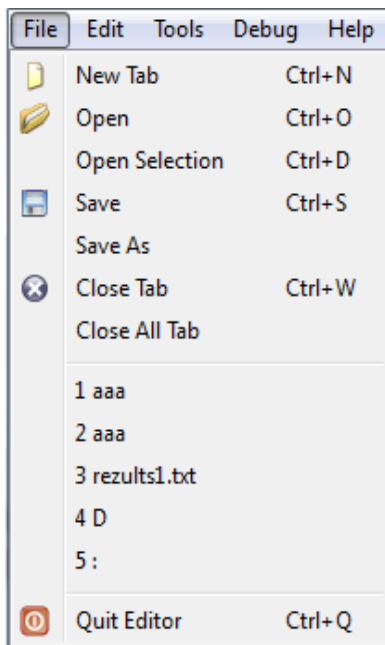


Figura 1.9. Meniul **File**

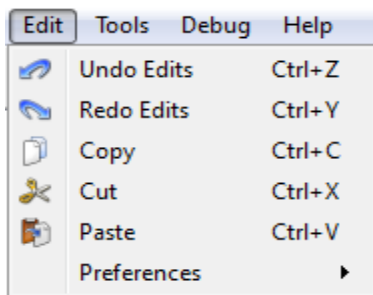


Figura 1.10. Meniul **Edit**

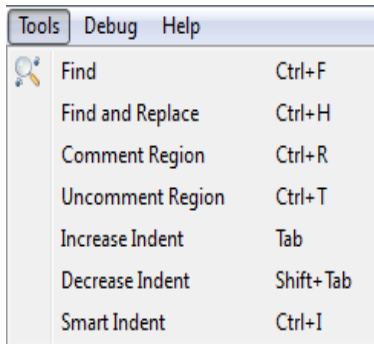


Figura 1.11. Meniul **Tools**

Meniul **File** conține comenzi pentru gestionarea fișierelor (figura 1.9):

New Tab: crează un fișier nou într-o pagină nouă;

Open: deschide un fișier existent;

Open Selection: deschide fișierul al cărui nume se află precizat într-un câmp selectat;

Save: salvează fișierul curent;

Save As: salvează fișierul curent sub un alt nume sau un alt format;

Close Tab: închide pagina curentă;

Close All Tab: închide toate paginile deschise;

Quit Editor: închide editorul de texte;

Meniul **Edit** conține comenzi pentru editorul de texte (figura 1.10):

Undo Edits: anulează ultimele comenzi de editare;

Redo Edits: anulează ultimul Undo Edits;

Copy: copiază textul selectat în memoria tampon;

Cut: mută textul selectat în memoria tampon;

Paste: copiază conținutul memoriei tampon în poziția curentă a cursorului;

Preferences: permite setarea fonturilor, a dimensiunii spațiului de aliniere la stânga a textului, precum și alte setări;

Meniul **Tools** conține următoarele comenzi (figura 1.11):

Find: se utilizează pentru căutare în text;

Find and Replace: se utilizează pentru căutare și înlocuire în text;

Comment Region: se utilizează pentru definirea unei regiuni de comentariu

Uncomment Region: se utilizează pentru transformarea unei regiuni de comentariu într-o zonă de program;

Increase Indent: se utilizează pentru deplasarea cursorului la dreapta cu un număr de spații (Tab orizontal);

Decrease Indent: se utilizează pentru deplasarea la stânga cu un număr de spații;

Smart Indent: se utilizează pentru alinierea față de marginea din stânga a mai multor linii de text;

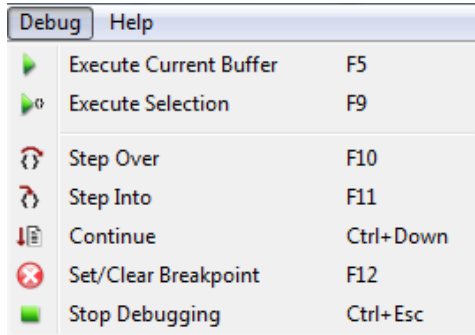


Figura 1.12. Meniul **Debug**

Meniul **Debug** conține comenzi pentru depanarea programelor (figura 1.12):

Execut Current Buffer: execută sursa aflată în memoria tampon;

Execute Selection: execută textul selectat;

Step Over: determină saltul peste o linie în procesul de depanare;

Step Into: se utilizează pentru saltul în interiorul apelului unei funcții;

Continue: permite continuarea procesului de rulare;

Set / Clear Breakpoint: selectează / deselectează un punct de întrerupere;

Stop Debugging: oprește procesul de depanare.

Meniul **Help** permite accesarea sistemului de informații ajutătoare.

1.1.4. Prezentare generală a mediului de programare MATLAB

MATLAB (**MAT**rix **LAB**oratory) este un software matematic, produs de firma The MathWorks Inc. Cu ajutorul **MATLAB**-ului pot fi realizate calcule numerice, programe, modelări și simulări numerice, prelucrări de date și reprezentări grafice cu precădere în știință și inginerie. **MATLAB** este alcătuit dintr-un **nucleu** și o serie de pachete de programe, intitulate **toolbox**-uri.

Nucleul are un interpretor propriu și conține comenzi de uz general, funcții matematice de bază, funcții de prelucrare a șirurilor de caractere, instrucțiuni de control, funcții pentru reprezentări grafice, etc.

Toolbox-urile reprezintă colecții de funcții **MATLAB** organizate în scopul rezolvării problemelor din diverse domenii, cum ar fi: calcul simbolic, optimizare, programare, grafică științifică și inginerescă, modelarea și simularea sistemelor dinamice, etc.

Sistemul **MATLAB** se compune din cinci părți principale:

Mediul de dezvoltare **MATLAB** se compune dintr-o colecție de instrumente și facilități ce permit utilizarea fișierelor și funcțiilor **MATLAB**.

Biblioteca de funcții matematice MATLAB este o colecție foarte bogată de funcții de la cele elementare (cum ar fi cele trigonometrice) până la funcții complexe (cum ar fi cele de generare și prelucrare a matricelor, transformata Fourier, etc).

Limbajul MATLAB este un limbaj de nivel înalt, cu instrucțiuni de control, funcții, structuri de date și elemente de programare orientată pe obiecte.

Grafica MATLAB (Handle Graphics) conține funcții pentru realizarea graficelor bi- și tridimensionale, pentru procesarea imaginilor, animație și prezentarea graficelor, personalizarea aspectului graficelor, construirea interfețelor grafice cu utilizatorul pentru aplicațiile **MATLAB**.

Interfața externă MATLAB (API – Application Programming Interface) reprezintă o bibliotecă care permite scrierea programelor în C sau Fortran care interacționează cu **MATLAB**.

Interfața mediului de dezvoltare **MATLAB** este alcătuită din mai multe ferestre (figura 1.13) și conține elemente comune tuturor aplicațiilor Windows, precum și o serie de elemente specifice.

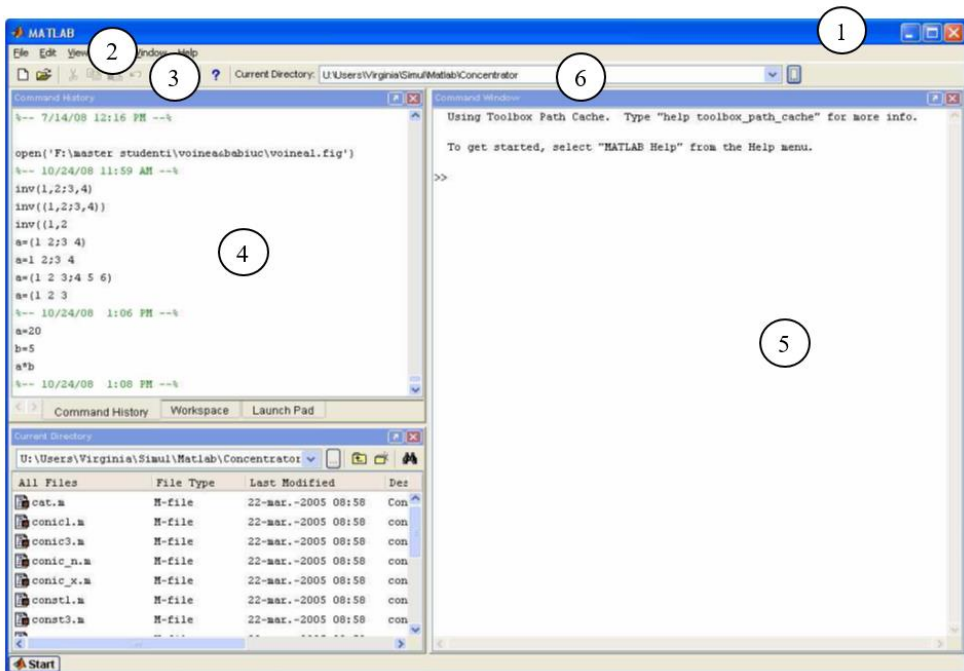


Figura 1.13. Interfața aplicației **MATLAB**

Elementele ferestrei de lucru a aplicației **MATLAB**: bara de titlu (1), meniul principal (2), bara cu instrumente (3), fereastra cu istoricul comenzilor (4), fereastra spațiului de lucru (5), afișarea directorului curent (6).

Cele mai utilizate ferestre sunt:

- *spațiul de lucru* (**Workspace**) conține setul de variabile (caracterizate prin numele și dimensiunea acestora) utilizate în sesiunea curentă de lucru și stocate în memorie.
- *fereastra de comenzi* (**Command Window**) se utilizează pentru introducerea în linia de comandă a instrucțiunilor, după prompter (reprezentat prin simbolul >>);
- *fereastra de editare a unui fișier* (**Editor**);
- *fereastra pentru reprezentări grafice* (**Figure**);
- *fereastra pentru accesarea informațiilor ajutătoare* (**Help**);
- *fereastra **Current Directory*** afișează fișierele directorului curent. Permite afișarea și schimbarea directorului curent.
- *fereastra **Command History*** conține o listă a ultimelor comenzi executate. Comenzile introduse anterior pot fi vizualizate, copiate sau executate;
- *fereastra **Launch Pad*** se utilizează pentru accesarea mijloacelor de lucru și lansarea pachetelor de programe.

Cu ajutorul tastelor cu săgeți pot fi realizate o serie de acțiuni de editare, astfel: **Săgeată sus** : recheamă comanda dată anterior; **Săgeată jos** : recheamă linia de comandă următoare; **Săgeată stânga** : deplasare la stânga cu un caracter; **Săgeată dreapta** : deplasare la dreapta cu un caracter; **Ctrl + săgeată stânga** : deplasare la stânga cu un cuvânt; **Ctrl + săgeată dreapta** : deplasare la dreapta cu un cuvânt; **Home** : deplasare la începutul liniei de comandă; **End** : deplasare la sfârșitul liniei de comandă; **Esc** : abandonează linia de comandă curentă; **Ins** : trece între modurile Insert și Overtyp; **Backspace** : șterge un caracter la stânga cursorului

Dacă se dorește introducerea unei comenzi (sau un text) a cărei lungime depășește lungimea liniei curente, inserați trei puncte (...) pe linia curentă și continuați editarea comenzii (sau a textului) pe linia următoare. Mediul de programare MATLAB este implicit sensibil la tipul de litere (mari sau mici), dar se poate modifica această setare. În mod obligatoriu, numele de funcții se scriu cu litere mici. Liniile de comentariu sunt precedate de caracterul %. Pentru obținerea informațiilor ajutătoare se tastează **help** pentru meniul întreg sau **help** urmat de denumirea funcției sau a fișierului **.m**.

1.2. Utilizarea **FreeMat** / **OCTAVE** / **MATLAB** ca și calculator

Mediul de dezvoltare **FreeMat** permite efectuarea de calcule matematice, fără a necesita scrierea unui program.

1.2.1. Expresii

La fel ca și celelalte limbaje de programare, **FreeMat** / **Octave** / **MATLAB** lucrează cu expresii matematice, dar spre deosebire de majoritatea limbajelor, aceste expresii implică la scară largă lucrul cu matrici. În cadrul unei expresii putem întâlni: *variabile*, *numere (constante)*, *operatori* și *funcții*.

Variabile: În toate limbajele de programare variabilele sunt caracterizate de trei elemente: nume (identificator), tip și valoare. Limbajele **FreeMat** / **Octave** / **MATLAB** nu necesită declararea dimensiunii variabilelor, deoarece la întâlnirea unui nou nume de variabilă se generează automat variabila respectivă și se alocă spațiul necesar de memorie. Numele unei variabile începe cu o literă și poate continua cu litere, cifre sau simboluri. Numărul maxim de caractere care pot fi utilizate în denumirea variabilelor este de 31. Limbajele **FreeMat** / **Octave** / **MATLAB** fac distincție între literele mici și cele mari, adică este *case sensitive*.

Numere: **FreeMat** / **Octave** / **MATLAB** utilizează notația zecimală, cu punct zecimal opțional și cu semn + sau -. Se poate utiliza și notația științifică cu litera *e* pentru a specifica o putere a lui 10. Reprezentarea numerelor imaginare este realizată cu litera *i* sau *j* ca sufix.

Operatori: Expresiile utilizează operatori aritmetici uzuali, cum ar fi: + (adunare), - (scădere), * (înmulțire), / (împărțire), \ (împărțire la stânga), ^ (ridicare la o putere), () (operatorul de specificare a ordinii de evaluare); operatori relationali : = = (egal); ~= (diferit) ; < (mai mic); > (mai mare) ; <= (mai mic sau egal) ; >= (mai mare sau egal); operatori logici : & (și); | (sau); ~ (not); xor (or)

1.2.2. Funcții **OCTAVE** / **MATLAB**

Funcții: **FreeMat** / **OCTAVE** / **MATLAB** pun la dispoziția utilizatorilor și a programatorilor un mare număr de funcții matematice elementare standard, astfel:

a) funcții trigonometrice: **sin** – sinus, **asin** – arcsinus, **cos** – cosinus, **acos** – arccosinus, **tan** – tangentă, **atan** – arctangentă, **cot** – cotangentă, **acot** – arccotangentă, **sec** – secantă, **asec** – arcsecantă, **csc** – cosecantă, **acsc** – arccosecantă;

b) funcții putere: **exp** – funcția exponențială, **log** – logaritm natural, **log2** – logaritm în baza 2, **log10** – logaritm în baza 10, **sqrt** – funcția radical;
 c) alte funcții: **abs** – valoarea absolută (modul), **min** – minimum, **max** – maximum, **pi** (3.14159265), **i** și **j** unitate imaginară, **eps** - precizia relativă în virgulă mobilă (2^{-52}), **realmin** - cel mai mic număr în virgulă mobilă (2^{-1022}), **realmax** - cel mai mare număr în virgulă mobilă, (2^{1023}), **Inf** – infinit, **NaN** (*Not-a-Number*) - nu este număr.

1.3. Tipurile de date din FreeMat / OCTAVE / MATLAB:

Tipul fundamental de dată în **FreeMat / OCTAVE / MATLAB** este **matricea**, elementele acesteia fiind de același tip. În figura 1.14. și tabelul 1.1 sunt ilustrate toate tipurile de date din **FreeMat / OCTAVE / MATLAB**.

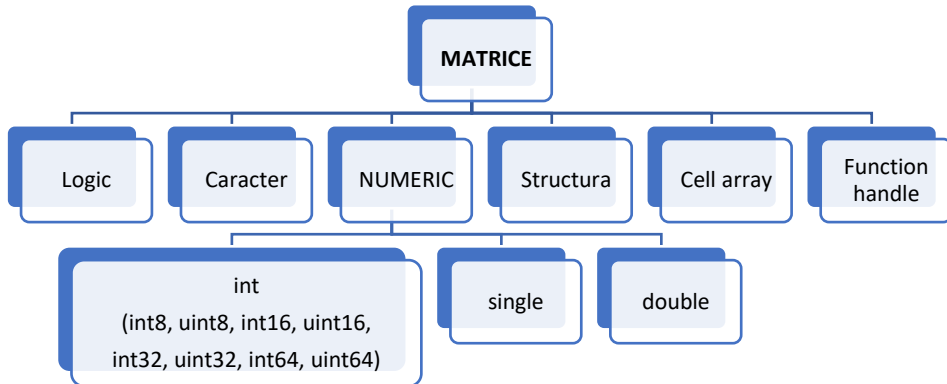


Figura 1.14. Tipurile de date din **FreeMat / OCTAVE / MATLAB**

Tipurile **structură** sau **celule de tablouri** pot să conțină date de tipuri diferite. Cu ajutorul claselor (uses classes și Java classes) pot fi create tipuri noi de date (date utilizator).

Tabelul 1.1. Tipurile de date din FreeMat / OCTAVE / MATLAB

Tip de dată:	Descriere:	Biți	Domeniu de valori:
int8	Întreg cu semn pe 8 biți	8	[-128, 127]
uint8	Întreg fără semn pe 8 biți	8	[0, 255]
int16	Întreg cu semn pe 16 biți	16	[-32,768, 32,767]
uint16	Întreg fără semn p 16 biți	16	[0, 65,535]
int32	Întreg cu semn pe 32 biți	32	[-2,147,483,648,2,147,483,647]
uint32	Întreg fără semn p 32 biți	32	[0, 4,294,967,295]
int64	Întreg cu semn pe 64 biți	64	[-9,223,372,036,854,775,808, 9,223,372,036,854,775,807]
uint64	Întreg fără semn p 64 biți	64	[0, 18,446,744,073,709,551,615]
Single	Real în simplă precizie.	32	[-3.40282e+038, 3.40282e+038]

Double	Real în dublă precizie	64	[-1.79769e+308, 1.79769e+308]
Logical	Tablou de valori logice	1	1 (adevărat) sau 0 (fals)
Char	Tablou de tip caracter sau șir de caractere	16	
function handle	Pointer la o funcție.		
Structure	Structura este o colecție de câmpuri, fiecare având un tip de dată.		
cell array	Tablou de celule indexate, fiecare celulă putând memora un tablou de orice tip sau mărime.		

1.4. Formatul datelor

În **FreeMat / OCTAVE / MATLAB** afișarea valorilor se realizează implicit cu 3 zecimale, însă această setare se poate modifica cu ajutorul comenzii *format*. Toate calculele sunt efectuate în dublă precizie.

Comanda **format** se poate utiliza pentru a comuta între diferite formate de afișare a datelor de ieșire :

- > format - formatul implicit (SHORT)
- > format short - reprezentare în virgulă fixă cu 5 cifre
- > format long - reprezentare în virgulă fixă cu 15 cifre
- > format short e – reprezentare în virgulă mobilă cu 5 cifre
- > format long e – reprezentare în virgulă mobilă cu 15 cifre
- > format short g - cea mai bună dintre reprezentările în virgulă fixă și mobilă cu 5 cifre
- > format long g - cea mai bună dintre reprezentările în virgulă fixă și mobilă cu 15 cifre
- > format hex - reprezentare în hexazecimal
- > format + - Simbolurile +, - și blank sunt utilizate pentru elemente pozitive, negative și nule; partea imaginară este ignorată
- > format bank - reprezentare pentru simboluri valutare (dolari și cenți)
- > format rat - aproximare printr-o fracție a unui întreg

Spațierea:

- > format compact – suprimă trecerea la linie nouă suplimentară
- > format loose - revine la introducerea unei linii suplimentare

Exemplu:	<pre>--> c = 0.123456789012 --> c c = 0.12346</pre>	<pre>--> format short e, c c = 1.2346e-001 --> format long e, c c = 1.2345678901e-001</pre>
----------	-----------------------------------------------------------	---------------------------------------------------------------------------------------------------



```
--> format long, c      --> format, c
c = 0.123456789012      c = 0.12346
```

Sistemul de informații ajutătoare: Pentru apelarea sistemului de informații ajutătoare se utilizează comanda **help** urmată de numele funcției pentru care se dorește obținerea informațiilor ajutătoare.

Exemple: --> help cos sau --> help plot

1.5. Caractere speciale în FreeMat / Octave / MATLAB

Tabelul 1.2. Caractere speciale în FreeMat / OCTAVE / MATLAB

Car.	Semnificație:	Car.	Semnificație
:	Se folosește la generarea diviziunilor	;	Separator între instrucțiuni pe aceeași linie de comandă (fără ecou pe ecran)
()	Folosite pentru “corpul” unei funcții	%	Se folosește pentru a anunța comentarii în program
[]	Pentru desemnarea unei matrice / vector	=	Asignare
.	Indicator de operație aritmetică “element cu element”	‘	Transpusa unei matrice
...	Continuarea unei comenzi pe linia de comandă următoare	[,]	Separator între elementele aceleiași linii într-o matrice / vector
,	Separator între instrucțiuni pe aceeași linie de comandă (cu ecou pe ecran)	[:]	Separator între liniile unei matrice

1.6. Exemple de utilizare a mediilor de programare FreeMat / OCTAVE / MATLAB ca și calculator

Operații aritmetice simple:

Exemplu:



Adunare	Scădere	Înmulțire	Împărțire	Împărțire
--> 5 + 3 ans = 8	--> 5 - 3 ans = 2	--> 5 * 3 ans = 15	--> 5 / 3 ans=1.6667	--> 5 \ 3 ans=0.60000

Observație: La scrierea unei expresii aritmetice care nu a fost atribuită unei variabile apare o variabilă ans în care programul introduce automat rezultatul.

Calculul unor expresii:

Exemple:



Calculul expresiei:

$$y = 6 \cdot x^3 - 5 \cdot x^2 + 3 \cdot x - 1$$

pentru valoare lui x : 12.3

$$\begin{aligned} \text{--> } x=12.3; y &= 6 \cdot x^3 - 5 \cdot x^2 + 3 \cdot x - 1 \\ y &= 1.0445e + 004 \end{aligned}$$

Inversarea valorilor a două variabile (varianta 1):

$$a = 3, b = 5.$$

Se utilizează o variabilă ajutătoare c

$$c = a, a = b, b = c.$$

În final $a=5, b=3$

$$\begin{aligned} \text{--> } a &= 3; b = 5 \\ \text{--> } c &= a, a = b, b = c \\ c &= 3, a = 5, b = 3 \end{aligned}$$

Inversarea valorilor a două variabile (varianta 2):

$$a = 3, b = 5.$$

Se utilizează formulele: $a=a+b, b=a-b, a=a-b$.

În final $a=5, b=3$

$$\begin{aligned} \text{--> } a &= 3; b = 5 \\ \text{--> } a &= a+b, b = a-b, a = a-b \\ a &= 8, b = 3, a = 5 \end{aligned}$$

Calculul diametrului, ariei și volumului unei sfere de raza R :

Se cunoaște raza sferei (R):	$\text{--> } R = 10; D = 2 * R$
$D = 2 * R$	$D = 20$
$A = 4 * \pi * R^2$	$\text{--> } A = 4 * \pi * R^2$
$V = 4 * \pi * R^3 / 3$	$A = 1256.6$
	$\text{--> } V = 4 * \pi * R^3 / 3$
	$V = 4188.8$

Calculul expresiei:
$$E = \frac{\sin \frac{\pi}{6} + \cos \frac{\pi}{4}}{\sqrt{5 + \ln(3) - 1,25^{0,25}}}$$

$$\begin{aligned} \text{--> } E &= (\sin(\pi/6) + \cos(\pi/4)) / (\text{sqrt}(5 + \log(3)) - 1.25^{0.25}) \\ E &= 0.85479 \end{aligned}$$

2. Prelucrarea șirurilor de date. Generarea și manipularea matricelor. Extragerea elementelor individuale dintr-un vector sau matrice

Introducere:



Matricea reprezintă tipul fundamental de dată în **FreeMat** / **OCTAVE** / **MATLAB**, elementele acesteia având același tip. La sfârșitul cursului anterior s-au prezentat câteva noțiuni introductive privind matricele. Vectorii reprezintă forme particulare ale matricelor, astfel: vectorul linie reprezintă o matrice cu o singură linie, respectiv vectorul coloană reprezintă o matrice cu o singură coloană.

Obiectivul acestui curs constă în prezentarea prin noțiuni teoretice și exemple a funcțiilor pentru: prelucrarea șirurilor; generarea și manipularea matricelor; respectiv extragerea unor elemente din șiruri și matrice.

Pentru atingerea obiectivului propus, cursul este structurat pe trei părți, în fiecare dintre acestea fiind prezentate noțiuni teoretice (explicațiile funcțiilor) și practice (exemple) necesare înțelegerii modului de lucru a funcțiilor prezentate.

Cele mai uzuale operații care se efectuează cu șirurile de date sunt: determinarea elementului minim / maxim, determinarea mediei aritmetice, suma / produsul unor elemente care îndeplinesc anumite condiții, ordonarea crescătoare / descrescătoare a elementelor șirului.

În prima parte sunt prezentate funcțiile **FreeMat** / **OCTAVE** / **MATLAB** pentru prelucrarea șirurilor.

În partea a doua sunt prezentate funcțiile pentru generarea și manipularea matricelor, fiind prezentate câteva funcții cu ajutorul cărora pot fi generate matrice particulare.

Ultima parte tratează funcțiile pentru extragerea unor elemente din vectori sau matrice.

Obiective:



După parcurgerea acestui material, studenții:

- vor ști să prelucreze și opereze cu șiruri de numere;
- vor ști să genereze și să manipuleze matrice;
- vor învăța să extragă elemente individuale dintr-un vector sau o matrice;

2.1. Prelucrarea șirurilor de date

Cele mai uzuale funcții utilizate în prelucrarea șirurilor sunt: **max** - determină cea mai mare componentă a unui vector sau matrice; **min** - determină cea mai mică componentă a unui vector sau matrice; **mean** sau **median** - calculează valoarea medie a componentelor unui vector; **sort** - sortează elementele în ordine crescătoare; **sum** - calculează suma componentelor unui vector sau matrice; **prod** - calculează produsul componentelor unui vector; **find** - caută elementele sau indicii elementelor care îndeplinesc o anumită condiție.



Exemplu:

Se consideră șirul format din primele 10 numere naturale. Se cere să se determine elementul maxim, minim, suma și produsul elementelor șirului, precum și media aritmetică a elementelor.

```
--> n = 1:10, max, min(n), prod(n), sum(n)
--> ma = sum(n)/max
n =
1 2 3 4 5 6 7 8 9 10
max = 10
ans = 1
ans = 3628800
ans = 55
ma = 5.5000
--> -sort(-n)
ans =
10 9 8 7 6 5 4 3 2 1
--> find(n >3)
ans =
4 5 6 7 8 9 10
```



Exemplu:

Se consideră șirul de numere reale: $a_n = \left(1 + \frac{1}{n}\right)^n$.

Se știe că $\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n = e$.

Se cere să se calculeze suma primilor 100, 200 și 300 de termeni ai șirului și să se compare cu valoarea lui e .

```
--> n=1:100; an=(1 + 1./n).^n ; lim=an(max(n))
lim = 2.7048
--> n=1:200; an=(1 + 1./n).^n ; lim=an(max(n))
lim = 2.7115
```



```
--> n=1:300; an=(1 + 1./n).^n ; lim=an(max(n))
lim = 2.7138
--> e
ans = 2.7183
```

Observație: Se observă că pe măsură ce crește numărul de termeni luați în considerare, limita se apropie de valoarea lui e.

Exemplu:



Se consideră dezvoltarea în serie de puteri : $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$

Se cere să se verifice egalitatea de mai sus, pt. $x = 0,5$ și 100 de termeni.

```
--> x=0.5; n=1:100;
--> ex=(x.^n) ./factorial(n) ; rez=1+sum(ex)
rez = 1.6487
--> e^x
ans = 1.6487
```

2.2. Generarea și manipularea matricelor

O matrice dreptunghiulară cu m linii și n coloane se reprezintă astfel:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n-1} & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n-1} & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n-1} & a_{3n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn-1} & a_{mn} \end{bmatrix}$$

Generarea matricelor: Matricile pot fi generate în mai multe moduri. Cel mai uzual constă în utilizarea parantezelor pătrate și a separatorilor. Ca separatori între elementele matricei se folosesc blankul sau virgula în interiorul unei linii, respectiv punct și virgula sau caracterul „newline” pentru a separa liniile.

Exemplu:



```
--> A = [5 7 9 1 -3 -7]
```

$$A = \begin{bmatrix} 5 & 7 & 9 \\ 1 & -3 & -7 \end{bmatrix}$$

```
--> B = [-1 2 5; 9 0 5]
```

$$B = \begin{bmatrix} -1 & 2 & 5 \\ 9 & 0 & 5 \end{bmatrix}$$

```
--> C = [0, 1; 3, -2; 4, 2]
```

$$C = \begin{bmatrix} 0 & 1 \\ 3 & -2 \\ 4 & 2 \end{bmatrix}$$

FreeMat / OCTAVE / MATLAB deține un set de funcții cu ajutorul cărora se pot construi matrice speciale, astfel : **zeros** – generează matricea nulă (cu elemente nule), **ones** – generează matricea formată din elemente 1, **eye** – matricea identică, **rand** – generează o matrice cu elemente numere aleatoare distribuite uniform, **randn** – numere aleatoare distribuite normal.

Sintaxa acestor funcții este aceeași, astfel că: **zeros(m,n)** generează o matrice cu elemente nule cu m linii și n coloane; **zeros(n)** generează o matrice cu elemente nule cu n linii și n coloane (matrice pătratică).

Exemplu:



<pre>--> zeros(3,2) ans = 0 0 0 0 0 0</pre>	<pre>--> zeros(2) ans = 0 0 0 0</pre>	<pre>--> ones(2) ans = 1 1 1 1</pre>
<pre>--> rand(2,3) ans = 0.73478 0.88761 0.57471 0.58542 0.32852 0.28011</pre>		
<pre>--> randn(2,3) ans = 1.32412 -0.12292 -0.47123 0.32081 -1.00701 0.64009</pre>		

Deoarece **FreeMat / OCTAVE / MATLAB** sunt programe bazate pe prelucrarea matricelor, este foarte important să se cunoască modul de introducere și de reprezentare ale matricelor (și implicit a vectorilor). Matricele sunt tipuri de date fundamentale în **FreeMat / OCTAVE / MATLAB**, ele sunt tablouri multidimensionale în dublă precizie. Cele mai des folosite sunt matricele bidimensionale cu m linii și n coloane. Vectorii linie (m = 1) și coloană (n = 1) sunt cazuri particulare ale matricelor bidimensionale.

Pentru generarea vectorilor cu pas liniar trebuie să se cunoască limitele intervalului (minimă și maximă) și pasul dintre două elemente consecutive.

Sintaxa utilizată pentru generarea vectorilor este:

$$v = v_min:pas:v_max,$$

unde **v_min**, respectiv **v_max** reprezintă valoarea minimă, respectiv maximă a intervalului, iar **pas** reprezintă pasul. Dacă nu se precizează pasul atunci acesta se consideră egal cu unitatea. Dacă $v_min > v_max$ vectorul va fi ordonat descrescător.

Exemplu:



Generarea vectorului $v1 = [1.00 \ 1.25 \ 1.50 \ 1.75 \ 2.00 \ 2.25 \ 2.50 \ 2.75 \ 3.00]$, respectiv $v2 = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10]$.

```
--> v1=1:0.25:3
v1 =
Columns 1 through 9:
1.0000  1.2500  1.5000  1.7500  2.0000  2.2500
2.5000  2.7500  3.0000

--> v2=1:10
v2 =
1 2 3 4 5 6 7 8 9 10
```

Dacă se dorește generarea unui vector pentru care se cunosc valoarea minimă (v_min), valoarea maximă (v_max) și numărul de elemente (n) se utilizează funcția `linspace`, astfel:

$$v = \text{linspace}(v_min, v_max, n)$$

Exemplu:



Generarea vectorului $v3 = [1.00 \ 1.25 \ 1.50 \ 1.75 \ 2.00 \ 2.25 \ 2.50 \ 2.75 \ 3.00]$

```
--> v3=linspace(1,3,9)
v3 =
Columns 1 through 9:
1.0000 1.2500 1.5000 1.7500 2.0000 2.2500 2.5000 2.7500 3.0000
```

Generarea unui vector cu n elemente cu pas logaritmic, între valorile v_min (minimă) și v_max (maximă) se realizează cu funcția `logspace`, astfel:

$$v = \text{logspace}(v_min, v_max, n)$$

Vectorul conține n elemente, distribuite logaritmic între $[10^{v_min}, 10^{v_max}]$. Dacă nu se precizează numărul de elemente va fi generat un vector cu 50 de elemente distribuite logaritmic între $[10^{v_min}, 10^{v_max}]$.

Exemplu:



```
--> v4=logspace(-2,2,5)
v4 =
1.0000e-002      1.0000e-001      1.0000e+000
1.0000e+001  1.0000e+002
```

Dacă se dorește generarea unui vector coloană, elementele acestuia trebuie separate prin punct-virgulă (;).

Exemplu:



```
--> v = [0;1;3;2;4]
v =
0
1
3
2
4
```

O altă categorie de funcții se utilizează în analiza matriceală, astfel: **inv** - calculează inversa unei matrice pătratice nesingulare; **rank** - calculează rangul unei matrice; **det** - calculează determinantul unei matrice pătratice; **trace** - calculează urma unei matrice; **norm** - calculează norma vectorului sau matricei; **size** - returnează dimensiunea unei matrice; **length** - returnează lungimea unui vector; **'** - calculează transpusa unei matrice / vector; **diag** - extrage din matrice matricea diagonală.

Exemplu:



Afișarea unei matrice pe ecran se realizează cu ajutorul funcției `disp`, sub forma `disp(A)`, unde `A` este matricea pe care dorim să

o afișăm. Se consideră matricea: $A = \begin{bmatrix} 1 & 2 & 3 \\ -4 & 2 & -5 \\ 6 & 0 & -4 \end{bmatrix}$

```
--> A = [1 2 3 ; -4 2 -5 ; 6 0 -4];
--> inv(A)
ans =
0.0588      -0.0588      0.1176
0.3382       0.1617      0.0514
0.0882      -0.0882     -0.0735

--> det(A)
ans = -136

--> size(A)
ans =
3 3

--> A'
ans =
1   -4    6
2    2    0
3   -5   -4
```

Alte funcții asociate matricelor sunt: **reshape** - schimbarea dimensiunii; **diag** - matrice diagonale și diagonale ale matricelor; **blkdiag** - matrice diagonală pe blocuri; **tril** - extragerea părții triunghiulare inferioare; **triu** - extragerea părții triunghiulare superioare; **fliplr** - rotire matrice în jurul axei de simetrie verticale; **flipud** - rotire matrice în jurul axei de simetrie orizontale; **rot90** - rotația unei matrice cu 90 de grade.

Exemplu:



Se consideră matricea M, cu elementele:

$$M = \begin{bmatrix} 1 & 3 & 2 & 4 \\ -2 & 0 & 1 & 3 \\ 0 & 2 & -3 & 1 \end{bmatrix}$$

```
--> M = [1 3 2 4; -2 0 1 3; 0 2 -3 1];
--> disp(M) % afișarea matricei M
1 3 2 4
-2 0 1 3
0 2 -3 1

--> reshape(M,4,3) % Redimensionarea matricei M
ans =
1 0 -3
-2 2 4
0 2 3
3 1 1

--> M2=[1 3 2; -2 0 1; 0 2 -3]
% Generarea matricei M2
M2 =
1 3 2
-2 0 1
0 2 -3

--> D1=tril(M2)
% Extragerea părții triunghiulare inf.
D1 =
1 0 0
-2 0 0
0 2 -3

--> D2=triu(M2)
% Extragerea părții triunghiulare sup.
D2 =
1 3 2
```

```

0      0      1
0      0     -3

% Rotirea matricei M2 cu 90° (1x90°), respectiv
cu 2x90° (2x90°)
--> rot90(M2,1)
ans =
2      1     -3
3      0      2
1     -2      0
--> rot90(M2,2)
ans =
-3      2      0
1      0     -2
2      3      1

```

2.3. Extragerea elementelor individuale dintr-un vector sau matrice

În cazul vectorilor elementele individuale se apelează prin numele vectorului urmat de indicele elementului dorit, plasat între paranteze rotunde.

Dacă se dorește să se extragă o succesiune de elemente dintr-un vector se utilizează forma $A(i:j)$, unde A – reprezintă numele vectorului, i și j reprezintă indicii elementelor de început, respectiv de sfârșit a subșirului care se extrage din vectorul A .

Pentru extragerea unei succesiuni de elemente cu un anumit pas, se utilizează forma $A(i:p:j)$, unde p reprezintă pasul.

Exemplu:



```

Se consideră șirul : A = [-2 1 3 0 6 4 -3 5 7 -1 6 8 2 -3 9]

--> A=[-2 1 3 0 6 4 -3 5 7 -1 6 8 2 -3 9]
A =
-2 1 3 0 6 4 -3 5 7 -1 6 8 2 -3 9

--> A(5) % Se extrage val. elem. cu indicele 5
ans = 6
--> A(10) % Se extrage val. elem cu indicele 10
ans = -1
--> A(3:6) % Se extrag elem. cu indicii 3,4,5,6
ans =
3 0 6 4
--> A(2:2:8) % Se extrag elem cu indicii 2,4,6,8
ans =
1 0 4 5

```

În cazul matricelor, elementele individuale se apelează cu numele matricei urmat de doi indici (unul pentru linie și unul pentru coloană) separați prin virgulă și cuprinși între paranteze rotunde.

Exemplu:



Se consideră matricea M, cu elementele: $M = \begin{bmatrix} 1 & 3 & 2 & 4 \\ -2 & 0 & 1 & 3 \\ 0 & 2 & -3 & 1 \\ -4 & 1 & -1 & 0 \end{bmatrix}$

```
--> M(2,3) % Extragerea elementului situat pe
linia 2 coloana 3
ans = 1
```

```
--> M(:,2) % Extragerea coloanei a doua
ans =
3
0
2
1
```

```
--> M(2,1:3) % Extrag. primelor trei elem.
de pe linia a doua
ans =
-2 0 1
```

```
--> C=M(2:4,1:3) % Extragerea unei matrice care
să conțină lin. 2,3,4 și
col. 1,2,3
C =
-2    0    1
 0    2   -3
-4    1   -1
```

3. Rezolvarea ecuațiilor algebrice și transcendente. Polinoame. Rezolvarea sistemelor de ecuații liniare și neliniare. Interpolarea și aproximarea datelor. Derivarea numerică. Integrarea numerică. Rezolvarea ecuațiilor diferențiale ordinare

Introducere:



Așa cum s-a arătat în primul capitol, **FreeMat / OCTAVE / MATLAB** oferă o serie de funcții cu ajutorul cărora se pot rezolva mai multe categorii de probleme din domeniul ingineresc.

În cadrul acestui curs se dorește prezentarea funcțiilor utile în rezolvarea unor categorii largi de probleme de matematică care apar în calculele din domeniul ingineresc. Cursul este împărțit în șapte părți, în fiecare dintre acestea fiind prezentate funcțiile cu ajutorul cărora se rezolvă o categorie de probleme. Astfel, în prima parte sunt prezentate funcțiile **roots**, **fzero** și **solve** pentru rezolvarea ecuațiilor algebrice și transcendente. În partea a doua sunt prezentate funcții și operatori pentru prelucrarea polinoamelor. În partea a treia sunt prezentate funcțiile utile în rezolvarea sistemelor de ecuații liniare și neliniare. Funcțiile pentru interpolarea și aproximarea datelor sunt prezentate în partea a patra a cursului. În calculele ingineresti avem nevoie adesea de efectuarea operațiilor de derivare sau/și integrare numerică. Funcțiile pentru derivare/integrare numerică sunt prezentate în subcapitolele cinci și șase ale cursului. O altă categorie de probleme des întâlnită în inginerie o reprezintă rezolvarea ecuațiilor / sistemelor de ecuații diferențiale ordinare. În ultima parte a cursului sunt prezentate funcțiile pentru rezolvarea ecuațiilor / sistemelor de ecuații diferențiale.

Obiective:



După parcurgerea acestui material, studenții vor ști să utilizeze mediile de programare pentru: rezolvarea ecuațiilor algebrice și transcendente; realizarea unor operații cu polinoame; rezolvarea sistemelor de ecuații liniare și neliniare; interpolarea și aproximarea datelor; derivarea și integrarea numerică; rezolvarea ecuațiilor diferențiale ordinare;

3.1. Rezolvarea ecuațiilor algebrice și transcendente

Se consideră ecuația $f(x) = 0$. Dacă ecuația poate fi adusă la o formă polinomială se numește algebrică în caz contrar ea se numește transcendentă.

Exemple:

- ecuații algebrice: $2 \cdot x^2 - 5 \cdot x + 3 = 0$; $3 \cdot x^4 - 6 \cdot x^3 + x^2 - 7 \cdot x + 9 = 0$;
- ecuații transcendente: $\sin(2 \cdot x) + \cos(x) - 0,5 = 0$; $e^{\ln(x)-x} = \pi$;

Pentru determinarea soluțiilor ecuațiilor transcendente se utilizează metode de aproximare.

Se consideră funcția $f: [x_{\min}, x_{\max}] \rightarrow \mathfrak{R}$, cu $[x_{\min}, x_{\max}] \subset \mathfrak{R}$.

Dacă există o valoare $\xi \in [x_{\min}, x_{\max}]$ astfel încât $f(\xi) = 0$, atunci ξ se numește **rădăcină exactă** a ecuației $f(x) = 0$ sau **zero** al funcției f . O valoare ξ' apropiată de ξ se numește rădăcină aproximativă a ecuației $f(x) = 0$. O rădăcină aproximativă se poate defini în două moduri:

- ξ' cu proprietatea: $|\xi - \xi'| < \varepsilon$, cu $\varepsilon \in \mathfrak{R}$, $\varepsilon > 0$;
- ξ' cu proprietatea: $|f(\xi')| < \varepsilon$;

Pentru determinarea soluțiilor unei ecuații de forma $f(x) = 0$ trebuie parcurse două etape:

A. determinarea intervalelor în care se găsesc soluții: constă în împărțirea intervalului de definiție al funcției în mai multe subintervale astfel încât fiecare interval să conțină cel mult o rădăcină a ecuației.

Pentru stabilirea intervalelor în care se găsesc rădăcinile ecuației $f(x) = 0$, se utilizează teorema conform căreia: dacă o funcție continuă $f(x)$ are valori de semn opus la capetele unui interval $[a, b]$ (adică dacă $f(a) \cdot f(b) < 0$) atunci în acel interval se găsește cel puțin o rădăcină a ecuației $f(x) = 0$. Rădăcina este unică în intervalul $[a, b]$ dacă derivata funcției $f(x)$ există și își păstrează semnul în acel interval, în caz contrar în intervalul considerat se pot afla mai multe rădăcini;

B. calculul rădăcinilor cu o anumită precizie impusă. Pentru determinarea rădăcinilor se pot utiliza diverse metode aproximative, cum ar fi: metoda biseției, metoda tangentei (Newton), metoda secantei, metoda falsei poziții, metoda aproximațiilor succesive.

Pentru rezolvarea ecuațiilor în **FreeMat** / **OCTAVE** / **MATLAB** se pot utiliza următoarele funcții:

A. Funcția **roots(v)**: pentru un vector **v** cu **n** elemente calculează rădăcinile ecuației polinomiale:

$$v_1 \cdot x^{n-1} + v_2 \cdot x^{n-2} + \dots + v_{n-1} \cdot x + v_n = 0$$

Exemplu:



Să se determine rădăcinile ecuației: $x^2 - 4 \cdot x + 3 = 0$;

```
--> v = [1, -4, 3]; roots(v)
ans =
3
1
```

B. Funcția **fzero**: determină rădăcina (zero-ul) unei funcții dependente de o variabilă reală. Forma generală a acestei funcții este:

[x, fval, info, output] = fzero (fun, x0, options)

unde: **fun** – reprezintă definiția sau numele funcției, **x0** – reprezintă un punct de start, **options** – reprezintă o structură utilizată adițional pentru specificarea unor opțiuni.

Funcția **fzero** recunoaște următoarele opțiuni: "FunValCheck" (verifică valorile invalide ale funcției obiectiv, poate avea valorile ' on ' sau ' off '), "OutputFcn" (apelează toate funcțiile de ieșire după fiecare iterație), "TolX" (toleranța parametrului), "MaxIter" (numărul maxim de iterații), "MaxFunEvals" (numărul maxim de evaluări ale funcției obiectiv – întreg pozitiv).

La ieșire funcția returnează: **x** – rădăcina aproximativă, **fval** – valoarea funcției în **x**, **info** – un parametru care poate avea valorile: (1) – algoritmul converge către o soluție, (0) – numărul maxim de iterații sau de evaluări a funcției a fost depășit, (-1) – algoritmul a fost întrerupt de către funcția de ieșire definită de utilizator, (-2) – o eroare neașteptată, (-3) – s-a întâlnit o valoare nereală, (-4) – s-a întâlnit o valoare NaN (Not – a – number).

Exemplu:



Să se determine rădăcinile ecuației: $-x^2 + \log(x) + 4 = 0$.

```
--> f = inline('-x.^2 + log(x) + 4 '),
--> x = fzero(f,0)
f =
f(x) = -x.^2+log(x)+4
x = 0.018322

--> f = inline('-x.^2 + log(x) + 4 '),
--> x=fzero(f,2)
f =
f(x) = -x.^2+log(x)+4
x = 2.1869
```



Exemplu:

Să se determine rădăcina ecuației $x + \log(x) = 0$, în intervalul $[0,1]$.

```
--> f=inline('x+log(x)'), x = fzero(f,[0 1])  
f =  
f(x) = x+log(x)  
x = 0.56714
```

C. Funcția **fsolve**: se utilizează pentru rezolvarea ecuațiilor / sistemelor de ecuații neliniare. Are forma generală:

[x, fval, info, output, fjac] = fsolve(fun, x0, options)

unde: **fun** – reprezintă definiția sau numele funcției, **x0** – reprezintă un punct de start, **options** – reprezintă o structură utilizată adițional pentru specificarea unor opțiuni.

La ieșire funcția returnează: **x** – rădăcina aproximativă, **fval** – valoarea funcției în **x**, **info** – un parametru care poate avea valorile: (1) – algoritmul converge către o soluție, (2) – pasul relativ este mai mic decât TolX, (3) – ultima valoare relativă descrește la o valoare mai mică decât TolF, (0) – s-a depășit numărul maxim de iterații, etc.

3.2. Polinoame în FreeMat / OCTAVE / MATLAB

În **FreeMat / OCTAVE / MATLAB** un polinom este reprezentat ca un vector ale cărui componente sunt coeficienții polinomului aranjați în ordine descrescătoare a puterilor. Pentru coeficienții puterilor care lipsesc se introduce 0.



Exemplu:

$P1(X) = X^3 - 3X^2 + 2X + 4$ --> $P1 = [1 \ -3 \ 2 \ 4]$

$P2(X) = X^5 - 2X^3 + 4$ --> $P2 = [1 \ 0 \ -2 \ 0 \ 0 \ 4]$

$P3(X) = X^{99} - X + 2$ --> $P3 = [1 \ \text{zeros}(1,97) \ -1 \ 2]$

FreeMat / MATLAB / OCTAVE oferă o serie de funcții utile în prelucrarea polinoamelor, astfel: **conv** - calculează produsul a două polinoame; **deconv** - calculează câtul și restul împărțirii a două polinoame; **poly** - calculează coeficienții unui polinom cu rădăcinile 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 în fracții simple raportul a două polinoame; **roots** - calculează rădăcinile unui polinom.

Exemplu:



Să considerăm două polinoame: $P(X) = X^2 - 2X + 1$ și $Q(X) = X - 1$

--> $p = [1 \quad -2 \quad 1]$, $q = [1 \quad -1]$;

Produsul polinoamelor: $P(X)*Q(X) = X^3 - 3X^2 + 3X - 1$

--> **conv (p , q)**

ans =
1 -3 3 -1

Derivata produsului: $[P(X)*Q(X)]' = 3X^2 - 6X + 3$

--> **polyder (conv (p , q))**

ans =
3 -6 3

Soluția ecuației: $P(X)*Q(X) = 0$ $P(X)*Q(X) = (X-1)^3$, adică: $X_1 = X_2 = X_3 = 1$

--> **roots (conv (p , q))**

ans =
1.00000 + 0.00000i
1.00000 + 0.00000i
1.00000 - 0.00000i

Funcții și operații cu polinoame:

Funcția: `polyout(p,"X")` : determină afișarea pe ecran a polinomului sub formă algebrică, astfel:

Exemplu:



$P1(X) = X^3 - 3X^2 + 2X + 4$

--> **P1=[1 -3 2 4]; polyout(P1,"X");**

$1*X^3 - 3*X^2 + 2*X^1 + 4$

Adunarea a două polinoame se realizează după regula adunării a doi vectori. În cazul în care cele două polinoame au grade diferite înainte de adunarea lor se aduc la aceeași dimensiune, astfel în vectorul care conține coeficienții de grad mai mic se introduce 0 pentru puterile care lipsesc.

Exemplu:




$P1(X) = X^3 - 3X^2 + 2X + 4$, $P2(X) = 3X^2 + 2$, $P3(X) = X^3 + 2X + 6$

--> **P1=[1 -3 2 4]; P2=[0 3 0 2]; P3=P1+P2;**

--> **polyout (P3, "X") ;**


$1 * X ^ 3 + 0 * X ^ 2 + 2 * X ^ 1 + 6$

Înmulțirea a două polinoame se realizează cu ajutorul funcției **conv**, astfel :

Exemplu:  $P1(X) = X^3 - 3X^2 + 2X + 4$, $P2(X) = 3X^2 + 2$, $P3(X) = 3X^5 - 9X^4 + 8X^3 + 6X^2 + 4X + 8$


```
--> P1=[1 -3 2 4];P2=[0 3 0 2];
--> P3=conv(P1,P2);polyout(P3,"X");
0*X^6 + 3*X^5 - 9*X^4 + 8*X^3 + 6*X^2 + 4*X^1
+ 8
```

Împărțirea a două polinoame se realizează cu ajutorul funcției **deconv**, astfel :

Exemplu:  $P1(X) = X^3 - 3X^2 + 2X + 4$, $P3(X) = 3X^5 - 9X^4 + 8X^3 + 6X^2 + 4X + 8$, $P4(X) = 3X^2 + 2$


```
--> P4 = deconv(P3,P1); polyout(P4,"X");
0*X^3 + 3*X^2 + 0*X^1 + 2
```

Rezolvarea ecuațiilor algebrice polinomiale se realizează cu ajutorul funcției **roots**:

Exemplu:  $P=[1 \ 3 \ 2]$; **roots(P)**


```
ans =
-2
-1
```

Cu ajutorul funcției **poly(v)** se construiește polinomul al cărui rădăcini sunt date sub forma vectorului **v**:

Exemplu:  $v=[1 \ 2]$; **P=poly(v)**; **polyout(P,"X")**;


```
1*X^2 - 3*X^1 + 2
```

Pentru determinarea derivatei unui polinom se utilizează una dintre funcțiile **polyder**, respectiv **polyderiv**, astfel:


Exemplu:  $P=[1 \ -3 \ 2 \ 4]$; **polyout(P,"X")**;

```
--> D=polyder(P);polyout(D,"X");
1*X^3 - 3*X^2 + 2*X^1 + 4
3*X^2 - 6*X^1 + 2
```


Integrala nedefinită a unui polinom (primitiva) se obține utilizând funcția **polyinteg** sau **polyint** astfel:

Exemplu:  `--> P=[1 -3 2 4];polyout(P,"X");`
`--> IP=polyint(P);polyout(IP,"X");`
 $1*X^3 - 3*X^2 + 2*X^1 + 4$
 $0.25*X^4 - 1*X^3 + 1*X^2 + 4*X^1 + 0$

Pentru calculul valorii unui polinom pentru una sau mai multe valori ale lui x, se utilizează funcția **polyval**, astfel :

Exemplu:  `--> P = [1 -3 2 4];polyval(P,0),`
`--> polyval(P,1),polyval(P,3)`
`ans = 4`
`ans = 4`
`ans = 10`


Polinomul caracteristic al unei matrice se poate obține utilizând funcția **poly**, astfel :

Exemplu:  `--> A=[1 2 3;-3 2 4;0 -2 1],pc = poly(A)`
`A =`
`1 2 3`
`-3 2 4`
`0 -2 1`
`pc =`
`1.0000 -4.0000 19.0000 -34.0000`

Aproximarea unui set de date printr-un polinom de grad **n** se realizează cu ajutorul funcției **polyfit** a cărei sintaxă este:

polyfit(x,y,n)

unde: **x** reprezintă variabila independentă, **y** reprezintă variabila dependentă, **n** reprezintă gradul polinomului.

Exemplu:  `--> x = -3:2,P=[1 2 1], y=polyval(P,x),`
`--> pn=polyfit(x,y,2)`
`--> val=polyval(pn,x)`
`x =`
`-3 -2 -1 0 1 2 3`
`P =`
`1 2 1`
`y =`
`4 1 0 1 4 9 16`
`pn =`

```

1.00000 2.00000 1.00000
val =
Columns 1 through 7:
4.0000 1.0000 -1.6653e-015 1.0000 4.0000
9.0000 1.6000e+001

```

3.3. Rezolvarea sistemelor liniare și neliniare

Se consideră următorul sistem de ecuații liniare:

$$\begin{cases} 3 \cdot x_1 + x_2 = -1 \\ x_1 - 2 \cdot x_2 + 3 \cdot x_3 = -2 \\ 3 \cdot x_2 + x_3 + 2 \cdot x_4 = 6 \\ x_3 - 3 \cdot x_4 = 4 \end{cases}$$

Pentru rezolvarea sistemului se construiește matricea coeficienților necunoscutelor A și vectorul coloană a termenilor liberi B . Pentru găsirea soluției se aplică relația: $X = A^{-1} \cdot B$.

Astfel obținem:

$$A = \begin{pmatrix} 3 & 1 & 0 & 0 \\ 1 & -2 & 3 & 0 \\ 0 & 3 & 1 & 2 \\ 0 & 0 & 1 & -3 \end{pmatrix} \quad B = \begin{pmatrix} -1 \\ -2 \\ 6 \\ 4 \end{pmatrix} \quad X = \begin{pmatrix} -1,078 \\ 2,233 \\ 1,181 \\ -0,940 \end{pmatrix}$$

În **FreeMat** / **OCTAVE** / **MATLAB** se utilizează următoarele funcții: **inv** – calculează inversa unei matrice patratice nesingulare; **pinv** – calculează pseudoinversa unei matrice; **rank** – calculează rangul unei matrice; **det** – calculează determinantul unei matrice patratice; **fsolve** – calculează rădăcinile reale pentru un sistem liniar sau neliniar.

Exemplu:



Rezolvarea unui sistem de ecuații liniare

```
--> A=[3 1 0 0;1 -2 3 0;0 3 1 2;0 0 1 -3];
```

```
--> B=[-1;-2;6;4]
```

```
--> X=A\B
```

```
X =
```

```
-1.0776
```

```
2.2328
```

```
1.1810
```

```
-0.9397
```

Exemplu:



Rezolvarea unui sistem de ecuații neliniare

Să se rezolve sistemul de ecuații neliniare:

$$\begin{cases} -2 \cdot x_1^2 + 3 \cdot x_1 \cdot x_2 + 4 \cdot \sin(x_2) = 6 \\ 3 \cdot x_1^2 - 2 \cdot x_1 \cdot x_2^2 + 3 \cdot \cos(x_1) = -4 \end{cases}$$

cu valorile de pornire $x = 1, y = 2$.

```
--> function y=f(x)
--> y(1)=-2*x(1)^2+3*x(1)*x(2)+4*sin(x(2))-6;
--> y(2)=3*x(1)^2-2*x(1)*x(2)^2+3*cos(x(1))+4;
--> endfunction
--> sol=fsolve("f",[1;2])
sol =
0.57983
2.54621
```

3.4. Interpolarea și aproximarea datelor

Se consideră o colecție de puncte $(x_0, y_0), \dots, (x_n, y_n)$. Prin **interpolare** se înțelege determinarea unei funcții φ - numită funcție de interpolare, cu proprietatea:

$$\varphi(x_i) = y_i, \quad i = \overline{0, n}$$

Dacă funcția de interpolare este **polinomială**, metoda se numește **interpolare polinomială**. Dacă funcția de interpolare este **trigonometrică**, metoda se numește **aproximare trigonometrică**. În cazul în care funcția de interpolare este **local polinomială**, metoda se numește **interpolare polinomială pe porțiuni** sau interpolare de tip **spline**.

Funcțiile de interpolare disponibile în **FreeMat / OCTAVE / MATLAB** sunt: **table1** - interpolează liniar și citește date din tabele unidimensionale; **spline** - interpolează prin metoda **spline** date din tabele unidimensionale; **polyfit** - aproximează un set de date cu un polinom de gradul **n**; **interp1** - interpolează liniar, spline sau cubic date din tabele unidimensionale; **table2** - interpolează liniar date și citește date din tabele bidimensionale; **interp2** - interpolează biliniar sau bicubic date din tabele bidimensionale; **interp3** - interpolează biarmonic valorile unei funcții de două variabile; **interp4** - interpolează biliniar valorile unei funcții de două variabile; **interp5** - interpolează bicubic valorile unei funcții de două variabile; **regress** - aproximează un set de date cu o funcție de mai multe variabile.

Exemplu:



În urma unui experiment a rezultat urmatorul tabel de date :

t (timp)	1	2	3	4	5
T(temperatura)	0	20	60	77	110

Să se determine expresia analitică a curbei polinomiale care aproximează funcția $T = f(t)$:

```
--> x = 1:5; y = [ 0.20 60 68 77 110]; plot(x,y)
--> coef = polyfit(x,y,1); y1= coef(1)*x+coef(2);
```

Exemplu:



În urma unui experiment se înregistrează datele: $ora=[0 \ 2.5 \ 5 \ 7.5]$, $temp=[1 \ -1 \ 0.5 \ 0]$.

Se cere să se reprezinte grafic variația lui $temp$ în intervalul $[0, 7.5]$, folosind `cubic_spline` de pas 0.25 . Se cere să se estimeze valoarea lui $temp$ la $ora = 0.25$.

```
--> ora=[0 2.5 5 7.5]; temp=[1 -1 0.5 0];
--> xi=0:0.25:7.5; >>> yi=spline(x,y,xi);
--> plot(ora,temp,'o',xi,yi)
--> valoarea_ceruta0 = spline(ora,temp,0.25)
```

3.5. Derivarea numerică

Funcțiile oferite de **FreeMat / OCTAVE / MATLAB** pentru derivarea numerică sunt: **diff** – evaluează diferența elementelor succesive ale unui vector sau ale coloanei matricei; **gradient** – calculează derivatele parțiale ale unei funcții de două variabile; **4*del2** – calculează Laplacianul unei funcții de două variabile.

Exemplu:



Să se calculeze derivatele parțiale ale funcției:

$$z(x,y) = x \cdot \exp(-x^2 - y^2)$$

```
--> [x,y]=meshgrid(-2:0.2:2, -2:0.2:2)
--> z=x.*exp(-x.^2-y.^2)
--> [px,py]=gradient(z,0.2,0.2)
```

Exemplu:



Să se calculeze valoarea aproximativă a **Laplacianului** pentru funcția $f(x,y)=3 \cdot x^3 + 2 \cdot y^2$ pe intervalul $[-4,4] \times [-4,4]$

```
--> [x,y]=meshgrid(-4:4, -4:4)
--> f=2*x.^3+2*y.^3
--> L=4*del2(f)
```

3.6. Integrarea numerică

Funcțiile oferite de **FreeMat** / **OCTAVE** / **MATLAB** pentru integrare numerică sunt: **quad** - calculează integrala prin metoda adaptiv-recursivă Simpson; **trapz** - calculează integrala prin metoda trapezelor.

Funcția **quad** utilizează expresia analitică a funcției de integrat f , $y = f(x)$. Variante ale sintaxei funcției **quad** sunt:

```
I=quad(ume_fisier,a,b)    sau
I=quad(ume_fisier,a,b,precizia)
```

unde: *ume_fisier* reprezintă un șir de caractere care definește numele fișierului-funcție în care a fost scrisă expresia funcției de integrat f ; a și b reprezintă limitele de integrare (capetele intervalului $[a,b]$ pe care se realizează integrarea); *precizia* este un argument opțional prin care se poate modifica precizia implicită 10^{-6} ; I reprezintă aproximarea integralei definite

$$I = \int_a^b f(x) \cdot dx.$$

Funcția **trapz** utilizează funcția de integrat sub formă de valori numerice $y=f(x)$ în puncte echidistante ale intervalului de integrat $[a,b]$. Sintaxa este următoarea:

```
I = trapz(x,y)
```

unde: x și y reprezintă vectorii valorilor funcției.

Exemplu:



Să se calculeze $I = \int_0^4 \frac{\sin(x)}{x} \cdot dx$ prin metoda trapezelor și prin metoda adaptiv-recursivă Simpson:

```
--> eps=1/10^3; x=eps:0.01:4;y=sin(x)./x;
--> I=trapz(x,y)
I = 1.7589

--> eps=1/10^3; f=inline('sin(x)./x '),
--> I=quad(f,eps,4)
f =
f(x) = sin(x)./x
I = 1.7572
```

Funcția **dblquad** se utilizează pentru calculul integralelor duble de forma $I = \int_{xa}^{xb} \int_{ya}^{yb} f(x, y) \cdot dx dy$. Forma generală a funcției poate fi:

dblquad (f, xa, xb, ya, yb)
dblquad (f, xa, xb, ya, yb, tol)

unde: f reprezintă o funcție de două variabile; xa, ya, xb, yb reprezintă limitele de integrare pentru x și y ; tol reprezintă toleranța absolută utilizată;

Exemplu:



Să se calculeze integrala dublă: $I = \int_0^1 \int_0^1 \sin(\pi \cdot x \cdot y) \cdot \sqrt{x \cdot y} dx dy$

--> `I = dblquad (@(x, y) sin(pi*x.*y) .* sqrt(x.*y), 0, 1, 0, 1)`
`I = 0.30022`

Funcția **triplequad** se utilizează pentru calculul integralelor triple de forma:

$$I = \int_{xa}^{xb} \int_{ya}^{yb} \int_{za}^{zb} f(x, y, z) dx dy dz$$

Forma generală a funcției poate fi:

triplequad ($f, xa, xb, ya, yb, za, zb$)
triplequad ($f, xa, xb, ya, yb, za, zb, tol$)

parametri funcțiilor având aceeași semnificație ca și la funcția **dblquad**.

Exemplu:



Să se calculeze integrala triplă:

$$I = \int_0^1 \int_{-3}^{-1} \int_{-2}^2 (x^2 y^2 z + 2z^2 x) dx dy dz$$

--> `I = triplequad(@(x, y, z) x.*x.*y.*y.*z.+2*z.*z.*x, 0, 1, -3, -1,-2, 2)`
`I = 10.667`

3.7. Rezolvarea ecuațiilor diferențiale ordinare

Funcțiile oferite de **FreeMat / OCTAVE / MATLAB** pentru rezolvarea ecuațiilor (sistemelor de ecuații) diferențiale sunt: **ode23** - rezolvă ecuațiile diferențiale sau sisteme de ecuații diferențiale prin metoda Runge-Kutta de ordinul 2 (3), respectiv **ode45** - rezolvă ecuații diferențiale sau sisteme de ecuații diferențiale prin metoda Runge-Kutta de ordinul 4 (5).

Cele două funcții au sintaxa asemănătoare, două variante fiind:

```
[xval,yval] = fct_M('nume_fisier',dom,y0)
[xval,yval] = fct_M('nume_fisier',dom,y0,optiuni)
```

unde:

- *fct_M* reprezintă numele funcției OCTAVE / MATLAB (ode23 sau ode45);
- *nume_fisier* reprezintă un șir de caractere care conține numele fișierului-funcție în care a fost definită expresia derivatei funcției-necunoscute. *Nume_fisier* conține vectorul expresiilor derivatelor de ordin I al funcțiilor-necunoscute în cazul unei ecuații diferențiale de ordinul I, sistemelor de ecuații diferențiale de ordinul I sau al ecuațiilor și sistemelor de ecuații diferențiale de ordin superior, care au fost aduse în prealabil la o formă echivalentă cu un sistem de ordinul I;
- *dom* reprezintă vectorul limitelor intervalului [a,b] al variabilei independente;
- *y0* reprezintă valoarea funcției-necunoscute din condiția inițială în cazul unei ecuații diferențiale de ordinul I, respectiv, vectorul valorilor funcțiilor-necunoscute din condițiile inițiale;
- *optiuni* reprezintă o structură care conține opțiuni de optimizare a calculării soluției / soluțiilor (opțional);
- *xval* reprezintă un vector ce conține valorile variabilei independente, în care se determină valorile soluției / soluțiilor;
- *yval* reprezintă vectorul valorilor funcției soluție în punctele definite prin *xval*.

Exemplu:



Să se integreze ecuația diferențială $y' = 3 \cdot t^2 - 4 \cdot t + 2$ pe intervalul [1,4], cu condițiile inițiale $y(2) = 0,5$.

Se editează mai întâi fișierul care conține definiția funcției de integrat:

```
% fisierul f1.m
function dy = f1m(t,y)
dy = 3*t^2-4*t+2;
```

apoi pe prompter se scrie următoarea secvență

```
--> [t,y] = ode23(@f1,[1 4],0.5);plot(t,y)
```



Exemplu:

Se consideră un sistem mecanic vibrant format dintr-o masă m și un element elastic, asupra masei acționând o forță armonică (figura 4.1):

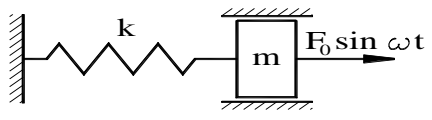


Figura 3.1. Sistem mecanic vibrant

Ecuția diferențială a mișcării este: $m\ddot{x} + kx = F_0 \cdot \sin \omega t$ a cărei soluție generală se exprimă astfel:

$$x = x_0 \cdot \cos pt + \frac{1}{p} \left(v_0 - \frac{q\omega}{p^2 - \omega^2} \right) \cdot \sin pt + \frac{q}{p^2 - \omega^2} \cdot \sin \omega t$$

unde: $p = \sqrt{\frac{k}{m}}$; $q = \frac{F_0}{m}$; x_0 , respectiv v_0 reprezentând condițiile inițiale.

Valorile numerice utilizate în exemplele care urmează sunt: $F_0 = 200$ [N], $\omega = 35$ [rad/s], $m = 250$ [kg], $k = 25.000$ [N/m].

Ecuția diferențială de ordinul n trebuie transformată într-un sistem de n ecuații diferențiale de ordinul întâi. Acest sistem va fi introdus într-un fișier de tip funcție care va returna derivatele ecuațiilor sistemului. Astfel, ecuația diferențială a fost scrisă ca un sistem cu două ecuații diferențiale de ordinul întâi:

$$\begin{aligned} yd1 &= y(2); \\ yd2 &= \frac{F_0 \cdot \sin(\omega \cdot x) - k \cdot y(1)}{m} \end{aligned}$$

Fișierul funcție scris pentru această aplicație are următoarea formă:

```
% Fisierul functie yprim.m
function ydot = yprim(t,y)
m=250; k=25000; F0=200; omg=35;
yd1=y(2);
yd2=(F0*sin(omg*t)-k*y(1))/m;
ydot=[yd1;yd2];
```

Forma în care funcția a fost utilizată este următoarea:

```
-->[t,y]=ode45(@yprim,[0 2],[0;0],0.00000025);
--> plot(t,y(:,1),t,y(:,2))
```

4. Grafică 2D și 3D

Introducere:



Așa cum am arătat în cursurile anterioare, mediile de programare **FreeMat / OCTAVE / MATLAB** oferă o serie de funcții cu ajutorul cărora se pot efectua calcule matematice. Funcțiile pot fi apelate direct în linia de comandă sau pot fi incluse în fișiere program.

Pe lângă funcțiile matematice **FreeMat / OCTAVE / MATLAB** oferă o serie de funcții cu ajutorul cărora pot fi realizate reprezentări grafice bidimensionale sau tridimensionale.

*Scopul acestui curs este acela de a cunoaște modul în care pot fi realizate reprezentările bidimensionale și tridimensionale în mediile de programare **FreeMat / OCTAVE / MATLAB**.*

Dacă în alte medii de programare, de exemplu Borland C, pentru reprezentarea grafică a unei funcții trebuie scris un program (sau o funcție), în **FreeMat / OCTAVE / MATLAB** este suficient să se apeleze una din funcțiile predefinite.

Funcțiile predefinite cu ajutorul cărora se pot realiza reprezentări grafice acoperă o gamă largă de tipuri de reprezentări, putând fi utilizate tipuri de linii, marcatori și culori diferite.

În prima parte a cursului sunt prezentate funcțiile pentru realizarea reprezentărilor bidimensionale, descrierea fiecărei funcții fiind însoțită de exemple.

În mod asemănător, în partea a doua a cursului sunt prezentate funcțiile pentru reprezentările tridimensionale.

Obiective:



După parcurgerea acestui material, studenții vor ști să utilizeze mediile de programare **FreeMat / OCTAVE / MATLAB** pentru: realizarea reprezentărilor bidimensionale în coordonate carteziene; realizarea reprezentărilor bidimensionale în coordonate polare; realizarea reprezentărilor bidimensionale logaritmice; realizarea reprezentărilor tridimensionale; realizarea reprezentărilor suprafețelor; realizarea reprezentărilor tridimensionale a funcțiilor dependente de două variabile;

4.1. Grafica 2D

Cu ajutorul funcțiilor dedicate, aplicațiile **FreeMat** / **OCTAVE** / **MATLAB** permit realizarea și modificarea cu ușurință a graficelor și figurilor.

Funcțiile utilizate de aplicația **FreeMat** / **OCTAVE** / **MATLAB** pentru generarea graficelor bidimensionale sunt prezentate în tabelul tabelul 4.1:

Tabelul 4.1. Funcțiile **FreeMat** / **OCTAVE** / **MATLAB** pentru generarea graficelor 2D

Denumire:	Semnificație:
plot	Desenează în coordonate carteziane graficul 2D al unei funcții $y=f(x)$
polar	Desenează în coordonate polare graficul 2D al unei funcții
bar	Desenează un grafic cu bare 2D
subplot	Permite desenarea mai multor ferestre grafice pe același ecran
semilogx	Desenează în coordonate semilogaritmice (logaritmice pe direcția Ox) graficul 2D
semilogy	Desenează în coordonate semilogaritmice (logaritmice pe direcția Oy) graficul 2D
loglog	Desenează graficul 2D în coordonate logaritmice (pe Ox și Oy)
compass	Desenează imagini de numere complexe sub formă trigonometrică
grid	Adauga graficului o rețea grid
legend	Adaugă graficului o legendă
title	Adaugă un titlu graficului
xlabel	Adaugă o etichetă (text) pe axa Ox
ylabel	Adaugă o etichetă (text) pe axa Oy
axis	Permite setarea manuală a axelor pentru graficul curent
axes	Permite crearea axelor în poziții arbitrare
text	Realizează plasarea unui text pe grafic în poziția impusă
hold	Realizează înghețarea graficului curent pe ecran
clf	Șterge fereastra grafică curentă

La trasarea graficelor se poate utiliza următoarea paletă de culori: b – albastru, g – verde, k – negru, m – magenta, r – roșu, y – galben, c – cyan. Caractere acceptate pentru trasarea graficului sunt: punct (.), cerc (o), cruce (x), stea (*), plus (+), pătrat (s), diamond (d), triunghi cu vârful în jos (

v), triunghi cu vârf în sus (^), triunghi cu vârf la stânga (<), triunghi cu vârf la dreapta (>), pentagram (p), hexagram (h).

Tipurile de linii ce pot fi utilizate sunt: continuă (-), punctată (:), linie-punct (- .), linie întreruptă (--).

Funcția **plot** : este o funcție de bază utilizată în generarea graficelor 2D în coordonate carteziene. Forma generală a funcției este :

plot (x, y, 'șir')

unde : x, y reprezintă coordonatele iar *șir* reprezintă un șir de caractere conținând de la 1 la 4 caractere ce desemnează o culoare, un stil de linie și un tip de marker (caracter).

Cea mai simplă formă de utilizare a funcției plot este *plot(y)* unde y reprezintă un vector. În această formă se realizează graficul luând pe abscisă indicii punctelor și pe ordonată valorile elementelor vectorului y.

Exemplu:



Construim graficul funcției date prin punctele definite de perechile de coordonate (figura 4.1) :

1	2	3	4	5	6	7	8	9	10
-8	-5	-3	-2	0	2	4	7	10	12

Rezolvare în OCTAVE:

```
>>> y=[-8 -5 -3 -2 0 2 4 7 10 12], plot(y)
```

```
y =  
-8 -5 -3 -2 0 2 4 7 10 12
```

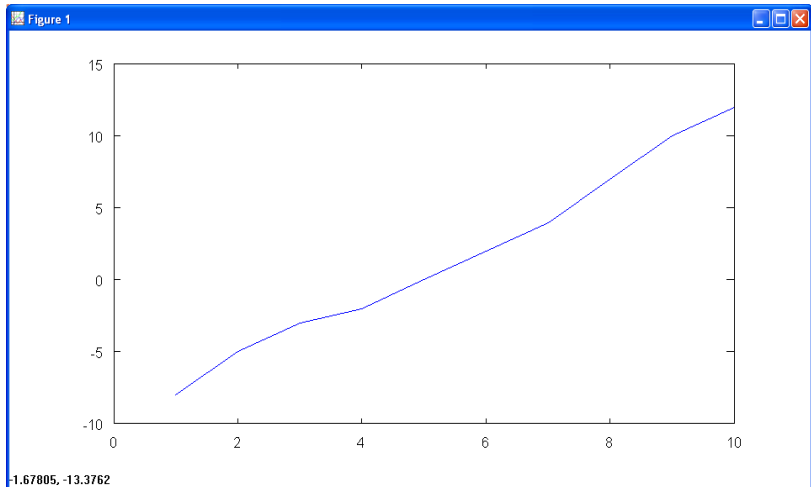


Figura 4.1

Exemplu:



Trasarea graficului funcției $y = f(x) = 3 \cdot x^2 - 4 \cdot x + 2$ pentru valorile lui $x \in [-4, 3]$, parcurs cu pasul de 1.

1. Vom calcula pe rând valorile funcției $f(x)$ pentru fiecare valoare a lui x din intervalul considerat (tabelul 4.2):

Tabelul 4.2. Valorile funcției $y = f(x) = 3 \cdot x^2 - 4 \cdot x + 2$ pentru $x \in [-4, 3]$ parcurs cu pasul de 1

x	f(x)	y = f(x)	Coordonatele punctelor (x,y)
-4	$3 \cdot (-4)^2 - 4 \cdot (-4) + 2$	66	(-4, 66)
-3	$3 \cdot (-3)^2 - 4 \cdot (-3) + 2$	41	(-3, 41)
-2	$3 \cdot (-2)^2 - 4 \cdot (-2) + 2$	22	(-2, 22)
-1	$3 \cdot (-1)^2 - 4 \cdot (-1) + 2$	9	(-1, 9)
0	$3 \cdot (0)^2 - 4 \cdot (0) + 2$	2	(0, 2)
1	$3 \cdot (1)^2 - 4 \cdot (1) + 2$	1	(1, 1)
2	$3 \cdot (2)^2 - 4 \cdot (2) + 2$	6	(2, 6)
3	$3 \cdot (3)^2 - 4 \cdot (3) + 2$	17	(3, 17)

2. Vom construi doi vectori \mathbf{x} și \mathbf{y} în care vom plasa valorile coordonatelor punctelor:

$\mathbf{x} = [-4 \ -3 \ -2 \ -1 \ 0 \ 1 \ 2 \ 3]$,

respectiv $\mathbf{y} = [66 \ 41 \ 22 \ 9 \ 2 \ 1 \ 6 \ 17]$;

3. Vom trasa graficul funcției utilizând comanda **plot** cu forma **plot(x,y)** (figura 4.2);

--> $\mathbf{x} = [-4 \ -3 \ -2 \ -1 \ 0 \ 1 \ 2 \ 3]$, $\mathbf{y} = [66 \ 41 \ 22 \ 9 \ 2 \ 1 \ 6 \ 17]$

$\mathbf{x} =$

-4 -3 -2 -1 0 1 2 3

$\mathbf{y} =$

66 41 22 9 2 1 6 17

--> **plot(x,y)**

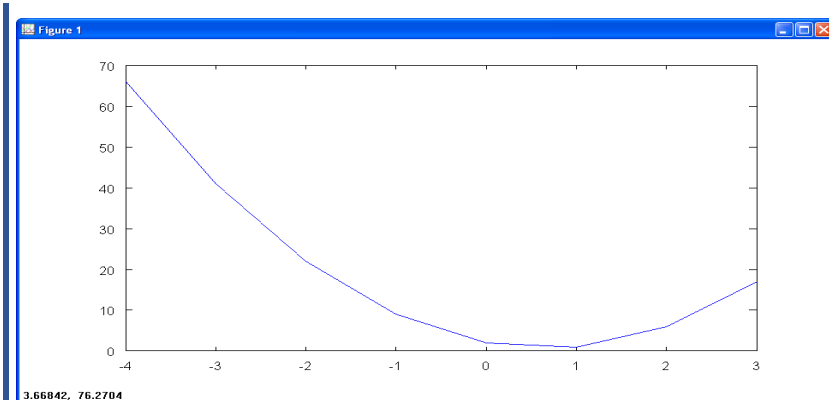


Figura 4.2.

Exemplu:



Trasarea graficului funcției $y = f(x) = \sin(x)$, unde $x \in [-\pi, \pi]$

```
--> x=-pi:2*pi/20:pi, plot(x,sin(x))
```

```
x =
```

```
-3.1416
```

```
-2.8274
```

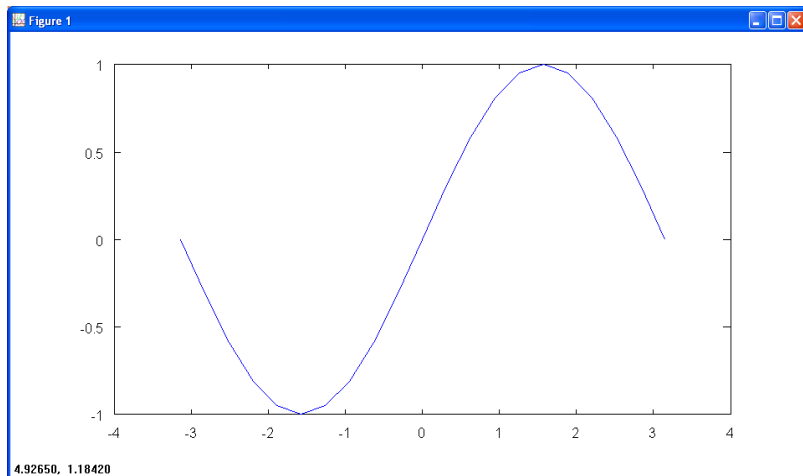


Figura 4.3

```
-2.5133
```

```
-2.1991
```

```
-1.8850
```

```
-1.5708
```

```
-1.2566
```

```
-0.9425
```

```
-0.6283
-0.3142 0.0000 0.3142 0.6283 0.9425 1.2566
1.5708
1.8850 2.1991 2.5133 2.8274
3.1416
```



Exemplu:

Vom trasa graficul funcției $\sin(x)$, modificând modalitatea de trasare a graficului, astfel vom alege următoarele caracteristici: tipul liniei: linie punctată (.), culoarea liniei: roșu (r), tipul marculatorului pentru puncte: cerc (o) (Figura 4.4).

Sintaxa comenzii este:

```
--> plot(x,sin(x),'.ro')
```

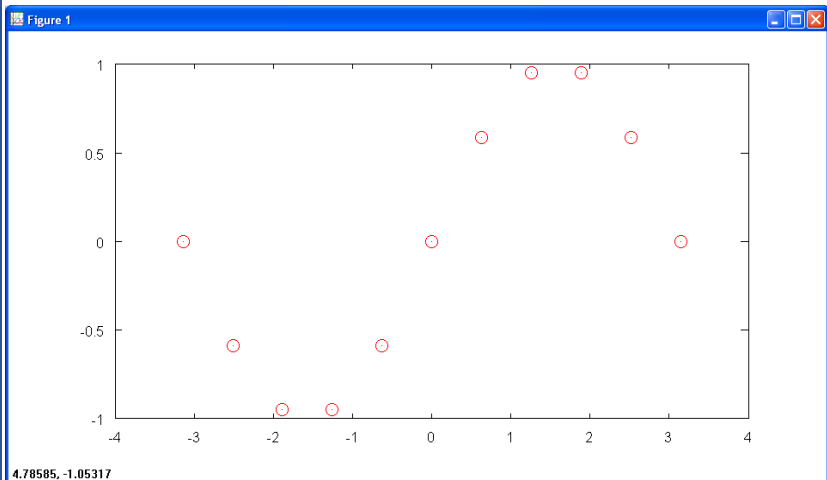


Figura 4.4.



Exemplu:

Vom trasa graficele funcțiilor $\sin(x)$ și $\cos(x)$ pe același grafic. Graficele vor avea următoarele caracteristici: graficul funcției sinus se va trasa cu linie continuă (-), de culoare albastră (b) iar punctele vor fi evidențiate prin cruciulițe (x); graficul funcției cosinus va fi trasat cu linie punctată (.), de culoare roșie (r), iar punctele vor fi marcate prin cerculețe (o) (Figura 4.5).

Sintaxa comenzii **plot** este:

```
--> plot(x, sin(x), '-bx', x, cos(x), 'r.o')
```

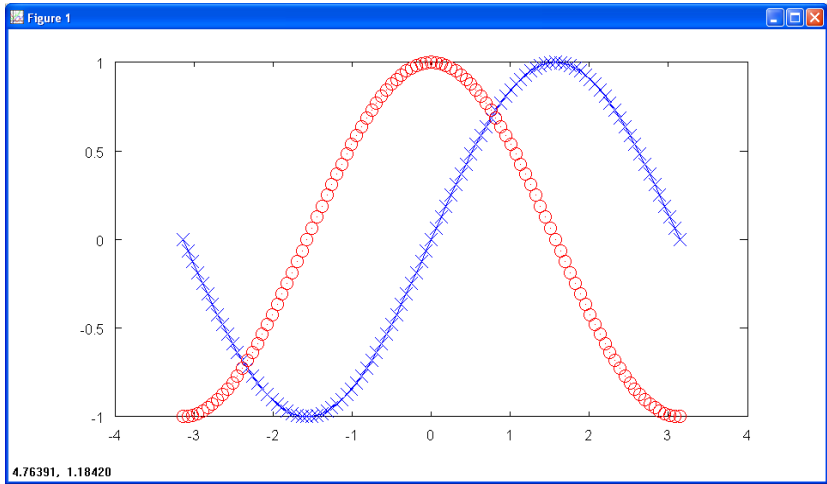


Figura 4.5.

Exemplu:



În figura 4.6 este ilustrat graficul unei curbe numită cardioidă, a cărei ecuație este: $r = a (1 + \cos(t))$, unde $t \in [0, 2\pi]$.

```
>t=0:pi/50:2*pi;a=2;r=a*(1+cos(t));polar(t,r)
```

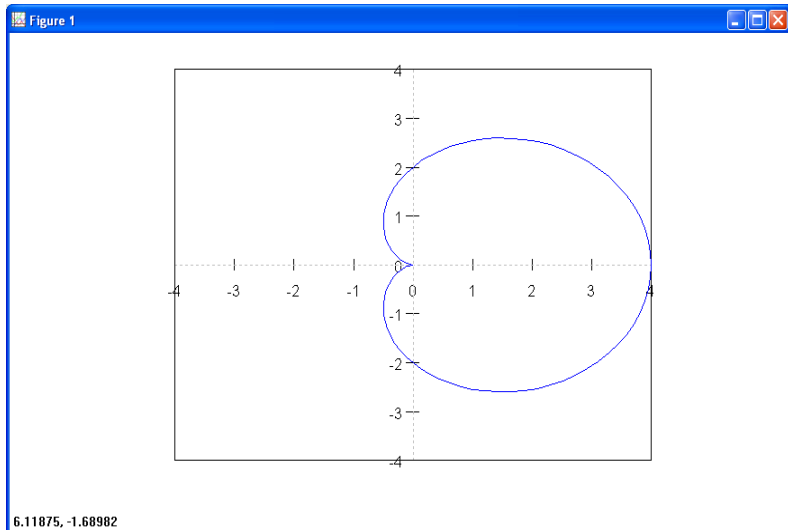


Figura 4.6

Comanda **plot** acceptă și argumente date sub formă matricială. În practică se pot întâlni patru situații posibile:

- ambele argumente (**x** și **y**) sunt vectori, prin urmare se trasează graficul cu valorile primului vector (**x**) pe abscisă și cu cele ale celui de-al doilea vector (**y**) pe ordonată;
- **x** este un vector de dimensiune **m** și **y** este o matrice **m x n**, comanda `plot(x,y)` suprapune graficele obținute din **x** și fiecare coloană (sau rând) din matricea **y**;
- **x** este o matrice **m x n** și **y** este un vector de dimensiune **m**, comanda `plot(x,y)` suprapune graficele obținute din fiecare coloană (sau rând) din matricea **x** și vectorul **y**;
- ambele argumente sunt matrice, situație în care sunt suprapuse graficele obținute din coloanele primului argument versus coloanele celui de al doilea argument. În acest caz cele două matrice trebuie să aibă același număr de linii și coloane.

În cazul în care argumentele nu sunt reale atunci părțile imaginare se ignoră.

Când funcția **plot** are un singur argument și acesta este un număr complex atunci sintaxa `plot(y)` este echivalentă cu `plot(real(y), imag(y))`.

Funcția **plot** mai poate primi și alte argumente, astfel: **LineWidth** (implicit 0,5 puncte) stabilește grosimea liniei, **MarkerSize** (implicit 6 puncte, unde un punct este 1/72 inch) stabilește dimensiunea marker-ului, **MarkerEdgeColor** stabilește culoarea liniilor de contur al marker-ului, **MarkerFaceColor** stabilește culoarea interiorului marker-ului.

Exemple:

```
plot(x,y,'m--^','LineWidth',3,'MarkerSize',5)
plot(x,y,'--rs','MarkerSize',20,'MarkerFaceColor','g')
```

Funcția **polar**: se utilizează pentru reprezentarea 2D a unei funcții în coordonate polare. Sintaxa comenzii este asemănătoare cu cea a funcției **plot**, astfel (figura 4.6):

```
polar(t,r,'şir')
```

unde **t** reprezintă unghiul polar iar **r** reprezintă raza polară, **şir** are aceeași semnificație ca și la funcția **plot**.

Funcția **bar**: se utilizează pentru realizarea graficelor 2D cu bare. Sintaxa funcției este asemănătoare cu cea a funcției `plot`, astfel:

```
bar(x,y,'şir')
```

unde argumentele **x**, **y** și **şir** au aceeași semnificație ca și în cazul funcției **plot**.

Exemplu:



Să considerăm șirurile de valori $x = [-4 -3 -2 -1 0 1 2 3]$, respectiv $y = [66 41 22 9 2 1 6 17]$;

Reprezentarea cu ajutorul funcției **bar** este următoarea (figura 4.7):

```
--> x = [ -4  -3  -2  -1  0  1  2  3 ];  
--> y = [ 66  41  22  9  2  1  6  17 ];  
--> bar(x , y)
```

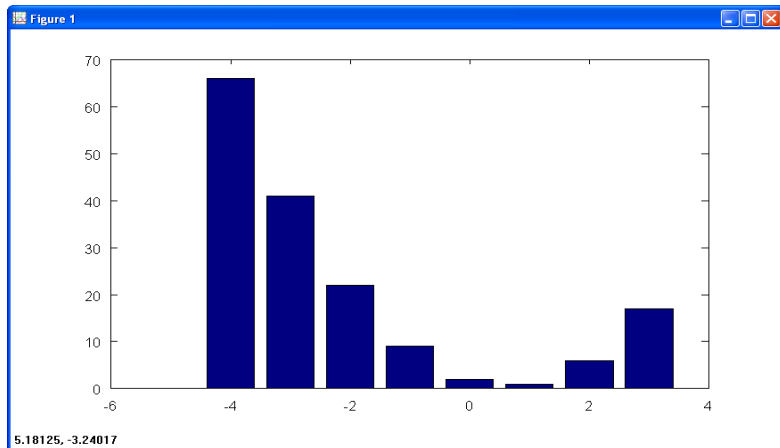


Figura 4.7

Funcțiile **grid**, **legend**, **title**, **xlabel**, **ylabel**, **axis**, **text**: acțiunea fiecărei funcții este explicată în tabelul 4.1.

Exemplu:



Reprezentarea graficului funcției sinus pe un anumit interval (figura 4.8). Efectuați pe rând fiecare din comenzile de mai jos și vizualizați după fiecare comandă graficul. Observați ce se întâmplă.

```
--> x=0:0.1:2*pi; y=sin(x); plot(x,y,'b-.')  
--> grid on;axis ([0 2*pi -1 1]);  
--> title(' Functia sinus')  
--> xlabel('x'),ylabel('sinus(x)');  
--> legend('sin')  
--> text(pi,sin(pi),' \leftarrow  
sin(\pi)', 'FontSize',18)
```

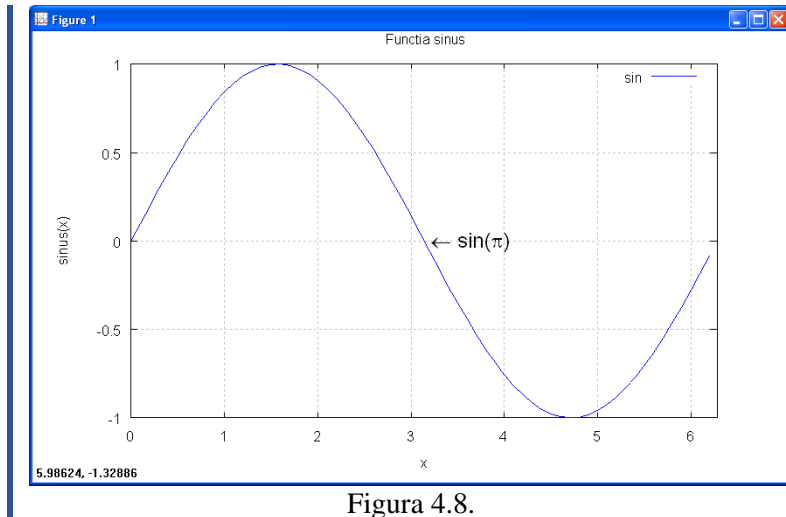


Figura 4.8.

Funcțiile **semilogx**, **semilogy**, **loglog**: aceste funcții se utilizează pentru reprezentări în coordonate logaritmice sau semilogaritmice. Funcția loglog scalează ambele axe utilizând logaritmul în baza 10, iar funcțiile semilogx și semilogy scalează logaritmice numai axa x, respectiv y în timp ce cealaltă axă este scalată liniar.

Exemplu:



Se reprezintă funcția $y = f(x) = 10^x$, pentru $x \in [0,10]$, pasul de 0.1 (Figura 4.9).

--> `x=0:0.1:10,y=10.^x,semilogy(x,y)`

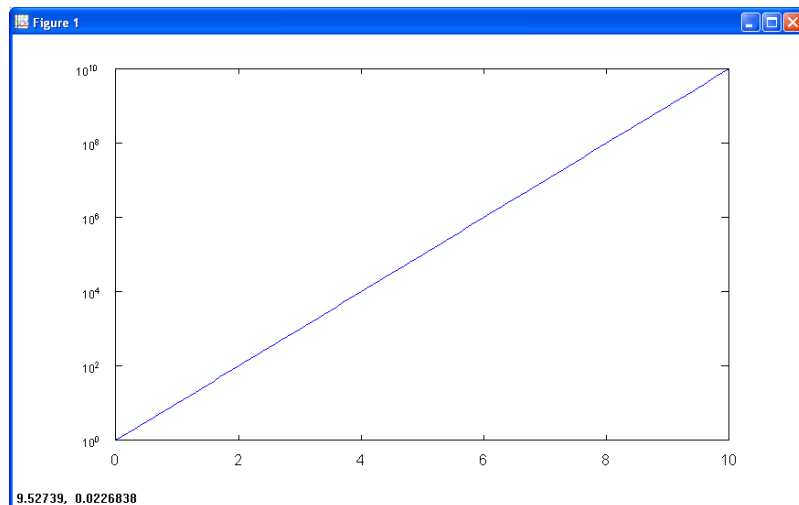


Figura 4.9.

Funcția **subplot**: se utilizează pentru plasarea mai multor grafice în aceeași fereastră. Sintaxa comenzii este: **subplot(mnp)**, sau **subplot(m,n,p)**. Fereastra se împarte într-o matrice $m \times n$ de grafice fiecare având propriile axe. Comanda de desenare se va aplica celei de a p-a regiune, modalitatea de parcurgere fiind de-a lungul primei linii, apoi de-a lungul celei de a doua linii, ș.a.m.d.

Exemplu:



Vom reprezenta într-o singură fereastră funcțiile sinus, cosinus și tangentă (figura 4.10).

```
--> x=0:0.1:2*pi;subplot(2,2,1),plot(x,sin(x))
--> subplot(2,2,2),plot(x,cos(x))
--> subplot(2,2,3:4),plot(x,tan(x))
```

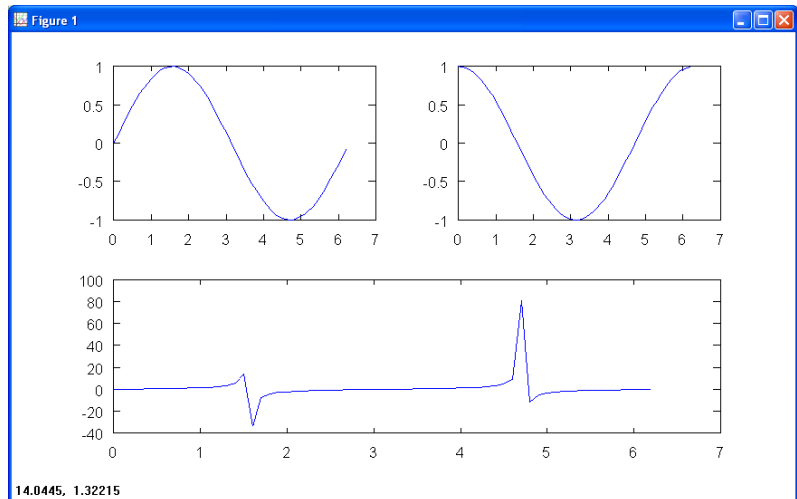


Figura 4.10

4.2. Grafica 3D

Funcțiile utilizate pentru reprezentarea tridimensională în **FreeMat** / **OCTAVE** / **MATLAB** sunt (tabelul 4.3):

Tabelul 4.3. Funcțiile OCTAVE pentru generarea graficelor 3D

Denumire:	Descriere:
plot3	Generează graficul 3D al unei funcții, analog cu funcția plot
bar3	Generează un grafic cu bare 3D
cylinder	Generează un cilindru
sphere	Desenează o sferă
elipsoid	Desenează un elipsoid de rotație
surf	Desenează suprafețe 3D colorate

surf	Desenează suprafețe cu contur
surfnorm	Desenează suprafețe 3D și normalele la ele
contour	Desenează proiecțiile intersecțiilor cu plane paralele cu xOy, în planul xOy
meshgrid	Generează o diviziune 2D
mesh	Realizează graficul unei suprafețe în 3D
meshc	Realizează graficul unei suprafețe cu contur în 3D
meshz	Graficul unei suprafețe cu plan de referință la cota z (pedestal)
waterfall	Produce un mesh de tip "waterfall"
comet3	Grafic 3D animat (efect de cometă)
quiver	Desenează vectorii tangenți la o suprafață

Funcția **plot3**: se utilizează pentru reprezentarea tridimensională a unor linii care trec prin puncte care pot fi definite de vectori sau de funcții.

Forma generală a funcției este:

plot3(x,y,z,'șir')

unde: **x**, **y**, **z** reprezintă vectori (matrici) de aceeași dimensiune ce conțin coordonatele **x**, **y**, **z** ale punctelor prin care trece linia, iar **șir** are aceeași semnificație ca și în cazul funcției plot.

În cazul în care argumentele *x,y,z* ale funcției sunt matrici, atunci se obține reprezentarea suprapusă a liniilor generate de tripletele de coloane $[X(:j), Y(:j), Z(:j)]$.

Exemplu:



Reprezentarea tridimensională a unei spirale (figura 4.11).

```
--> t=0:pi/100:10*pi; plot3(sin(t),cos(t),t,'-b')
```

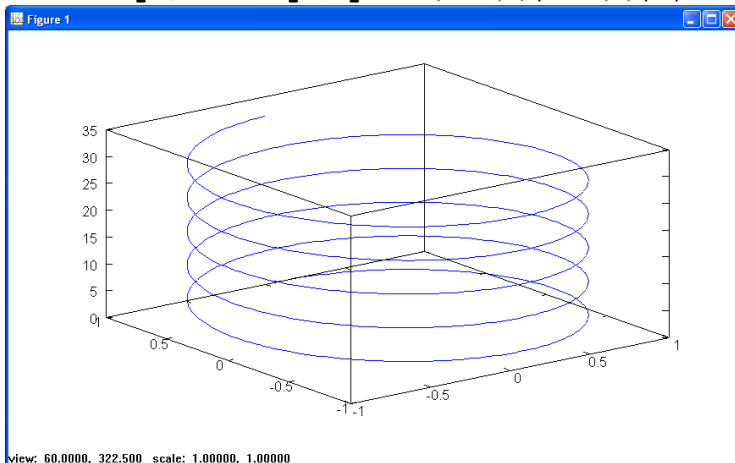


Figura 4.11. Reprezentarea tridimensională a unei spirale

Reprezentarea 3D a suprafețelor se poate face fie sub forma unei rețele (mesh) fie sub forma suprafețelor netede.

Funcțiile **mesh**, **meshc**, **meshz**: se utilizează pentru reprezentarea suprafețelor sub forma unor rețele. Sintaxele funcțiilor sunt:

mesh (X , Y , Z)	mesh (Z)	meshc (...)
mesh (X , Y , Z , C)	mesh (Z , C)	meshz (...)

În cazul cel mai general funcția **mesh** are patru matrice ca parametri de intrare și reprezintă grafic suprafața $Z(X, Y)$, respectiv matricea culorilor C .

De cele mai multe ori X și Y sunt vectori. Pentru ca reprezentarea să fie corectă, aceștia trebuie să fie ordonați crescător și cu pas constant.

În cazul în care argumentele X și Y lipsesc, reprezentarea se realizează pe baza indicilor matricei Z .

Dacă matricea C lipsește, se consideră $C = Z$, adică culoarea este proporțională cu înălțimea suprafeței.

Poziția din care este observată suprafața reprezentată grafic poate fi controlată prin funcția **view**.

Modalitatea de gradare a axelor este stabilită de intervalele X , Y și Z sau de setarea curentă a axelor, cu ajutorul funcțiilor **axis** sau **axes**.

Stabilirea culorilor utilizate se realizează cu ajutorul funcției *caxis*. Valorile definite ale scalei de culori sunt utilizate ca indici ai unui tabel de culori.

Funcția **meshc** are formă asemănătoare cu funcția **mesh** și permite reprezentarea 3D a suprafețelor cărora li se asociază liniile de contur, trasate ca proiecții în planul bazei.

Funcția **meshz** se utilizează asemănător cu **mesh** și **meshc** și permite reprezentarea 3D a suprafețelor, însă se trasează suplimentar un plan de referință la valoarea minimă (pedestal).

Exemplu:



Reprez. funcției $z = x \cdot e^{(-x^2 - y^2)}$ pt. domeniul $D = [-2, 2] \times [-2, 2]$.

```
--> [X, Y]=meshgrid(-2:0.2:2, -2:0.2:2);
--> Z=X.*exp(-X.^2-Y.^2)
--> subplot(221);mesh(X, Y, Z);
--> subplot(222);meshc(X, Y, Z)
--> subplot(223);meshz(X, Y, Z)
```

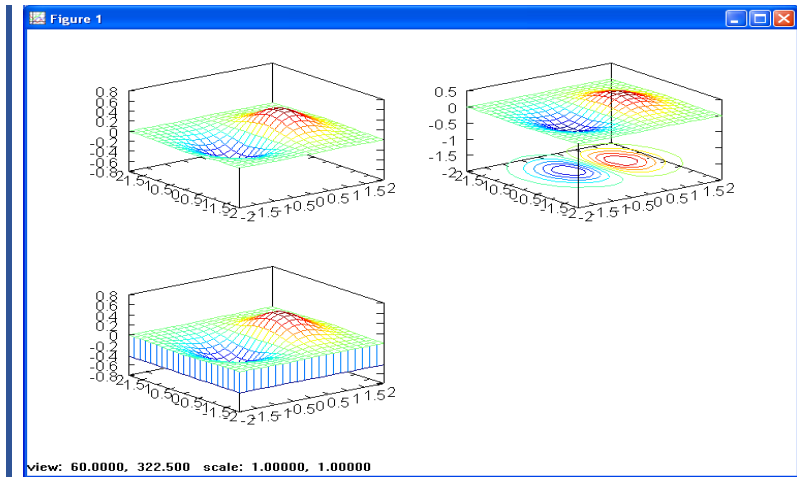


Figura 4.12. Exemple de utilizare a funcțiilor mesh, meshc și meshz

Funcția **cylinder**: se utilizează pentru generarea unui cilindru. Forma generală a funcției este:

$$[\mathbf{x}, \mathbf{y}, \mathbf{z}] = \text{cylinder}(r, n)$$

unde r reprezintă un vector cu două elemente prin care se specifică raza cercului la partea inferioară și la cea superioară, iar n reprezintă numărul de puncte prin care se aproximează cercul bazei.

Funcția **sphere**: se utilizează pentru reprezentarea obiectelor de tip sferă. Forma generală a funcției este:

$$[\mathbf{x}, \mathbf{y}, \mathbf{z}] = \text{sphere}(n)$$

unde: $[x, y, z]$ reprezintă trei matrice $(n+1) \times (n+1)$ care conțin coordonatele sferei, iar n reprezintă numărul de puncte, implicit $n = 20$. Sferile pot fi reprezentate cu ajutorul funcțiilor **surf(x,y,z)** sau **mesh(x,y,z)**.

Exemplu:



Să se reprezinte un cilindru cu razele $r_{inf} = 1$, $r_{sup} = 0.5$, înălțimea $h = 3$ și numărul de puncte prin care se aproximează cercul bazei $n = 35$ (figura 4.13).

```
--> rinf=1;rsup=0.5;h=3;n=35;
--> [xc,yc,zc]=cylinder([rinf rsup],n);
--> zc = zc*h; surface(xc, yc, zc);
```

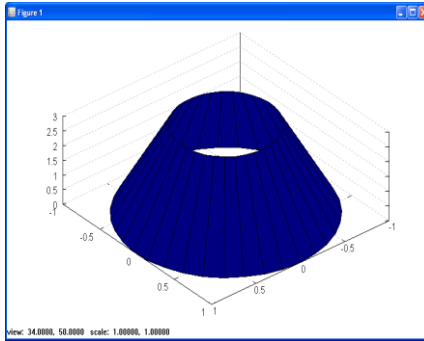


Figura 4.13. Reprezentarea unui cilindru

Exemplu:



Reprezentarea unei sfere:

--> `[x,y,z] = sphere(20); mesh(x,y,z); grid`

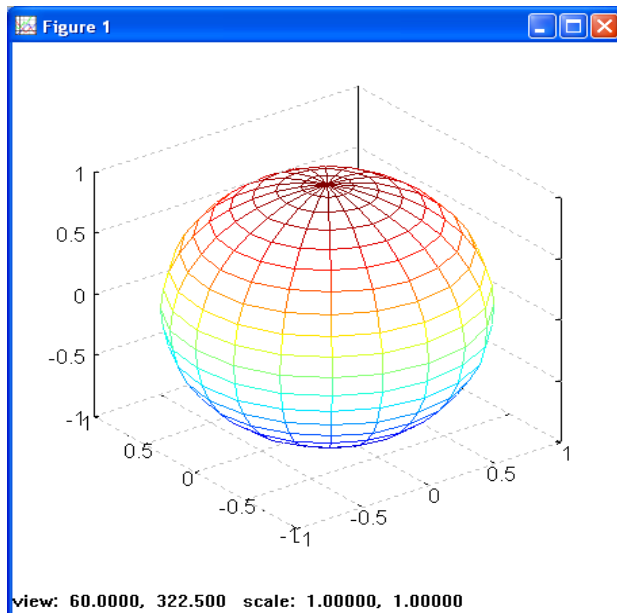


Fig. 4.14. Reprezentarea unei sfere

5. Declarații și variabile. Formatul datelor de ieșire. Operatori. Instrucțiuni de introducere și extragere de date. Fișiere script. Fișiere funcție. Instrucțiuni de decizie.

Introducere:



Mediile de programare **FreeMat** / **OCTAVE** / **MATLAB** reprezintă instrumente foarte utile celor care doresc să realizeze aplicații specifice pentru rezolvarea unor probleme cu caracter tehnic.

Prin multitudinea de funcții puse la dispoziția utilizatorilor și programatorilor pot fi realizate relativ ușor aplicații diverse care să ușureze munca inginerilor din domeniul tehnic și nu numai.

În cadrul acestui curs, sunt prezentate noțiunile de bază referitoare la programarea în **FreeMat** / **OCTAVE** / **MATLAB**.

În prima parte a cursului sunt prezentate noțiuni legate de forma generală a unui program **FreeMat** / **OCTAVE** / **MATLAB**, noțiuni despre tipurile de date, declarații de variabile, operatori precum și despre instrucțiuni de intrare / ieșire. În continuare sunt prezentate noțiuni despre fișierele de tip script și funcție. Noțiunile sunt însoțite de exemple.

În ultima parte a cursului sunt prezentate instrucțiunile de decizie ale limbajelor **FreeMat** / **OCTAVE** / **MATLAB**, atât aspectele teoretice cât și o serie de exemple.

Scopul acestui curs este acela de a prezenta noțiunile de bază privind programarea în mediile de programare **FreeMat** / **OCTAVE** / **MATLAB**. De asemenea, prin parcurgerea acestui curs studenții vor învăța să scrie programe simple în care vor folosi apeluri de funcții uzuale și instrucțiuni de decizie.

Obiective:



După parcurgerea acestui material, studenții vor ști să utilizeze mediile de programare **OCTAVE** / **MATLAB** pentru:

- realizarea fișierelor script cu ajutorul cărora să poată să scrie programe cu diferite grade de complexitate;

- crearea fișierelor de tip funcție cu ajutorul cărora pot realiza funcții pentru rezolvarea unor probleme punctuale. Prin crearea de funcții noi se îmbogățește mediul de programare și astfel pot fi create noi biblioteci de funcții;
- rezolvarea unor probleme simple de programare (în care se efectuează numai calcule) sau a unor probleme în care se utilizează și instrucțiunile de decizie;

De asemenea, studenții vor cunoaște:

- tipurile de date ale limbajelor FreeMat / OCTAVE / MATLAB;
- modalitatea de afișare a datelor;
- operatorii limbajelor FreeMat / OCTAVE / MATLAB;
- instrucțiuni de introducere / extragere de date;

5.1. Etapele de rezolvare a unei probleme

A. Definirea problemei: necesită cunoașterea problemei în cele mai mici detalii. În această etapă se determină care sunt informațiile sau datele de intrare, modelul matematic care trebuie aplicat și care sunt informațiile sau datele de ieșire pe care le va furniza programul.

B. Întocmirea algoritmului de calcul: În această etapă se analizează problema de rezolvat, se determină algoritmi cunoscuți care pot fi utilizați sau se indică pas cu pas rezolvarea.

C. Scrierea programului: se editează programul (sub forma unui fișier text), avantajul scrierii în **FreeMat / OCTAVE / MATLAB** derivă din faptul că acesta este adaptat algoritmilor numerici astfel că de cele mai multe ori fiecare pas al algoritmului se transformă într-o comandă **FreeMat / OCTAVE / MATLAB**.

D. Testarea și validarea rezultatelor: Se rulează programul cu o serie de date de intrare și se verifică rezultatele intermediare și cele finale dacă acestea se cunosc.

5.2. Declarații și variabile

FreeMat / OCTAVE / MATLAB este un limbaj de expresii. Expresiile tipărite de utilizator sunt interpretate și evaluate. Instrucțiunile **FreeMat / OCTAVE / MATLAB** sunt, în general, de forma:

variabila = expresie

sau mai simplu:

expresie

Expresiile pot fi compuse din constante, operatori sau alte caractere speciale, funcții și variabile. Evaluarea unei expresii are ca rezultat o matrice, care este afișată pe ecran și atribuită unei variabile. Dacă numele variabilei și semnul egal nu sunt precizate, rezultatul se atribuie automat unei variabile cu numele “ans”.

Orice instrucțiune se termină cu **ENTER**. Dacă după o instrucțiune sau expresie se plasează caracterul punct-virgulă (;), instrucțiunea este executată însă tipărirea rezultatului pe ecran este suprimată.

În situația în care expresia nu încapă pe o singură linie, se utilizează semnul “ ... ” (trei puncte), urmat de **ENTER** și se continuă editarea expresiei pe linia următoare.

Numele de variabile sau funcții trebuie să aibă ca prim caracter o literă care poate fi urmată de alte litere sau cifre sau caracterul underline (“_”). Limbajele **FreeMat** / **OCTAVE** / **MATLAB** fac deosebirea între litere mari și mici, cu ajutorul funcției **casesen (on / off)** se realizează setarea în modul sensibil / nesensibil.

Tipul fundamental de dată în **FreeMat** / **OCTAVE** / **MATLAB** este **matricea**, elementele acesteia fiind de același tip.

În figura 5.1 sunt prezentate tipurile de date din **OCTAVE** / **MATLAB**.

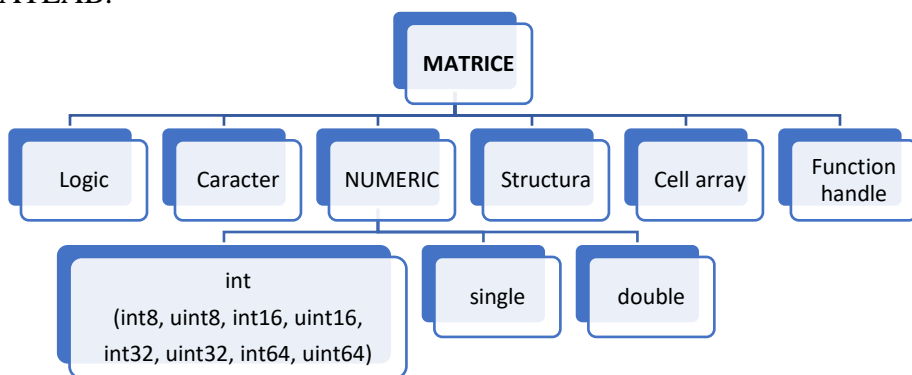


Figura 5.1. Tipurile de date din **OCTAVE** / **MATLAB**

Tipurile **structură** sau **celule de tablouri** pot să conțină date de tipuri diferite. Cu ajutorul claselor (uses classes și Java classes) pot fi create tipuri noi de date (date utilizator).

Tipurile fundamentale de date sunt prezentate în tabelul 5.1.

Tabelul 5.1.

Tip de dată:	Descriere:	Biți	Domeniu de valori:
int8	Întreg cu semn pe 8 biți	8	[-128, 127]
uint8	Întreg fără semn pe 8 biți	8	[0, 255]
int16	Întreg cu semn pe 16 biți	16	[-32,768, 32,767]

uint16	Întreg fără semn p 16 biți	16	[0, 65,535]
int32	Întreg cu semn pe 32 biți	32	[-2,147,483,648,2,147,483,647]
uint32	Întreg fără semn p 32 biți	32	[0, 4,294,967,295]
int64	Întreg cu semn pe 64 biți	64	[-9,223,372,036,854,775,808, 9,223,372,036,854,775,807]
uint64	Întreg fără semn p 64 biți	64	[0, 18,446,744,073,709,551,615]
Single	Real în simplă precizie.	32	[-3.40282e+038, 3.40282e+038]
Double	Real în dublă precizie	64	[-1.79769e+308, 1.79769e+308]
Logical	Tablou de valori logice	1	1 (adevărat) sau 0 (fals)
Char	Tablou de tip caracter sau șir de caractere	16	
function handle	Pointer la o funcție.		
Structure	Structura este o colecție de câmpuri, fiecare având un tip de dată.		
cell array	Tablou de celule indexate, fiecare celulă putând memora un tablou de orice tip sau mărime.		

5.3. Formatul datelor de ieșire

Cu ajutorul funcției **format** se poate preciza formatul de afișare a datelor de ieșire. Prin intermediul acestei funcții se specifică modul de afișare a datelor și nu precizia acestora.

Forma generală a funcției este:

format opțiune

unde **opțiune** poate fi una dintre variantele prezentate în tabelul de mai jos (tabelul 5.2):

Tabelul 5.2

Opțiune:	Mod de afișare:	Exemplu:
short	reprezentare în virgulă fixă cu 5 cifre	--> a=123456789.12345678901234 5678 --> format short --> a a = 1.2346e+008
long	reprezentare în virgulă fixă cu 15 cifre	--> format long --> a a = 123456789.123457
short e	reprezentare în virgulă mobilă cu 5 cifre	--> format short e --> a a = 1.2346e+008

long e	reprezentare în virgulă mobilă cu 15 cifre	--> format long e --> a a = 1.23456789123457e+008
short g	cea mai bună dintre reprezentările în virgulă fixă și mobilă cu 5 cifre	--> format short g --> a a = 1.2346e+008
long g	cea mai bună dintre reprezentările în virgulă fixă și mobilă cu 15 cifre	--> format long g --> a a = 123456789.123457
hex	reprezentare în hexazecimal	--> format hex --> a a = 419d6f34547e6b75
+, -, _	sunt utilizate pentru elemente pozitive, negative și nule; partea imaginară este ignorată	
bank	reprezentare pentru simboluri valutare (dolari și cenți)	--> format bank --> a a = 123456789.12
rat	aproximare printr-o fracție a unui întreg	--> format rat --> 22.33 ans = 2233/100

5.4. Operatori

În **FreeMat / OCTAVE / MATLAB** există trei tipuri de operatori: **aritmetici, relaționali și logici**.

Operatori aritmetici sunt: + (adunare), - (scădere), * (înmulțire), / (împărțire la dreapta), \ (împărțire la stânga), ^ (ridicare la putere), ' (transpusa complex conjugată), () (operatorul de specificare a ordinii de evaluare).

Operatori relaționali : == (egal); ~= (diferit) ; < (mai mic); > (mai mare) ; <= (mai mic sau egal) ; >= (mai mare sau egal);

Operatori logici : & (și); | (sau); ~ (not); xor (or). Tabelul operatorilor logici poate fi sintetizat astfel:

Prioritatea operatorilor specifică ordinea de evaluare a unei expresii. Prioritatea operatorilor este următoarea:

1. Parantezele () – au prioritate maximă,
2. Transpusa ('), ridicare la putere (^) pt. scalari și matrice
3. plus unar (+), minus unar (-), negare logică (~)

4. înmulțire (*), împărțire la dreapta (/), împărțire la stânga (\) pt. scalari și matrice
5. adunare (+), scădere (-)
6. operatorul două puncte (:)
7. operatori relaționali și de egalitate: mai mic (<), mai mic sau egal (<=), mai mare (>), mai mare sau egal (>=), egalitate (==), diferit (~=)
8. AND (&)
9. OR(|)

5.5. Instrucțiuni de introducere și extragere de date

Funcții pentru citirea de la tastatură / afișarea pe ecran

Funcția **input** – Se utilizează pentru citirea datelor de la tastatură. Forma generală a funcției este :

```
n = input('sir_de_caractere',tip)
```

Funcția afișează pe ecran șirul de caractere 'sir_de_caractere' și atribuie valoarea răspunsului oferit de utilizator variabilei **n**, iar **tip** reprezintă tipul conversiei aplicate la citire.

Exemplu:



Citirea unei valori numerice

```
>>> n=input('Introdu numarul de elemente:')
Introdu numarul de elemente: 2
n = 2
```

Funcția **disp** – realizează afișarea datelor sau a șirurilor de caractere.

Exemplu:



Afișarea valorii citire anterior:

```
>>> disp('Numarul de elemente este:'),
disp(n)
Numarul de elemente este:
2
```

Funcții pentru scrierea / citirea fișierelor

Funcția **save** – permite scrierea datelor într-un fișier.

Exemplu:



Scrierea unei matrice într-un fișier. Vizualizați conținutul fișierului creat.

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

```

4 5 6
7 8 9
>>> save fis_mat.txt A

```

Funcția **load** – se realizează pentru citirea datelor din fișiere (citirea matricelor din fișiere binare sau a valorilor din fișiere text)

Exemplu:



Citirea unei matrice dintr-un fișier (matricea din fișierul creat anterior):

```

>>> BB=load('fis_mat.txt')
BB =
1 2 3
4 5 6
7 8 9

```

Funcții pentru crearea(deschiderea) / închiderea fișierelor

Funcția **fopen** – se utilizează pentru crearea unui fișier, forma generală a funcției fiind:

fopen(nume , mod)

unde: - **nume** reprezintă numele fișierului;

- **mod** reprezintă modalitatea de creare / deschidere a fișierului, astfel: 'r' – deschide un fișier pt. citire, 'w' – deschide un fișier pentru scriere, 'a' – deschide sau crează un fișier pentru adăugare la sfârșitul acestuia, 'r+' deschide un fișier pt. citire / scriere, 'w+' deschide un fișier pt. citire / scriere conținutul fișierului înainte de deschidere se va pierde.

Funcția **fclose** – se utilizează pentru închiderea unui fișier.

Funcții de intrare / ieșire în stilul limbajului C

Funcția **puts(string)** – afișează pe ecran un șir de caractere.

Funcția **fputs(fid,string)** – scrie un șir de caractere într-un fișier.

Funcția **fgets(fid,len)** - citește un șir de caractere cu lungimea len, iar dacă aceasta nu este precizată citirea se face până se întâlnește caracterul « new-line » sau EOF (end-of-file).

Funcția **printf(sir_format,...)** se utilizează pentru afișarea datelor sub controlul formatului de afișare.

Funcția **fprintf(fid,sir_format,...)** este asemănătoare cu funcția printf, cu deosebirea că scrierea se face în fișierul definit de identificatorul **fid**.

Funcția **sprintf(sir_format,...)** este o altă formă a funcției printf, deosebirea constând în faptul că afișarea se realizează sub forma unui șir de caractere.

Argumentul **sir_format** poate conține: șiruri de caractere, secvențe ESCAPE și descriptori de format.

Șirurile de caractere sunt afișate așa cum sunt scrise, fără să sufere modificări.

Secvențele ESCAPE încep cu caracterul backslash ('****') și determină execuția unei acțiuni, astfel: **\n** – determină saltul la linia următoare, **\t** – reprezintă tab-ul orizontal, **\r** – determină saltul la începutul rândului, etc.

Descriptorii de format au forma generală:

%[flags][lățime][.precizie] tip

Parantezele drepte nu trebuie scrise, ele sugerează faptul că elementele cuprinse între [] sunt opționale, elementele componente având următoarea semnificație:

% : reprezintă caracterul prin care se specifică descriptorul de format, nu trebuie să lipsească;

tip : reprezintă o literă care specifică tipul conversiei aplicate datelor la afișare, nu poate lipsi din specificatorul de format. Pentru **tip** se folosesc: **c** – afișarea unui caracter; **s** – afișarea unui șir de caractere; **i**, **d** – afișarea unui număr întreg (în baza zece) cu semn; **u** – afișarea unui număr întreg (în baza zece) fără semn; **f** – afișarea unui număr real cu semn - notație zecimală; **e**, **E** – afișarea unui număr real cu semn (notație exponențială); **g** – afișarea unui număr real (cea mai scurtă reprezentare între f și e); **x** – afișarea unui număr hexazecimal întreg fără semn; **o** – afișarea unui număr octal întreg fără semn; **flags** : Poate fi unul din caracterele: + (plus): se afișează semnul datei (plus sau minus); - (minus): data este aliniată la stânga în câmpul de scriere, dacă lipsește alinierea se face la dreapta; _ (spațiu): valorile pozitive sunt scrise fără semn lăsându-se un spațiu în fața valorii afișate, iar în cazul celor negative data se scrie cu semn (semnul minus) fără a se lăsa nici un spațiu; # (diez): determină folosirea formei alternative de scriere, pentru datele octale și hexazecimale se scrie un zero nesemnificativ, respectiv 0x sau 0X în cazul datelor hexazecimale.

lățime : constă dintr-un număr întreg care specifică dimensiunea minimă a câmpului în care va fi afișată data. Dacă sunt necesare mai multe poziții, data va fi afișată pe câte poziții trebuie. Dacă data are mai puține caractere ea va fi aliniată la dreapta în câmpul de afișare. Spațiile rămase libere se completează cu spațiu. Dacă se plasează un 0 (zero) înaintea cifrei ce reprezintă lățimea, spațiile rămase libere vor fi completate cu cifra 0.

.precizie : indică precizia de scriere a datei, astfel: pentru %e, %E, %f indică numărul de zecimale; pentru %d, %i, %u indică numărul minim de cifre pe care va fi reprezentată data de tip întreg. pentru %s indică numărul maxim de caractere care se afișează; pentru %c nu are efect.

Observații: Dacă în specificatorul de format după caracterul % se scrie un alt caracter care nu este admis atunci primul caracter % va fi ignorat, deci nu va fi luat în considerare ca un specificator de format iar caracterele care urmează după el vor fi afișate pe ecran așa cum apar ele. Între specificatorii de format și parametri trebuie să existe o concordanță de număr, ordine și tip.

Exemplu:



Într-un fișier scrieți următoarele instrucțiuni :

```
m = 250; n = 23.4567;
printf('\n Format zecimal : %8d \t
%8i', m, m);
printf('\n Format octal : %o', m);
printf('\n Format octal : =%6o=', m);
printf('\n Format hexazecimal : %8x', m);
printf('\n Format hexazecimal : %%4x \t
%5x', m);
printf('\n Format real : %f', n);
printf('\n Format real : %7.3f', n);
printf('\n Format real : %09.4f', n);
printf('\n Format real : %.0f', n);
printf('\n Format real : %.2f', n);
printf('\n Numarul real este : %e ', -n);
printf('\n Numarul real este : %g ', -n);
```

După lansarea în execuție a programului, rezultatul va fi prezentat sub forma:

```
Format zecimal : 250 250
Format octal : 372
Format octal : = 372=
Format hexazecimal : fa
Format hexazecimal : %4x fa
Format real : 23.456700
Format real : 23.457
Format real : 0023.4567
Format real : 23
Format real : 23.46
Numarul real este : -2.345670e+001
Numarul real este : -23.4567
```

5.6. Fișiere script

Cele trei medii de programare permit două modalități de lucru: în modul *linie de comandă*: în această variantă fiecare linie este prelucrată imediat după ce a fost scrisă, rezultatele fiind afișate imediat sau *utilizând programe scrise în fișiere*.

Fișierele care conțin instrucțiuni **FreeMat / OCTAVE / MATLAB** poartă numele de fișiere **M** sau **M-files**, denumirea provenind de la extensia acestora, adică extensia **.m**. În **FreeMat / OCTAVE / MATLAB** un fișier **M** poate fi de două tipuri: *script* sau *funcție*. Cu ajutorul fișierelor **M** pot fi create noi funcții care le pot completa pe cele existente.

Fișierele script: sunt fișiere de tip text în care sunt înșiruite secvențe de comenzi **FreeMat / OCTAVE / MATLAB**. Prin apelarea numelui fișierului, se realizează execuția secvențelor de comenzi **FreeMat / OCTAVE / MATLAB** conținute în fișier. Variabilele utilizate în cadrul fișierelor script rămân în spațiul de lucru (workspace) după execuția completă a acestuia.

Aceste fișiere nu permit integrarea în programe mari, realizate pe principiul modularizării, ele pot fi utilizate însă, acolo unde este nevoie de un număr relativ mare de instrucțiuni iar introducerea acestora de la tastatură ar fi greoaie.

5.7. Exemple de fișiere script OCTAVE / MATLAB

Exemplu:



Inversarea valorilor a două variabile

Se deschide un fișier **m** nou. Se scrie următoarea secvență de program:

```
a = input(' Introduceți valoarea lui a:')
b = input(' Introduceți valoarea lui b:')
c = a; a = b; b = c;
printf("\n a = %f \t b = %f",a,b)
```

Se lansează în execuție programul (comanda Run - Run sau tastați F5). Execuția programului va fi următoarea:

```
--> Introduceți valoarea lui a: 12
a = 12
Introduceți valoarea lui b: 33
b = 33
a = 33.000000 b = 12.000000
```

Exemplu:



Să se scrie un program pentru calculul diametrului, ariei și volumului unei sfere pentru care se cunoaște raza R . Fișierul m va avea următorul conținut:

```
% Program pentru calculul diametrului, ariei si  
% volumului unei sfere de raza R
```

```
R = input('Introduceti raza sferei:');  
D=2*R;S=4*PI*R*R;V=4*PI*R*R*R/3;  
printf("\n Diametrul D = %7.3f",D);  
printf("\n Aria S = %7.3f",S);  
printf("\n Volumul V = %7.3f",V);
```

Rezultatul execuției programului fiind:

```
-->Introduceti raza sferei: 10.25  
R = 10.250  
Diametrul D = 20.500  
Aria S = 1320.253  
Volumul V = 4510.865
```

Exemplu:



Program pentru calculul valorilor funcției $y = f(x) = x^3 - 4x^2 + 6x - 3$, unde $x \in [a, b]$, parcurs cu pasul h . Să se reprezinte grafic funcția în intervalul considerat.

```
a = input('Introduceti valoarea lui a: a =');  
b = input('Introduceti valoarea lui b: b =');  
h = input('Introduceti valoarea lui h: h =');  
x = a:h:b;  
y = polyval([1 -4 6 -3],x);  
plot(x,y);  
grid on;  
title('Graficul functiei y = x^3-4x^2+6x-3')
```

Rularea programului:

```
--> Introduceti valoarea lui a: a = -1  
--> Introduceti valoarea lui b: b = 1  
--> Introduceti valoarea lui h: h = 0.05
```

5.8. Fișiere funcție

Fișierele funcție sunt utilizate pentru dezvoltarea aplicațiilor mai complexe, sau pentru “îmbogățirea” mediului de programare cu funcții. Forma generală a unei funcții este:

```
function [param_ieșire] = nume_funcție(param_intrare)
```

unde: **function** reprezintă un cuvânt cheie care specifică faptul că fișierul este de tip funcție; **nume_funcție** reprezintă numele funcției, adică numele sub care se salvează fișierul; **param_ieșire** reprezintă o listă de parametri, separați prin virgulă (,) și cuprinși între paranteze drepte ([]); **param_intrare** reprezintă o listă de argumente, separate prin virgulă și cuprinse între paranteze rotunde. Din punct de vedere al existenței valorilor returnate sau a argumentelor se pot întâlni 4 categorii de funcții:

A. Funcții care nu returnează valori și nu au parametrii:

Exemplu:



Funcție care afișează un text predefinit :

```
function antet()  
printf("\\n UNIVERSITATEA TEHNICA din CLUJ-  
NAPOCA");  
printf("\\n Facultatea CONSTRUCTII DE  
MASINI");  
endfunction
```

Apelul funcției și rezultatul apelării fiind:

```
--> antet  
UNIVERSITATEA TEHNICA din CLUJ-NAPOCA  
Facultatea CONSTRUCTII DE MASINI
```

B. Funcții care nu returnează valori dar au parametrii:

Exemplu:



Funcție care afișează un număr impus de steluțe

```
function stea(n)  
    for i=1:n  
        printf("*");  
    endfor  
endfunction
```

Apelul funcției și rezultatul fiind:

```
--> stea(5)  
*****
```


C. Funcții care returnează valori dar nu au parametri

Exemplu:



Funcție care citește un număr strict pozitiv

```
function n = strpoz()  
do  
n = input('Introduceti o valoare strict  
pozitiva: n = ');  
until (n>0)  
endfunction
```

Apelul funcției și rezultatul fiind:

```
--> a=strpoz  
Introduceti o valoare strict pozitiva: n = -2  
Introduceti o valoare strict pozitiva: n = 0  
Introduceti o valoare strict pozitiva: n = 3  
a = 3
```

D. Funcții care returnează valori și au parametri

Exemplu:



Funcție care citește un număr mai mare decât o valoare impusă

```
function n = maimare(m)  
do  
n = input(' Introduceti o valoare: n = ');  
until (n>m)  
endfunction
```

Apelul funcției și rezultatul fiind:

```
--> c = maimare(3)  
Introduceti o valoare: n = 1  
Introduceti o valoare: n = 4  
c = 4
```

5.9. Exemple de fișiere funcție în OCTAVE / MATLAB

Exemplu:



Să se calculeze valorile funcției $y = f(x) = 3x^2 - 5x + 7$, $x \in [a, b]$, parcurs cu pasul h .

Se va construi o funcție pentru calculul valorii lui y și va fi salvată într-un fișier fl.m:

```
function [y] = f1(x)  
    y = 3*x.^2 - 5*x + 7;  
endfunction
```

Programul care utilizează funcția f1

```
% Program care utilizeaza functia f1
a = input(' Introduceți valoarea lui a: a=');
b = input(' Introduceți valoarea lui b: b=');
h = input(' Introduceți valoarea lui h: h=');
x = a:h:b;
y = f1(x)
plot(x,y)
grid on
```

Apelul funcției și rezultatul fiind:

```
--> Introduceți valoarea lui a: a = -10
Introduceți valoarea lui b: b = 10
Introduceți valoarea lui h: h = 1
y =
Columns 1 through 13:
357 295 239 189 145 107 75 49 29 15 7 5 9
Columns 14 through 21:
19 35 57 85 119 159 205 257
```

5.10. Instrucțiuni de decizie.

5.10.1. Instrucțiunea if ... end

Corespunde deciziei cu o ramură din schema logică (figura 5.2) și are forma generală:

```
if (expresie)
    instrucțiune
end
```

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucțiune** poate fi o instrucțiune simplă sau o instrucțiune compusă.

Mod de lucru: se evaluează **expresie**, dacă are valoare diferită de zero (adevărată) se execută **instrucțiune**, iar dacă are valoare nulă (falsă) nu se execută **instrucțiune**.

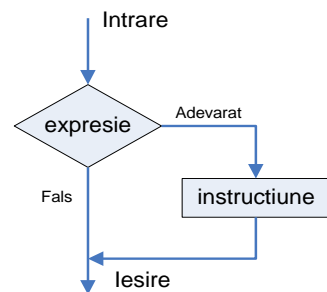


Figura 5.2.

5.10.2. Instrucțiunea if... else ... end

Corespunde deciziei cu două ramuri din schema logică (figura 5.3) și are forma generală:

```
if (expresie)
    instrucțiune_1
else
    instrucțiune_2
end
```

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucțiune_1** și **instrucțiune_2** pot fi instrucțiuni simple sau compuse.

Mod de lucru: se evaluează **expresie**; dacă are valoare diferită de zero (adevărată) se execută **instrucțiune_1**, iar dacă are valoare nulă (falsă) se execută **instrucțiune_2**.

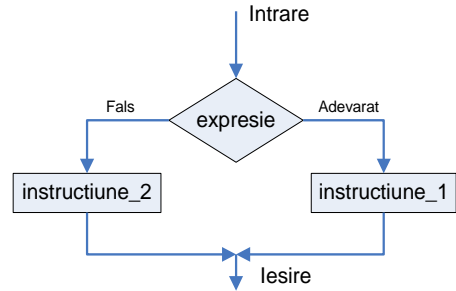


Figura 5.3.

Exemplu:

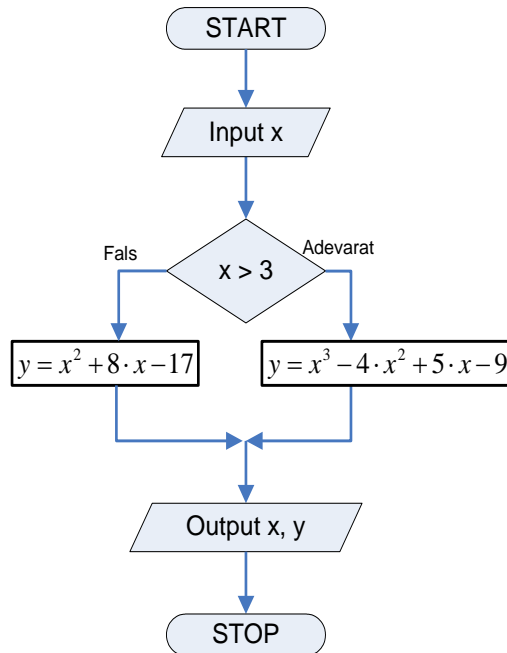


Calculul valorilor funcției

$$y = \begin{cases} x^3 - 4x^2 + 5x - 9 & x > 3 \\ x^2 + 8x - 17 & x \leq 3 \end{cases}$$

pentru un x introdus de utilizator de la tastatură.

Schema logică :



Programul în **OCTAVE / MATLAB** :

```
x = input("\n Introdu x, x = ")
if ( x > 3 )
y = x*x*x - 4*x*x + 5*x - 9;
else
y = x*x + 8*x - 17;
endif
printf("\nx=%7.3f \t y=%7.3f",x,y);
```

Rularea programului :

```
--> Introduceti x, x = 0
x = 0.000 y = -17.000
--> Introduceti x, x = 4
x = 4.000 y = 11.000
```

5.10.3. Instrucțiuni if... else suprapuse

Au forma generală :

```

if (expresie_1)
    if (expresie_2)
        instrucțiune_1
    else
        instrucțiune_2
    end
else
    instrucțiune_3
end
    
```

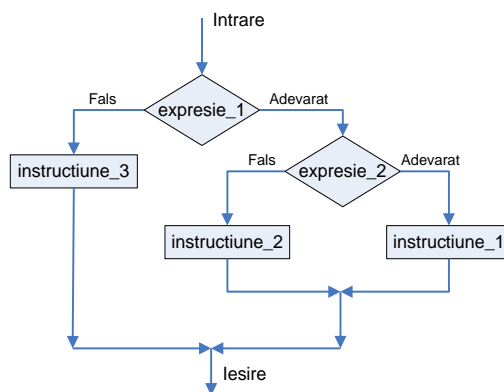


Figura 5.4.

Mod de lucru: se evaluează **expresie_1**; dacă are valoare diferită de zero (adevărată) se trece la executarea celei de a doua instrucțiuni **if**, iar dacă este falsă se execută **instrucțiune_3**. Dacă **expresie_2** este adevărată se execută **instrucțiune_1** iar dacă este falsă se execută **instrucțiune_2**.

Exemplu:



Să se scrie un program pentru calculul valorilor funcției

$$y = \begin{cases} x^4 - 6x^2 + 4x - 11 & x > 2 \\ x^2 - 5x - 7 & -1 \leq x \leq 2 \\ x^3 + 13 \cdot x^2 - 4 & x < -1 \end{cases}$$

pentru un **x** introdus de utilizator de la tastatură.

Program în **OCTAVE / MATLAB**:

```

x = input("\n Introduceți x, x = ");
if ( x <= 2 )
    if ( x >= -1) y = x * x - 5 * x - 7;
    else y = x ^ 3 + 13 * x ^ 2 - 4;
    end
else y = x ^ 4 - 6 * x ^ 2 + 4 * x - 11;
endif
printf("\n x = %7.3f \t y = %7.3f",x,y)
    
```

Rularea programului:

```

--> Introduceți x, x = 0
x = 0.000 y = -7.000
--> Introduceți x, x = 1
x = 1.000 y = -11.000
--> Introduceți x, x = 3
x = 3.000 y = 28.000
    
```

5.10.4. Instrucțiunea de decizie multiplă elseif

O selecție multiplă se poate realiza cu mai multe instrucțiuni decizionale în cascadă (figura 5.5). În cazul general în care există „n+1” alternative posibile selectate pe baza a „n” condiții, se recomandă folosirea acestei structuri.

Forma generală este:

```

...
if( expresie_1 )
    instrucțiune_1
elseif( expresie_2 )
    instrucțiune_2
...
elseif( expresie_n )
    instrucțiune_n
else
    instrucțiune
end
...

```

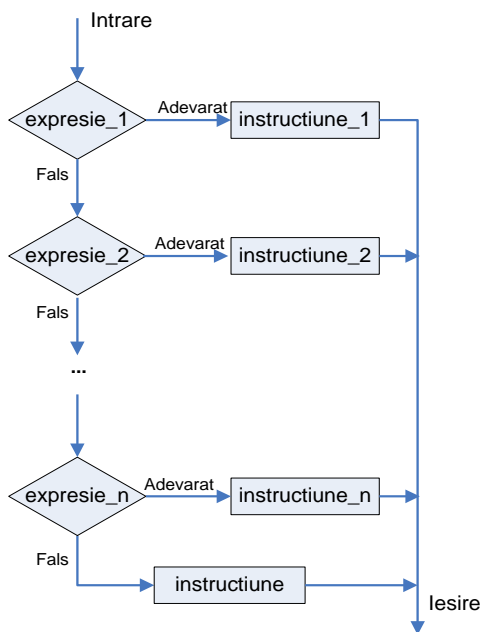


Figura 5.5.

Mod de lucru: Expresiile se evaluează în ordinea în care sunt scrise. Dacă se întâlnește o expresie adevărată, atunci se execută instrucțiunea care îi este asociată și astfel se încheie întregul lanț. Instrucțiunea de după ultimul **else** se execută atunci când nici una dintre expresii nu a fost adevărată.

Exemplu:



Calculul valorilor funcției:
$$y = \begin{cases} x^2 + 2x + 3 & \text{daca } x \geq 1 \\ x - 8 & \text{daca } -1 < x < 1 \\ x^3 - 6x + 8 & \text{daca } x \leq -1 \end{cases}$$

pentru un x introdus de utilizator de la tastatură.

```

a = input(' Limita stanga a interv.: a = ');
b = input(' Limita dreapta a interv.: b = ');
h = input(' Pasul de parc. a interv.: h = ');
for x = a:h:b;
    if(x>=1) y = x^2 + 2*x + 3;
    elseif (x>-1) y = x-8;
    else y = x^3 - 6*x + 8; end
    printf("\n x = %7.3f \t y = %7.3f",x,y);
endfor

```

Rularea programului:

```
>>> Limita stanga a interv.: a = -2  
Limita dreapta a interv.: b = 2  
Pasul de parcurg. a interv.: h = 1  
x = -2.000 y = 12.000  
x = -1.000 y = 13.000  
x = 0.000 y = -8.000  
x = 1.000 y = 6.000  
x = 2.000 y = 11.000
```

6. Instrucțiunea de ciclare for. Cicluri for suprapuse.

Introducere:



În acest curs sunt prezentate noțiunile teoretice și exemple practice de utilizare a instrucțiunilor de ciclare for în limbajele de programare OCTAVE / MATLAB.

Instrucțiunea de ciclare **for** se utilizează pentru programarea ciclurilor cu variabilă conducătoare (contor), atunci când se cunosc numărul de repetări.

Exemplele prezentate în lucrare ilustrează modul de utilizare a instrucțiunilor for pentru calculul unor sume, pentru calculul valorilor unei funcții într-un interval cunoscut, parcurs cu un pas cunoscut. De asemenea, sunt prezentate exemple de utilizare a instrucțiunii for pentru realizarea unor operații cu șiruri.

În corpul unei instrucțiuni for poate să fie plasată altă instrucțiune for, situație în care cele două cicluri for sunt suprapuse sau imbricate.

În lucrare sunt prezentate exemple de utilizare a ciclurilor for imbricate pentru calculul unor suma sau pentru parcurgerea matricelor (patratice sau dreptunghiulare).

Obiective:



Scopul acestui curs este acela de a prezenta studenților aspectele teoretice și practice legate de instrucțiunea for. Prin exemplele prezentate se oferă informații privind utilizarea instrucțiunilor de ciclare for (inclusiv cicluri for imbricate) pentru calculul valorilor unor funcții, pentru calculul unor sume sau pentru operații cu șiruri și matrice.

6.1. Instrucțiunea de ciclare *for*.

Instrucțiunea de ciclare **for** este folosită pentru programarea ciclurilor care au o variabilă conducătoare (contor), și are forma generală:

.....

```
for index = expresie  
    instrucțiune  
end
```

.....

unde: **index** reprezintă numele contorului, **expresie** este o matrice, un vector sau un scalar, **instrucțiune** poate fi o instrucțiune simplă sau compusă.

În general, expresie este de cele mai multe ori de forma:

```
index = inițial:pas:final
```

unde: - **inițial** reprezintă prima valoare a indexului;

- **final** reprezintă ultima valoare a indexului;

- **pas** reprezintă pasul indexului (dacă nu este precizat se consideră 1);

Observații:

- contorul (indexul) trebuie să fie o variabilă întreagă sau reală;
- dacă **expresie** are o valoare nulă, **instrucțiune** nu se execută;
- dacă **expresie** este un scalar, **instrucțiune** se execută o singură dată, cu **indexul** dat de valoarea scalarului;
- dacă **expresie** este un vector linie, **instrucțiune** se execută de un număr egal cu numărul de elemente ale vectorului, **indexul** primind pe rând valorile elementelor vectorului;
- dacă **expresie** este o matrice, **indexul** va avea la fiecare iterație valorile conținute în următoarea coloană a matricei;
- la terminarea ciclului **indexul** are ultima valoare utilizată;
- când indicele este definit sub forma:

```
indice = initial : pas : final
```

numărul de repetări este dat de relația:

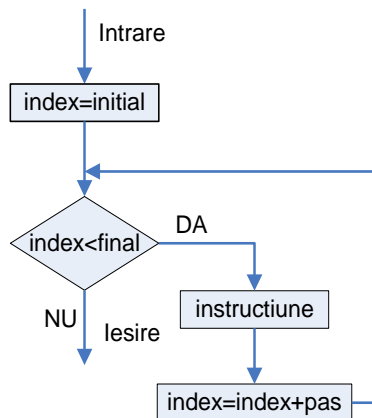


Figura 6.1.

$$n = \left[\frac{\text{final} - \text{initial}}{\text{pas}} \right] + 1,$$

notația [] semnificând partea întreagă.

6.2. Exemple de programe cu ciclul for

Exemplu:



Calculul sumei primelor 10 numere naturale, $S = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$.

Se inițializează suma cu 0.

Se folosește o variabilă ajutătoare "i", aceasta va lua valori de la 1 la 10, suma se calculează astfel:

```
% Calculul sumei primelor 10 numere naturale
s=0;
for i=1:10
    s=s+i
end
printf("\n Suma este: S = %d",s)
```

Exemplu:



Calculul valorilor funcției $y = 3,4 \cdot x^3 + 5,25 \cdot x^2 + 3 \cdot x + 1,6$ în intervalul [a,b] parcurs cu pasul h.

Programul în FreeMat / OCTAVE / MATLAB:

```
a = input(' Introdu limita din stanga: a=' );
b = input(' Introdu limita din dreapta: b=' );
h = input(' Introdu pasul: h = ');
for x=a:h:b
    y=3.4*x^3 + 5.25*x^2 + 3*x + 1.6;
    printf("\n x = %7.3f \t y = %7.3f",x,y);
end
```

Rularea programului:

```
--> Introdu limita din stanga: a = -3
    Introdu limita din dreapta: b = 4
    Introdu pasul: h = 1
x = -3.000 y = -51.950
x = -2.000 y = -10.600
x = -1.000 y = 0.450
x = 0.000 y = 1.600
x = 1.000 y = 13.250
x = 2.000 y = 55.800
x = 3.000 y = 149.650
x = 4.000 y = 315.200
```

Exemplu:



Calculați valorile funcției $y = \begin{cases} x^4 - 6x^2 + 4x - 11 & x > 2 \\ x^2 - 5x - 7 & x \leq 2 \end{cases}$

pentru $x \in [-a, a]$ parcurs cu pasul h .

Rezultatele se vor scrie într-un fișier text.

Programul în **FreeMat / OCTAVE / MATLAB**:

```
fid = fopen("fis_ext.txt","w+");
a = input("Introduceți valoarea lui a: a= ");
h = input("Introduceți valoarea lui h: h= ");
for x=-a:h:a
    if (x>2) y = x^4 - 6*x^2 + 4*x - 11;
    else y = x^2 - 5*x - 7;
    end
fprintf(fid,'x=%7.3f \t y=%7.3f \n',x,y);
end
fclose(fid);
```

Rularea programului:

```
-->Introduceți valoarea lui a: a= 3
Introduceți valoarea lui h: h= 1
```

Conținutul fișierului:

```
x = -3.000    y = 17.000
x = -2.000    y = 7.000
x = -1.000    y = -1.000
x = 0.000     y = -7.000
x = 1.000     y = -11.000
x = 2.000     y = -13.000
x = 3.000     y = 28.000
```

Exemplu:



Să se scrie un program pentru calculul sumei: $S = (1) + (1 + 2) + (1 + 2 + 3) + (1 + 2 + 3 + 4) + \dots$
luând în considerare n termeni (maxim 25 de termeni).

Pentru calculul sumei se va utiliza o formulă de recurență:
 $t[i] = t[i-1] + i$. În această situație se inițializează suma cu valoarea primului element, calculul sumei realizându-se pornind de la al doilea element.

Varianta I. Rezolvare utilizând funcțiile sum și cumsum:

```
--> n=6;i=1:n; s=sum(cumsum(i))  
s = 56
```

Varianta II. Rezolvare cu ajutorul instrucțiunii de ciclare **for**:

Program în **FreeMat / OCTAVE / MATLAB**:

```
n = input(' Introdu numarul de termeni:');  
t=1;s=t;  
for i=2:n  
    t=t+i; s=s+t;  
end  
printf("\n S (%2d) = %d",n,s);
```

Rularea programului:

```
--> Introdu numarul de termeni: 6  
S ( 6) = 56
```

Exemplu:



Suma elementelor unui șir de numere reale

```
v=[-1 6 2 -3 8 4 5 -6 1 0 -2]  
s=0;  
for i=1:length(v)  
    s=s+v(i);  
end  
printf("\n Suma elementelor este: %d",s)
```

Rularea programului:

```
--> v = -1 6 2 -3 8 4 5 -6 1 0 -2  
Suma elementelor este: 14
```

Exemplu:



Produsul elementelor strict pozitive ale unui șir de numere reale:

```
v=[-1 6 2 -3 8 4 5 -6 1 0 -2]  
p=1;  
for i=1:length(v)  
    if v(i)>0  
        p=p*v(i);  
    end  
end  
printf("\n Prod. elem. este strict pozitive  
este: P = %d",p)
```

```
--> v = -1 6 2 -3 8 4 5 -6 1 0 -2  
Prod. elem. este strict pozitive este: P = 1920
```

6.3. Cicluri for suprapuse

Dacă instrucțiunile care compun corpul unui ciclu **for** (numit *ciclu for exterior*) conțin alt ciclu **for** (numit *ciclu for interior*), cele două cicluri **for** se numesc *suprapuse*, *imbricate* sau *incluse*.

Forma generală a ciclurilor suprapuse este următoarea:

```

...
for index1 = expresie1
    ...
    for index2=expresie2
        instrucțiune2
    end
    ...
end
...

```

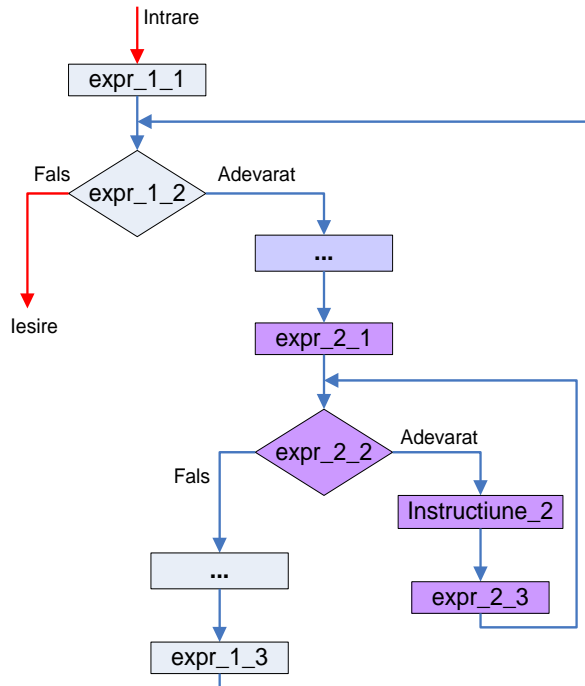


Figura 6.2

Observație : *Ciclu for interior* trebuie să fie cuprins complet în corpul *cicluului for exterior*.

De obicei formele $\text{index1} = \text{expresie1}$, respectiv $\text{index2} = \text{expresie2}$ sunt de forma:

variabila = valoare_inițială : pas : valoare_finală

O modalitate de reprezentare în schema logică este ilustrată în figura alăturată (figura 6.2), unde:

- **expr_1_1**: reprezintă expresia prin care lui index1 i se atribuie valoarea inițială a sa;
- **expr_1_2**: reprezintă de obicei o expresie logică și reprezintă expresia prin care se verifică dacă index1 a depășit valoarea finală a acestuia;
- **expr_1_3**: reprezintă expresia prin care valoarea lui index1 se modifică cu pasul de variație al acestuia.
- **expr_2_1**: reprezintă expresia prin care lui index2 i se atribuie valoarea inițială a sa;
- **expr_2_2**: reprezintă de obicei o expresie logică și reprezintă expresia prin care se verifică dacă index2 a depășit valoarea finală a acestuia;

- **expr_2_3**: reprezintă expresia prin care valoarea lui index2 se modifică cu pasul de variație al acestuia.

Mod de lucru (conform cu reprezentarea din schema logică din figura 6.2):

1. se evaluează **expr_1_1**, adică index1 primește valoarea inițială a sa;
2. se verifică **expr_1_2**, dacă valoarea de adevăr a acesteia este Fals se părăsește ciclul, în caz contrar se continuă cu pasul următor;
3. se execută (dacă există) instrucțiunile de pe ramura Adevărat până la întâlnirea expresiei **expr_2_1**;
4. se evaluează **expr_2_1**;
5. se verifică **expr_2_2**, dacă valoarea de adevăr a acesteia este Fals se părăsește corpul ciclului și se continuă cu instrucțiunile următoare din schema logică (pasul 8) , iar dacă este Adevărat se continuă cu pasul următor;
6. se execută **instrucțiune_2**;
7. se evaluează **expr_2_3**, după care se revine la pasul 5;
8. se evaluează **expr_1_3** după care se revine la pasul 2;

Observație: Toate instrucțiunile aferente ciclului condus de variabila index2 sunt incluse în corpul ciclului condus de variabila index1.

6.4. Exemple de utilizare a ciclurilor for suprapuse

Exemplu:



Afișarea tuturor posibilităților de a obține 100 de lei utilizând monede de 10, 20 și 50 de lei.

Considerăm următoarele variabile **a** – numărul de monede de 10 lei; **b** – numărul de monede de 20 lei, **c** – numărul de monede de 50 lei.

Intervalul de valori pentru fiecare variabilă este: $a = 0 \div 10$; $b = 0 \div 5$; $c = 0 \div 2$;

```
% Afișarea tuturor posibilitatilor de a
obține 100 de lei % utilizand monede de 10,
20 si 50 de lei
for a=0:10
    for b=0:5
        for c=0:2
            if (a*10+b*20+c*50==100)
printf("\n %2d*10+%2d*20+%2d*50=100", a,b,c) ;
            end
        end
    end
end
end
```

Exemplu:



Suma elem. strict pozitive ale unei matrice dreptunghiulare $A(m \times n)$.

Program în **FreeMat / OCTAVE / MATLAB**:

```
A=[1 2 -3 4 -5 6;0 1 -2 4 3 5;2 1 -3 -4 6 5;5
3 ...
-1 0 2 4;1 -2 -1 0 3 2]
S=0;
for i=1:5
    for j=1:6
        if(A(i,j)>0)
            S = S + A(i,j);
        end
    end
end
printf("\n Suma elem. str. poz. este: S=%5d",S)
```

Exemplu:



Calculul sumei $S = (1) + (1 + 2) + (1 + 2 + 3) + (1 + 2 + 3 + 4) + \dots$ luând în considerare n termeni (maxim 25 de termeni). Pentru calculul sumei se va utiliza un ciclu (condus de variabila i), fiecare termen adăugându-se pe rând sumei s . Pentru calculul fiecărui termen se utilizează un al doilea ciclu (condus de variabila j), fiecare termen fiind la rândul lui o sumă exprimată

prin relația: $T_i = \sum_{j=1}^i j$.

Program în **FreeMat / OCTAVE / MATLAB**:

```
n = input('Introduceti numarul de termeni:');
S=0;
for i=1:n
    st=0;
    for j=1:i
        st = st + j;
    end
    S = S + st;
endfor
printf("\n S(%2d) = %6d",n,S)

-->Introduceti numarul de termeni: 6
S(6) = 56
```

7. Instrucțiunea de ciclare **while...end**. Instrucțiunile **break**, **continue**, **error**, **switch**.

Introducere:



Prezentul curs continuă cursurile precedente și oferă noțiuni teoretice și exemple privind utilizarea instrucțiunilor de ciclare.

În prima parte este prezentată instrucțiunea de ciclare **while .. end**. Sunt prezentate o serie de noțiuni teoretice (forma generală, mod de lucru, corespondent în schema logică) precum și o serie de exemple de utilizare a acestei instrucțiuni. În partea a doua a cursului sunt prezentate instrucțiunile **break**, **continue** și **error**. Ca și la instrucțiunile precedente și în cazul acestor instrucțiuni, cursul oferă informații teoretice și exemple practice de utilizare a acestor instrucțiuni.

În ultima parte este prezentată instrucțiunea de selecție multiplă **switch**, fiind trecute în revistă atât aspectele teoretice (formă generală, mod de lucru) cât și un exemplu sugestiv de utilizare a acestei instrucțiuni.

Obiective:



După parcurgerea acestui material, studenții vor fi și să utilizeze în cadrul programelor scrise în **FreeMat / OCTAVE / MATLAB**:

- instrucțiunea de ciclare **while ... end**;
- instrucțiunile **break**, **continue**, **error**;
- instrucțiunea de selecție multiplă **switch**;

7.1. Instrucțiunea de ciclare **while ... end**

Se utilizează pentru programarea unor operații de un număr de ori necunoscut. Este o instrucțiune de ciclare *condiționată anterior*.

Forma generală este următoarea:

```
...  
while (expresie)  
    instrucțiune;  
end  
...
```

Interpretarea sintaxei este următoarea:

“atât timp cât ...”.

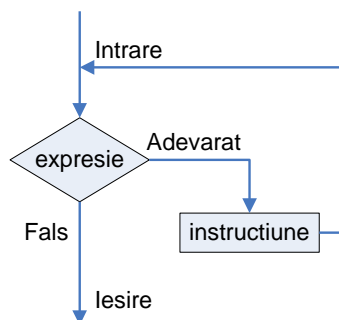


Figura 7.1.

Funcționare : Se evaluează expresia **expresie**, dacă aceasta este nenulă se execută **instrucțiune**, după care se revine la evaluarea expresiei. În cazul în care expresie este nulă se trece la instrucțiunea următoare din program.

Exemplu:



Suma primelor “n” numere naturale:

```
...
s=0; i=0;
while (i<n)
    s = s + i;
    i = i + 1;
end
```

Observații:

1. **expresie** este de cele mai multe ori o expresie logică;
2. dacă **expresie** are valoarea 0 de la început, corpul ciclului nu se execută;
3. dacă valoarea expresiei este întotdeauna $\neq 0$ atunci ciclul este **infinit**.

Exemplu:



Ordonarea crescătoare a elementelor unui șir de numere reale.

Varianta I. Utilizarea funcției **sort**

```
--> x=[5 -3 2 0 4 -1 3 -2 1]; sort(x)
ans =
-3 -2 -1 0 1 2 3 4 5
```

Varianta II. Crearea unui program utilizând instrucțiunea **while**.

```
x=[5 -3 2 0 4 -1 3 -2 1]; sch = 1;
while (sch~=0)
    sch = 0;
    for i = 1: length(x)-1
        if (x(i)>x(i+1))
            aux=x(i); x(i)=x(i+1); x(i+1)=aux; sch=sch+1;
        end
    end
end
disp(x);
```

Rularea programului:

```
--> -3 -2 -1 0 1 2 3 4 5
```

Exemplu:



Calculul sumei:

$$\pi = 2 \cdot \left(1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \frac{1 \cdot 2 \cdot 3 \cdot 4}{3 \cdot 5 \cdot 7 \cdot 9} + \dots \right)$$

până când diferența dintre doi termeni consecutivi este mai mică decât o valoare impusă.

Program în **FreeMat / OCTAVE / MATLAB**:

```
EPS = 0.000001;
t(1)=1./3; i = 2; t(i)=t(i-1)*i/(2*i+1); S =
1+t(1)+t(2);
    while((t(i-1)-t(i)) > EPS)
        i=i+1; t(i)=t(i-1)*i/(2*i+1); S = S + t(i);
    end
printf('\n PI_M = %10.8f \',pi)
printf('\n PI_C = %10.8f dupa %3d etape
\n',2*S,i-1)
```

Rularea programului:

```
PI_M = 3.14159265
PI_C = 3.14159117 dupa 17 etape
```

7.2. Instrucțiunea de ciclare *do-until*

Instrucțiunea de ciclare *do-until* este similară cu instrucțiunea **while**, interpretarea fiind însă: **execută ... până când**. Diferența dintre cele două instrucțiuni constă în faptul că **do-until** este o instrucțiune de ciclare condiționată posterior. Aceasta înseamnă că mai întâi se execută corpul ciclului și abia apoi se verifică condiția.

Forma generală a instrucțiunii este:

```
do
instrucțiune
until (expresie)
```

unde: **instrucțiune** reprezintă o instrucțiune simplă sau compusă iar **expresie** este în general o expresie logică.

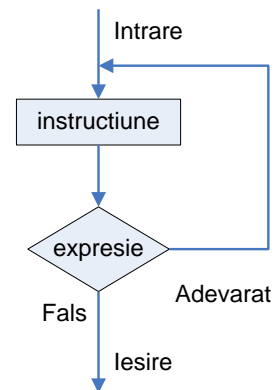


Figura 7.2.

Mod de lucru: Se execută **instrucțiune**, apoi se evaluează **expresie**. Dacă **expresie** este falsă (adică este zero) se revine la execuția **instrucțiune**. Dacă **expresie** este adevărată (adică are valoare diferită de zero), se părăsește ciclul.

Observație: Dacă expresie este un vector sau o matrice, se consideră adevărată numai dacă tabloul (vectorul sau matricea) nu este gol și dacă elementele au valoare diferită de zero.

Exemplu:



Citirea unui număr pozitiv:

```
do
a = input('Introdu o valoare intregă: a=');
until (a>0)
```

Exemplu:



Ordonarea crescătoare a unui șir de numere:

```
x=[5 -3 2 0 4 -1 3 -2 1];
do
    sch = 0;
    for i = 1: length(x)-1
        if(x(i)>x(i+1))
aux=x(i); x(i)=x(i+1); x(i+1)=aux; sch=sch+1;
        end
    end
until (sch==0)
disp(x);
```

7.3. Exemple de programe cu instrucțiuni while ... end și do – until

Exemplu:



Se consideră două numere întregi: a și b ($a > b$). Se cere să se determine câtul q și restul r al împărțirii numărului a la b. Rezultatele se vor afișa sub forma: $a = b * q + r$.

Fișierul script:

```
do
    a=input(' Introduceți primul număr: a=');
    b=input(' Introduceți primul număr: b=');
    c = floor(a/b); r = rem(a,b);
    printf("\n %d = %d x %d + %d \n",a,b,c,r);
until (a == 0)
```

Rularea programului:

```
--> Introduceți primul număr: a = 23
Introduceți primul număr: b = 4
23 = 4 x 5 + 3
```

Exemplu:



Să se calculeze și să se afișeze termenii din dezvoltarea:

$$S_n = \frac{1}{1+2} + \frac{1}{2+3} + \frac{1}{3+4} + \dots$$

până când diferența dintre doi termeni consecutivi este mai mică decât o valoare impusă (epsilon).

Fișierul script:

```
epsilon = input(' Introduceți precizia de
calcul: ');
i = 1; t(1) = 1/3; s=t(1);
do i = i+1; t(i)=1/(i+(i+1)); s=s+t(i);
until(t(i-1)-t(i)<epsilon)
printf("\n Suma este: s = %12.9f",s);
printf("\n Termeni: n = %d",i);
```

Rularea programului:

```
--> Introduceți precizia de calcul: 0.001
Suma este: s = 1.570818086
Termeni: n = 23
```

Exemplu:



Se citesc de la tastatură un număr natural **n** și o valoare **b**, $b \in [2,9]$. Se cere să se întocmească schema logică și program pentru conversia numărului natural **n** din baza **10** în baza **b**.

Algoritmul de conversie a unui număr din baza zece în baza **b** este următorul:

Se împarte numărul **n** la baza **b**.

Se obține un cât și un rest.

Restul astfel obținut va reprezenta ultima cifră a numărului **n** în baza **b**.

Câtul astfel obținut se va împărți la baza **b** și se vor obține iarăși un cât și un rest.

Restul va reprezenta penultima cifră a numărului **n** în baza **b** ș.a.m.d..

Procedeul se repetă până când câtul obținut la împărțire devine nul.

Pașii pe care trebuie să-I parcurgem sunt:

1. Se citește numărul n de la tastatură. De exemplu, se citește de la tastatură $n = 35$.
2. Se citește valoarea bazei b , de exemplu 2 .
3. Se notează valoarea numărului n în baza b cu Nb și se inițializează cu 0 .
4. Se utilizează o variabilă $fact$ cu ajutorul căreia se va asigura plasarea cifrelor numărului Nb pe pozițiile corespunzătoare. Valoarea acesteia se inițializează cu 1 .
5. Se verifică valoarea lui n , dacă este pozitivă, se calculează restul și câtul împărțirii numărului n la baza b . Se atribuie valoarea câtului variabilei n și se calculează valoarea variabilei Nb cu relația $Nb := Nb + fact * r$. Se modifică valoarea variabilei $fact$.
6. Se revine la pasul 5

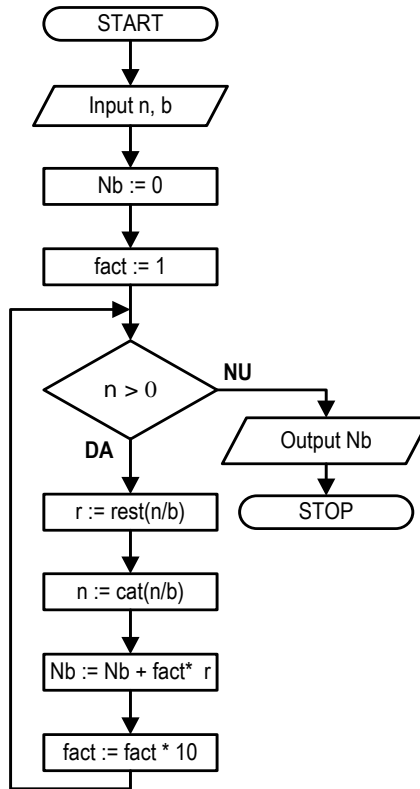


Figura 7.5

Observație: Pașii 5 – 7 se repetă în cadrul fiecărei etape.

Etapa	n	b	cât	r	Nb	fact
0	35	2	0	0	0	1
1	35	2	17	1	1	10
2	17	2	8	1	11	100
3	8	2	4	0	011	1000
4	4	2	2	0	0011	10000
5	2	2	1	0	00011	100000
6	1	2	0	1	100011	1000000

Valorile obținute pentru variabile sunt prezentate în tabelul următor:

Deoarece numărul **n** devine nul, în pasul etapa 6, pasul 6 nu se mai îndeplinește condiția impusă, continuarea se va face pe ramura cu **NU**, astfel că se va afișa valoarea variabilei **Nb** (adică **1**) și se încheie derularea etapelor.

Fișierul script:

```
s=input(' Date de intrare :');  
n=input(' Introduceți un număr natural:');  
Nb=0; fact=1;  
b=input(' Introduceți baza în care doriți  
conversia:');  
while(n>0)  
    r=mod(n,b); n = fix(n/b); Nb=Nb+fact*r;  
    fact=fact*10;  
end  
printf('\n Reprezentarea în baza %d este %d  
\n' ,b,Nb);
```

Rularea programului:

```
--> Introduceți un număr natural: 35  
Introduceți baza în care doriți conversia: 2  
Reprezentarea în baza 2 este 10011
```

Exemplu:



Se citește de la tastatură un număr natural n . Se cere să se întocmească schema logică și program pentru afișarea reversului numărului.

Mod de lucru: Determinarea reversului se desfășoară într-un număr finit de etape, numărul de etape fiind egal cu numărul de cifre al lui n .

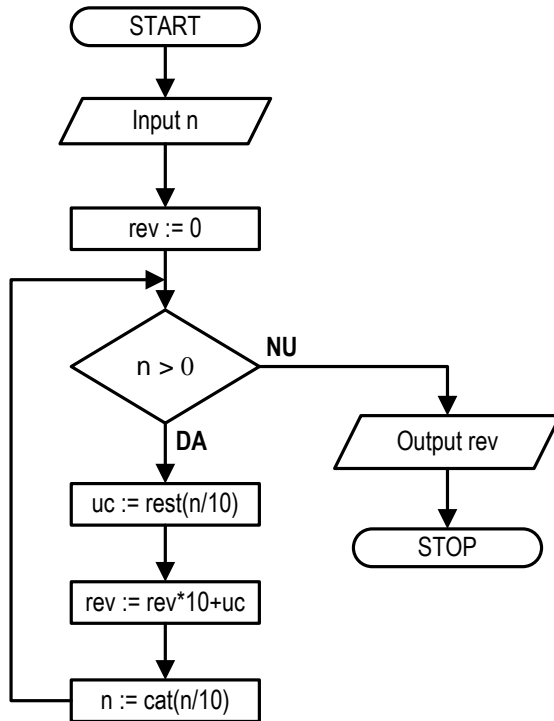


Figura 7.3

Pașii pe care trebuie să-i parcurgem sunt:

1. Se citește numărul **n** de la tastatură. De exemplu, se citește de la tastatură **n = 3562**.
2. Variabila **rev** reprezintă variabila care va memora valoarea reversului numărului **n**. Se inițializează valoarea acesteia cu 0 (zero).
3. Se verifică dacă numărul citit de la tastatură **n** este strict pozitiv. Dacă **DA** se continuă cu pașii următori, dacă **NU** se afișează valoarea variabilei **rev** și se finalizează algoritmul.
4. Se determină ultima cifră, notată cu **uc**, a numărului **n** prin calcularea restului împărțirii numărului **n** la 10. În cazul nostru, prin împărțirea numărului **n** ($= 3562$) la 10 se obține restul **2**. Deci, **uc** va fi egal cu **2**.
5. Se atribuie variabilei **rev** valoarea dată de relația **rev*10+uc**. Astfel **rev** $= 0*10 + 2$, adică **rev** $= 2$.
6. Se recalculează valoarea lui **n** astfel: valoarea nouă a lui **n** este cățul împărțirii lui **n** la 10, astfel $n = \text{cat}(3562 / 10) = 356$.
7. Se revine la pasul 3.

Observație: Pașii 3 – 7 se repetă în cadrul unei etape.

Valorile obținute pentru variabile sunt prezentate în tabelul alăturat:

Etapa	uc	rev	n
0	0	0	3562
1	2	2	356
2	6	26	35
3	5	265	3
4	3	2653	0

Deoarece numărul **n** devine nul, în pasul 3 nu se mai îndeplinește condiția impusă, continuarea se va face pe ramura cu **NU**, astfel că se va afișa valoarea variabilei **rev** (adică **2653**) și se încheie derularea etapelor.

Fișierul script

```
s=input(' Date de intrare:');  
n=input(' Introduceți un numar natural:');  
rev=0;  
while(n>0)  
uc=mod(n,10); rev=rev*10+uc; n = fix(n/10);  
end  
printf('\n Reversul numarului este %d \n',rev)
```

Rularea programului:

```
--> Introduceți un numar natural: 3562  
Reversul numarului este 2653
```

Exemplu:



Se citește de la tastatură un număr natural **n**. Se cere să se întocmească schema logică și program pentru determinarea sumei cifrelor numărului.

Mod de lucru: Calculul sumei se desfășoară într-un număr finit de etape, numărul de etape fiind egal cu numărul de cifre al lui **n**. Nu se știe însă câte cifre are numărul **n**.

Pașii pe care trebuie să-i parcurgem sunt:

1. Se citește numărul **n** de la tastatură. De exemplu, se citește de la tastatură **n = 3562**.
2. Variabila **S** reprezintă variabila care va memora valoarea sumei cifrelor numărului **n**. Se inițializează valoarea acesteia cu 0 (zero).
3. Se verifică dacă numărul citit de la tastatură **n** este strict pozitiv. Dacă **DA** se continuă cu pașii următori, dacă **NU** se afișează valoarea variabilei **S** și se finalizează algoritmul.

4. Se determină ultima cifra, notată cu **uc**, a numărului **n** prin calcularea restului împărțirii numărului **n** la 10. În cazul nostru, prin împărțirea numărului **n** ($= 3562$) la 10 se obține restul **2**. Deci, **uc** va fi egal cu **2**.
5. Se atribuie variabilei **S** valoarea dată de relația $S := S + uc$.
6. Se recalculează valoarea lui **n** astfel: valoarea nouă a lui **n** este câtul împărțirii lui **n** la 10, astfel $n = \text{cat}(3562 / 10) = 356$.
7. Se revine la pasul 3.

Etapa	uc	S	n
0	0	0	3562
1	2	2	356
2	6	8	35
3	5	13	3
4	3	16	0

Observație: Pașii 3 – 7 se repetă în cadrul unei etape.

Valorile obținute pentru variabile sunt prezentate în tabelul alăturat:

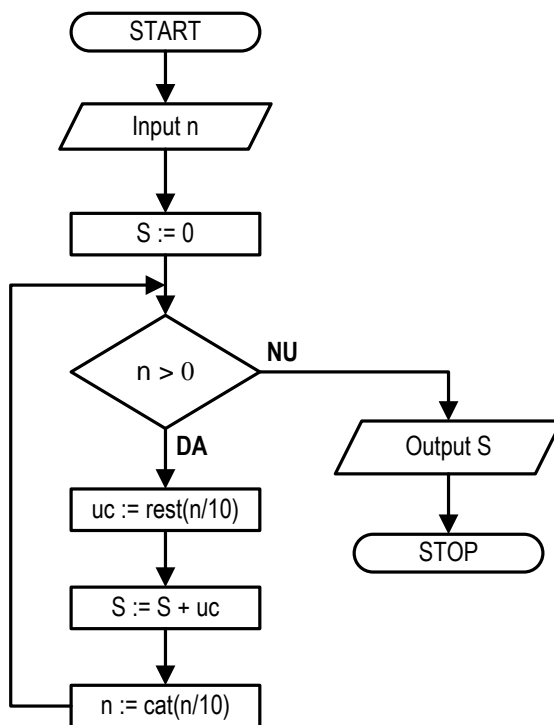


Figura 7.4

Deoarece numărul **n** devine nul, în pasul 3 nu se mai îndeplinește condiția impusă, continuarea se va face pe ramura cu **NU**, astfel că se va afișa valoarea variabilei **S** (adică **16**) și se încheie derularea etapelor.

Fișierul script:

```
s=input(' Date de intrare :');  
n=input(' Introduceți un număr natural:');  
S=0;  
while (n>0)  
    uc=mod(n,10); S=S+uc; n = fix(n/10);  
end  
printf('\n Suma cifrelor numărului este S = %d  
\n' ,S);
```

Rularea programului:

```
--> Introduceți un număr natural: 3562  
Suma cifrelor numărului este S = 16
```

7.4. Instrucțiunea break

Se utilizează pentru părăsirea unui ciclu înainte de a se fi terminat. În cazul unor cicluri imbricate instrucțiunea **break** determină părăsirea ciclului cel mai interior.

Exemplu:



Părăsirea unui ciclu for atunci când se îndeplinește o condiție, în acest caz când variabila conducătoare primește valoarea 8.

```
for i=1:10  
    if(i==8)  
        break;  
    else  
        printf(' i = %2d \n' ,i);  
    end  
end
```

Rularea programului:

```
-->  
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7
```

7.5. Instrucțiunea continue

Instrucțiunea **continue** cedează controlul iterației următoare în cazul unei instrucțiuni de ciclare (**for** sau **while**).

Exemplu:



Program pentru calculul valorilor funcției:

$$y = f(x) = \frac{x^2 - 5x + 3}{x - 2}, \quad x \in [-a, a + 2], \text{ parcurs cu pasul } h.$$

```
s=input('Date de intrare:');
a=input('Introduceti valoarea lui a: a=');
h=input('Introduceti valoarea lui h: h=');
for x=-a:h:a+2
    if(x==2) continue;
    else
        y=(x^2-5*x+3)/(x-2);
        printf(' x = %7.2f \t y = %7.2f \n',x,y);
    end
end
```

Rularea programului:

```
-->Introduceti valoarea lui a: a= 2
Introduceti valoarea lui h: h= 1
x = -2.00 y = -4.25
x = -1.00 y = -3.00
x = 0.00 y = -1.50
x = 1.00 y = 1.00
x = 3.00 y = -3.00
x = 4.00 y = -0.50
```

7.6. Instrucțiunea error

Permite afișarea unor mesaje la întâlnirea unor erori. Forma generală a funcției este: **error('mesaj')**.

După afișarea textului 'mesaj' se cedează controlul tastaturii.

Exemplu:



Program pentru calculul valorilor funcției:

$$y = f(x) = \frac{x^2 - 5x + 3}{x - 2}, \quad x \in [-a, a + 2], \text{ parcurs cu pasul } h.$$

```
s=input('Date de intrare:');
a=input('Introduceti valoarea lui a: a=');
h=input('Introduceti valoarea lui h: h=');
for x=-a:h:a+2
    if(x==2) error('Nu se poate calcula:');
        continue;
    else
        y=(x^2-5*x+3)/(x-2);
        printf(' x = %7.2f \t y = %7.2f \n',x,y);
    end
end
```

Rularea programului:

```
-->Introduceti valoarea lui a: a= 2
Introduceti valoarea lui h: h= 1
x = -2.00 y = -4.25
x = -1.00 y = -3.00
x = 0.00 y = -1.50
x = 1.00 y = 1.00
Nu se poate calcula
x = 3.00 y = -3.00
x = 4.00 y = -0.50
```

7.7. Instrucțiunea switch

Se utilizează pentru programarea decizilor multiple. Se utilizează în program pentru a comuta execuția la un anumit bloc de instrucțiuni din cadrul structurii ei, în funcție de rezultatul unui test. Pentru fiecare valoare posibilă a expresiei test poate fi stabilit un caz, deci se poate asocia un bloc de instrucțiuni care va fi executat.

Instrucțiunea **switch** are forma generală:

```
switch (expresie)
    case val_1 instruct_1
    case val_2 instruct_2
    ...
    case val_n instruct_n
    otherwise bloc
end
```

Mod de lucru: Se evaluează **expresie** de după cuvântul cheie **switch**. Se compară pe rând valoarea **expresie** cu **val_1, val_2, ..., val_n**. Dacă valoarea **expresie** este egală cu valoarea uneia dintre acestea se execută blocul de instrucțiuni (**instruct_1, instruct_2, ...**), în caz se execută blocul de instrucțiuni aferent variantei **otherwise** (adică **bloc**).

Exemplu:



Program pentru realizarea unor operații aritmetice cu doi operanzi, în funcție de simbolul operatorului introdus de la tastatură.

De la tastatură se introduce primul operand, al doilea operand și operatorul. Operațiile efectuate sunt: + (adunare), - (scădere), * (înmulțire), / (împărțire), \$ (media aritmetică), ^ (ridicare la putere).

```
s = input(' Date de intrare :');
a = input(' Introdu primul operand:');
b = input(' Introdu al doilea operand:');
op = input(' Introdu operatorul:', 's');
switch op
case '+' rez = a + b; % adunare
printf('%8.3f %s %8.3f=%8.3f\n', a, op, b, rez);
case '-' rez = a - b; % scadere
printf('%8.3f %s %8.3f=%8.3f\n', a, op, b, rez);
case '*' rez = a * b; % inmultire
printf('%8.3f %s %8.3f=%8.3f\n', a, op, b, rez);
case '/' rez = a / b; % impartire
printf('%8.3f %s %8.3f=%8.3f\n', a, op, b, rez);
case '$' rez = (a + b)/2;% media aritmetica
printf('%8.3f %s %8.3f=%8.3f\n', a, op, b, rez);
case '^' rez = a^b; % a la puterea b
printf('%8.3f %s %8.3f=%8.3f\n', a, op, b, rez);
otherwise printf(' Operator gresit!!!! \n');
end
```

Rularea programului:

```
--> Date de intrare :
Introdu primul operand: 4
Introdu al doilea operand: 7
Introdu operatorul: +

4 + 7 = 11
```

--> Date de intrare :
Introdu primul operand: 7
Introdu al doilea operand: 4
Introdu operatorul: -

$$7 - 4 = 3$$

--> Date de intrare :
Introdu primul operand: 7
Introdu al doilea operand: 4
Introdu operatorul: *

$$7 * 4 = 28$$

--> Date de intrare :
Introdu primul operand: 8
Introdu al doilea operand: 4
Introdu operatorul: /

$$8 / 4 = 2$$

--> Date de intrare :
Introdu primul operand: 8
Introdu al doilea operand: 4
Introdu operatorul: \$

$$8 \$ 4 = 6$$

--> Date de intrare :
Introdu primul operand: 2
Introdu al doilea operand: 4
Introdu operatorul: ^

$$2 ^ 4 = 16$$

8. Aplicații FreeMat / OCTAVE / MATLAB

8.1. Analiza mecanismului bielă-manivelă

Mecanismul bielă-manivelă (figura 8.1) se utilizează pentru transformarea mișcării de translație alternativă în mișcare de rotație continuă (motoare cu ardere internă), respectiv pentru transformarea mișcării de rotație continuă în mișcare de translație alternativă (compresoare, pompe, prese). Mecanismul bielă-manivelă se realizează în două variante constructive: dezaxat (axa de rotație a manivelei deplasată cu excentricitatea “e” față de direcția ghidajului), respectiv dezaxat (excentricitatea “e” este nulă).

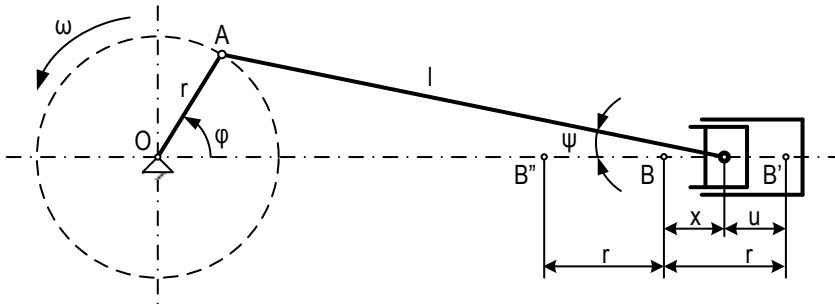


Figura 8.1.

În figura 8.1 notațiile au următoarea semnificație: l – lungimea bielei, r – raza manivelei, A – centru de articulație dintre bielă și axa cilindrului, B' și B'' – pozițiile extreme ale capătului bielei, x – deplasarea capătului bielei la un anumit moment. Manivela se rotește cu viteza unghiulară constantă: $\omega = \varphi'$, iar cupla de translație B are cursa $B'B''=2r$.

Poziția mecanismului este determinată de unghiul φ , care este o funcție de timp, astfel:

$$\varphi = \omega \cdot t = \frac{\pi \cdot n}{30} \cdot t \quad (8.1)$$

Poziția pistonului se raportează față de punctul mort exterior (B'), astfel coordonata punctului B față de punctul limită B' este:

$$u = r \cdot (1 - \cos \varphi) + l(1 - \cos \psi); \quad (8.2)$$

Ținând seama de relația geometrică:

$$\cos \psi = \sqrt{1 - \left(\frac{r}{l} \sin \varphi\right)^2} \quad (8.3)$$

și dezvoltând în serie de puteri expresia (8.3), și considerând numai primii doi termeni, se obține legea de mișcare a pistonului:

$$u \cong r \cdot \left(1 - \cos \varphi + \frac{\lambda}{2} \sin^2 \varphi \right) \quad (8.4)$$

În expresia (8.4) au fost neglijați termenii cu puteri mai mari decât 2 și s-a notat $\lambda = \frac{r}{l}$, raport denumit caracteristica structurală a mecanismului.

Pentru determinarea legii de variație a vitezei, se derivează legea de mișcare în raport cu timpul:

$$v = \frac{du}{dt} = \frac{du}{d\varphi} \cdot \frac{d\varphi}{dt} = \omega \cdot \frac{du}{d\varphi}$$

rezultă:

$$v(\varphi) = \omega \cdot r \cdot \left(\sin \varphi + \frac{1}{2} \cdot \lambda \cdot \sin 2\varphi \right), \quad (8.5)$$

$$v(t) = \omega \cdot r \cdot \left(\sin(\omega \cdot t) + \frac{1}{2} \cdot \lambda \cdot \sin(2 \cdot \omega \cdot t) \right)$$

Legea de variație a accelerației se determină derivând viteza în raport cu timpul:

$$a = \frac{dv}{dt} = \frac{dv}{d\varphi} \cdot \frac{d\varphi}{dt} = \omega \cdot \frac{dv}{d\varphi}$$

$$a(\varphi) = \omega^2 \cdot r \cdot (\cos \varphi + \lambda \cdot \cos 2\varphi) \quad (8.6)$$

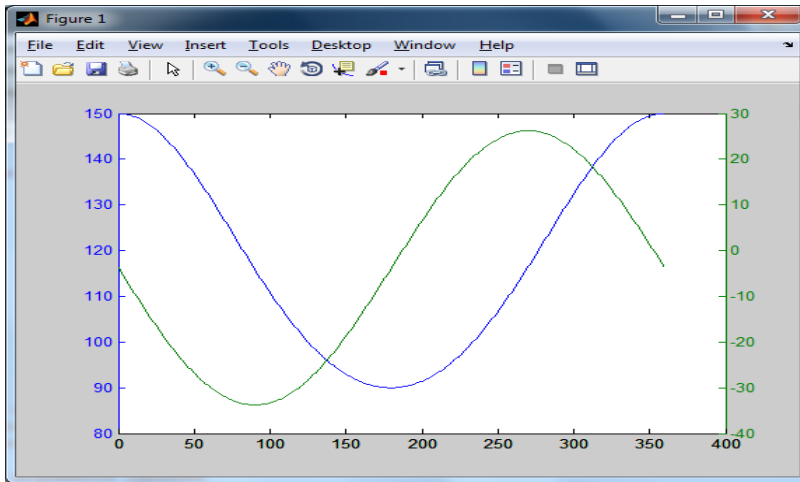
$$a(t) = \omega^2 \cdot r \cdot (\cos(\omega \cdot t) + \lambda \cdot \cos(2 \cdot \omega \cdot t))$$

Studiu cinematic presupune trasarea graficelor deplasării (x), a vitezei (v), respectiv a accelerației (a) în funcție de timp.

Fișierul script:

```
s =input(' Date de intrare : ');
lm=input(' Introduceți lungimea manivelei [mm]: ');
lb=input(' Introduceți lungimea bielei [mm]: ');
ex=input(' Introduceți excentricitatea [mm]: ');
pu=input(' Introduceți pasul unghiular [grade]: ');
fig=0:pu:360;
s=lm*cos(fig*pi/180.0)+sqrt(lb*lb-...
(lm*sin(fig*pi/180.0)+ex).^2);
v = -lm*sin(fig*pi/180.0) - ...
lm*(lm*sin(fig*pi/180.0)+ex).*sin(fig*pi/180.0)/sqrt(lb
*lb- ...
(lm*sin(fig*pi/180.0)+ex).^2);
plotyy(fig,s,fig,v)
```

Graficul generat în urma rulării programului:



8.2. Rezolvarea unei integrale prin metoda cuadratură adaptivă

Se consideră integrala:

$$I_c = \int_0^1 \frac{\cos(x)}{\sqrt{x}} dx$$

Valoarea exactă a integralei este: $I_c = 1,809048476$

Deoarece în $x = 0$, datorită numitorului se ajunge la o situație de nedeterminare care conduce la o eroare a limitei de recursivitate.

Mesajul obținut este:

??? Maximum recursion limit of 500 reached. Use set(0,'RecursionLimit',N) to change the limit. Be aware that exceeding your available stack space can crash MATLAB and/or your computer. Error in ==> q at 5 y = cos(x)./sqrt(x);

Funcțiile utilizate sunt:

```
function I=trapez(f,a,b,n);
h=(b-a)/n;
I=(f(a)+f(b)+2*sum(f([1:n-1]*h+a)))*h/2;
```

```
function I=adaptquad(f,a,b,eps)
m=4; % sau 5
I1=trapez(f,a,b,m);
I2=trapez(f,a,b,2*m);
```

```

if abs(I1-I2) < eps
    I=I2;
    return
else
    I=adaptquad(f,a,(a+b)/2,eps)+adaptquad(f,(a+b)/2,b,eps);
end

function y = p(x)
if x==0      y=1;
else      y = cos(x)./sqrt(x);
end

```

Rezultatul obținut este:

```
--> I=adaptquad(@p,0,1,1e-8)
```

```
I = 1.80904915475805
```

8.3. Rezolvarea unei integrale prin metoda Gauss-Legendre

Se consideră integrala:

$$I_c = \int_0^1 \frac{\cos(x)}{\sqrt{x}} dx$$

Se face schimbarea de variabilă: $t = \sqrt{x}$ și se obține: $dt = \frac{dx}{2\sqrt{x}} = \frac{dx}{2t}$

, adică: $dx = 2 \cdot t \cdot dt$, prin urmare integrala se poate scrie:

$$I_c = \int_0^1 2 \cdot \cos(t^2) dt$$

Valoarea exactă a integralei este: $I_c = 1,809048476$

Funcțiile utilizate sunt:

```

function [x,A] = gaussN(n,tol)
if nargin < 2; tol = 1.0e4*eps; end
A = zeros(n,1); x = zeros(n,1);
nra = fix(n + 1)/2;
for i = 1:nra
    t = cos(pi*(i - 0.25)/(n + 0.5));
    for j = i:30
        [p,dp] = legendre(t,n);
        dt = -p/dp; t = t + dt;
    end
end

```

```

        if abs(dt) < tol
            x(i) = t; x(n-i+1) = -t; A(i) = 2/(1-t^2)/dp^2;
            A(n-i+1) = A(i); break
        end
    end
end

function [p,dp] = legendre(t,n)
% Evaluează polinomul Legendre p de grad n si derivata
dp în x = t.
p0 = 1.0; p1 = t;
for k = 1:n-1
    p = ((2*k + 1)*t*p1 - k*p0)/(k + 1); p0 = p1;p1 = p;
end
dp = n *(p0 - t*p1)/(1 - t^2);

function I = gaussQuad(func,a,b,n)
% Cuadratura Gauss-Legendre
% func = numele functiei
% a,b = limitele de integrare
% n = ordinul de integrare
c1 = (b + a)/2; c2 = (b - a)/2;
[x,A] = gaussN(n);
sum = 0;
for i = 1:length(x)
y = feval(func,c1 + c2*x(i));
sum = sum + A(i)*y;
end
I = c2*sum;

% Fisierul functie fex
function y = fex(x)
y = 2*cos(x^2);

% Fisierul script
a = 0; b = 1; Ie = 1.809048476;
for n = 2:12
    I = gaussQuad(@fex,a,b,n);
    if abs(I - Ie) < 0.000000001
        I
        n
        break
    end
end
end

```

Rezultatul obținut este:

$I = 1.80904847548611$

$n = 6$

8.4. Program pentru verificarea validității Codului Numeric Personal

Codul numeric personal sau **CNP** este un cod unic asocia fiecărei persoane născute în România și este format din 13 cifre. Este atribuit la naștere fiecărui nou născut și se înscrie pe certificatul de naștere. De asemenea, CNP figurează pe buletinul de identitate sau cartea de identitate precum și pe permisul de conducere.

Cele 13 cifre ale **CNP** au următoarea semnificație:

<u>S</u>	<u>AA</u>	<u>LL</u>	<u>ZZ</u>	<u>JJ</u>	<u>NNN</u>	<u>C</u>
Codul numeric personal						

S reprezintă sexul și secolul în care s-a născut persoana care posedă acel CNP. Persoanelor de sex masculin le sunt atribuite numerele impare iar persoanelor de sex feminin numerele pare.

Prima cifră a CNP-ului este: (sex bărbătesc / sex femeiesc): 1 / 2 - născuți între 1 ianuarie 1900 și 31 decembrie 1999; 3 / 4 - născuți între 1 ianuarie 1800 și 31 decembrie 1899; 5 / 6 - născuți între 1 ianuarie 2000 și 31 decembrie 2099; 7 / 8 - pentru persoanele străine rezidente în România; 9 - pentru persoanele străine.

AA este un număr format din 2 cifre și reprezintă ultimele 2 cifre din anul nașterii. O persoană născută în anul 1970 va avea la AA 70. (SAA = 170). Dacă o persoană va avea prima cifră cu una din valorile 7,8 (rezidenti) sau 9, atunci se va considera secolul 20. ex SAA = 771 anul nașterii va fi 1971

LL este un număr format din 2 cifre și reprezintă luna nașterii persoanei: 01 – ianuarie, 02 – februarie, 03 – martie, 04 – aprilie, 05 – mai, 06 – iunie, 07 – iulie, 08 – august, 09 – septembrie, 10 – octombrie, 11 – noiembrie, 12 – decembrie.

ZZ reprezintă ziua nașterii în format de 2 cifre. Pentru zilele de la 1 la 9 se adaugă 0 înaintea datei. Spre exemplificare, o persoană născută în prima zi a lunii va avea codul 01.

JJ este un număr format din două cifre și este reprezentat de codul județului sau sectorului în care s-a născut persoana ori în care avea domiciliul sau reședința în momentul acordării C.N.P.

Codurile județelor sunt: 01 – Alba, 02 – Arad, 03 – Argeș, 04 – Bacău, 05 – Bihor, 06 – Bistrița-Năsăud, 07 – Botoșani, 08 – Brașov, 09 – Brăila, 10 – Buzău, 11 – Caraș-Severin, 12 – Cluj, 13 – Constanța, 14 – Covasna, 15 – Dâmbovița, 16 – Dolj, 17 – Galați, 18 – Gorj, 19 – Harghita, 20 – Hunedoara, 21 – Ialomița, 22 – Iași, 23 – Ilfov, 24 – Maramureș, 25 – Mehedinți, 26 – Mureș, 27 – Neamț, 28 – Olt, 29 – Prahova, 30 – Satu-Mare, 31 – Sălaș, 32 – Sibiu, 33 – Suceava, 34 – Teleorman, 35 – Timiș, 36 – Tulcea, 37 – Vaslui, 38 – Vâlcea, 39 – Vrancea, 40 – București, 41 – București S.1., 42 - București S.2., 43 - București S.3., 44 - București S.4., 45 - București S.5., 46 - București S.6., 51 - Călărași, 52 – Giurgiu.

NNN este un număr format din 3 cifre din intervalul 001 - 999. Numerele din acest interval se împart pe județe, birourilor de evidență a populației, astfel încât un anumit număr din acel interval să fie alocat unei singure persoane într-o anumită zi.

C este cifră de control aflată în relație cu toate celelate 12 cifre ale CNP-ului. Cifra de control este calculată după cum urmează: fiecare cifră din CNP este înmulțită cu cifra de pe aceeași poziție din numărul 279146358279; rezultatele sunt însumate, iar rezultatul final este împărțit cu rest la 11. Dacă restul este 10, atunci cifra de control este 1, altfel cifra de control este egală cu restul împărțirii.

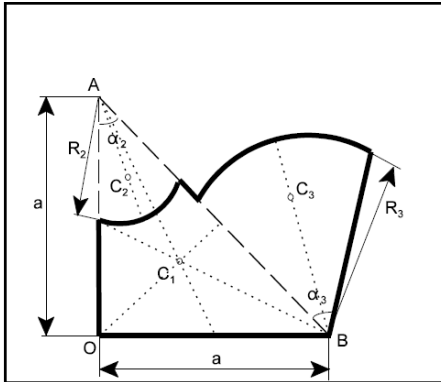
Fișierul script:

```
s = input('Date de intrare :');
cnp = input('Introduceti CNP:'); s=0;
cod_control = '279146358279';
for i=1:12
    s = s + str2num(cnp(i)) * str2num(cod_control(i));
end
cifra_control = str2num(cnp(13));
if mod(s,11) == cifra_control
printf(' CNP VALID !!!! \n ');
else
printf(' CNP INVALID !!!! \n ');
end
```

Observație: CNP se va introduce ca un șir de caractere

8.5. Program pentru calculul coordonatelor centrului de greutate al unei plăci plane

Se consideră placa omogenă din figura alăturată (figura 8.2). Se cere să se scrie un program pentru calculul coordonatelor centrului de greutate.



$$a = 10 + 0,5 * n \text{ [cm];}$$

$$R_2 = 7 + 0,1 * n \text{ [cm];}$$

$$R_3 = 8 + 0,2 * n \text{ [cm];}$$

$$\alpha_2 = 45^\circ$$

$$\alpha_3 = 40 + n^\circ$$

unde n – numărul de ordine al studentului în grupă;

Considerații teoretice:

Placa omogenă cu o formă mai complicată poate fi împărțită în bucăți sau părți și tratată ca o placă compusă omogenă.

În acest caz, placa se împarte în trei părți:

- o parte de forma unui triunghi dreptunghic isoscel AOB care are centrul de greutate C_1 aflat la intersecția medianelor;
- o parte sub forma unui sector circular de rază R_2 care însă lipsește și are centrul de greutate C_2 aflat pe axa sa de simetrie;
- o parte sub forma unui sector circular cu raza R_3 care se adaugă și care are centrul de greutate C_3 pe axa sa de simetrie.

Relațiile cu ajutorul cărora se vor determina coordonatele centrului de greutate sunt:

$$x_C = \frac{x_{C1}A_1 - x_{C2}A_2 + x_{C3}A_3}{A_1 - A_2 + A_3}, \quad y_C = \frac{y_{C1}A_1 - y_{C2}A_2 + y_{C3}A_3}{A_1 - A_2 + A_3}$$

Se observă că partea care lipsește se scrie cu semnul minus.

Fișierul script:

```
s = input(' Date de intrare: ');
n = input(' Introdu numarul de ordine:, n = ');
a = 10 + 0.5 * n; R2 = 7 + 0.1 * n; R3 = 8 + 0.2 * n; %[cm]
alfa1g = (45 + 0) / 2.; alfa2g = (40 + m) / 2.; %[grade]
```

```

printf(' Cota a = %9.4f [cm] \n',a);
printf(' Raza R2 = %9.4f [cm] \n',R2);
printf(' Raza R3 = %9.4f [cm] \n',R3);
printf(' Unghiul alfa1 = %9.4f [grade] \n',alfa1g);
printf(' Unghiul alfa2 = %9.4f [grade] \n',alfa2g);
alfa1r = alfa1g * pi()/ 180; alfa2r = alfa2g * pi()/180;
A1 = a * a / 2.; A2 = alfa1r * R2 * R2; A3 = alfa2r*R3*R3;
x1 = a / 3.; y1 = a / 3.; x2 = sin(alfa1r) * 2 * R2 *
sin(alfa1r) / (3 * alfa1r);
y2 = a - cos(alfa1r) * 2 * R2 * sin(alfa1r) / (3 *
alfa1r);
x3 = a - (2 * R3 * sin(alfa2r) / (3 * alfa2r)) * cos(alfa2r
+ 2 * alfa1r);
y3 = (2 * R3 * sin(alfa2r) / (3 * alfa2r)) * sin(alfa2r
+ 2 * alfa1r);
printf(' x1 = %f \t y1 = %f \t A1 = %f \n',x1,y1,A1);
printf(' x2 = %f \t y2 = %f \t A2 = %f \n',x2,y2,A2);
printf(' x3 = %f \t y3 = %f \t A3 = %f \n',x3,y3,A3);
xc = (A1*x1-A2*x2+A3*x3)/(A1-A2+A3);
yc = (A1*y1-A2*y2+A3*y3)/(A1-A2+A3);
printf('\n Coordonatele centrului de greutate sunt: \n');
printf('\n XC = %9.4f \t YC = %9.4f \n',xc,yc);

```

Rularea programului:

```

--> Introdu numarul de ordine, n = 12
Cota a = 16.0000 [cm]
Raza R2 = 8.2000 [cm]
Raza R3 = 10.4000 [cm]
Unghiul alfa1 = 22.5000 [grade]
Unghiul alfa2 = 26.0000 [grade]
x1 = 5.333333 y1 = 5.333333 A1 = 128.000000
x2 = 2.038647 y2 = 11.078271 A2 = 26.405086
x3 = 13.819404 y3 = 6.332911 A3 = 49.081451
Coordonatele centrului de greutate sunt:
XC = 8.6750 YC = 4.6522

```


Bibliografie:

- [1] Ghinea, M., Fireteanu, V., *MATLAB (calcul numeric-grafică-aplicații)*, Editura Teora, București, 2003, ISBN 973-601-275-1, 304 pg.
- [2] Chapman, S.J., *Essentials MATLAB Programming 2nd Edition*, Cengage Learning, USA, 2006, ISBN-13: 978-0-495-29568-6, 429 pg.
- [3] Chapra, S. C., *Applied Numerical Methods with MATLAB for Engineers and Scientists – Second Edition*, McGraw-Hill, 2006, ISBN 978-0-073-13290-7, 584 pg.
- [4] Curteanu, S., *Initiere in MatLab*, Editura POLIROM, 2008, ISBN 978-973-46-0920-8, 315 pg.
- [5] Gilat, A., *MATLAB An Introduction with Applications*, J Wiley & Sons, Inc., 2004, ISBN 0-471-43997-5
- [6] Hahn, B.D., Valentine, D.T., *Essential MATLAB for Engineers and Scientists*, Elsevier, 2007
- [7] Hunt, B.R., Lipsman, R.L., Rosenberg, J.M., *A Guide to MATLAB - For Beginners and Experienced Users*, Second Edition, Cambridge University Press, Cambridge UK, 2006, ISBN-13 978-0-521-61565-5,
- [8] Eato, J. W., *GNU Octave: A high-level interactive language for numerical computations*, Edition 3 for Octave 2.1.x., February 1997, <http://pcmap.unizar.es/softpc/OctaveManual.pdf>
- [9] Kalechman, M., *Practical MATLAB basics for engineers*, CRC Press, Boca Raton, Florida, 2007, ISBN 978-1-4200-4774-5, 736 pg.
- [10] Quarteroni, A., Saleri, F., Gervasio, P. – *Scientific Computing eith MATLAB and OCTAVE*, Third Edition, Springer – Verlag, 2010, ISSN 1611-0994, ISBN 978-3-642-12429-7, e-ISBN 978-3-642-12430-3
- [11] Smith, D.M., *Engineering Computation with MATLAB 2nd Edition*, Pearson Education, Inc., 2010.
- [12] Trâmbițaș, R.T., *Analiza numerică – MATLAB*, Presa Universitară Clujeană, 2005, ISBN 973-610-388

Link-uri utile:

<http://octave.sourceforge.net/>
<http://sourceforge.net/projects/octave/>
<http://www.mathworks.com/products/matlab/>
http://www.tmt.ugal.ro/crios/Support/ANPT/Tutoriale/MATLAB_IN_INGINERIE.pdf
<http://www.e-learn.ro/tutoriale/matlab/33.htm>