Radu Mircea MORARIU-GLIGOR

PROGRAMAREA CALCULATOARELOR ÎN MATLAB / FREEMAT / OCTAVE

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

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 Detești
Prof.dr.ing.mat. Nicolae Ursu-Fischer

ISBN 978-606-737-249-6

Prefaţă

Evoluţia din ultimii ani a limbajelor şi a mediilor de programare a condus la apariţia unor medii de programare noi, bazate pe limbaje de programare deja consacrate.

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

În lucrare sunt prezentate asemănările şi diferenţele dintre aceste medii de programare, precum şi generalităţi ale acestor medii de programare.

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

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

În al treilea capitol sunt prezentate modalităţile de rezolvare a ecuaţiilor algebrice şi transcendente, aspecte privind operaţii cu polinoame, rezolvarea sistemelor de ecuaţii liniare şi neliniare, interpolarea şi aproximarea datelor, derivarea şi integrarea numerică, rezolvarea ecuaţiilor diferenţiale ordinare, noţiunile teoretice fiind completate de exemple.

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

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

Capitolul şase al lucrării tratează instrucţiunea de ciclare for, noţiunile teoretice fiind însoţite de exemple sugestive.

Capitolul al şaptelea al lucrării prezintă instrucţiunile de ciclare while...end, do ... until, precum şi instrucţiunile break, continue, error şi switch. Exemplele care însoţesc noţiunile teoretice au rolul de a facilita modul de înţelegere a noţiunilor prezentate.

Ultimul capitol prezintă câteva aplicaţii, al căror nivel de dificultate este uşor-mediu, utile pentru înţelegerea noţiunilor prezentate anterior.

Cuprins:

1. Prezentare generală a mediilor de programare FreeMat / OCTAVE / MATLAB. Expresii. Formatul datelor. Caractere speciale. Vectori şi matrice. 9
1.1. Generalităţi 10
1.1.1. Asemănări şi diferenţe 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 şi calculator 20
1.2.1. Expresii 20
1.2.2. Funcţii 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 şi calculator 23
2. Prelucrarea şirurilor de date. Generarea şi manipularea matricelor. Extragerea elementelor individuale dintr-un vector sau matrice 25
2.1. Prelucrarea şirurilor de date 26
2.2. Generarea şi manipularea matricelor 27
2.3. Extragerea elementelor individuale dintr-un vector sau matrice 32
3. Rezolvarea ecuaţiilor algebrice şi transcendente. Polinoame. Rezolvarea sistemelor de ecuaţii liniare şi neliniare. Interpolarea şi aproximarea datelor. Derivarea numerică. Integrarea numerică. Rezolvarea ecuaţiilor diferenţiale ordinare 34
3.1. Rezolvarea ecuaţiilor algebrice şi transcendente 35
3.2. Polinoame în FreeMat / OCTAVE / MATLAB 37
3.3. Rezolvarea sistemelor liniare şi neliniare 41
3.4. Interpolarea şi aproximarea datelor 42
3.5. Derivarea numerică 43
3.6. Integrarea numerică 44
3.7. Rezolvarea ecuaţiilor diferenţiale ordinare 46
4. Grafică 2D şi 3D 48
4.1. Grafica 2D 49
4.2. Grafica 3D 58
5. Declaraţii şi variabile. Formatul datelor de ieşire. Operatori. Instrucţiuni de introducere şi extragere de date. Fişiere script. Fişiere funcţie. Instrucţiuni de decizie. 63
5.1. Etapele de rezolvare a unei probleme 64
5.2. Declaraţii şi variabile 64
5.3. Formatul datelor de ieşire 66
5.4. Operatori 67

5.5. Instrucţiuni de introducere şi extragere de date 68
5.6. Fişiere script 72
5.7. Exemple de fişiere script OCTAVE / MATLAB 72
5.8. Fişiere funcţie 74
5.9. Exemple de fişiere funcţie în OCTAVE / MATLAB 75
5.10. Instrucţiuni de decizie 76
5.10.1. Instrucţiunea if … end 76
5.10.2. Instrucţiunea if ... else ... end 77
5.10.3. Instrucţiuni if ... else suprapuse 79
5.10.4. Instrucţiunea de decizie multiplă elseif 80
6. Instrucţiunea de ciclare for. Cicluri for suprapuse 82
6.1. Instrucţiunea de ciclare for. 83
6.2. Exemple de programe cu ciclu for 84
6.3. Cicluri for suprapuse 87
6.4. Exemple de utilizare a ciclurilor for suprapuse 88
7. Instrucţiunea de ciclare while…end. Instrucţiunile break, continue, error, switch. 90
7.1. Instrucţiunea de ciclare while ... end 90
7.2. Instrucţiunea de ciclare do-until 92
7.3. Exemple de programe cu instrucţiuni while ... end şi do – until 93
7.4. Instrucţiunea break 101
7.5. Instrucţiunea continue 102
7.6. Instrucţiunea error 102
7.7. Instrucţiunea switch 103
8. Aplicaţii 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 validităţii 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 şi matrice.

Introducere:

În cadrul acestui curs se urmăreşte familiarizarea studenţilor cu mediile de programare **FreeMat**, **OCTAVE** şi **MATLAB**. După parcurgerea acestui curs studenţii vor fi capabili să utilizeze mediile de programare pentru efectuarea unor calcule matematice simple şi pentru rezolvarea unor probleme de dificultate scăzută.
Cursul este structurat în şase părţi, în fiecare dintre acestea fiind prezentate noţiuni teoretice şi practice necesare înţelegerii modului de organizare şi prezentare a mediilor de programare. În prima parte sunt prezentate noţiuni generale privind cele trei medii de programare, sunt prezentate interfeţele celor trei medii, precum şi asemănări şi diferenţe între cele trei medii de programare. A doua parte descrie modalitatea de utilizare a mediilor de programare **FreeMat**, **OCTAVE** şi **MATLAB** ca şi calculator, fiind prezentate câteva funcţii uzuale, utile în scrierea şi interpretarea unor expresii matematice. Următoarele două părţi prezintă într-o formă succintă formatul şi tipurile de date din **FreeMat** / **MATLAB**. În următoarea parte sunt prezentate câteva din caracterele speciale şi modul de utilizare a acestora. Partea a şasea conţine o serie de exemple de utilizare a mediilor de programare **FreeMat** / **MATLAB** ca şi calculator.

Obiective:

Prin parcurgerea acestui material, studenţii: vor şti să utilizeze mediile de programare **FreeMat / OCTAVE / MATLAB** ca şi calculator; vor învăţa să utilizeze câteva din funcţiile oferite de aceste medii de programare; se vor familiariza cu tipurile de date şi cu formatul datelor;

1.1. Generalităţi

1.1.1. Asemănări şi diferenţe

Aceste limbaje de programare sunt limbaje de nivel înalt, interpretate şi structurate, bazate pe calculul matriceal şi sunt utilizate în special pentru calcule ştiinţifice şi inginereşti. Programele în **FreeMat** / **OCTAVE** / **MATLAB** constau dintr-o listă de funcţii 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 afişat pe ecran. Caracterul virgulă (**,**) se utilizează pentru separarea comenzilor în cadrul unei linii, cu afişarea rezultatelor comenzilor pe ecran.

Pentru continuarea unei comenzi pe linia următoare se utilizează **…** .

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

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

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

Limbajele **MATLAB** şi **OCTAVE / FreeMat** sunt asemănătoare în mare parte, existând însă şi câteva deosebiri în ceea ce priveşte mediul de programare **OCTAVE**, cum ar fi: instrucţiunile **if**, **for**, **while** se pot termina cu **end** dar şi cu **endif**, **endfor**, **endwhile**; funcţiile pot fi scrise direct în linia de comandă înainte de utilizare şi se termină cu **endfunction**; începutul unei linii de program de comentariu poate fi marcată cu simbolul **%,** dar şi 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 funcţii sau un script. Sintaxa este bazată pe matrice şi oferă diverse funcţii pentru operaţiile 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 diferenţe între cele două medii de programare, însă cu o programare atentă a scriptului se poate obţine rularea corectă atât în **OCTAVE** cât şi în **MATLAB**.

QtOCTAVE este un front-end pentru **OCTAVE** şi este un program interactiv pentru calcule numerice de înaltă performanţă şi vizualizări. **QtOCTAVE** integrează toate acestea într-un mediu uşor de învăţat şi folosit, în care enunţurile problemelor şi rezolvările acestora sunt exprimate în modul cel mai natural posibil, aşa cum sunt scrise matematic, fără a fi necesară programarea tradiţională.

Pentru că **OCTAVE** este disponibil sub Licenta Publica Generala GNU, poate fi copiat de pe internet de la adresa **http://www.octave.org** şi 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 afişarea variabilelor locale şi a funcţiilor disponibile (2), o fereastră tip navigator (1), o fereastră tip editor (3), o fereastră de comenzi (terminal) (4) şi o fereastră pentru afişarea comenzilor lansate în terminal, şi a rezultatelor acestora (5).

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

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

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

Meniul View cuprinde următoarele opţiuni: **Clear terminal** – se utilizează pentru curăţarea ferestrei de comenzi (terminal); **Dock tools** – conţine comenzi pentru activarea ferestrelor aplicaţiei; **Windows Layout** – permite configurarea interfeţei aplicaţiei; **Show/Hide Objects** – permite afişarea / ascunderea ferestrelor sau a bărilor cu butoane.

Meniul Analysis cuprinde opţiunile: **Integrate function** – pentru integrarea funcţiilor; **Ordinary Differential Equations** – pentru rezolvarea ecuaţiilor diferenţiale.

Meniul Data conţine următoarele opţiuni: **Table** – se utilizează pentru crearea tabelelor de date (vectori sau matrice);
Format of the displayed output – se utilizează pentru stabilirea formatului de afişare; **Load matrix from file** – pentru încărcarea unei matrice dintr-un fişier; **Save matrix to file** – pentru salvarea unei matrice într-un fişier.

Meniul Equations conţine comenzi pentru rezolvarea ecuaţilor: **Solve equation by bisection method** – rezolvarea ecuaţilor prin metoda bisecţiei; **Linear equation** – rezolvarea ecuaţilor liniare; **Nonlinear equation** – rezolvarea ecuaţilor neliniare.

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

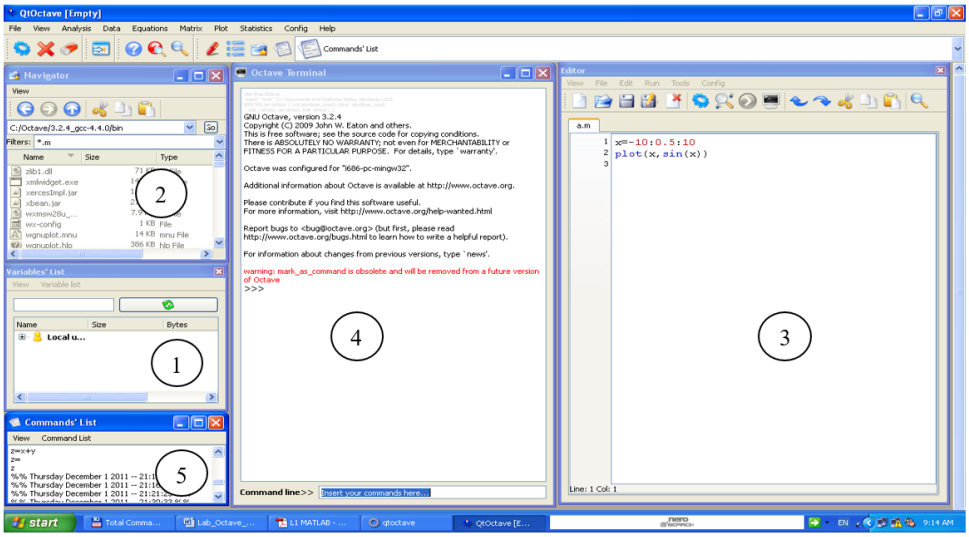


Figura 1.1. Interfaţa aplicaţiei **OCTAVE**

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

Meniul Statistics conţine comenzi pentru prelucrări statistice.

Meniul Config permite configurarea aplicaţiei.

Meniul Help permite utilizarea sistemului de informaţii ajutătoare.

1.1.3. Prezentare generală a mediului de programare FreeMat

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

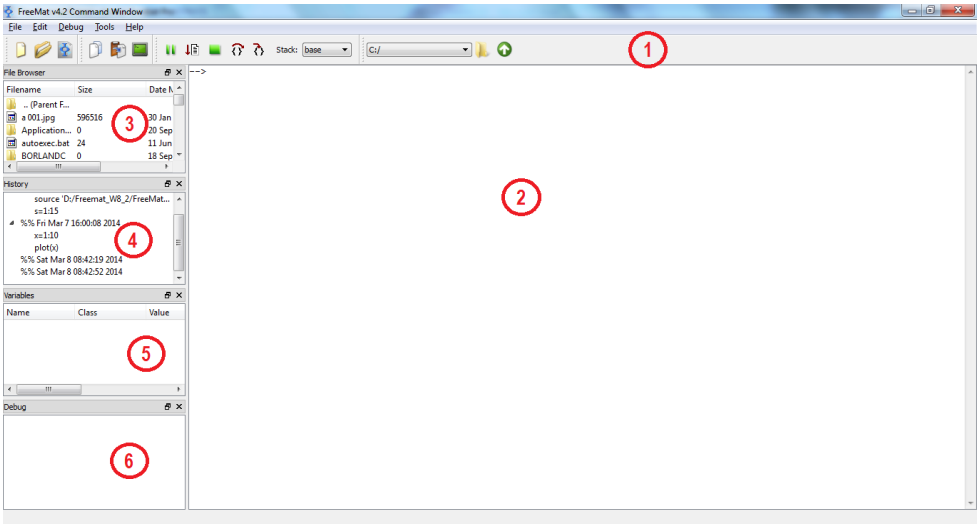


Figura 1.2. Interfaţa aplicaţiei **FreeMat**

Interfaţa aplicaţiei **FreeMat** conţine următoarele elemente (figura 1.2): 1 – meniul principal şi bara cu unelte; 2 – fereastra de comenzi: permite introducerea comenzilor, definirea variabilelor sau a constantelor, precum şi definirea funcţiilor; 3 – browser-ul de fişiere: permite vizualizarea directoarelor şi a fişierelor curente; 4 – fereastra de istoric: ilustrează în ordine cronologică toate comenzile introduse precum şi 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 acţiunile rezultate în urma procesului de depanare;

Meniul principal conţine următoarele meniuri:

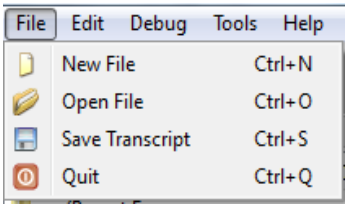


Fig. 1.3. Meniul **File**

Meniul **File** conţine comenzi pentru operarea cu fişiere (figura 1.3):

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

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

Save Transscript (Ctrl + S): Salvează conţinutul ferestrei de comenzi într-un fişier text.
Quit (Ctrl + Q): se utilizează pentru închiderea aplicaţiei **FreeMat**.

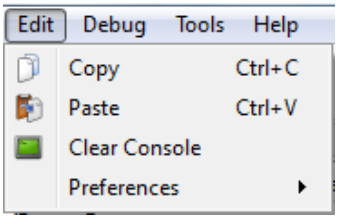


Fig. 1.4. Meniul **Edit**

Meniul **Edit** conţine comenzi de editare (figura 1.4):
Copy (Ctrl + C): copiază în memoria tampon textul selectat;
Paste (Ctrl + V): inserează în poziţia curentă a cursorului, conţinutul memoriei tampon;
Clear console: curăţă conţinutul ferestrei de comenzi;
Preferences: permite stabilirea dimensiunilor fonturilor şi a numărului de linii scrise anterior care pot fi apelate;

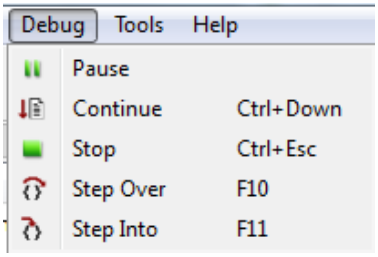


Fig. 1.5. Meniul **Debug**

Meniul **Debug** conţine 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 funcţii.

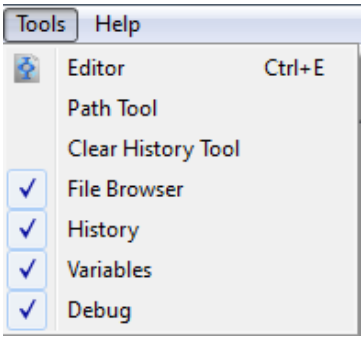
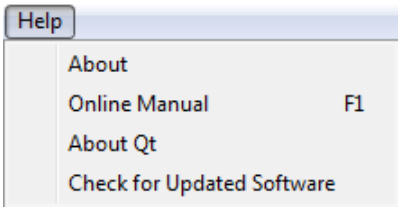


Fig. 1.6. Meniul **Tools**

Meniul **Tools** conţine comenzi pentru gestionarea instrumentelor (figura 1.6), astfel:
Editor: deschide editorul de texte;
Path Tools: permite stabilirea directorului curent;
Clear History Tool: şterge conţinutul istoricului;
File Browser: activează / dezactivează browser-ul de fişiere;
History: activează / dezactivează fereastra de istoric;
Variables: activează / dezactivează fereastra “Variables”;
Debug: activează / dezactivează fereastra “Debug”;



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

Fig. 1.7. Meniul Help

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

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

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

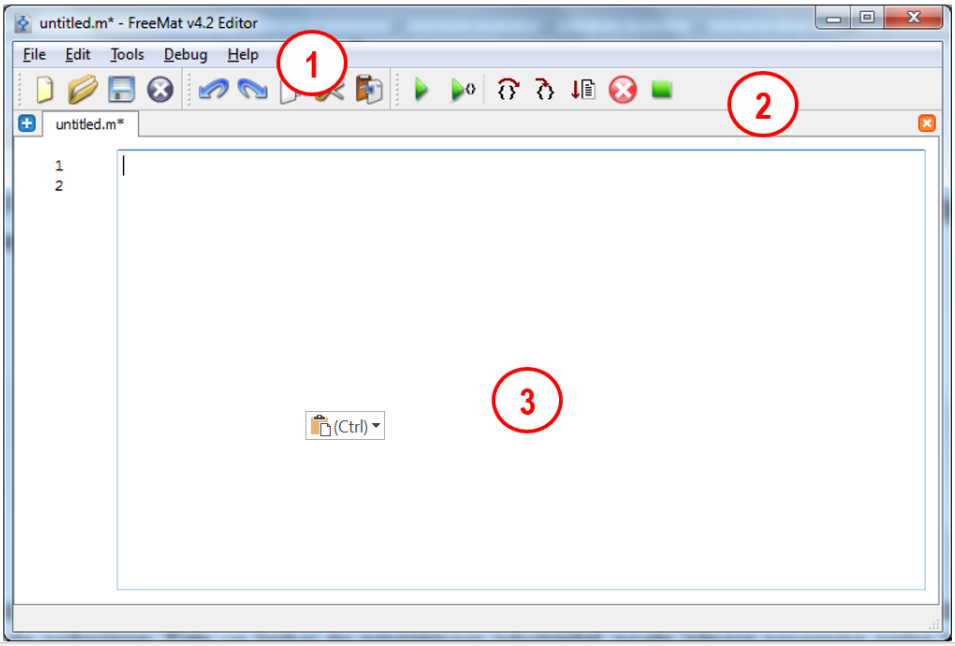


Figura 1.8. Interfaţa editorului de texte al mediului de programare **FreeMat**

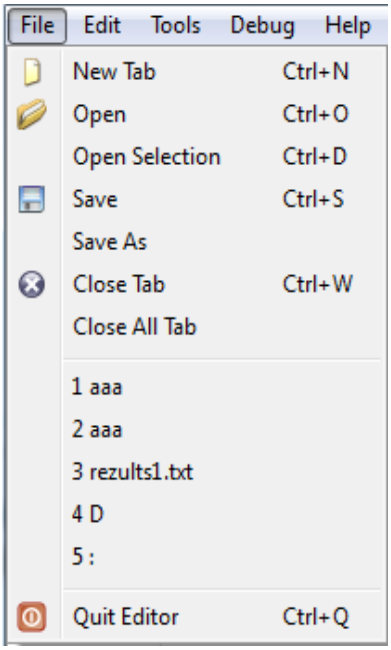


Figura 1.9. Meniul **File**

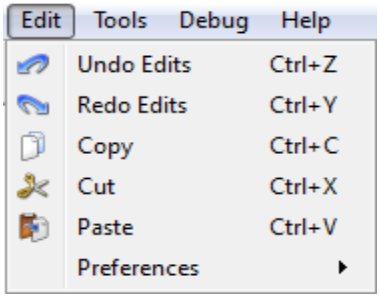


Figura 1.10. Meniul **Edit**

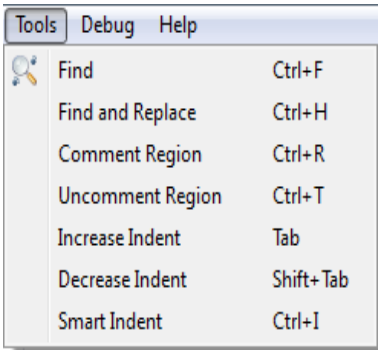


Figura 1.11. Meniul **Tools**

Meniul **File** conţine comenzi pentru gestionarea fişierelor (figura 1.9):
New Tab: crează un fişier nou într-o pagină nouă;
Open: deschide un fişier existent;
Open Selection: deschide fişierul al cărui nume se află precizat într-un câmp selectat;
Save: salvează fişierul curent;
Save As: salvează fişierul 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** conţine 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 conţinutul memoriei tampon în poziţia curentă a cursorului;
Preferences: permite setarea fonturilor, a dimensiunii spaţiului de aliniere la stânga a textului, precum şi alte setări;

Meniul **Tools** conţine următoarele comenzi (figura 1.11):
Find: se utilizează pentru căutare în text;
Find and Replace: se utilizează pentru căutare şi 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 spaţii (Tab orizontal);
Decrease Indent: se utilizează pentru deplasarea la stânga cu un număr de spaţii;
Smart Indent: se utilizează pentru alinierea fată de marginea din stănga a mai multor linii de text;

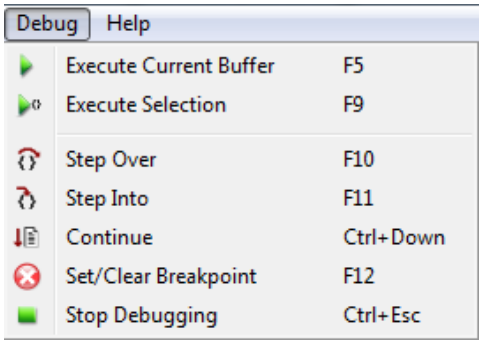


Figura 1.12. Meniul **Debug**

Meniul **Debug** conţine 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 funcţii;
Continue: permite continuarea procesului de rulare;
Set / Clear Breakpoint: selectează / deselectează un punct de întrerupere;
Stop Debugging: opreşte procesul de depanare.

Meniul **Help** permite accesarea sistemului de informaţii 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 şi simulări numerice, prelucrări de date şi reprezentări grafice cu precădere în ştiinţă şi inginerie. **MATLAB** este alcătuit dintr-un **nucleu** şi o serie de pachete de programe, intitulate **toolbox**-uri.

Nucleul are un interpretor propriu şi conţine comenzi de uz general, funcţii matematice de bază, funcţii de prelucrare a şirurilor de caractere, instrucţiuni de control, funcţii pentru reprezentări grafice, etc.
Toolbox-urile reprezintă colecţii de funcţii **MATLAB** organizate în scopul rezolvării problemelor din diverse domenii, cum ar fi: calcul simbolic, optimizare, programare, grafică ştiinţifică şi inginerească, modelarea şi simularea sistemelor dinamice, etc.
Sistemul **MATLAB** se compune din cinci părţi principale:

Mediul de dezvoltare ***MATLAB*** se compune dintr-o colecţie de instrumente şi facilităţi ce permit utilizarea fişierelor şi funcţiilor **MATLAB**.

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

Limbajul ***MATLAB*** este un limbaj de nivel înalt, cu instrucţiuni de control, funcţii, structuri de date şi elemente de programare orientată pe obiecte.

Grafica ***MATLAB (Handle Graphics)*** conţine funcţii pentru realizarea graficelor bi- şi tridimensionale, pentru procesarea imaginilor, animaţie şi prezentarea graficelor, personalizarea aspectului graficelor, construirea interfeţelor grafice cu utilizatorul pentru aplicaţiile **MATLAB**.

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

Interfaţa mediului de dezvoltare **MATLAB** este alcătuită din mai multe ferestre (figura 1.13) şi conţine elemente comune tuturor aplicaţiilor Windows, precum şi o serie de elemente specifice.

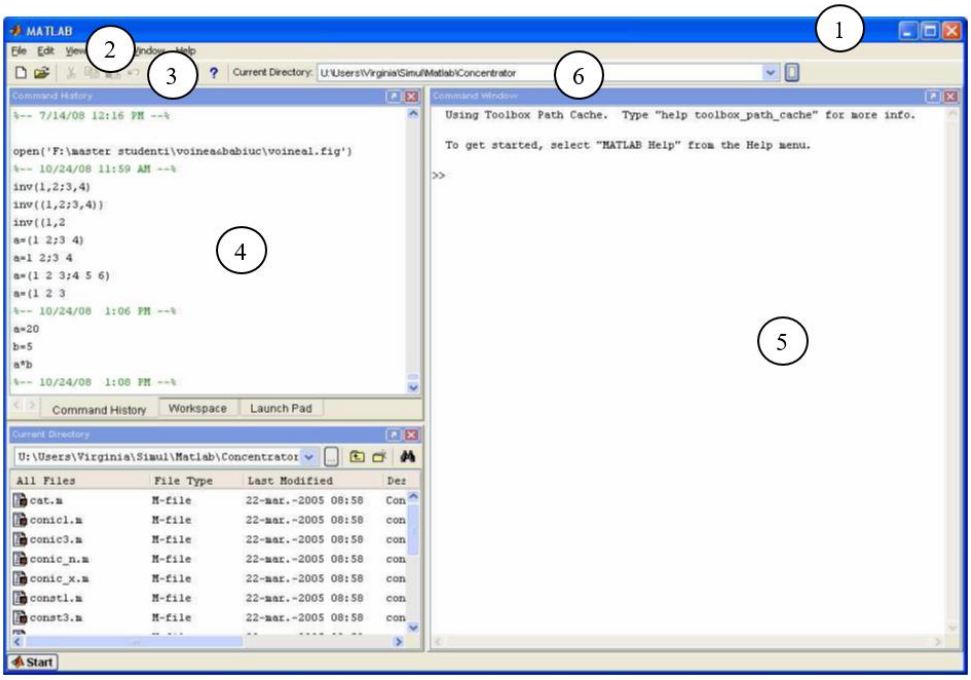


Figura 1.13. Interfaţa aplicaţiei **MATLAB**

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

Cele mai utilizate ferestre sunt:

- *spaţiul de lucru (**Workspace**)* conţine setul de variabile (caracterizate prin numele şi dimensiunea acestora) utilizate în sesiunea curentă de lucru şi stocate în memorie.
- *fereastra de comenzi* (***Command Window***) se utilizează pentru introducerea în linia de comandă a instrucţiunilor, după prompter (reprezentat prin simbolul >>);
- *fereastra de editare a unui fişier* (***Editor***);
- *fereastra pentru reprezentări grafice* (***Figure***);
- *fereastra pentru accesarea informaţiilor ajutătoare* (***Help***);
- *fereastra **Current Directory*** afişează fişierele directorului curent. Permite afişarea şi schimbarea directorului curent.
- *fereastra **Command History*** conţine 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 şi lansarea pachetelor de programe.

Cu ajutorul tastelor cu săgeţi pot fi realizate o serie de acţiuni 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 sfârşitul liniei de comandă; **Esc** : abandonează linia de comandă curentă; **Ins** : trece între modurile Insert şi Overtype; **Backspace** : şterge un caracter la stânga cursorului

Dacă se doreşte introducerea unei comenzi (sau un text) a cărei lungime depăşeşte lungimea liniei curente, inseraţi trei puncte (**…**) pe linia curentă şi continuaţi 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 funcţii se scriu cu litere mici. Liniile de comentariu sunt precedate de caracterul %. Pentru obţinerea informaţiilor ajutătoare se tastează **help** pentru meniul întreg sau **help** urmat de denumirea funcţiei sau a fişierului **.m**.

1.2. Utilizarea FreeMat / OCTAVE / MATLAB ca şi calculator

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

1.2.1. Expresii

La fel ca şi 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* şi *funcţii*.

Variabile: În toate limbajele de programare variabilele sunt caracterizate de trei elemente: nume (identificator), tip şi valoare. Limbajele **FreeMat** / **Octave** / **MATLAB** nu necesită declararea dimensiunii variabilelor, deoarece la întâlnirea unui nou nume de variabilă se generează automat variabila respectivă şi se alocă spaţiul necesar de memorie. Numele unei variabile începe cu o literă şi 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 distincţie între literele mici şi cele mari, adică este *case sensitive*.

Numere*: FreeMat** / **Octave** / **MATLAB** utilizează notaţia zecimală, cu punct zecimal opţional şi cu semn + sau -. Se poate utiliza şi notaţia ştiinţifică 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), * (înmulţire), / (împărţire), \ (împărţire la stânga), ^ (ridicare la o putere), ( ) (operatorul de specificare a odinii de evaluare); operatori relationali : = = (egal); ~= (diferit) ; < (mai mic); > (mai mare) ; <= (mai mic sau egal) ; >= (mai mare sau egal); operatori logici : & (şi); | (sau); ~ (not); xor (or)

1.2.2. Funcţii OCTAVE / MATLAB

Funcţii: **FreeMat** / **OCTAVE** / **MATLAB** pun la dispoziţia utilizatorilor şi a programatorilor un mare număr de funcţii matematice elementare standard, astfel:

a) funcţii 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) funcţii putere: **exp** – funcţia exponenţială, **log** – logaritm natural, **log2** – logaritm în baza 2, **log10** – logaritm în baza 10, **sqrt** – funcţia radical;
c) alte funcţii: **abs** – valoarea absolută (modul), **min** – minimum, **max** – maximum, **pi** (3.14159265), **i** şi **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** (**N***ot*-**a**-**N***umber*) - 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 acelaşi tip. În figura 1.14. şi tabelul 1.1 sunt ilustrate toate tipurile de date din **FreeMat** / **OCTAVE** / **MATLAB**.

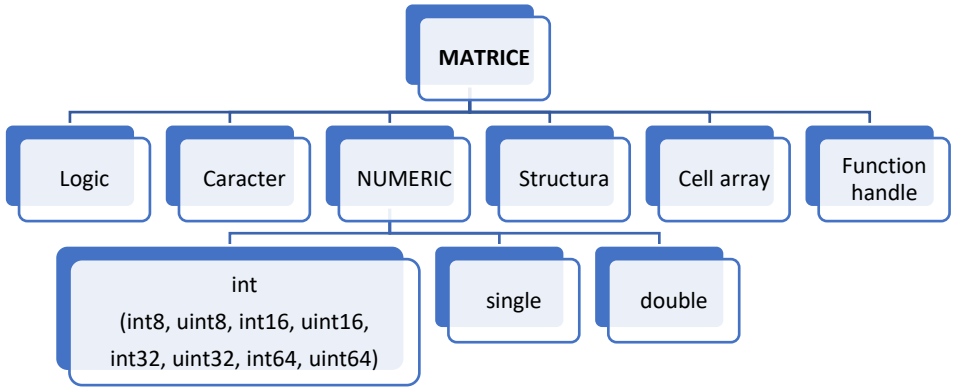


Figura 1.14. Tipurile de date din **FreeMat** / **OCTAVE** / **MATLAB**

Tipurile **structură** sau **celule de tablouri** pot să conţină date de tipuri diferite. Cu ajutorul claselor (uses classes şi 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:	Biţi	Domeniu de valori:
int8	Întreg cu semn pe 8 biţi	8	[-128, 127]
uint8	Întreg fără semn pe 8 biţi	8	[0, 255]
int16	Întreg cu semn pe 16 biţi	16	[-32,768, 32,767]
uint16	Întreg fără semn p 16 biţi	16	[0, 65,535]
int32	Întreg cu semn pe 32 biţi	32	[-2,147,483,648,2,147,483,647]
uint32	Întreg fără semn p 32 biţi	32	[0, 4,294,967,295]
int64	Întreg cu semn pe 64 biţi	64	[-9,223,372,036,854,775,808, 9,223,372,036,854,775,807]
uint64	Întreg fără semn p 64 biţi	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 şir de caractere	16	
function handle	Pointer la o funcţie.		
Structure	Structura este o colecţie 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** afişarea 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 afişare a datelor de ieşire :

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

Spaţierea:

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

Exemplu:

```
--> c =                          --> format short e, c
 0.123456789012                  c = 1.2346e-001
--> c                            --> format long e, c
 c = 0.12346                     c = 1.2345678901e-001
```

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

Sistemul de informaţii ajutătoare: Pentru apelarea sistemului de informaţii ajutătoare se utilizează comanda **help** urmată de numele funcţiei pentru care se doreşte obţinerea informaţiilor 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.	Semnificaţie:	Car.	Semnificaţie
:	Se foloseşte la generarea diviziunilor	;	Separator între instrucţiuni pe aceiaşi linie de comandă (fără ecou pe ecran)
( )	Folosite pentru "corpul" unei funcţii	%	Se foloseşte pentru a anunţa comentarii în program
[ ]	Pentru desemnarea unei matrice / vector	=	Asignare
.	Indicator de operaţie aritmetică "element cu element"	‘	Transpusa unei matrice
…	Continuarea unei comenzi pe linia de comandă următoare	[,]	Separator între elementele aceleiaşi linii într-o matrice / vector
,	Separator între instrucţiuni pe aceiaşi 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 şi calculator

Operaţii aritmetice simple:

Exemplu:

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

Observaţie***:*** 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

```
--> x=12.3; y = 6*x^3 - 5*x^2 + 3*x-1
y = 1.0445e + 004
```

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

```
--> a = 3; b = 5
--> c = a, a = b, b = c
c = 3, a = 5, b = 3
```

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

```
--> a = 3; b = 5
--> a=a+b, b=a-b, a=a-b
a = 8, b = 3, a = 5
```

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

Se cunoaşte raza sferei (R):
D = 2 * R
A = 4 * π * R^2
V = 4 * π * R^3 / 3

```
--> R = 10; D = 2 * R
D = 20
--> A = 4 * pi * R^2
A = 1256.6
--> V = 4*pi*R^3/3
V = 4188.8
```

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

```
--> E=(sin(pi/6)+cos(pi/4))/(sqrt(5+log(3))-
1.25^0.25)
E = 0.85479
```

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

Introducere:

Matricea reprezintă tipul fundamental de dată în **FreeMat** / **OCTAVE** / **MATLAB**, elementele acesteia având acelaşi tip. La sfârşitul cursului anterior s-au prezentat câteva noţiuni 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 noţiuni teoretice şi exemple a funcţiilor pentru: prelucrarea şirurilor; generarea şi manipularea matricelor; respectiv extragerea unor elemente din şiruri şi matrice.

Pentru atingerea obiectivului propus, cursul este structurat pe trei părţi, în fiecare dintre acestea fiind prezentate noţiuni teoretice (explicaţiile funcţiilor) şi practice (exemple) necesare înţelegerii modului de lucru a funcţiilor prezentate.

Cele mai uzuale operaţii care se efectuează cu şirurile de date sunt: determinarea elementului minim / maxim, determinarea mediei aritmetice, suma / produsul unor elemente care îndeplinesc anumite condiţii, ordonarea crescătoare / descrescătoare a elementelor şirului.

În prima parte sunt prezentate funcţiile **FreeMat** / **OCTAVE** / **MATLAB** pentru prelucrarea şirurilor.

În partea a doua sunt prezentate funcţiile pentru generarea şi manipularea matricelor, fiind prezentate câteva funcţii cu ajutorul cărora pot fi generate matrice particulare.

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

Obiective:

După parcurgerea acestui material, studenţii:

- vor şti să prelucreze şi opereze cu şiruri de numere;
- vor şti să genereze şi să manipuleze matrice;
- vor învăţa să extragă elemente individuale dintr-un vector sau o matrice;

2.1. Prelucrarea şirurilor de date

Cele mai uzuale funcţii utilizate în prelucrarea şirurilor 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 in 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 condiţie.

Exemplu:

Se consideră şirul format din primele 10 numere naturale. Se cere să se determine elementul maxim, minim, suma şi produsul elementelor şirului, precum şi 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 şirul de numere reale: $a_n = \left(1+\frac{1}{n}\right)^n$.

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

Se cere să se calculeze suma primilor 100, 200 şi 300 de termeni ai şirului şi 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
```

Observaţie: Se observă că pe măsură ce creşte numărul de termeni luaţi în considerare, limita se apropie de valoarea lui e.

Exemplu:

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

Se cere să se verifice egalitatea de mai sus, pt. x = 0,5 şi 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 şi manipularea matricelor

O matrice dreptunghiulară cu m linii şi 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: Matricele pot fi generate în mai multe moduri. Cel mai uzual constă în utilizarea parantezelor pătrate şi a separatorilor. Ca separatori între elementele matricei se folosesc blancul sau virgula în interiorul unei linii, respectiv punct şi 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 deţine un set de funcţii 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 funcţii este aceeaşi, astfel că: **zeros(m,n)** generează o matrice cu elemente nule cu m linii şi n coloane; **zeros(n)** generează o matrice cu elemente nule cu n linii şi n coloane (matrice pătratică).

Exemplu:

```
--> zeros(3,2)
ans =
0  0
0  0
0  0
```

```
--> zeros(2)
ans =
0  0
0  0
```

```
--> ones(2)
ans =
1  1
1  1
```

```
--> rand(2,3)
ans =
0.73478  0.88761  0.57471
0.58542  0.32852  0.28011
```

```
--> randn(2,3)
ans =
1.32412  -0.12292  -0.47123
0.32081  -1.00701   0.64009
```

Deoarece **FreeMat / OCTAVE / MATLAB** sunt programe bazate pe prelucrarea matricelor, este foarte important să se cunoască modul de introducere şi de reprezentare ale matricelor (şi 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 şi n coloane. Vectorii linie (m = 1) şi coloană (n = 1) sunt cazuri particulare ale matricelor bidimensionale.

Pentru generarea vectorilor cu pas liniar trebuie să se cunoască limitele intervalului (minimă şi maximă) şi 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) şi numărul de elemente (n) se utilizează funcţia linspace, astfel:

v = 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) şi v_max (maximă) se realizează cu funcţia logspace, astfel:

v = logspace(v_min,v_max,n)

Vectorul conţine 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 doreşte generarea unui vector coloană, elemnetele acestuia trebuie separate prin punct-virgulă (;).

Exemplu:

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

O altă categorie de funcţii 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:

Afişarea unei matrice pe ecran se realizează cu ajutorul funcţiei disp, sub forma disp(A), unde A este matricea pe care dorim să o afişăm. 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 funcţii asociate matricelor sunt : **reshape** - schimbarea dimensiunii; **diag** - matrice diagonale şi diagonale ale matricelor; **blkdiag** - matrice diagonală pe blocuri; **tril** - extragerea părţii triunghiulare inferioare; **triu** - extragerea părţii triunghiulare superioare; **fliplr** - rotire matrice în jurul axei de simetrie verticale; **flipud** - rotire matrice în jurul axei de simetrie orizontale; **rot90** - rotaţia 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)          % afişarea 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 părţii triunghiulare inf.
D1 =
 1     0     0
-2     0     0
 0     2    -3

--> D2=triu(M2)
% Extragerea părţii 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 doreşte să se extragă o succesiune de elemente dintr-un vector se utilizează forma A(i :j), unde : A – reprezintă numele vectorului, i şi j reprezintă indicii elementelor de început, respectiv de sfârşit a subşirului 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 şirul : 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 şi unul pentru coloană) separaţi prin virgulă şi cuprinşi 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 conţină lin.                          2,3,4 şi
col. 1,2,3
C =
-2     0     1
 0     2    -3
-4     1    -1
```

3. Rezolvarea ecuaţiilor algebrice şi transcendente. Polinoame. Rezolvarea sistemelor de ecuaţii liniare şi neliniare. Interpolarea şi aproximarea datelor. Derivarea numerică. Integrarea numerică. Rezolvarea ecuaţiilor diferenţiale ordinare

Introducere*:*

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

În cadrul acestui curs se doreşte prezentarea funcţiilor utile în rezolvarea unor categorii largi de probleme de matematică care apar în calculele din domeniul ingineresc. Cursul este împărţit în şapte părţi, în fiecare dintre acestea fiind prezentate funcţiile cu ajutorul cărora se rezolvă o categorie de probleme. Astfel, în prima parte sunt prezentate funcţiile **roots**, **fzero** şi **solve** pentru rezolvarea ecuaţiilor algebrice şi transcendente. În partea a doua sunt prezentate funcţii şi operatori pentru prelucrarea polinoamelor. În partea a treia sunt prezentate funcţiile utile în rezolvarea sistemelor de ecuaţii liniare şi neliniare. Funcţiile pentru interpolarea şi aproximarea datelor sunt prezentate în partea a patra a cursului. În calculele inginereşti avem nevoie adesea de efectuarea operaţiilor de derivare sau/şi integrare numerică. Funcţiile pentru derivare/integrare numerică sunt prezentate în subcapitolele cinci şi şase ale cursului. O altă categorie de probleme des întâlnită în inginerie o reprezintă rezolvarea ecuaţiilor / sistemelor de ecuaţii diferenţiale ordinare. În ultima parte a cursului sunt prezentate funcţiile pentru rezolvarea ecuaţiilor / sistemelor de ecuaţii diferenţiale.

Obiective:

După parcurgerea acestui material, studenţii vor şti să utilizeze mediile de programare pentru: rezolvarea ecuaţiilor algebrice şi transcendente; realizarea unor operaţii cu polinoame; rezolvarea sistemelor de ecuaţii liniare şi neliniare; interpolarea şi aproximarea datelor; derivarea şi integrarea numerică; rezolvarea ecuaţiilor diferenţiale ordinare;

3.1. Rezolvarea ecuaţiilor algebrice şi transcendente

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

Exemple:

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

Pentru determinarea soluţiilor ecuaţiilor transcendente se utilizează metode de aproximare.

Se consideră funcţia $f : [x_{\min}, x_{\max}] \to \Re$, cu $[x_{\min}, x_{\max}] \subset \Re$.

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

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

Pentru determinarea soluţiilor unei ecuaţii de forma *f(x) = 0* trebuie parcurse două etape:

A. determinarea intervalelor în care se găsesc soluţii: constă în împărţirea intervalului de definiţie al funcţiei în mai multe subintervale astfel încât fiecare interval să conţină cel mult o rădăcină a ecuaţiei.

Pentru stabilirea intervalelor în care se găsesc rădăcinile ecuaţiei *f(x) = 0*, se utilizează teorema conform căreia: dacă o funcţie continuă *f(x)* are valori de semn opus la capetele unui interval *[a,b]* (adică dacă *f(a)*f(b)<0*) atunci în acel interval se găseşte cel puţin o rădăcină a ecuaţiei *f(x) = 0*. Rădăcina este unică în intervalul *[a,b]* dacă derivata funcţiei *f(x)* există şi îşi 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 bisecţiei, metoda tangentei (Newton), metoda secantei, metoda falsei poziţii, metoda aproximaţiilor succesive.

Pentru rezolvarea ecuaţiilor în **FreeMat** / **OCTAVE / MATLAB** se pot utiliza următoarele funcţii:

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

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

Exemplu:

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

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

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

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

unde: **fun** – reprezintă definiţia sau numele funcţiei, **x0** – reprezintă un punct de start, **options** – reprezintă o structură utilizată adiţional pentru specificarea unor opţiuni.

Funcţia **fzero** recunoaşte următoarele opţiuni: "FunValCheck" (verifică valorile invalide ale funcţiei obiectiv, poate avea valorile ‘ on ’ sau ‘ off ’), "OutputFcn" (apelează toate funcţiile de ieşire după fiecare iteraţie), "TolX" (toleranţa parametrului), "MaxIter" (numărul maxim de iteraţii), "MaxFunEvals" (numărul maxim de evaluări ale funcţiei obiectiv – întreg pozitiv).

La ieşire funcţia returnează: **x** – rădăcina aproximativă, **fval** – valoarea funcţiei în x, **info** – un parametru care poate avea valorile: (1) – algoritmul converge către o soluţie, (0) – numărul maxim de iteraţii sau de evaluări a funcţiei a fost depăşit, (-1) – algoritmul a fost întrerupt de către funcţia de ieşire definită de utilizator, (-2) – o eroare neaşteptată, (-3) – s-a întâlnit o valoare nereală, (-4) – s-a întâlnit o valoare NaN (Not – a – number).

Exemplu:

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

```
--> 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 ecuaţiei $x + \log(x) = 0$, în intervalul [0,1].

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

C. Funcţia **fsolve**: se utilizează pentru rezolvarea ecuaţiilor / sistemelor de ecuaţii neliniare. Are forma generală:

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

unde: **fun** – reprezintă definiţia sau numele funcţiei, **x0** – reprezintă un punct de start, **options** – reprezintă o structură utilizată adiţional pentru specificarea unor opţiuni.

La ieşire funcţia returnează: **x** – rădăcina aproximativă, **fval** – valoarea funcţiei în x, **info** – un parametru care poate avea valorile: (1) – algoritmul converge către o soluţie, (2) – pasul relativ este mai mic decât TolX, (3) – ultima valoare relativă descreşte la o valoare mai mică decât TolF, (0) – s-a depăşit numărul maxim de iteraţii, etc.

3.2. Polinoame în FreeMat / OCTAVE / MATLAB

În **FreeMat / OCTAVE / MATLAB** un polinom este reprezentat ca un vector ale cărui componente sunt coeficienţii polinomului aranjaţi în ordine descrescătoare a puterilor. Pentru coeficienţii 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 zeros(1,97) -1 2]`

FreeMat / MATLAB / OCTAVE oferă o serie de funcţii utile în prelucrarea polinoamelor, astfel: **conv** - calculează produsul a două polinoame; **deconv** - calculează câtul şi restul împărţirii a două polinoame; **poly** - calculează coeficienţii 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 fracţii simple raportul a două polinoame; **roots** - calculează rădăcinile unui polinom.

Exemplu:

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

```
--> p = [ 1  -2  1 ] , q = [ 1  -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
```

Soluţia ecuaţiei: 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
```

Funcţii şi operaţii cu polinoame:

Funcţia: polyout(p,"X") : determină afişarea 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 aceeaşi dimensiune, astfel în vectorul care conţine coeficienţii 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
```

Înmulţirea a două polinoame se realizează cu ajutorul funcţiei **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
```

Împărţirea a două polinoame se realizează cu ajutorul funcţiei **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 ecuaţiilor algebrice polinomiale se realizează cu ajutorul funcţiei **roots**:

Exemplu:

```
--> P=[1 3 2]; roots(P)
ans =
-2
-1
```

Cu ajutorul funcţiei **poly(v)** se construieşte 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 funcţiile **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 obţine utilizând funcţia **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 funcţia **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 obţine utilizând funcţia **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 funcţiei **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 şi neliniare

Se consideră următorul sistem de ecuaţii 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 construieşte matricea coeficienţilor necunoscutelor A şi vectorul coloană a termenilor liberi B. Pentru găsirea soluţiei se aplică relaţia: $X = A^{-1}\cdot B$.

Astfel obţinem:

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

În **FreeMat / OCTAVE / MATLAB** se utilizează următoarele funcţii: **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 ecuaţii 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 ecuaţii neliniare

Să se rezolve sistemul de ecuaţii 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 şi aproximarea datelor

Se consideră o colecţie de puncte $(x_0, y_0), \ldots, (x_n, y_n)$. Prin **interpolare** se înţelege determinarea unei funcţii φ - numită funcţie de interpolare, cu proprietatea:

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

Dacă funcţia de interpolară este **polinomială**, metoda se numeşte **interpolare polinomială**. Dacă funcţia de interpolare este **trigonometrică**, metoda se numeşte **aproximare trigonometrică**. În cazul în care funcţia de interpolare este **local polinomială**, metoda se numeşte **interpolare polinomială pe porţiuni** sau interpolare de tip **spline**.

Funcţiile de interpolare disponibile în **FreeMat / OCTAVE / MATLAB** sunt: **table1** - interpolează liniar şi citeşte date din tabele unidimensionale; **spline - i**nterpolează prin metoda **spline** date din tabele unidimensionale; **polyfit - a**proximează un set de date cu un polinom de gradul **n**; **interp1 - i**nterpolează liniar, spline sau cubic date din tabele unidimensionale; **table2** - interpolează liniar date şi citeşte date din tabele bidimensionale; **interp2 -** interpolează biliniar sau bicubic date din tabele bidimensionale; **interp3 -** interpolează biarmonic valorile unei funcţii de două variabile; **interp4 -** interpolează biliniar valorile unei funcţii de două variabile; **interp5** - interpolează bicubic valorile unei funcţii de două variabile; **regress** - aproximează un set de date cu o funcţie 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 funcţia **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 variaţia 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

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

Exemplu:

Să se calculeze derivatele partiale ale functiei:
z(x,y) = x*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 functia **f(x,y)=3*x^3+2*y^2** pe intervalul **[-4,4] x [-4,4]**

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

3.6. Integrarea numerică

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

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

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

unde: *nume_fisier* reprezintă un şir de caractere care defineşte numele fişierului-funcţie în care a fost scrisă expresia funcţiei de integrat f; *a* şi *b* reprezintă limitele de integrare (capetele intervalului [a,b] pe care se realizează integrarea); *precizia* este un argument opţional prin care se poate modifica precizia implicită 10^{-6}; *I* reprezintă aproximarea integralei definite

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

Funcţia **trapz** utilizează funcţia 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* şi *y* reprezintă vectorii valorilor funcţiei.

Exemplu:

Să se calculeze $I = \int_0^4 \frac{\sin(x)}{x} \cdot dx$ prin **metoda trapezelor** şi 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
```

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

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

unde: *f* reprezintă o funcţie de două variabile; *xa*, *ya*, *xb*, *yb* reprezintă limitele de integrare pentru x şi y; *tol* reprezintă toleranţa absolută utilizată;

Exemplu:

Să se calculeze integrala dublă: $I = \int_{0}^{1}\int_{0}^{1} \sin(\pi\cdot x\cdot y)\cdot\sqrt{x\cdot y}\,dxdy$

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

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

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

Forma generală a funcţiei poate fi:

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

parametri funcţiilor având aceeaşi semnificaţie ca şi la funcţia dblquad.

Exemplu:

Să se calculeze integrala triplă:

$$I = \int_{0}^{1}\int_{-3}^{-1}\int_{-2}^{2} (x^2y^2z + 2z^2x)\,dxdydz$$

```
--> 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 ecuaţiilor diferenţiale ordinare

Funcţiile oferite de **FreeMat / OCTAVE / MATLAB** pentru rezolvarea ecuaţiilor (sistemelor de ecuaţii) diferenţiale sunt: **ode23 -** rezolvă ecuaţiile diferenţiale sau sisteme de ecuaţii diferenţiale prin metoda Runge-Kutta de ordinul 2 (3), respectiv **ode45 -** rezolvă ecuaţii diferenţiale sau sisteme de ecuaţii diferenţiale prin metoda Runge-Kutta de ordinul 4 (5).

Cele două funcţii 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 funcţiei OCTAVE / MATLAB (ode23 sau ode45);
- *nume_fisier* reprezintă un şir de caractere care conţine numele fişierului-funcţie în care a fost definită expresia derivatei funcţiei-necunoscute. *Nume_fisier* conţine vectorul expresiilor derivatelor de ordin I al funcţiilor-necunoscute în cazul unei ecuaţii diferenţiale de ordinul I, sistemelor de ecuaţii diferenţiale de ordinul I sau al ecuaţiilor şi sistemelor de ecuaţii diferenţiale 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 funcţiei-necunoscute din condiţia iniţială în cazul unei ecuaţii diferenţiale de ordinul I, respectiv, vectorul valorilor funcţiilor-necunoscute din condiţiile iniţiale;
- *optiuni* reprezintă o structură care conţine opţiuni de optimizare a calculării soluţiei / soluţiilor (opţional);
- *xval* reprezintă un vector ce conţine valorile variabilei independente, în care se determină valorile soluţiei / soluţiilor;
- *yval* reprezintă vectorul valorilor funcţiei soluţie în punctele definite prin *xval*.

Să se integreze ecuaţia diferenţială $y' = 3 \cdot t^2 - 4 \cdot t + 2$ pe intervalul [1,4], cu condiţiile iniţiale y(2) = 0,5.
Se editează mai întâi fişierul care conţine definiţia funcţiei de integrat:

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

apoi pe prompter se scrie următoarea secvenţă

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

Exemplu:

Se consideră un sistem mecanic vibrant format dintr-o masă **m** şi un element elastic, asupra masei acţionând o forţă armonică (figura 4.1):

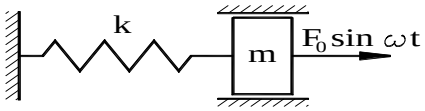


Figura 3.1. Sistem mecanic vibrant

Ecuaţia diferenţială a mişcării este: $m\ddot{x} + kx = F_0 \cdot \sin \omega t$ a cărei soluţie 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 condiţiile iniţiale.

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

Ecuaţia diferenţială de ordinul n trebuie transformată într-un sistem de n ecuaţii diferenţiale de ordinul întâi. Acest sistem va fi introdus într-un fişier de tip funcţie care va returna derivatele ecuaţiilor sistemului. Astfel, ecuaţia diferenţială a fost scrisă ca un sistem cu două ecuaţii diferenţiale de ordinul întâi:

$$yd1 = y(2);$$

$$yd2 = \frac{F_0 \cdot \sin(\omega \cdot x) - k \cdot y(1)}{m}$$

Fişierul funcţie scris pentru această aplicaţie 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 funcţia 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 şi 3D

Introducere:

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

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

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

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

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

În prima parte a cursului sunt prezentate funcţiile pentru realizarea reprezentărilor bidimensionale, descrierea fiecărei funcţii fiind însoţită de exemple.

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

Obiective:

După parcurgerea acestui material, studenţii vor şti 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 suprafeţelor; realizarea reprezentărilor tridimensionale a funcţiilor dependente de două variabile;

4.1. Grafica 2D

Cu ajutorul funcţiilor dedicate, aplicaţiile **FreeMat** / **OCTAVE** / **MATLAB** permit realizarea şi modificarea cu uşurinţă a graficelor şi figurilor.

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

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

Denumire:	Semnificaţie:
plot	Desenează în coordonate carteziene graficul 2D al unei funcţii y=f(x)
polar	Desenează în coordonate polare graficul 2D al unei funcţii
bar	Desenează un grafic cu bare 2D
subplot	Permite desenarea mai multor ferestre grafice pe acelaşi ecran
semilogx	Desenează în coordonate semilogaritmice (logaritmice pe direcţia Ox) graficul 2D
semilogy	Desenează în coordonate semilogaritmice (logaritmice pe direcţia Oy) graficul 2D
loglog	Desenează graficul 2D în coordonate logaritmice (pe Ox şi Oy)
compass	Desenează imagini de numere complexe sub formă trigonometrică
grid	Adauga graficului o reţea 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 poziţii arbitrare
text	Realizează plasarea unui text pe grafic în poziţia impusă
hold	Realizează îngheţarea graficului curent pe ecran
clf	Şterge fereastra grafică curentă

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

v), triunghi cu vârf in 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 (--).

Funcţia **plot** : este o funcţie de bază utilizată în generarea graficelor 2D în coordonate carteziene. Forma generală a funcţiei este :

```
plot (x, y, 'şir')
```

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

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

Exemplu:

Construim graficul funcţiei 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
```

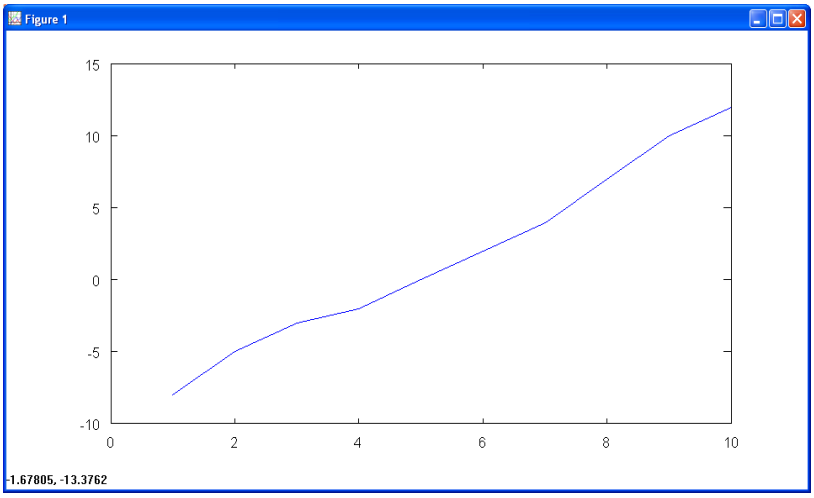


Figura 4.1

Exemplu:

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

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

Tabelul 4.2. Valorile funcţiei $y = f(x) = 3 \cdot x^2 - 4 \cdot x + 2$ 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 **x** şi **y** în care vom plasa valorile coordonatelor punctelor:
x = [-4 -3 -2 -1 0 1 2 3],
respectiv y = [66 41 22 9 2 1 6 17];
3. Vom trasa graficul funcţiei utilizând comanda **plot** cu forma **plot(x,y)** (figura 4.2);

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

x =
-4 -3 -2 -1 0 1 2 3
y =
66 41 22 9 2 1 6 17

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

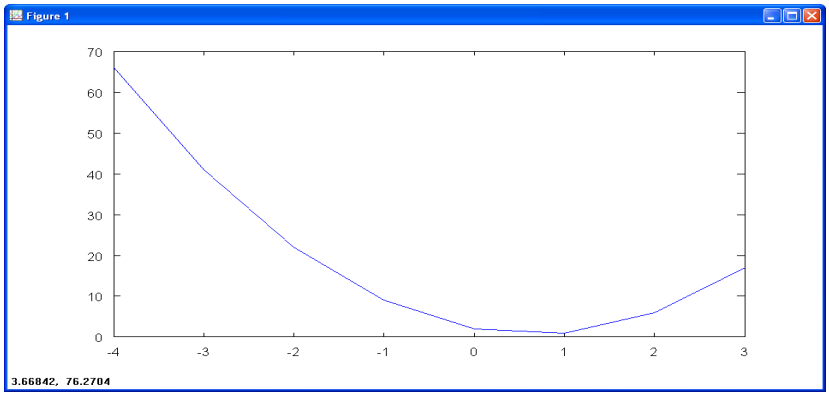


Figura 4.2.

Exemplu:

Trasarea graficului funcţiei 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
```

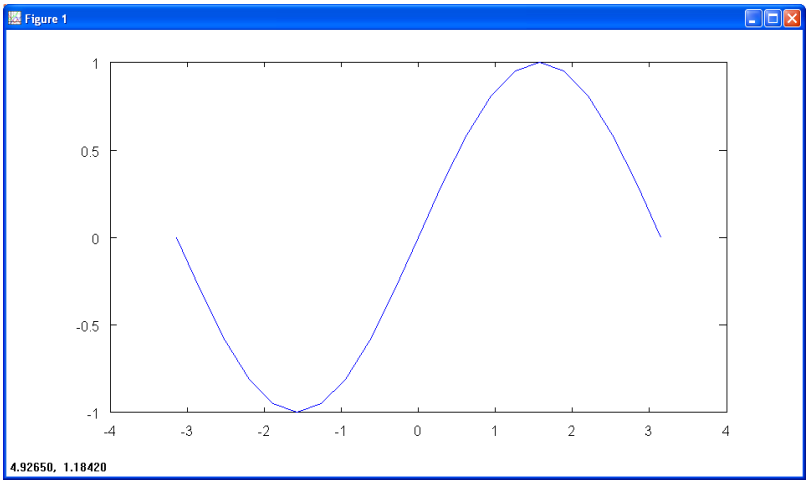


Figura 4.3

```
-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 funcţiei sin(x), modificând modalitatea de trasare a graficului, astfel vom alege următoarele caracteristici: tipul liniei: linie punctată (.), culoarea liniei: roşu (r), tipul marcatorului pentru puncte: cerc (o) (Figura 4.4).
Sintaxa comenzii este:

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

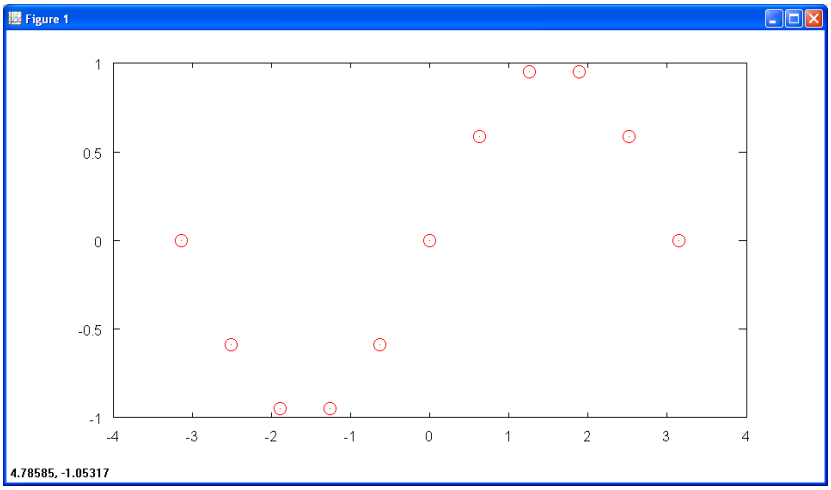


Figura 4.4.

Exemplu:

Vom trasa graficele funcţilor sin(x) şi cos(x) pe acelaşi grafic. Graficele vor avea următoarele caracteristici: graficul funcţiei sinus se va trasa cu linie continuă (-), de culoare albastră (b) iar punctele vor fi evidenţiate prin cruciuliţe (x); graficul funcţiei cosinus va fi trasat cu linie punctată (.), de culoare roşie (r), iar punctele vor fi marcate prin cerculeţe (o) (Figura 4.5).
Sintaxa comenzii **plot** este:

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

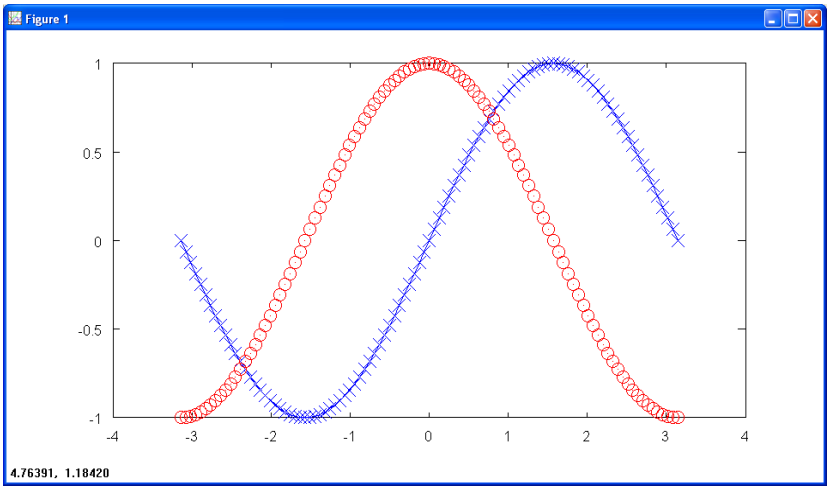


Figura 4.5.

Exemplu:

În figura 4.6 este ilustrat graficul unei curbe numită cardioidă, a cărei ecuaţie 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)
```

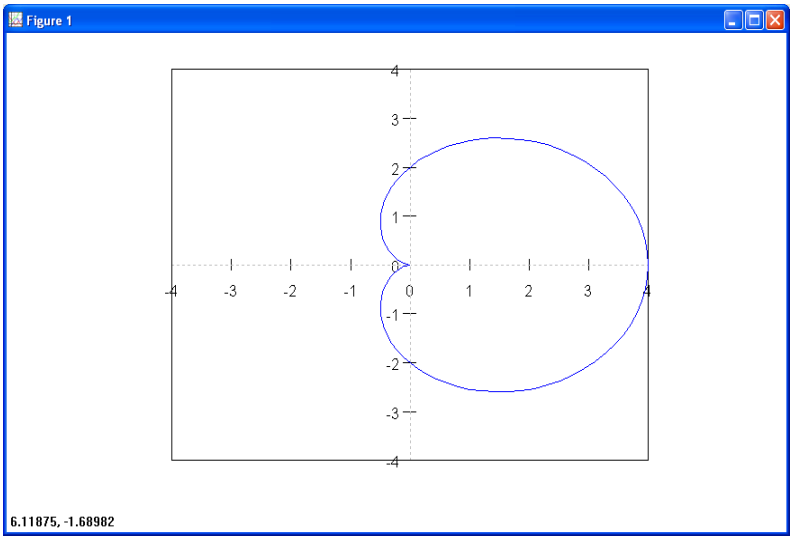


Figura 4.6

Comanda **plot** acceptă şi argumente date sub formă matricială. În practică se pot întâlni patru situaţii posibile:
- ambele argumente (**x** şi **y**) sunt vectori, prin urmare se trasează graficul cu valorile primului vector (**x**) pe abscisă şi cu cele ale celui de-al doilea vector (**y**) pe ordonată;
- **x** este un vector de dimensiune **m** şi **y** este o matrice **m x n**, comanda plot(x,y) suprapune graficele obţinute din **x** şi fiecare coloană (sau rând) din matricea **y**;
- **x** este o matrice **m x n** şi **y** este un vector de dimensiune **m**, comanda plot(x,y) suprapune graficele obţinute din fiecare coloană (sau rând) din matricea **x** şi vectorul **y**;
- ambele argumente sunt matrice, situaţie în care sunt suprapuse graficele obţinute din coloanele primului argument versus coloanele celui de al doilea argument. În acest caz cele două matrice trebuie să aibă acelaşi număr de linii şi coloane.

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

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

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

Exemple:

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

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

```
polar(t,r, 'şir')
```

unde **t** reprezintă unghiul polar iar **r** reprezintă raza polară, **şir** are aceeaşi semnificaţie ca şi la funcţia **plot**.

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

```
bar(x,y,'şir')
```

unde argumentele **x**, **y** şi **şir** au aceeaşi semnificaţie ca şi în cazul funcţiei **plot**.

Exemplu:

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

Reprezentarea cu ajutorul funcţiei **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)
```

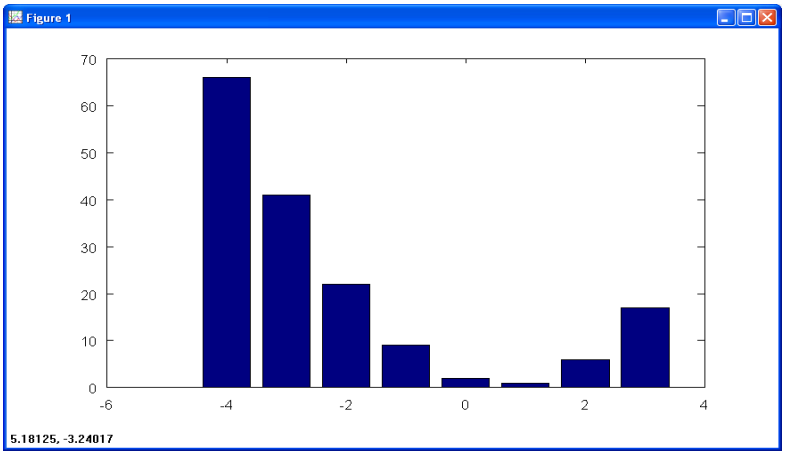


Figura 4.7

Funcţiile **grid**, **legend**, **title**, **xlabel**, **ylabel**, **axis**, **text**: acţiunea fiecărei funcţii este explicată în tabelul 4.1.

Exemplu:

Reprezentarea graficului funcţiei sinus pe un anumit interval (figura 4.8). Efectuaţi pe rând fiecare din comenzile de mai jos şi vizualizaţi după fiecare comandă graficul. Observaţi 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)
```

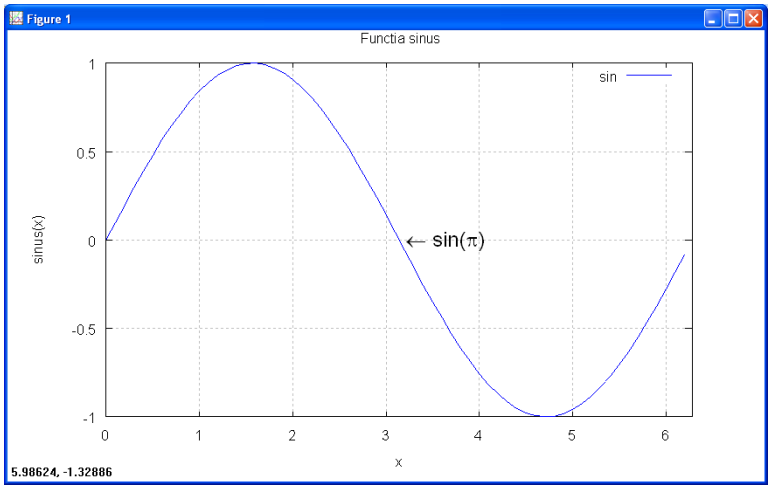


Figura 4.8.

Funcţiile **semilogx**, **semilogy**, **loglog**: aceste funcţii se utilizează pentru reprezentări în coordonate logaritmice sau semilogaritmice. Funcţia loglog scalează ambele axe utilizând logaritmul în baza 10, iar funcţiile semilogx şi semilogy scalează logaritmic numai axa x, respectiv y în timp ce cealaltă axă este scalată liniar.

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

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

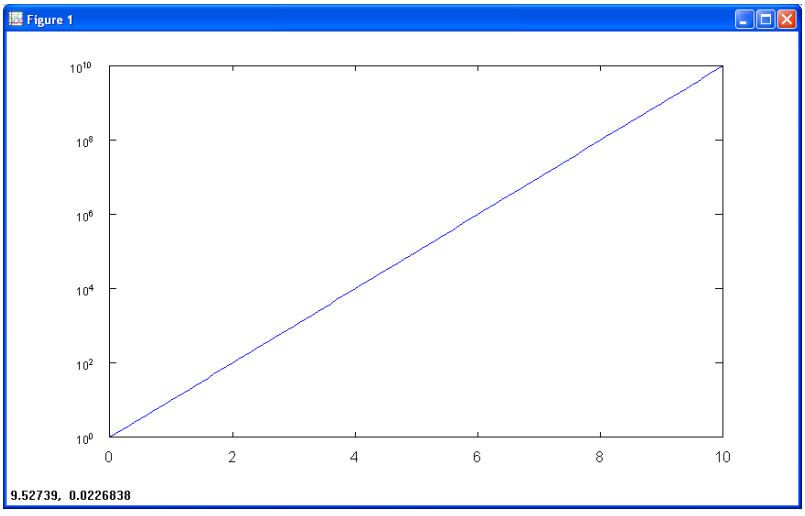


Figura 4.9.

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

Exemplu:

Vom reprezenta într-o singură fereastră funcţiile sinus, cosinus şi 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))
```

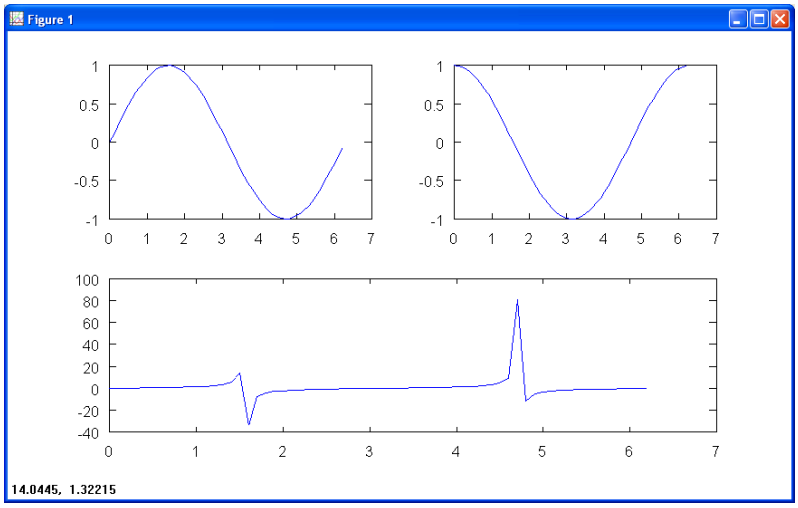


Figura 4.10

4.2. Grafica 3D

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

Tabelul 4.3. Funcţiile OCTAVE pentru generarea graficelor 3D

Denumire:	Descriere:
plot3	Generează graficul 3D al unei funcţii, analog cu funcţia plot
bar3	Generează un grafic cu bare 3D
cylinder	Generează un cilindru
sphere	Desenează o sferă
ellipsoid	Desenează un elipsoid de rotaţie
surf	Desenează suprafeţe 3D colorate

surfc	Desenează suprafeţe cu contur
surfnorm	Desenează suprafeţe 3D şi normalele la ele
contour	Desenează proiecţiile intersecţiilor cu plane paralele cu xOy, în planul xOy
meshgrid	Generează o diviziune 2D
mesh	Realizează graficul unei suprafeţe în 3D
meshc	Realizează graficul unei suprafeţe cu contur în 3D
meshz	Graficul unei suprafete cu plan de referinţă la cota z (piedestal)
waterfall	Produce un mesh de tip “waterfall”
comet3	Grafic 3D animat (efect de cometă)
quiver	Desenează vectorii tangenţi la o suprafaţă

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

Forma generală a funcţiei este:

```
plot3(x,y,z,'şir')
```

unde: **x**, **y**, **z** reprezintă vectori (matrici) de aceeaşi dimensiune ce conţin coordonatele **x**, **y**, **z** ale punctelor prin care trece linia, iar **şir** are aceeaşi semnificaţie ca şi în cazul funcţiei plot.

În cazul în care argumentele *x,y,z* ale funcţiei sunt matrice, atunci se obţine 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')
```

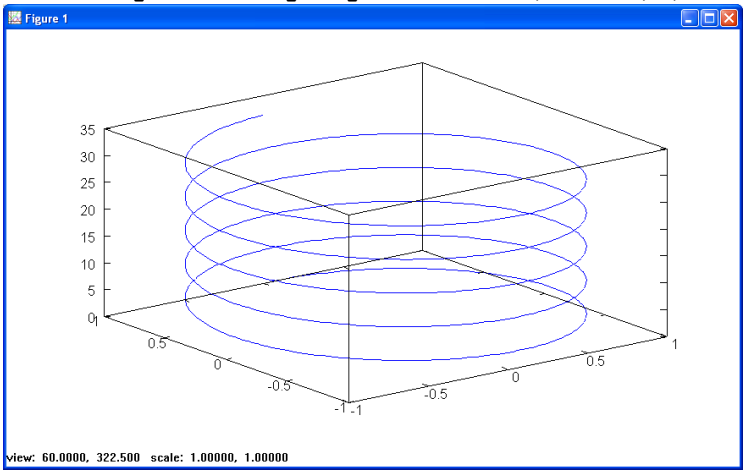


Figura 4.11. Reprezentarea tridimensională a unei spirale

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

Funcţiile **mesh, meshc, meshz**: se utilizează pentru reprezentarea suprafeţelor sub forma unor reţele. Sintaxele funcţiilor sunt:

```
mesh(X, Y, Z)          mesh(Z)                meshc( … )
   mesh(X, Y, Z, C)       mesh(Z, C)          meshz( … )
```

În cazul cel mai general funcţia **mesh** are patru matrice ca parametri de intrare şi reprezintă grafic suprafaţa *Z(X, Y),* respectiv matricea culorilor *C*.

De cele mai multe ori *X* şi *Y* sunt vectori. Pentru ca reprezentarea să fie corectă, aceştia trebuie să fie ordonaţi crescător şi cu pas constant.
În cazul în care argumentele *X* şi *Y* lipsesc, reprezentarea se realizează pe baza indicilor matricei *Z*.

Dacă matricea *C* lipseşte, se consideră *C* = *Z*, adică culoarea este proporţională cu înălţimea suprafeţei.

Poziţia din care este observată suprafaţa reprezentată grafic poate fi controlată prin funcţia **view**.

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

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

Funcţia **meshc** are formă asemănătare cu funcţia **mesh** şi permite reprezentarea 3D a suprafeţelor cărora li se asociază liniile de contur, trasate ca proiecţii în planul bazei.

Funcţia **meshz** se utilizează asemănător cu **mesh** şi **meshc** şi permite reprezentarea 3D a suprafeţelor, însă se trasează suplimentar un plan de referinţă la valoarea minimă (piedestal).

Exemplu:

Reprez. funcţiei $z = x \cdot e^{\left(-x^2-y^2\right)}$ pt. domeniul D=[-2,2]x[-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)
```

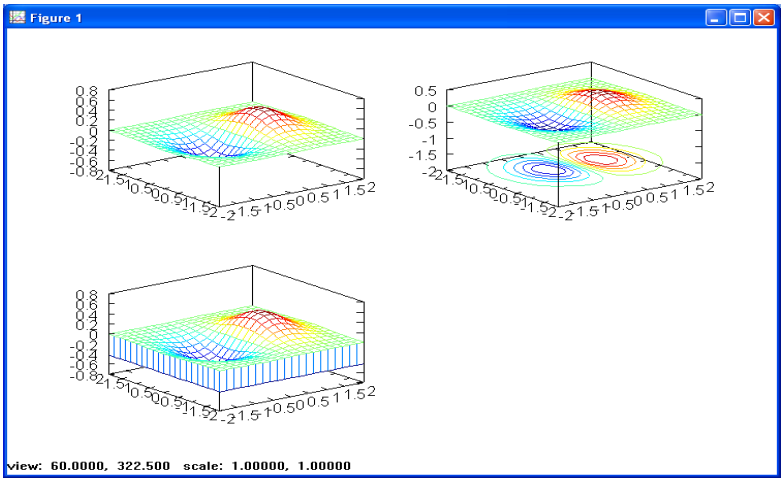


Figura 4.12. Exemple de utilizare a funcţiilor mesh, meshc şi meshz

Funcţia **cylinder**: se utilizează pentru generarea unui cilindru. Forma generală a funcţiei este:

```
[x,y,z] = cylinder(r,n)
```

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

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

```
[x, y, z] = sphere(n)
```

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

Exemplu:

Să se reprezinte un cilidru cu razele rinf = 1, rsup = 0.5, înălţimea h = 3 şi 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);
```

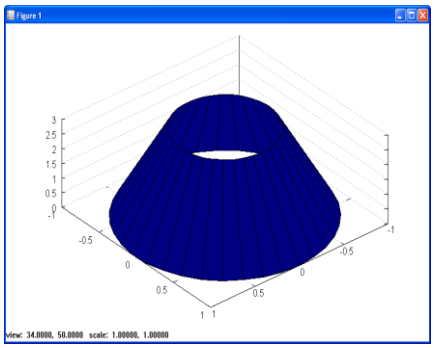


Figura 4.13. Reprezentarea unui cilindru

Exemplu:

Reprezentarea unei sfere:

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

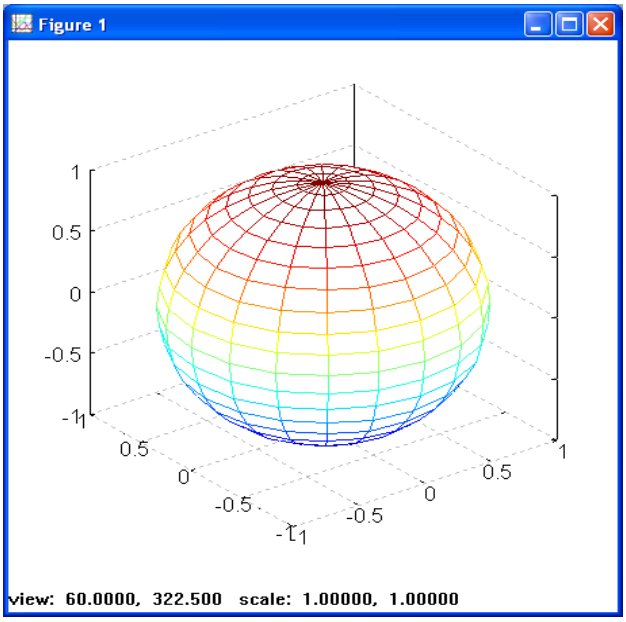


Fig. 4.14. Reprezentarea unei sfere

5. Declaraţii şi variabile. Formatul datelor de ieşire. Operatori. Instrucţiuni de introducere şi extragere de date. Fişiere script. Fişiere funcţie. Instrucţiuni de decizie.

Introducere:

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

Prin multitudinea de funcţii puse la dispoziţia utilizatorilor şi programatorilor pot fi realizate relativ uşor aplicaţii diverse care să uşureze munca inginerilor din domeniul tehnic şi nu numai.

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

În prima parte a cursului sunt prezentate noţiuni legate de forma generală a unui program **FreeMat / OCTAVE / MATLAB**, noţiuni despre tipurile de date, declaraţii de variabile, operatori precum şi despre instrucţiuni de intrare / ieşire.În continuare sunt prezentate noţiuni despre fişierele de tip script şi funcţie. Noţiunile sunt însoţite de exemple.

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

Scopul acestui curs este acela de a prezenta noţiunile de bază privind programarea în mediile de programare **FreeMat / OCTAVE / MATLAB**. De asemenea, prin parcurgerea acestui curs studenţii vor învăţa să scrie programe simple în care vor folosi apeluri de funcţii uzuale şi instrucţiuni de decizie.

Obiective:

După parcurgerea acestui material, studenţii vor şti să utilizeze mediile de programare OCTAVE / MATLAB pentru:

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

- crearea fişierelor de tip funcţie cu ajutorul cărora pot realiza funcţii pentru rezolvarea unor probleme punctuale. Prin crearea de funcţii noi se îmbogăţeşte mediul de programare şi astfel pot fi create noi biblioteci de funcţii;
- rezolvarea unor probleme simple de programare (în care se efectuează numai calcule) sau a unor probleme în care se utilizează şi instrucţiunile de decizie;

De asemenea, studenţii vor cunoaşte:

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

5.1. Etapele de rezolvare a unei probleme

A. **Definirea prolemei:** necesită cunoaşterea problemei în cele mai mici detalii. În această etapă se determină care sunt informaţiile sau datele de intrare, modelul matematic care trebuie aplicat şi care sunt informaţiile sau datele de ieşire pe care le va furniza programul.

B. **Întocmirea algoritmului de calcul:** În această etapă se analizează problema de rezolvat, se determină algoritmii cunoscuţi care pot fi utilizaţi sau se indică pas cu pas rezolvarea.

C. **Scrierea programului:** se editează programul (sub forma unui fişier 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 şi validarea rezultatelor:** Se rulează programul cu o serie de date de intrare şi se verifică rezultatele intermediare şi cele finale dacă acestea se cunosc.

5.2. Declaraţii şi variabile

FreeMat / OCTAVE / MATLAB este un limbaj de expresii. Expresiile tipărite de utilizator sunt interpretate şi evaluate. Instrucţiunile **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, funcţii şi variabile. Evaluarea unei expresii are ca rezultat o matrice, care este afişată pe ecran şi atribuită unei variabile. Dacă numele variabilei şi semnul egal nu sunt precizate, rezultatul se atribuie automat unei variabile cu numele “ans”.

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

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

Numele de variabile sau funcţii 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 şi mici, cu ajutorul funcţiei **casesen (on / off)** se realizează setarea în modul senzitiv / nesenzitiv.

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

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

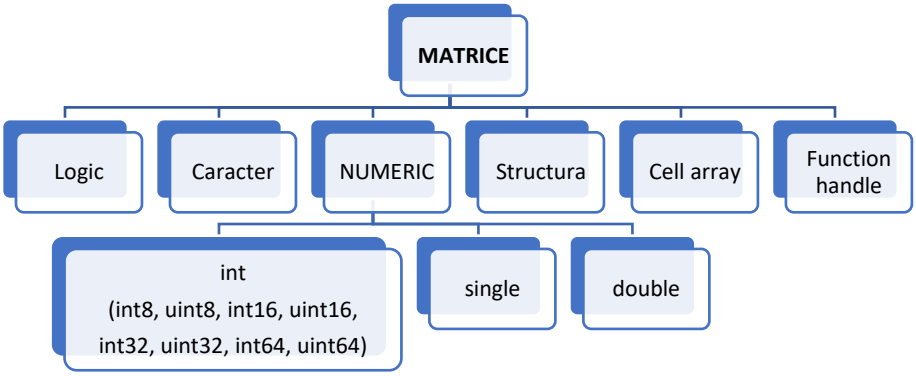


Figura 5.1. Tipurile de date din OCTAVE / MATLAB

Tipurile **structură** sau **celule de tablouri** pot să conţină date de tipuri diferite. Cu ajutorul claselor (uses classes şi 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:	Biţi	Domeniu de valori:
int8	Întreg cu semn pe 8 biţi	8	[-128, 127]
uint8	Întreg fără semn pe 8 biţi	8	[0, 255]
int16	Întreg cu semn pe 16 biţi	16	[-32,768, 32,767]

uint16	Întreg fără semn p 16 biţi	16	[0, 65,535]
int32	Întreg cu semn pe 32 biţi	32	[-2,147,483,648,2,147,483,647]
uint32	Întreg fără semn p 32 biţi	32	[0, 4,294,967,295]
int64	Întreg cu semn pe 64 biţi	64	[-9,223,372,036,854,775,808, 9,223,372,036,854,775,807]
uint64	Întreg fără semn p 64 biţi	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 şir de caractere	16	
function handle	Pointer la o funcţie.		
Structure	Structura este o colecţie 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 ieşire

Cu ajutorul funcţiei **format** se poate preciza formatul de afişare a datelor de ieşire. Prin intermediul acestei funcţii se specifică modul de afişare a datelor şi nu precizia acestora.

Forma generală a funcţiei este:

```
format opţiune
```

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

Tabelul 5.2

Opţiune:	Mod de afişare:	Exemplu:

short	reprezentare în virgulă fixă cu 5 cifre	`--> a=123456789.123456789012345678`
		`--> 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 şi mobilă cu 5 cifre	`--> format short g`
		`--> a`
		`a = 1.2346e+008`

long g	cea mai bună dintre reprezentările în virgulă fixă şi 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 şi nule; partea imaginară este ignorată`	

bank	reprezentare pentru simboluri valutare (dolari şi cenţi)	`--> format bank`
		`--> a`
		`a = 123456789.12`

rat	aproximare printr-o fracţie a unui întreg	`--> format rat`
		`--> 22.33`
		`ans = 2233/100`

5.4. Operatori

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

Operatori aritmetici sunt: + (adunare), - (scădere), * (înmulţire), / (împărţire la dreapta), \ (împărţire la stânga), ^ (ridicare la putere), ‘ (transpusa complex conjugată), () (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 : & (şi); **|** (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 şi matrice
3. plus unar (+), minus unar (-), negare logică (~)

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

5.5. Instrucţiuni de introducere şi extragere de date

Funcţii pentru citirea de la tastatură / afişarea pe ecran

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

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

Funcţia afişează pe ecran şirul de caractere **'sir_de_caractere'** şi atribuie valoarea răspunsului oferit de utilizator varabilei **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
```

Funcţia **disp** – realizează afişarea datelor sau a şirurilor de caractere.

Exemplu:

Afişarea valorii citire anterior:

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

Funcţii pentru scrierea / citirea fişierelor

Funcţia **save** – permite scrierea datelor într-un fişier.

Exemplu:

Scrierea unei matrice într-un fişier. Vizualizaţi conţinutul fişierului 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
```

Funcţia **load** – se realizează pentu citirea datelor din fişiere (citirea matricelor din fisiere binare sau a valorilor din fişiere text)

Exemplu:

Citirea unei matrice dintr-un fişier (matricea din fişierul creat anterior):

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

Funcţii pentru crearea(deschiderea) / închiderea fişierelor

Funcţia **fopen** – se utilizează pentru crearea unui fişier, forma generală a funcţiei fiind:

fopen(nume, mod)

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

- **mod** reprezintă modalitatea de creare / deschidere a fişierului, astfel: ‘r’ – deschide un fişier pt. citire, ‘w’ – deschide un fişier pentru scriere, ‘a’ – deschide sau crează un fişier pentru adăugare la sfârşitul acestuia, ‘r+’ deschide un fişier pt. citire / scriere, ‘w+’ deschide un fişier pt. citire / scriere conţinutul fişierului înainte de deschidere se va pierde.

Funcţia **fclose** – se utilizează pentru închiderea unui fişier.

Funcţii de intrare / ieşire în stilul limbajului C

Funcţia **puts(string)** – afişează pe ecran un şir de caractere.

Funcţia **fputs(fid,string)** – scrie un şir de caractere într-un fişier.

Funcţia **fgets(fid,len)** - citeşte un şir de caractere cu lungimea len, iar dacă aceasta nu este precizată citirea se face până se întâlneşte caracterul « new-line » sau EOF (end–of-file).

Funcţia **printf(sir_format,...)** se utilizează pentru afişarea datelor sub controlul formatului de afişare.

Funcţia **fprintf(fid,sir_format,...)** este asemănătoare cu funcţia printf, cu deosebirea că scrierea se face în fişierul definit de identificatorul **fid**.

Funcţia **sprintf(sir_format,...)** este o altă formă a funcţiei printf, deosebirea constând în faptul că afişarea se realizează sub forma unui şir de caractere.

Argumentul **sir_format** poate conţine: şiruri de caractere, secvenţe ESCAPE şi descriptori de format.

Şirurile de caractere sunt afişate aşa cum sunt scrise, fără să sufere modificări.

Secvenţele ESCAPE încep cu caracterul backslash ('\') şi determină execuţia unei acţiuni, 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][lăţime][.precizie] tip
```

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

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

tip : reprezintă o literă care specifică tipul conversiei aplicate datelor la afişare, nu poate lipsi din specificatorul de format. Pentru **tip** se folosesc: **c** – afişarea unui caracter; **s** – afişarea unui şir de caractere; **i, d** – afişarea unui număr întreg (în baza zece) cu semn; **u** – afişarea unui număr întreg (în baza zece) fără semn; **f** – afişarea unui număr real cu semn - notaţie zecimală; **e, E** – afişarea unui număr real cu semn (notaţie exponenţială); **g** – afişarea unui număr real (cea mai scurtă reprezentare între f şi e); **x** – afişarea unui număr hexazecimal întreg fără semn; **o** – afişarea unui număr octal întreg fără semn;

flags : Poate fi unul din caracterele: **+** (plus): se afişează semnul datei (plus sau minus); **-** (minus): data este aliniată la stânga în câmpul de scriere, dacă lipseşte alinierea se face la dreapta; **_** (spaţiu): valorile pozitive sunt scrise fără semn lăsându-se un spaţiu în faţa valorii afişate, iar în cazul celor negative data se scrie cu semn (semnul minus) fără a se lăsa nici un spaţiu; **#** (diez): determină folosirea formei alternative de scriere, pentru datele octale şi hexazecimale se scrie un zero nesemnificativ, respectiv 0x sau 0X în cazul datelor hexazecimale.

lăţime : constă dintr-un număr întreg care specifică dimensiunea minimă a câmpului în care va fi afişată data. Dacă sunt necesare mai multe poziţii, data va fi afişată pe câte poziţii trebuie. Dacă data are mai puţine caractere ea va fi aliniată la dreapta în câmpul de afişare. Spaţiile rămase libere se completează cu spaţiu. Dacă se plasează un 0 (zero) înaintea cifrei ce reprezintă lăţimea, spaţiile 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 afişează; pentru %c nu are efect.

Observaţii: *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 afişate pe ecran aşa cum apar ele. Între specificatorii de format şi parametri trebuie să existe o concordanţă de număr, ordine şi tip.*

Exemplu:

Într-un fişier scrieţi următoarele instrucţiuni :

```
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 execuţie 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. Fişiere script

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

Fişierele care conţin instructiuni **FreeMat / OCTAVE / MATLAB** poartă numele de fişiere **M** sau **M-files**, denumirea provenind de la extensia acestora, adică extensia ***.m***. În **FreeMat / OCTAVE / MATLAB** un fişier **M** poate fi de două tipuri: ***script*** sau ***funcţie***. Cu ajutorul fişierelor **M** pot fi create noi funcţii care le pot completa pe cele existente.

Fişierele script: sunt fişiere de tip text în care sunt înşiruite secvenţe de comenzi **FreeMat / OCTAVE / MATLAB**. Prin apelarea numelui fişierului, se realizează execuţia secvenţele de comenzi **FreeMat / OCTAVE / MATLAB** conţinute în fişier. Variabilele utilizate în cadrul fişierelor script rămân în spaţiul de lucru (workspace) după execuţia completă a acestuia.

Aceste fişiere 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 instrucţiuni iar introducerea acestora de la tastatură ar fi greoaie.

5.7. Exemple de fişiere script OCTAVE / MATLAB

Exemplu:

Inversarea valorilor a două variabile

Se deschide un fişier **m** nou. Se scrie următoarea secvenţă de program:

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

Se lansează în execuţie programul (comanda Run - Run sau tastaţi F5). Execuţia programului va fi următoarea:

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

Exemplu:

Să se scrie un program pentru calculul diametrului, ariei şi volumului unei sfere pentru care se cunoaşte raza R. Fişierul m va avea următorul conţinut:

```
% 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 execuţiei 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 funcţiei $y = f(x) = x^3 - 4x^2 + 6x - 3$, unde $x \in [a,b]$, parcurs cu pasul *h*. Să se reprezinte grafic funcţia 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. Fişiere funcţie

Fişierele funcţie sunt utilizate pentru dezvoltarea aplicaţiilor mai complexe, sau pentru "îmbogăţirea" mediului de programare cu funcţii. Forma generală a unei funcţii este:

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

unde: **function** reprezintă un cuvânt cheie care specifică faptul că fişierul este de tip funcţie; **nume_functie** reprezintă numele funcţiei, adică numele sub care se salvează fişierul; **param_ieşire** reprezintă o listă de parametri, separaţi prin virgulă (,) şi cuprinşi între paranteze drepte ([ ]); **param_intrare** reprezintă o listă de argumente, separate prin virgulă şi cuprinse între paranteze rotunde. Din punct de vedere al existenţei valorilor returnate sau a argumentelor se pot întâlni 4 categorii de funcţii:

A. Funcţii care nu returnează valori şi nu au parametrii:

Exemplu:

Funcţie care afişează un text predefinit :

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

Apelul funcţiei şi rezultatul apelării fiind:

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

B. Funcţii care nu returnează valori dar au parametrii:

Exemplu:

Funcţie care afişează un număr impus de steluţe

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

Apelul funcţiei şi rezultatul fiind:

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

C. Funcţii care returnează valori dar nu au parametri

Exemplu:

Funcţie care citeşte un număr strict pozitiv

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

Apelul funcţiei şi 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. Funcţii care returnează valori şi au parametri

Exemplu:

Funcţie care citeşte un număr mai mare decât o valoare impusă

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

Apelul funcţiei şi rezultatul fiind:

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

5.9. Exemple de fişiere funcţie în OCTAVE / MATLAB

Exemplu:

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

Se va construi o funcţie pentru calculul valorii lui *y* şi va fi salvată într-un fişier f1.m:

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

Programul care utilizează funcţia f1

```
% Program care utilizeaza functia f1
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 = f1(x)
plot(x,y)
grid on
```

Apelul funcţiei şi rezultatul fiind:

```
--> Introduceti valoarea lui a: a = -10
Introduceti valoarea lui b: b = 10
Introduceti 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. Instrucţiuni de decizie.

5.10.1. Instrucţiunea if ... end

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

if (expresie)
instrucţiune
end

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucţiune** poate fi o instrucţiune simplă sau o instrucţiune compusă.
Mod de lucru: se evaluează **expresie**, dacă are valoare diferită de zero (adevărată) se execută **instrucţiune**, iar dacă are valoare nulă (falsă) nu se execută **instrucţiune**.

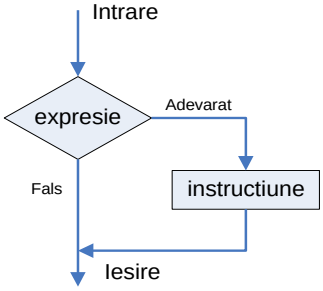


Figura 5.2.

5.10.2. Instrucţiunea if … else ... end

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

```
if (expresie)
        instrucţiune_1
else
        instrucţiune_2
end
```

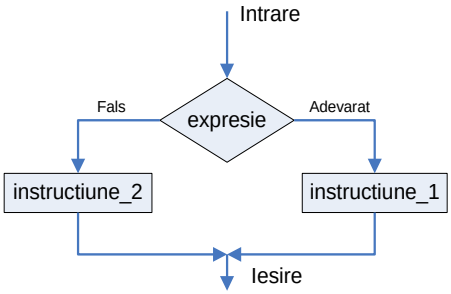


Figura 5.3.

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucţiune_1** şi **instrucţiune_2** pot fi instrucţiuni simple sau compuse.
Mod de lucru: se evaluează **expresie**; dacă are valoare diferită de zero (adevărată) se execută **instrucţiune_1**, iar dacă are valoare nulă (falsă) se execută **instrucţiune_2**.

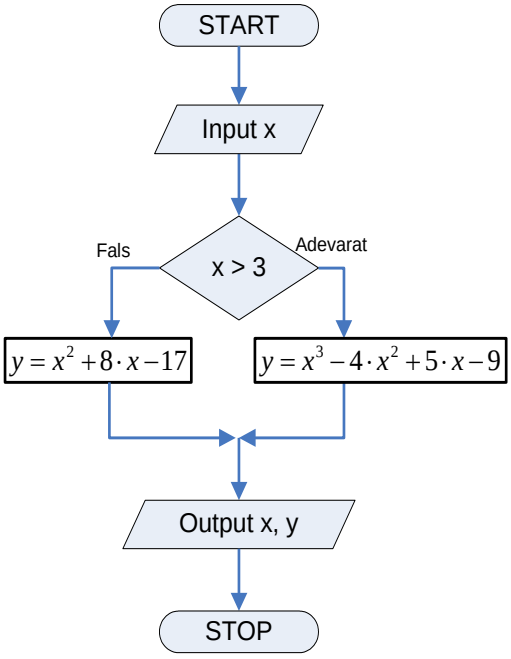
Exemplu:

Calculul valorilor funcţiei

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

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. Instrucţiuni if ... else suprapuse

Au forma generală :

```
if (expresie_1)
        if (expresie_2)
              instrucţiune_1
        else
              instrucţiune_2
        end
else
       instrucţiune_3
end
```

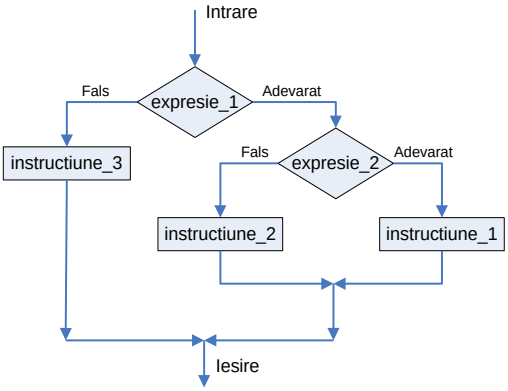


Figura 5.4.

Mod de lucru: se evaluează **expresie_1**; dacă are valoare diferită de zero (adevărată) se trece la executarea celei de a doua instrucţiuni **if**, iar dacă este falsă se execută **instrucţiune_3**. Dacă **expresie_2** este adevărată se execută **instrucţiune_1** iar dacă este falsă se execută **instrucţiune_2**.

Exemplu:

Să se scrie un program pentru calculul valorilor funcţiei

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

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

Program în **OCTAVE / MATLAB**:

```
x = input("\n Introduceti 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:

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

5.10.4. Instrucţiunea de decizie multiplă elseif

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

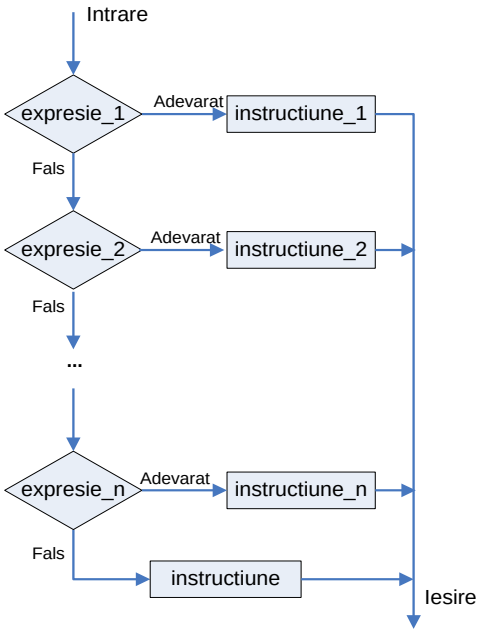


Figura 5.5.

Forma generală este:

```
...
if( expresie_1 )
      instructiune_1
elseif( expresie_2 )
      instructiune_2
...
elseif( expresie_n )
      instructiune_n
else
      instructiune
end
...
```

Mod de lucru: Expresiile se evaluează în ordinea în care sunt scrise. Dacă se întâlneşte o expresie adevărată, atunci se execută instrucţiunea care îi este asociată şi astfel se încheie întregul lanţ. Instrucţiunea de după ultimul **else** se execută atunci când nici una dintre expresii nu a fost adevărată.

Exemplu:

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

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. Instrucţiunea de ciclare for. Cicluri for suprapuse.

Introducere:

În acest curs sunt prezentate noţiunile teoretice şi exemple practice de utilizare a instrucţiunilor de ciclare for în limbajele de programare OCTAVE / MATLAB.
Instrucţiunea 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 instrucţiunilor for pentru calculul unor sume, pentru calculul valorilor unei funcţii într-un interval cunoscut, parcurs cu un pas cunoscut. De asemenea, sunt prezentate exemple de utilizare a instrucţiunii for pentru realizarea unor operaţii cu şiruri.
În corpul unei instrucţiuni for poate să fie plasată altă instrucţiune for, situaţie 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 studenţilor aspectele teoretice şi practice legate de instrucţiunea for. Prin exemplele prezentate se oferă informaţii privind utilizarea instrucţiunilor de ciclare for (inclusiv cicluri for imbricate) pentru calculul valorilor unor funcţii, pentru calculul unor sume sau pentru operaţii cu siruri şi matrice.

6.1. Instrucţiunea de ciclare ***for.***

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

```
…..
        for index = expresie
                instrucţiune
        end
…..
```

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

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

```
index = iniţial:pas:final
```

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

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

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

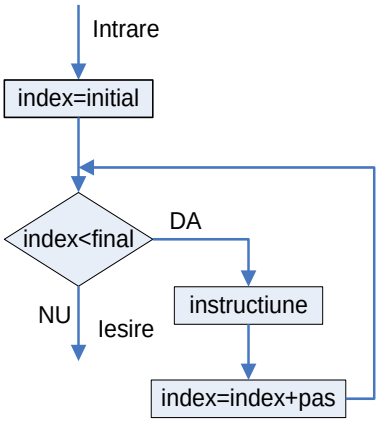


Figura 6.1.

Observaţii:

- contorul (indexul) trebuie să fie o variabilă întreagă sau reală;
- dacă **expresie** are o valoare nulă, **instrucţiune** nu se execută;
- dacă **expresie** este un scalar, **instrucţiune** se execută o singură dată, cu **indexul** dat de valoarea scalarului;
- dacă **expresie** este un vector linie, **instrucţiune** 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 iteraţie valorile conţinute 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 relaţia:

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

notaţia [ ] semnificând partea întreagă.

6.2. Exemple de programe cu ciclu for

Exemplu:

Calculul sumei primelor 10 numere naturale, S = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10.
Se iniţializează suma cu 0.
Se foloseşte 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 funcţiei $y = 3{,}4 \cdot x^3 + 5{,}25 \cdot x^2 + 3 \cdot x + 1{,}6$ 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:

Calculaţi valorile funcţiei $y = \begin{cases} x^4 - 6x^2 + 4x - 11 & x > 2 \\ x^2 - 5x - 7 & x \le 2 \end{cases}$ pentru $x \in [-a, a]$ parcurs cu pasul h.
Rezultatele se vor scrie într-un fişier text.

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

```
fid = fopen("fis_ext.txt","w+");
a = input("Introduceti valoarea lui a: a= ");
h = input("Introduceti 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:

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

Conţinutul fişierului:

```
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) + ...
luând în considerare n termeni (maxim 25 de termeni).

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

Varianta I. Rezolvare utilizând funcţiile sum şi cumsum:

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

Varianta II. Rezolvare cu ajutorul instrucţiunii 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 şir 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 şir 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 instrucţiunile care compun corpul unui ciclu **for** (numit *ciclu for exterior*) conţin 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
      instrucţiune2
   end
   …
end
…
```

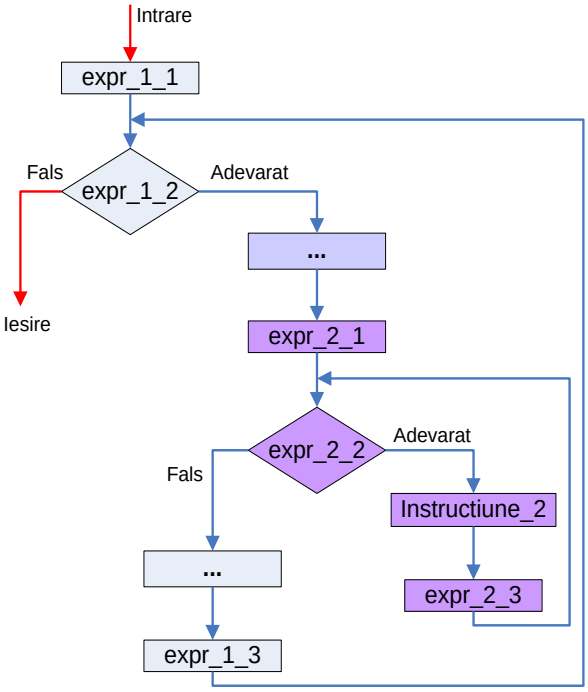


Figura 6.2

Observaţie : *Ciclul for interior* trebuie să fie cuprins complet în corpul *ciclului for exterior*.

De obicei formele index1 = expresie1, respectiv index2 = expresie2 sunt de forma:

```
variabila = valoare_iniţială : 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 iniţială a sa;
- **expr_1_2**: reprezintă de obicei o expresie logică şi reprezintă expresia prin care se verifică dacă index1 a depăşit valoarea finală a acestuia;
- **expr_1_3**: reprezintă expresia prin care valoarea lui index1 se modifică cu pasul de variaţie al acestuia.
- **expr_2_1**: reprezintă expresia prin care lui index2 i se atribuie valoarea iniţială a sa;
- **expr_2_2**: reprezintă de obicei o expresie logică şi reprezintă expresia prin care se verifică dacă index2 a depăşit valoarea finală a acestuia;

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

Mod de lucru (conform cu reprezentarea din schema logică din figura 6.2):
1. se evaluează **expr_1_1**, adică index1 primeşte valoarea iniţială a sa;
2. se verifică **expr_1_2**, dacă valoarea de adevăr a acesteia este Fals se părăseşte ciclul, în caz contrar se continuă cu pasul următor;
3. se execută (dacă există) instrucţiunile 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 părăseşte corpul ciclului şi se continuă cu instrucţiunile următoare din schema logică (pasul 8) , iar dacă este Adevărat se continuă cu pasul următor;
6. se execută **instrucţiune_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;

Observaţie: Toate instrucţiunile 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:

Afişarea tuturor posibilităţilor de a obţine 100 de lei utilizând monede de 10, 20 şi 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 ÷ 10; b = 0 ÷ 5; c = 0 ÷ 2;

```
% Afisarea tuturor posibilitatilor de a
obtine 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
```

Exemplu:

Suma elem. strict pozitive ale unei matrice dreptunghiulare A(mxn).

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) + ... 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 relaţia: $T_i = \sum_{j=1}^{i} j$.

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. Instrucţiunea de ciclare while…end. Instrucţiunile break, continue, error, switch.

Introducere:

Prezentul curs continuă cursurile precedente şi oferă noţiuni teoretice şi exemple privind utilizarea instrucţiunilor de ciclare.
În prima parte este prezentată instrucţiunea de ciclare **while .. end**. Sunt prezentate o serie de noţiuni teoretice (forma generală, mod de lucru, corespondent în schema logică) precum şi o serie de exemple de utilizare a acestei instrucţiuni.
În partea a doua a cursului sunt prezentate instrucţiunile **break**, **continue** şi **error**. Ca şi la instrucţiunile precedente şi în cazul acestor instrucţiuni, cursul oferă informaţii teoretice şi exemple practice de utilizare a acestor instrucţiuni.
În ultima parte este prezentată instrucţiunea de selecţie multiplă **switch**, fiind trecute în revistă atât aspectele teoretice (formă generală, mod de lucru) cât şi un exemplu sugestiv de utilizare a acestei instrucţiuni.

Obiective:

După parcurgerea acestui material, studenţii vor fi şti să utilizeze în cadrul programelor scrise în **FreeMat / OCTAVE / MATLAB:**
- instrucţiunea de ciclare while … end;
- instrucţiunile break, continue, error;
- instrucţiunea de selecţie multiplă switch;

7.1. Instrucţiunea de ciclare while ... end

Se utilizează pentru programarea unor operaţii de un număr de ori necunoscut. Este o instrucţiune de ciclare *condiţionată anterior*.

Forma generală este următoarea:

```
…
while (expresie)
    instrucţiune;
end
…
```

Interpretarea sintaxei este următoarea:
"atât timp cât … ".

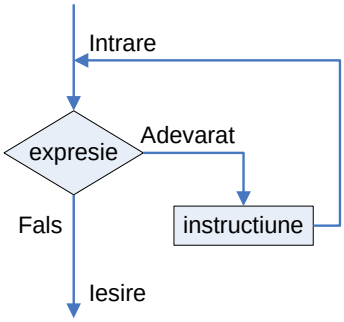


Figura 7.1.

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

Exemplu:

Suma primelor "n" numere naturale:

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

Observaţii:

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 ≠ 0 atunci ciclul este **infinit**.

Exemplu:

Ordonarea crescătoare a elementelor unui şir de numere reale.

Varianta I. Utilizarea funcţiei **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 instrucţiunea **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:

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

până când diferenţa 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. Instrucţiunea de ciclare do-until

Instrucţiunea de ciclare do-until este similară cu instrucţiunea **while**, interpretarea fiind însă: **execută ... până când**. Diferenţa dntre cele două instrucţiuni constă în faptul că **do-until** este o instrucţiune de ciclare condiţionată posterior. Aceasta înseamnă că mai întâi se execută corpul ciclului şi abia apoi se verifică condiţia.

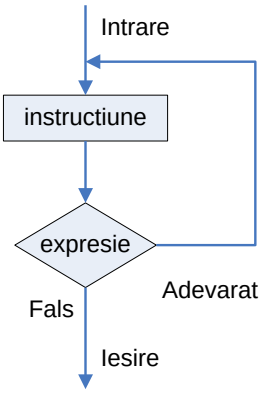


Figura 7.2.

Forma generală a instrucţiunii este:

```
do
instructiune
until (expresie)
```

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

Mod de lucru: Se execută **instructiune**, apoi se evaluează **expresie**. Dacă **expresie** este falsă (adică este zero) se revine la execuţia **instructiune**. Dacă **expresie** este adevărată (adică are valoare diferită de zero), se părăseşte ciclul.
Observaţie: Dacă expresie este un vector sau o matrice, se consideră adevărată numai dacă tabloul (vectorul sau matricea) nu este gol şi dacă elementele au valoare diferită de zero.

Exemplu: 

Citirea unui număr pozitiv:

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

Exemplu: 

Ordonarea crescătoare a unui şir 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 instrucţiuni while ... end şi do – until

Exemplu: 

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

Fişierul script:

```
do
   a=input(' Introduceti primul numar: a=');
   b=input(' Introduceti primul numar: 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:

```
--> Introduceti primul numar: a = 23
Introduceti primul numar: b = 4
23 = 4 x 5 + 3
```

Exemplu:

Să se calculeze şi să se afişeze termenii din dezvoltarea:

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

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

Fişierul script:

```
epsilon = input(' Introduceti 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:

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

Exemplu:

Se citesc de la tastaură un număr natural **n** şi o valoare **b**, $b \in [2,9]$. Se cere să se întocmească schema logică şi 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 obţine un cât şi un rest.
Restul astfel obţinut va reprezenta ultima cifră a numărului **n** în baza **b**.
Câtul astfel obţinut se va împărţi la baza **b** şi se vor obţine iarăşi un cât şi un rest.
Restul va reprezenta penultima cifră a numărului **n** în baza **b** ş.a.m.d..

Procedeul se repetă până când câtul obţinut la împărţire devine nul.

Paşii pe care trebuie să-I parcurgem sunt:
1. Se citeşte numărul **n** de la tastatură. De exemplu, se citeşte de la tastatură **n = 35**.
2. Se citeşte valoarea bazei b, de exemplu **2**.
3. Se notează valoarea numărului n în baza b cu Nb şi se iniţializează cu 0.
4. Se utilizează o variabilă fact cu ajutorul căreia se va asigura plasarea cifrelor numărului Nb pe poziţiile corespunzătoare. Valoarea acesteia se iniţializează cu 1
5. Se verifică valoarea lui n, dacă este pozitivă, se calculează restul şi câtul împărţirii numărului n la baza b. Se atribuie valoarea câtului variabilei n şi se calculează valoarea variabilei Nb cu relaţia Nb := Nb + fact * r. Se modifică valoarea variabilei fact.
6. Se revine la pasul 5

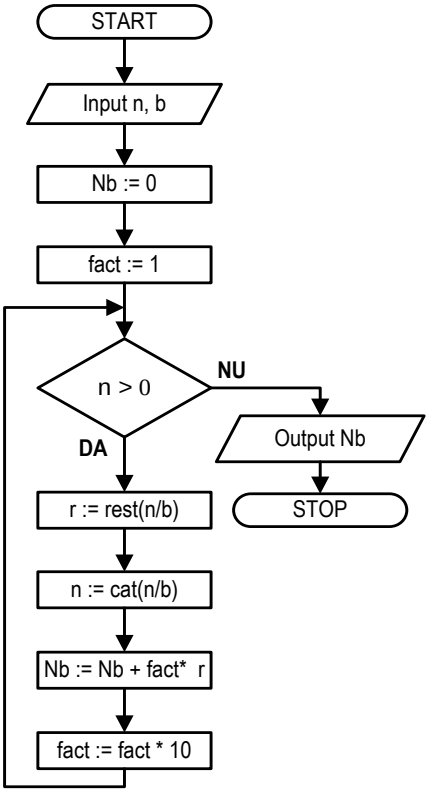


Figura 7.5

Observaţie: Paşii 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 obţinute pentru variabile sunt prezentate în tabelul următor:
Deoarece numărul **n** devine nul, în pasul etapa 6, pasul 6 nu se mai îndeplineşte condiţia impusă, continuarea se va face pe ramura cu **NU**, astfel că se va afişa valoarea variabilei **Nb** (adică **1**) şi se încheie derularea etapelor.

Fişierul script:

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

Rularea programului:

```
--> Introduceti un numar natural: 35
Introduceti baza in care doriti conversia: 2
Reprezentarea in baza 2 este 100011
```

Exemplu:

Se citeşte de la tastatură un număr natural n. Se cere să se întocmească schema logică şi program pentru afişarea reversului numărului.

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

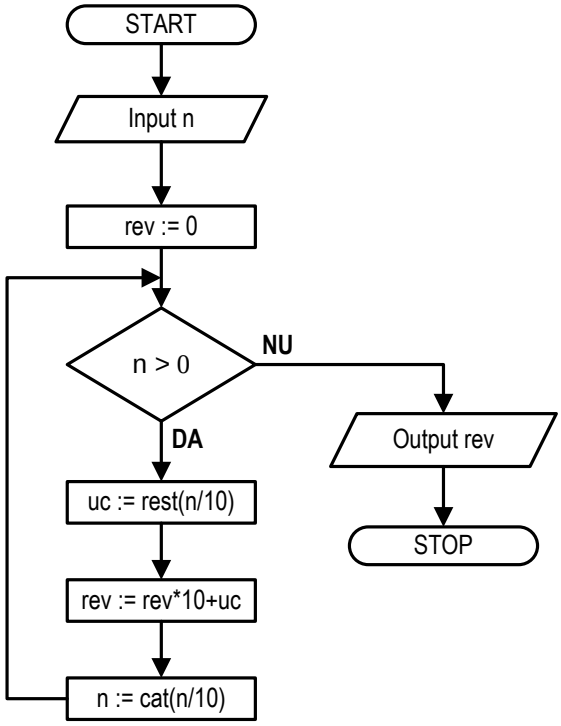


Figura 7.3

Paşii pe care trebuie să-i parcurgem sunt:
1. Se citeşte numărul **n** de la tastatură. De exemplu, se citeşte de la tastatură **n = 3562**.
2. Variabila **rev** reprezintă variabila care va memora valoarea reversului numărului **n**. Se iniţializează valoarea acesteia cu 0 (zero).
3. Se verifică dacă numărul citit de la tastatură **n** este strict pozitiv. Dacă **DA** se continuă cu paşii următori, dacă **NU** se afişează valoarea variabilei **rev** şi se finalizează algoritmul.
4. Se determină ultima cifra, notată cu **uc**, a numărului **n** prin calcularea restului împărţirii numărului **n** la 10. În cazul nostru, prin împărţirea numărului **n** (= 3562) la 10 se obţine restul **2**. Deci, **uc** va fi egal cu **2**.
5. Se atribuie variabilei **rev** valoarea dată de relaţia **rev*10+uc**. Astfel **rev** = 0*10 + 2, adică **rev** = 2.
6. Se recalculează valoarea lui **n** astfel: valoarea nouă a lui **n** este cătul împărţirii lui n la 10, astfel n = cat(3562 / 10) = 356.
7. Se revine la pasul 3.

Observaţie: Paşii 3 – 7 se repetă în cadrul unei etape.

Valorile obţinute 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 îndeplineşte condiţia impusă, continuarea se va face pe ramura cu **NU**, astfel că se va afişa valoarea variabilei rev (adică **2653**) şi se încheie derularea etapelor.

Fişierul script

```
s=input(' Date de intrare:');
n=input(' Introduceti 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:

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

Exemplu:

Se citeşte de la tastatură un număr natural n. Se cere să se întocmească schema logică şi program pentru determinarea sumei cifrelor numărului.
Mod de lucru: Calculul sumei se desfăşoară într-un număr finit de etape, numărul de etape fiind egal cu numărul de cifre al lui **n**. Nu se ştie însă câte cifre are numărul **n**.

Paşii pe care trebuie să-i parcurgem sunt:
1. Se citeşte numărul **n** de la tastatură. De exemplu, se citeşte de la tastatură **n = 3562**.
2. Variabila **S** reprezintă variabila care va memora valoarea sumei cifrelor numărului **n**. Se iniţializează valoarea acesteia cu 0 (zero).
3. Se verifică dacă numărul citit de la tastatură **n** este strict pozitiv. Dacă **DA** se continuă cu paşii următori, dacă **NU** se afişează valoarea variabilei **S** şi se finalizează algoritmul.

4. Se determină ultima cifra, notată cu **uc**, a numărului **n** prin calcularea restului împărţirii numărului **n** la 10. În cazul nostru, prin împărţirea numărului **n** (= 3562) la 10 se obţine restul **2**. Deci, **uc** va fi egal cu **2**.
5. Se atribuie variabilei **S** valoarea dată de relaţia **S := S + uc**.
6. Se recalculează valoarea lui **n** astfel: valoarea nouă a lui **n** este cătul împărţirii lui n la 10, astfel n = cat(3562 / 10) = 356.
7. Se revine la pasul 3.

Observaţie: Paşii 3 – 7 se repetă în cadrul unei etape.
Valorile obţinute pentru variabile sunt prezentate în tabelul alăturat:

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

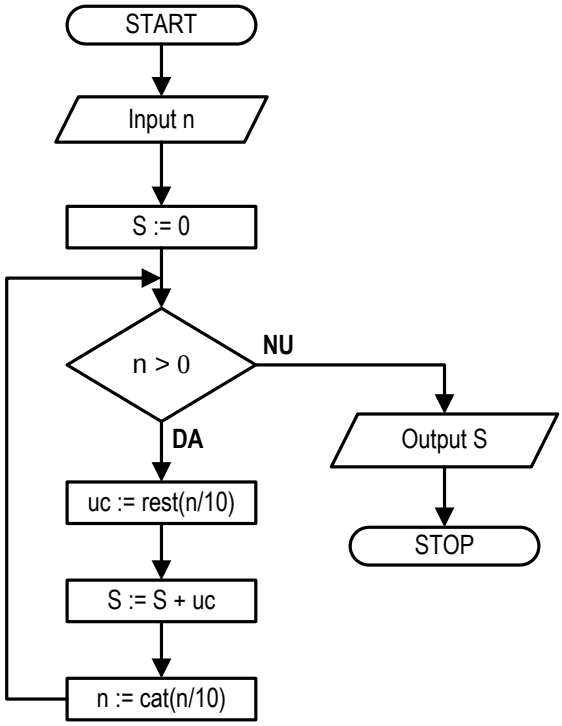


Figura 7.4

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

Fişierul script:

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

Rularea programului:

```
--> Introduceti un numar natural: 3562
Suma cifrelor numarului este S = 16
```

7.4. Instrucţiunea break

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

Exemplu:

Părăsirea unui ciclu for atunci când se îndeplineşte o condiţie, în acest caz când variabila conducătoare primeşte 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. Instrucţiunea continue

Instrucţiunea **continue** cedează controlul iteraţiei următoare în cazul unei instrucţiuni de ciclare (**for** sau **while**).

Exemplu:

Program pentru calculul valorilor funcţiei: $y = f(x) = \dfrac{x^2 - 5x + 3}{x - 2}$, $x \in [-a, a+2]$, 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. Instrucţiunea error

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

După afişarea textului 'mesaj' se cedează controlul tastaturii.

Exemplu:

Program pentru calculul valorilor funcţiei: $y = f(x) = \frac{x^2 - 5x + 3}{x - 2}$, $x \in [-a, a+2]$, 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. Instrucţiunea switch

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

Instrucţiunea **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 instrucţiuni (**instruct_1, instruct_2**, …), în caz se execută blocul de instrucţiuni aferent variantei **otherwise** (adică **bloc**).

Exemplu:

Program pentru realizarea unor operaţii aritmetice cu doi operanzi, în funcţie de simbolul operatorului introdus de la tastatură.
De la tastatură se introduce primul operand, al doilea operand şi operatorul. Operaţiile efectuate sunt: + (adunare), - (scădere), * (înmulţire), / (împărţire), $ (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. Aplicaţii 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" faţă de direcția ghidajului), respectiv dezaxat (excentricitatea "e" este nulă).

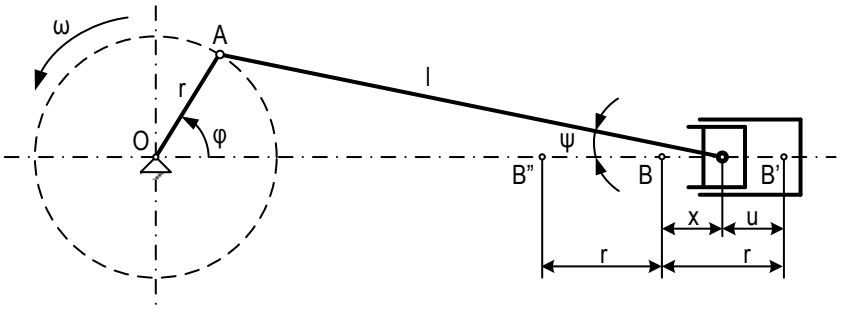


Figura 8.1.

În figura 8.1 notaţiile au următoarea semnificaţie: I – lungimea bielei, r – raza manivelei, A – centru de articulaţie dintre bielă şi axa cilindrului, B' şi B" – poziţiile extreme ale capătului bielei, x – deplasarea capătului bielei la un anumit moment. Manivela se roteşte cu viteza unghiulară constantă: ω = φ', iar cupla de translaţie B are cursa B'B"=2r.

Poziţia mecanismului este determinată de unghiul φ, care este o funcţie de timp, astfel:

$$\varphi = \omega \cdot t = \frac{\pi \cdot n}{30} \cdot t \tag{8.1}$$

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

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

Ţinând seama de relaţia geometrică:

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

şi dezvoltând în serie de puteri expresia (8.3), şi considerând numai primii doi termeni, se obţine legea de mişcare a pistonului:

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

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

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:

$$\begin{gathered} v(\varphi) = \omega \cdot r \cdot \left(\sin\varphi + \frac{1}{2} \cdot \lambda \cdot \sin 2\varphi\right), \\ v(t) = \omega \cdot r \cdot \left(\sin(\omega \cdot t) + \frac{1}{2} \cdot \lambda \cdot \sin(2 \cdot \omega \cdot t)\right) \end{gathered} \tag{8.5}$$

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

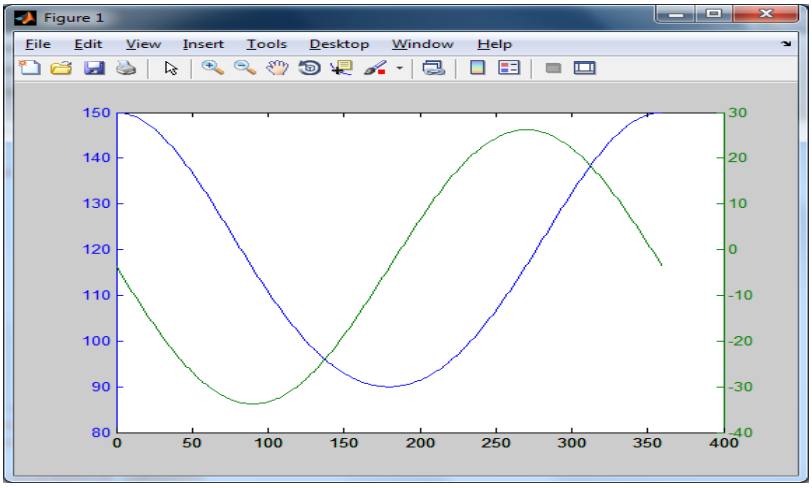
$$\begin{gathered} a = \frac{dv}{dt} = \frac{dv}{d\varphi} \cdot \frac{d\varphi}{dt} = \omega \cdot \frac{dv}{d\varphi} \\ a(\varphi) = \omega^2 \cdot r \cdot (\cos\varphi + \lambda \cdot \cos 2\varphi) \\ a(t) = \omega^2 \cdot r \cdot (\cos(\omega \cdot t) + \lambda \cdot \cos(2 \cdot \omega \cdot t)) \end{gathered} \tag{8.6}$$

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

Fişierul script:

```
s =input(' Date de intrare :');
lm=input(' Introduceti lungimea manivelei [mm]:');
lb=input(' Introduceti lungimea bielei [mm]:');
ex=input(' Introduceti excentricitatea [mm]:');
pu=input(' Introducei 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 situaţie de nedeterminare care conduce la o eroare a limitei de recursivitate.

Mesajul obţinut 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);

Funcţiile 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 obţinut 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: $t = \sqrt{x}$ şi se obţine: $dt = \frac{dx}{2\sqrt{x}} = \frac{dx}{2t}$

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

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

Valoarea exactă a integralei este: I_c = 1,809048476

Funcţiile 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;
```

```
        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)
% Evalueează 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
```

Rezultatul obţinut este:

```
I = 1.80904847548611
n = 6
```

8.4. Program pentru verificarea validităţii Codului Numeric Personal

Codul numeric personal sau **CNP** este un cod unic asocia fiecărei persoane născute în România şi 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 semnificaţie:

S	AA	LL	ZZ	JJ	NNN	C
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 persoana va avea prima cifra cu una din valorile 7,8 (rezidenti) sau 9, atunci se va considera secolul 20. ex SAA =771 anul nasterii 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 – Argeş, 04 – Bacău, 05 – Bihor, 06 – Bistriţa-Năsăud, 07 – Botoşani, 08 – Braşov, 09 – Brăila, 10 – Buzău, 11 – Caraş-Severin, 12 – Cluj, 13 – Constanţa, 14 – Covasna, 15 – Dâmboviţa, 16 – Dolj, 17 – Galaţi, 18 – Gorj, 19 – Harghita, 20 – Hunedoara, 21 – Ialomiţa, 22 – Iaşi, 23 – Ilfov, 24 – Maramureş, 25 – Mehedinţi, 26 – Mureş, 27 – Neamţ, 28 – Olt, 29 – Prahova, 30 – Satu-Mare, 31 – Sălaş, 32 – Sibiu, 33 – Suceava, 34 – Teleorman, 35 – Timiş, 36 – Tulcea, 37 – Vaslui, 38 – Vâlcea, 39 – Vrancea, 40 – Bucureşti, 41 – Bucureşti S.1., 42 - Bucureşti S.2., 43 - Bucureşti S.3., 44 - Bucureşti S.4., 45 - Bucureşti S.5., 46 - Bucureşti S.6., 51 - Călăraşi, 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 împărţirii.

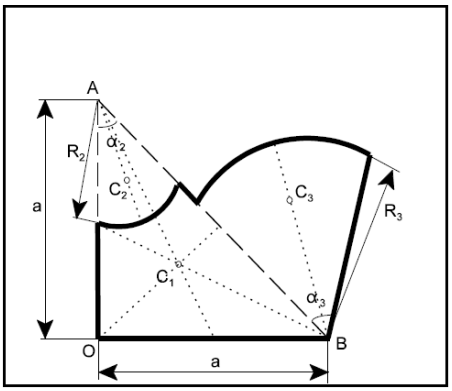
Fişierul script:

```
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
```

Observaţie: CNP se va introduce ca un şir 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 [cm];
R_2 = 7 + 0,1*n [cm];
R_3 = 8 + 0,2*n [cm];
α_2 = 45°
α_3 = 40 + n°

unde n – numărul de ordine al studentului în grupă;

Consideraţii teoretice:

Placa omogenă cu o formă mai complicată poate fi împărţită în bucăţi sau părţi şi tratată ca o placă compusă omogenă.

În acest caz, placa se împarte în trei părţi:
- o parte de forma unui triunghiu dreptunghic isoscel AOB care are centrul de greutate C_1 aflat la intersecţia medianelor;
- o parte sub forma unui sector circular de rază R_2 care însă lipseşte şi 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 şi care are centrul de greutate C_3 pe axa sa de simetrie.

Relaţiile 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 lipseşte se scrie cu semnul minus.

Fişierul 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-aplicaţii)*, Editura Teora, Bucureşti, 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] Trâmbiţaş, 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