

CUPRINS

CAP. 1. PREZENTARE GENERALĂ

CAP. 2. FUNDAMENTELE PROGRAMĂRII ÎN MATLAB

- 2.1. Expresii fundamentale
- 2.2. Help on-line, formatul datelor, opțiuni de salvare
- 2.3. Crearea fișierelor MATLAB (.m files)
- 2.4. Matrici, vectori și polinoame
- 2.5. Operațiuni elementare cu matrici și funcții

CAP. 3. MEDIUL DE LUCRU MATLAB

- 3.1. Startul și terminarea sesiunilor de lucru
- 3.2. Fereastra de comandă (fereastra principală)
 - 3.2.1. Editarea liniilor de comandă în fereastra principală
 - 3.2.2. Descrierea ferestrei de comandă
 - 3.2.3. Spațiul de lucru al MATLAB-ului (workspace)
 - 3.2.4. Căi de căutare
- 3.3. Fereastra grafică (figure)
- 3.4. Importul și exportul de date
- 3.5. Utilizarea memoriei

CAP. 4. EDITORUL/DEBUGGER-UL ȘI PROFILER-UL MATLAB

- 4.1. Editorul/Debugger-ul MATLAB
- 4.2. Profiler-ul MATLAB

CAP. 5. MATRICI, ALGEBRĂ LINIARĂ, POLINOAME, TEHNICI DE INTERPOLARE

- 5.1. Matricile în MATLAB
- 5.2. Rezolvarea ecuațiilor liniare
- 5.3. Inverse și determinanți
- 5.4. Funcții de matrice. Valori proprii
- 5.5. Reprezentarea polinoamelor. Interpolarea

CAP. 6. REPREZENTAREA FUNCȚIILOR. ECUAȚII DIFERENȚIALE

- 6.1. Reprezentarea și plotarea funcțiilor matematice
- 6.2. Minimizarea funcțiilor și găsirea zerourilor
- 6.3. Integrarea numerică
- 6.4. Rezolvarea ecuațiilor diferențiale

CAP. 7. PROGRAMAREA ÎN LIMBAJUL MATLAB

- 7.1. Fișiere MATLAB
- 7.2. Tipuri de date și operatori
- 7.3. Instrucțiuni de salt și bucle
- 7.4. Evaluarea datelor de tip caracter
- 7.5. Reprezentarea și manipularea informațiilor despre dată și timp
- 7.6. Intrări utilizator

CAP. 8. GRAFICE ȘI INTERFEȚE GRAFICE ÎN MATLAB

- 8.1. Tehnici de plotare
 - 8.1.1. Plotări 2 D elementare
 - 8.1.2. Grafice 2 D specializate
 - 8.1.3. Plotări tridimensionale (3 D)
- 8.2. Handle Graphics și Interfețe Grafice în MATLAB (GUI)
 - 8.2.1. Graficele privite ca obiecte. Ierarhia obiectelor
 - 8.2.2. Proprietățile obiectelor grafice
 - 8.2.3. Funcții de generare a obiectelor grafice

CAP. 9. PREZENTAREA TOOLBOX-URILOR MATLAB

- 9.1. Toolbox-ul Comunicații (Communications Toolbox)
- 9.2. Toolbox-ul pentru Sisteme de Conducere Automată (Control System)
- 9.3. Toolbox-ul pentru Baze de Date (Database Toolbox)
- 9.4. Toolbox-ul de Procesare a Semnalelor (Signal Processing Toolbox)
- 9.5. Toolbox-ul DSP Blockset
- 9.6. Toolbox-ul Finanțe (Financial Toolbox)
- 9.7. Toolbox-ul de Procesare a Imaginilor (Image Processing Toolbox)
- 9.8. Toolbox-ul Optimizare (Optimization Toolbox)
- 9.9. Toolbox-ul pentru Sisteme de Putere (Power System Blockset)
- 9.10. Toolbox-ul Stateflow (Diagrame de stare)
- 9.11. Toolbox-ul de Statistică (Statistics Toolbox)
- 9.12. Toolbox-ul pentru Calcul Simbolic (Symbolic Math Toolbox)

CAP. 10. PACHETUL DE MODELARE ȘI SIMULARE SIMULINK

- 10.1. Rularea unui model SIMULINK demonstrativ
- 10.2. Crearea modelelor SIMULINK
- 10.3. Rularea simulărilor în SIMULINK
- 10.4. Modul de lucru al unui program SIMULINK

BIBLIOGRAFIE

APLICAȚII MATLAB

1. PREZENTARE GENERALĂ

MATLAB® = Limbaj de înaltă performanță pentru proiectarea asistată de calculator
MATLAB este în același timp un *limbaj de programare* și un *sistem de dezvoltare* care integrează calculul, vizualizarea și programarea într-un mediu ușor de utilizat (easy-to-use), problemele și soluțiile acestor probleme fiind exprimate într-un limbaj matematic accesibil.

Domenii de utilizare:

- Matematică și calcul numeric
- Dezvoltarea algoritmilor
- Modelare, simulare și testarea prototipurilor
- Analiza și vizualizarea datelor
- Grafica inginerescă și din științele aplicate
- Dezvoltarea de aplicații, inclusiv GUI

➤ MATLAB = sistem interactiv care are ca element de bază tabloul, matricea, ceea ce permite rezolvarea problemelor de calcul numeric, în special cele care necesită prelucrarea de vectori sau matrici.

➤ Numele MATLAB provine de la *Matrix laboratory*
Firma producătoare este **The MathWorks, Inc., SUA**

➤ MATLAB-ul a evoluat:

- în mediul universitar unde este pachetul standard pentru cursurile introductive și avansate de matematică, inginerie și științe
- în industrie, unde este utilizat pentru cercetarea de înalt randament, dezvoltare și producție

➤ MATLAB permite dezvoltarea unei familii de aplicații sub forma *toolbox*-urilor. Aceste *toolbox-uri* permit învățarea și aplicarea tehnologiilor specializate din diverse domenii. Sunt disponibile toolbox-uri pentru domenii cum ar fi: procesarea numerică a semnalelor, sisteme de conducere automată, rețele neurale, logică fuzzy, wavelet, simulare (**SIMULINK**), identificare etc.

Sistemul MATLAB constă în cinci părți principale:

- ❑ **Limbajul MATLAB**
- ❑ **Mediul de lucru MATLAB**
- ❑ **Handle Graphics®**
- ❑ **Biblioteca de funcții matematice a MATLAB-ului**
- ❑ **Interfața de aplicații program a MATLAB-ului (API)**

Limbajul MATLAB: Reprezintă un limbaj de nivel înalt de tip matrice/tablou cu instrucțiuni de control al salturilor, funcții, structuri de date, intrări/ieșiri și cu proprietăți de programare orientată pe obiecte. Facilitățile de programare sunt organizate pe 6 directoare:

ops	Operators and special characters.
Lang	Programming language constructs.
Strfun	Character strings.

Iofun	File input/output.
Timefun	Time and dates.
Datatypes	Data types and structures.

Mediul de lucru MATLAB: Reprezintă un set de facilități care permit manevrarea variabilelor în spațiul de lucru, importul și exportul de date, dezvoltarea, manipularea, editarea și depanarea fișierelor MATLAB (.m) și a aplicațiilor MATLAB. Aceste facilități sunt organizate în directorul:

general	General purpose commands.
---------	---------------------------

Handle Graphics®: Reprezintă sistemul grafic al MATLAB-ului. Cuprinde comenzi de înalt nivel pentru vizualizarea datelor bi și tri-dimensionale, procesarea imaginilor, animație, prezentări de grafice. Permite de asemenea utilizarea unor comenzi de nivel scăzut pentru crearea unor interfețe grafice GUI. Funcțiile grafice sunt organizate în 5 directoare:

graph2d	Two-dimensional graphs.
Graph3d	Three-dimensional graphs.
Specgraph	Specialized graphs.
Graphics	Handle Graphics.
Uitools	Graphical user interface tools.

Biblioteca de funcții matematice a MATLAB-ului: Reprezintă o colecție complexă de algoritmi de calcul pornind de la funcții elementare (sinus, cosinus etc.) până la funcții sofisticate (inversarea de matrice, valori proprii, funcții Bessel, FFT etc.). Funcțiile matematice sunt organizate în 8 directoare:

elmat	Elementary matrices and matrix manipulation.
Elfun	Elementary math functions.
Specfun	Specialized math functions.
Matfun	Matrix functions – numerical linear algebra.
Datafun	Data analysis and Fourier transforms.
Polyfun	Interpolation and polynomials.
Funfun	Function functions and ODE solvers.
Sparfun	Sparse matrices.

Interfața de aplicații program a MATLAB-ului (API) este o bibliotecă care permite scrierea de programe în C sau Fortran care interacționează cu MATLAB-ul. Include facilități pentru apelarea rutinelor din MATLAB, apelarea MATLAB-ului ca mașină de calcul, scrierea și citirea fișierelor de tip .MAT .

Pachetul SIMULINK

- SIMULINK® este un pachet software atașat MATLAB-ului și reprezintă un sistem interactiv pentru simularea dinamicii sistemelor neliniare (bineînțeles și a celor liniare). Este conceput sub forma unei interfețe grafice care permite crearea unui model prin “trasarea” schemei bloc a sistemului și apoi simularea dinamicii sistemului.
- SIMULINK poate lucra cu sisteme liniare, neliniare, continue, discrete, multivariabile etc.

- SIMULINK bebeneficiază de așa-numitele *Blockset*-uri care sunt de fapt biblioteci suplimentare care conțin aplicații specializate din domenii cum ar fi: comunicații, procesarea semnalelor etc.
- *Real-time Workshop*® este un program foarte important care permite generarea unui cod C pentru schemele bloc create în SIMULINK și prin urmare permite rularea de aplicații în timp real de o mare diversitate.

Toolbox-urile MATLAB

Toolbox-urile reprezintă o familie de aplicații care permit învățarea și aplicarea tehnologiilor specializate din diverse domenii. Aceste toolbox-uri sunt colecții de funcții MATLAB (functions) (M-files) care extind mediul MATLAB pentru rezolvarea unor clase particulare de probleme. Câteva din cele mai utilizate aplicații sunt prezentate în figura următoare.

The MATLAB Product Family

How The MathWorks products fit together

MATLAB is the foundation for all The MathWorks products. MATLAB combines numeric computation, 2-D and 3-D graphics, and language capabilities in a single, easy-to-use environment.

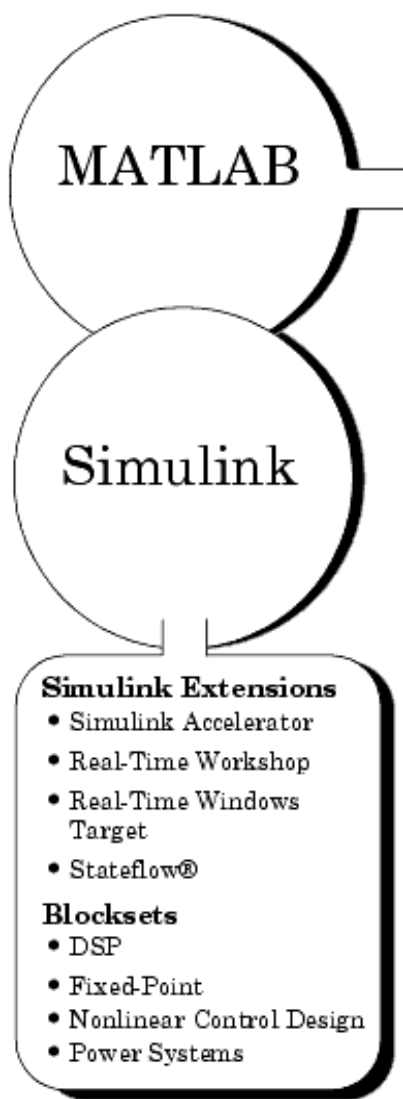
MATLAB Extensions are optional tools that support the implementation of systems developed in MATLAB.

Toolboxes are libraries of MATLAB functions that customize MATLAB for solving particular classes of problems. Toolboxes are open and extensible; you can view algorithms and add your own.

Simulink is a system for nonlinear simulation that combines a block diagram interface and “live” simulation capabilities with the core numeric, graphics, and language functionality of MATLAB.

Simulink Extensions are optional tools that support the implementation of systems developed in Simulink

Blocksets are collections of Simulink blocks designed for use in specific application areas.



MATLAB Extensions

- MATLAB Compiler
- MATLAB C/C++ Math Libraries
- MATLAB Web Server
- MATLAB Report Generator

Toolboxes

- Control System
- Communications
- Database
- Financial
- Frequency Domain System Identification
- Fuzzy Logic
- Higher Order Spectral Analysis
- Image Processing
- LMI Control
- Model Predictive Control
- μ -Analysis and Synthesis
- NAG® Foundation
- Neural Network
- Optimization
- Partial Differential Equation
- QFT Control Design
- Robust Control
- Signal Processing
- Spline
- Statistics
- Symbolic Math
- System Identification
- Wavelet

Contact The MathWorks or visit www.mathworks.com for an up-to-date product list.

2. FUNDAMENTELE PROGRAMĂRII ÎN MATLAB

2.1. Expresii fundamentale

MATLAB-ul lucrează cu expresii matematice ca și celelalte limbaje de programare, dar spre deosebire de majoritatea acestor limbaje, aceste expresii implică la scară largă lucrul cu matrici.

Expresiile sunt alcătuite cu ajutorul următoarelor tipuri:

- Variabile
- Numere
- Operatori
- Funcții

Variable

- MATLAB-ul nu necesită declararea dimensiunii variabilelor, deoarece la întâlnirea unui nou nume de variabilă generează automat variabila respectivă și alocă spațiul necesar de memorie.
- Numele unei variabile este o literă, urmată de un număr oricât de mare de litere, cifre sau simboluri. Din acest număr “oricât de mare” sunt oprite primele 31 de caractere.
- MATLAB-ul este *case sensitive* - face distincție între literele mici și cele mari.
- Exemplu:

```
» a = 30
```

crează o matrice 1 x 1 cu numele *a* și stochează valoarea acesteia 30 într-o singură locație corespunzătoare singurului element al matricei.

Numere

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

```
3          -99          0.0001
9.6397238  1.60210e-20   6.02252e23
1i         -3.14159j     3e5i
```

- Toate numerele sunt stocate intern utilizând formatul *long* specificat de standardul IEEE în virgulă mobilă (*precizie* de 16 zecimale semnificative în domeniul 10^{-308} la 10^{+308}).

Operatori

Expresiile utilizează operatori aritmetici uzuali:

+	Adunare
-	Scădere
*	Înmulțire
/	Împărțire
\	Împărțire la stânga
^	Ridicarea la o putere
'	Transpusa complex conjugată
()	Operatorul de specificare a ordinii de evaluare

Funcții

MATLAB-ul furnizează un mare număr de funcții matematice elementare standard (*abs*, *sqrt*, *exp*, *sin* ...).

Există și funcții matematice avansate (funcții Bessel, gama etc.), multe dintre acestea acceptând argumente complexe.

Pentru vizualizarea funcțiilor elementare se poate tasta:

```
» help elfun
```

Pentru a vedea lista funcțiilor avansate se poate tasta:

```
» help specfun  
» help elmat
```

- ♦ O parte din funcții (cum ar fi *sqrt*, *sin*) sunt de tip *built-in*, adică sunt o parte a nucleului MATLAB, au o mare eficiență, dar detaliile constructive nu sunt accesibile utilizatorului.
- ♦ Alte funcții sunt implementate ca fișiere MATLAB (M-files) și pot fi chiar modificate.
- ♦ Câteva funcții furnizează valorile unor constante universale:

pi	3.14159265
I	Imaginary unit, -1
J	Same as I
Eps	Floating-point relative precision, 2^{-52}
Realmin	Smallest floating-point number, 2^{-1022}
Realmax	Largest floating-point number, 2^{1023}
Inf	Infinity
NaN	Not-a-number

- Numele funcțiilor nu sunt rezervate și deci este posibilă suprascrierea lor.

Exemplu:

```
eps = 1.e-6
```

Funcția originală este reconstituită prin comanda:

```
» clear eps
```

Expresii

Exemple de expresii și rezultatele corespunzătoare ale evaluării acestor expresii:

```
» rho = (1+sqrt(5))/2
```

```
rho =  
    1.6180
```

```
» a = abs(3+4i)
```

```
a =  
    5
```

```
» z = sqrt(besselk(4/3,rho-i))
```

```
z =  
    0.3730+ 0.3214i
```

```
» huge = exp(log(realmax))
```

```
huge =  
    1.7977e+308
```

```
» toobig = pi*huge
```

```
toobig =  
    Inf
```

2.2. Help on-line, formatul datelor, opțiuni de salvare

Help on-line

Pentru rularea MATLAB pe un PC trebuie pur și simplu executat un dublu click cu mouse-ul pe icon-ul MATLAB. Dacă sistemul de operare nu este de tip Windows (este de tip UNIX) trebuie tastat `matlab` după prompter-ul sistemului de operare.

- Limbajul MATLAB este mult mai simplu de învățat dacă se renunță la inspectarea aridă a listelor cu variabile, funcții și operatori și se utilizează în schimb comenzile `help`, `helpdesk`, `demo` tastate direct de la prompterul MATLAB.
- Pentru aflarea tuturor informațiilor utile despre o comandă sau o funcție se tastează `help` urmat de numele comenzii sau funcției respective.
- Pachetul MATLAB dispune de asemenea de informații complete despre utilizare sub forma unei documentații tip `.pdf`.
- În cazuri particulare se poate apela la INTERNET, existând o legătură la pagina Web a firmei producătoare.
- Alte comenzi utile pentru aflarea de informații sunt: `helpwin`, `lookfor`, `help help`.

Exemple sugestive de utilizare a comenzii `help`:

```
» help sin
```

```
SIN      Sine.  
        SIN(X) is the sine of the elements of X.
```

```
Overloaded methods  
    help sym/sin.m
```

```
» help exp
```

```
EXP      Exponential.  
        EXP(X) is the exponential of the elements of X, e  
        to the X.  
        For complex Z=X+i*Y, EXP(Z) =  
        EXP(X) * (COS(Y)+i*SIN(Y)).
```

```
        See also LOG, LOG10, EXPM, EXPINT.
```

```
Overloaded methods  
    help sym/exp.m  
    help demtseries/exp.m
```

```
» help plot
```

```
PLOT     Linear plot.
```

`PLOT(X,Y)` plots vector `Y` versus vector `X`. If `X` or `Y` is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever line up. If `X` is a scalar and `Y` is a vector, `length(Y)` disconnected points are plotted.

`PLOT(Y)` plots the columns of `Y` versus their index. If `Y` is complex, `PLOT(Y)` is equivalent to `PLOT(real(Y),imag(Y))`. In all other uses of `PLOT`, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with `PLOT(X,Y,S)` where `S` is a character string made from one element from any or all the following 3 columns:

y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot

r	red	+	plus	--	dashed
g	green	*	star		
b	blue	s	square		
w	white	d	diamond		
k	black	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

For example, `PLOT(X,Y,'c+:')` plots a cyan dotted line with a plus at each data point; `PLOT(X,Y,'bd')` plots blue diamond at each data point but does not draw any line.

`PLOT(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...)` combines the plots defined by the (X,Y,S) triples, where the X 's and Y 's are vectors or matrices and the S 's are strings.

For example, `PLOT(X,Y,'y-',X,Y,'go')` plots the data twice, with a solid yellow line interpolating green circles at the data points.

The `PLOT` command, if no color is specified, makes automatic use of the colors specified by the axes `ColorOrder` property. The default `ColorOrder` is listed in the table above for color systems where the default is yellow for one line, and for multiple lines, to cycle through the first six colors in the table. For monochrome systems, `PLOT` cycles over the axes `LineStyleOrder` property.

`PLOT` returns a column vector of handles to `LINE` objects, one handle per line.

The X,Y pairs, or X,Y,S triples, can be followed by parameter/value pairs to specify additional properties of the lines.

See also `SEMILOGX`, `SEMILOGY`, `LOGLOG`, `GRID`, `CLF`, `CLC`, `TITLE`, `XLABEL`, `YLABEL`, `AXIS`, `AXES`, `HOLD`, `COLORDEF`, `LEGEND`, and `SUBPLOT`.

Formatul datelor

MATLAB-ul afișează numerele cu 5 zecimale (setare implicită). Această setare se poate modifica cu ajutorul comenzii `format`:

`FORMAT` Set output format.

All computations in MATLAB are done in double precision.

`FORMAT` may be used to switch between different output display formats as follows:

`FORMAT` Default. Same as `SHORT`.

`FORMAT SHORT` Scaled fixed point format with 5 digits.

`FORMAT LONG` Scaled fixed point format with 15 digits.

`FORMAT SHORT E` Floating point format with 5 digits.

`FORMAT LONG E` Floating point format with 15 digits.

`FORMAT SHORT G` Best of fixed or floating point format with 5 digits.

`FORMAT LONG G` Best of fixed or floating point format with 15 digits.

`FORMAT HEX` Hexadecimal format.

`FORMAT +` The symbols `+`, `-` and blank are printed for positive, negative and zero elements. Imaginary parts are ignored.

`FORMAT BANK` Fixed format for dollars and cents.

`FORMAT RAT` Approximation by ratio of small integers.

Spacing:

`FORMAT COMPACT` Suppress extra line-feeds.

`FORMAT LOOSE` Puts the extra line-feeds back in.

Example:

```
» c=1.333456789233
c =
    1.3335
```

```

» format long
» c
c =
    1.33345678923300

» format short e
» c
c =
    1.3335e+000

» format long e
» c
c =
    1.333456789233000e+000

» format
» c
c =
    1.3335

```

Opțiuni de salvare

- Pentru salvarea variabilelor curente cu care se lucrează în MATLAB la încheierea unei sesiuni de lucru se poate utiliza comanda `save`.
- Această comandă va salva toate variabilele curente generate de către utilizator într-un fișier numit *matlab.mat* . Dacă se dorește se poate da un nume fișierului de date în care se salvează variabilele.

Exemplu:

```
» save date c determ A
```

realizează salvarea datelor `c`, `determ` și `A` într-un fișier *date.mat* .

- Pentru restituirea variabilelor într-o sesiune de lucru ulterioară se folosește comanda `load` . Exemplu:

```
» load date
```

- Dacă se dorește aflarea variabilelor curente se pot utiliza comenzile `who`, `whos`:

```

» who
Your variables are:

```

```
A           c           determ
```

```

» whos
  Name           Size           Bytes   Class
  ----
  A              2x2              32   double array
  c              1x1               8   double array
  determ         1x1               8   double array

```

```
Grand total is 6 elements using 48 bytes
```

- Pentru ștergerea tuturor variabilelor curente din memoria de lucru se poate utiliza comanda `clear`.

2.3. Crearea fișierelor MATLAB (.m files)

Deoarece este mult mai comod și util decât introducerea comenzilor linie după linie la prompterul MATLAB, se lucrează cu fișiere text care conțin aceste linii program cu comenzile necesare.

Aceste fișiere conțin cod în limbajul MATLAB și sunt denumite .m files (sau M-files). Fișierele se creează utilizând un editor de text și apoi se utilizează ca o comandă MATLAB obișnuită.

Sunt două tipuri de fișiere .m:

- Fișiere Script, care nu acceptă argumente de intrare și nu returnează argumente de ieșire. Aceste fișiere operează cu datele din spațiul de lucru.
- Rutine (funcții), care acceptă argumente de intrare și returnează argumente de ieșire. Variabilele utilizate sunt variabile locale (interne) ale funcției.

Pentru a vedea conținutul unui fișier MATLAB, de exemplu *evolutie_studii.m*, se folosește comanda:

```
» type evolutie_studii
```

Fișiere Script

Atunci când se apelează la un fișier script, MATLAB-ul execută comenzile găsite în fișierul respectiv. Fișierele script pot lucra cu date din spațiul de lucru (workspace) sau pot crea date noi cu care operează. Script-urile nu furnizează argumente de ieșire, iar variabilele create rămân în workspace, pentru a fi eventual folosite în calculele ulterioare.

Fișierele script pot furniza ieșiri grafice folosind funcții cum ar fi `plot`, `bar`.

Exemplu de fișier script: *magicrank.m*, cu următoarele comenzi MATLAB:

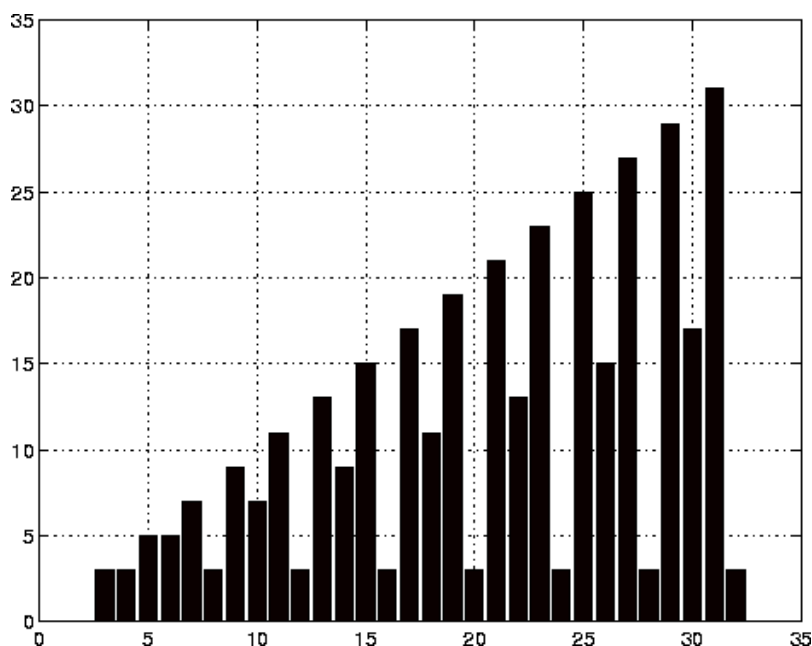
```
% Investigate the rank of magic squares
r = zeros(1,32);
for n = 3:32
    r(n) = rank(magic(n));
end
r
bar(r)
```

La tastarea numelui fișierului script (fără extensia .m):

```
» magicrank
```

MATLAB-ul execută comenzile, calculează rangul unor matrici (matricile magice), și trasează graficul cu rezultatele calculului. După ce se termină execuția fișierului, variabilele `n` și `r` rămân în spațiul de lucru.

Graficul rezultat este prezentat în continuare:



Funcții (rutine)

Aceste fișiere acceptă argumente de intrare și furnizează argumente de ieșire. Numele fișierului MATLAB (M-file) și cel al funcției (subrutinei) respective trebuie să fie identice. Funcțiile (subrutinele) lucrează cu variabile proprii separate de spațiul de lucru uzual al MATLAB-ului.

Exemplu: funcția rank. Fișierul M-file rank.m este disponibil în directorul

toolbox/matlab/matfun

Se poate vizualiza fișierul cu comanda:

```
> type rank

function r = rank(A,tol)
% RANK Matrix rank.
% RANK(A) provides an estimate of the number of
% linearly independent rows or columns of a matrix A
% RANK(A,tol) is the number of singular values of A
% that are larger than tol.
% RANK(A) uses the default
% tol = max(size(A)) * norm(A) * eps.

s = svd(A);
if nargin==1
    tol = max(size(A)) * max(s) * eps;
end
r = sum(s > tol);
```

Prima linie a unei funcții M-file începe cu cuvântul cheie `function`. Această linie dă numele funcției, ordinea și numărul argumentelor.

Liniile următoare (care încep cu caracterul `%`) sunt linii de comentariu, care de fapt sunt și liniile afișate atunci când se apelează la comanda

```
> help rank
```

Restul liniilor sunt executabile. Variabila `s`, ca și `r`, `A`, `tol` sunt variabile locale ale funcției și sunt separate de variabilele din `workspace`.

Funcția rank poate fi utilizată în diferite moduri:

```
> rank(A)
> r = rank(A)
> r = rank(A,1.e-6)
```

Variabile globale

Dacă se dorește ca mai multe astfel de subrutine să utilizeze o anume variabilă comună, se declară variabila respectivă ca globală utilizând comanda `global` în toate funcțiile respective.

Exemplu: fișierul falling.m:

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

Se introduc apoi în mod interactiv liniile:

```
> global GRAVITY
> GRAVITY = 32;
> y = falling((0:.1:5)');
```

Funcția eval

Funcția `eval` lucrează cu variabilă text pentru implementarea unei facilități puternice de tip macro text.

Expresia

```
eval(s)
```

folosește interpreter-ul MATLAB pentru evaluarea expresiei sau execuția declarației din șirul de caractere `s`.

Vectorizarea

Pentru a obține o viteză de calcul mare, este foarte importantă așa-numita vectorizare a algoritmilor în fișierele MATLAB. Acolo unde alte limbaje folosesc bucle de tip `for` sau `do`, MATLAB-ul poate utiliza operații matriceale sau vectoriale.

Un exemplu simplu este următorul:

```
x = 0;
for k = 1:1001
    y(k) = log10(x);
    x = x + .01;
end
```

Versiunea vectorizată a aceluiași program este

```
x = 0:.01:10;
y = log10(x);
```

Programatorii MATLAB spun uneori:

"Viața este prea scurtă pentru a ți-o petrece scriind bucle!"

Atunci când nu se poate elimina complet folosirea unei bucle se utilizează procedura de prealocare.

Funcții de funcții

În MATLAB există o clasă de funcții care lucrează cu funcții neliniare ca argument. Funcțiile de funcții includ:

- Găsirea zerourilor
- Optimizare
- Integrare numerică
- Ecuații diferențiale ordinare

MATLAB-ul reprezintă funcția neliniară ca o funcție M-file care poate fi ulterior utilizată ca argument de alte funcții MATLAB.

Exemplu:

Următorul fișier creează o funcție neliniară:

```
function y = humps(x)
y = 1./((x-.3).^2 + .01) + 1./((x-.9).^2 + .04) - 6;
```

Această funcție poate fi evaluată pentru un set de puncte în intervalul $0 \leq x \leq 1$ cu programul:

```
x = 0:.002:1;
y = humps(x);
```

și apoi se poate reprezenta grafic funcția cu comanda

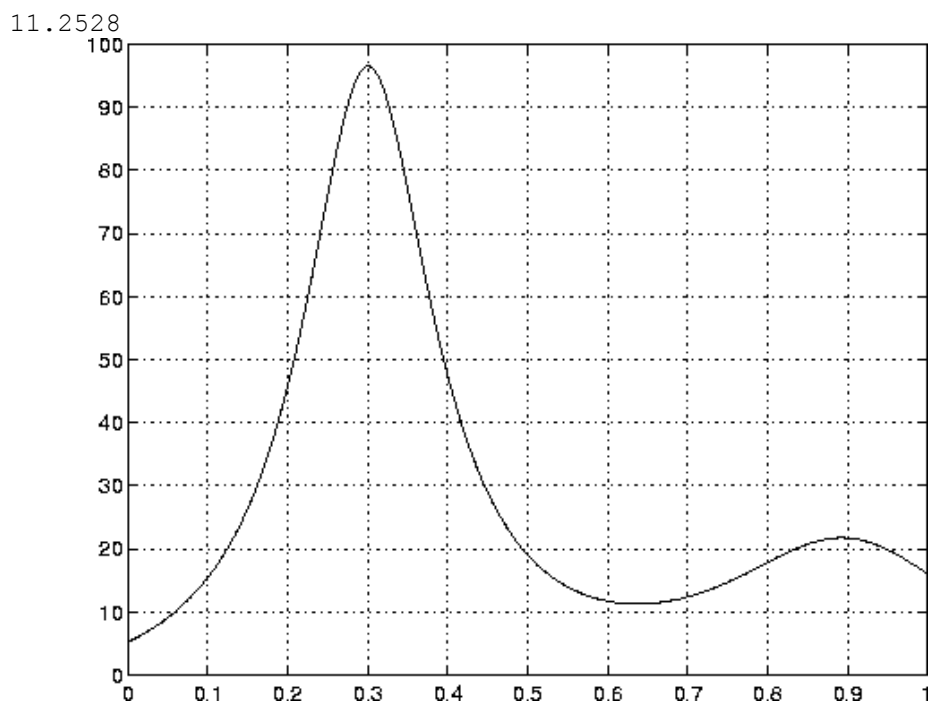
```
plot(x,y)
```

Graficul arată că funcția are un minim local la aproximativ $x = 0.6$. Dacă de exemplu utilizăm funcția `fmins` putem găsi imediat valoarea exactă a lui x . Primul argument al funcției `fmins` este chiar numele funcției pentru care calculăm minimul (al doilea parametru este o aproximare grosieră a localizării minimului).

```
» p = fmins('humps',.5)
p =
    0.6370
```

Se poate acum evalua valoarea funcției în punctul de minim local:

```
» humps(p)
ans =
```

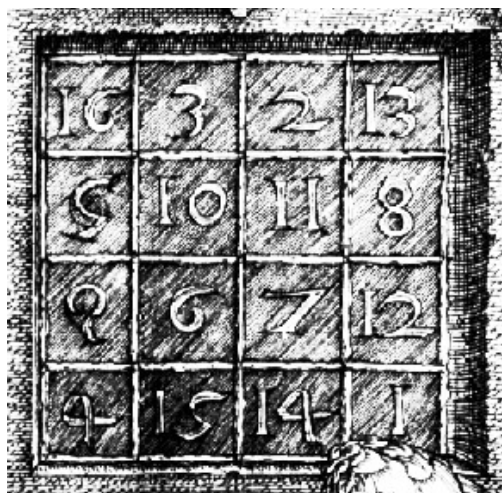
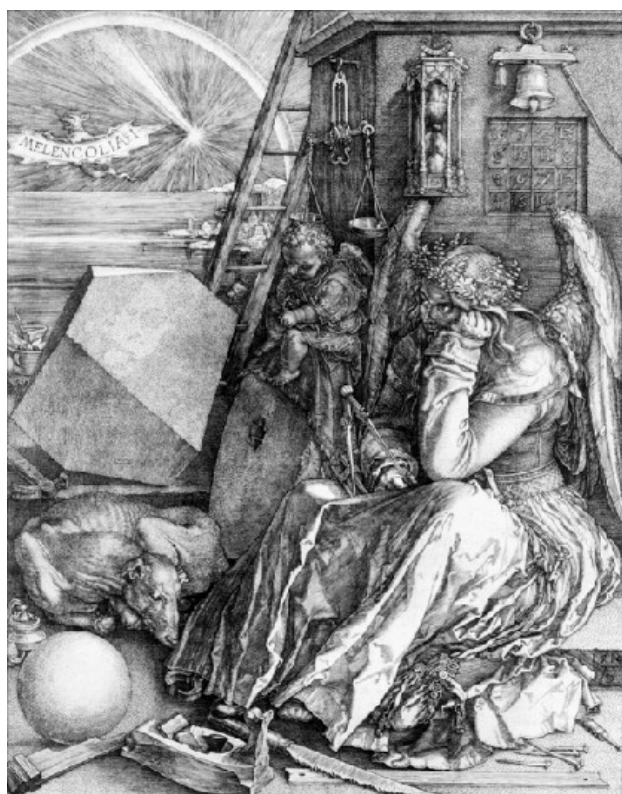


2.4. Matrici, vectori și polinoame

Pentru a lucra ușor și bine cu limbajul MATLAB trebuie în primul rând să se învețe manipularea matricilor. În MATLAB, o matrice este un tablou dreptunghiular de numere. Scalarii de exemplu sunt matrici 1×1 , iar matricile cu o singură linie sau coloană sunt de fapt vectori.

Un exemplu celebru de matrice apare în gravura renescentistă *Melancholia* realizată de marele artist și matematician amator Albrecht Dürer. Gravura este încărcată de simbolism matematic și la o atentă observare a acesteia se poate distinge în colțul din dreapta sus o matrice.

Matricea respectivă este cunoscută sub numele de pătrat magic și în timpul lui Dürer se considera că are proprietăți magice.



Introducerea matricilor

Matricile se pot introduce în mai multe moduri.

- Introducerea unei liste explicite cu elementele matricei.
- Încărcarea unor date din fișere externe de date.
- Generarea de matrici utilizând funcții built-in.
- Crearea de matrici în fișierele M-files.

Vom introduce matricea lui Dürer mai întâi ca o listă de elemente.

Trebuie respectate câteva convenții simple:

- ❑ Elementele unei linii sunt separate prin virgule sau spații.
- ❑ Sfârșitul unei linii se marchează cu punct și virgulă.
- ❑ Lista de elemente care formează matricea se delimitează cu paranteze drepte:

[]

Pentru introducerea matricii lui Dürer tastăm:

```
» A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB-ul va afișa matricea:

```
A =  
    16     3     2    13  
     5    10    11     8  
     9     6     7    12  
     4    15    14     1
```

O dată introdusă, matricea este memorată în workspace și poate fi apelată simplu, ca A.

Să vedem acum: de ce este magică?

sum, transpose, diag

Caracterul magic derivă din faptul că prin efectuarea unor operații asupra elementelor matrici rezultă numere interesante și surprinzătoare.

Dacă de exemplu însumăm elementele pe orice linie sau coloană sau de pe cele două diagonale, vom obține același număr.

Să verificăm acest lucru cu MATLAB-ul. Suma elementelor de pe cele 4 coloane se calculează rapid cu:

```
» sum(A)  
ans =  
    34    34    34    34
```

Pentru calcularea sumelor pe linii, efectuăm întâi transpunerea matricii și apoi aplică din nou comanda sum.

Transpusa se calculează cu:

```
» A'  
ans =  
    16     5     9     4  
     3    10     6    15  
     2    11     7    14  
    13     8    12     1
```

și apoi

```
» sum(A')'  
ans =  
    34
```

```
34
34
34
```

Suma elementelor de pe diagonală se calculează cu tot cu funcția `sum`, dar după ce în prealabil vom sorta cu funcția `diag` elementele de pe diagonală principală:

```
» diag(A)
ans =
    16
    10
     7
     1
» sum(diag(A))
ans =
    34
```

Un anumit element al matricii, de exemplu elementul din linia i coloana j se notează $A(i, j)$.

Prin urmare o altă cale (mai puțin rapidă) de a calcula suma de pe patra coloană de exemplu este următoarea:

```
» A(1,4) + A(2,4) + A(3,4) + A(4,4)
ans =
    34
```

Dacă specificăm un element care nu există în matrice, primim un mesaj de eroare:

```
» t = A(4,5)
Index exceeds matrix dimensions.
```

Operatorul :

Operatorul `:` este foarte important. De exemplu, expresia

```
» 1:10
```

este un vector linie

```
ans =
     1     2     3     4     5     6     7     8     9    10
```

Alte exemple:

```
» 100:-7:50
ans =
    100    93    86    79    72    65    58    51

» 0:pi/4:pi
ans =
     0    0.7854    1.5708    2.3562    3.1416
```

Expresia

```
A(1:k,j)
```

Se referă la primele k elemente ale coloanei j a lui A .

Dacă este utilizat în paranteze operatorul `:` atunci înseamnă că ne referim la toate elementele unei linii sau coloane

```
» sum(A(:,3))
```

calculează suma elementelor din coloana a treia a lui A :

```
ans =
    34
```

O altă proprietate interesantă a pătratului magic este că suma magică 34 este obținută și prin însumarea elementelor matricii și prin împărțirea la dimensiunea matricii (4):

```
» sum(1:16)/4
ans =
    34
```


Observație: suma magică pentru orice pătrat magic $n \times n$ este $(n^3 + n)/2$ (se poate calcula cu ajutorul *Symbolic Math Toolbox*).

Funcția `magic`

MATLAB-ul are o funcție built-in care creează pătrate magice de orice dimensiune (funcție pe care deja am utilizat-o):

```
>> B = magic(4)
B =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Această matrice este aproape identică cu matricea lui Dürer singura diferență fiind că cele două coloane din mijloc sunt schimbate între ele. Pentru obținerea din B a matricii lui Dürer se poate utiliza următoarea comandă MATLAB:

```
>> A = B(:, [1 3 2 4])
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Polinoame

◆ Polinoamele sunt descrise în MATLAB prin vectori linie ale căror elemente sunt de fapt coeficienții polinoamelor în ordinea descrescătoare a puterilor.

Exemplu: polinomul $p(x)=x^3+5x+6$ este reprezentat în MATLAB astfel:

```
p = [1 0 5 6]
```

◆ Un polinom poate fi evaluat pentru o valoare a lui x cu ajutorul funcției `polyval`:

```
>> polyval(p,1)
ans=
    12
```

În exemplul de mai sus este evaluat polinomul p în punctul $x=1$.

◆ Se pot afla cu ușurință rădăcinile polinomului folosind funcția `roots`:

```
>> r=roots(p)
r =
    0.5000 + 2.3979i
    0.5000 - 2.3979i
   -1.0000
```

◆ Există numeroase alte funcții și comenzi care se ocupă cu operații asupra polinoamelor, funcții care vor fi abordate într-un capitol special. Dintre acestea amintim comanda care permite înmulțirea a două polinoame, și anume `conv`:

```
>> p1=[1 3 5]
p1 =
     1     3     5
>> p2=[2 0 1 0 5]
p2 =
     2     0     1     0     5
>> p3=conv(p1,p2)
p3 =
     2     6    11     3    10    15    25
```

2.5. Operațiuni elementare cu matrici și funcții

MATLAB-ul operează cu matricile cu aceeași ușurință cu care lucrează cu scalarii. Pentru adunarea a două matrici de exemplu se folosește pur și simplu semnul + ca la o adunare obișnuită. Bineînțeles că matricile trebuie să aibă aceleași dimensiuni pentru a putea fi adunate.

Exemplu:

```
>> A=[2 3;15 -3]
A =
     2     3
    15    -3

>> B=[11 -21; 12 4]
B =
    11    -21
    12     4

>> C=A+B
C =
    13    -18
    27     1
```

Pentru înmulțirea a două matrici se folosește operatorul *, valabil de altfel și pentru operațiile cu scalari. Exemplu:

```
>> D=A*B
D =
    58    -30
   129   -327
```

Dacă dimensiunile matricilor care se înmulțesc nu sunt corespunzătoare, atunci va fi furnizat un mesaj de eroare:

```
>> E=[1 23; -12 2;1 2]
E =
     1    23
    -12     2
     1     2

>> F=A*E
??? Error using ==> *
Inner matrix dimensions must agree.
```

Pentru “depanarea” programului în cazul unor astfel de greșeli se poate utiliza comanda `size` care ne dă informații despre dimensiunile matricilor respective și permite corectarea erorilor:

```
>> size(A)
ans =
     2     2

>> size(E)
ans =
     3     2
```

MATLAB-ul include multe alte funcții care operează cu matrici și care vor fi descrise și utilizate intensiv în capitolele următoare. Amintim aici câteva: `det`, `inv`, `rank`, `eig` etc.

O facilitate interesantă a MATLAB-ului este aceea că lucrează cu matricile cu operatori logici și relaționali într-un mod asemănător acestor operații efectuate cu scalari.

De exemplu, pentru operațiunea scalară

```
>> r=17>55
r =
     0
```

MATLAB-ul răspunde cu `r = 0`, adică *fals*. Dacă dorim de exemplu să comparăm fiecare element al matricii A cu elementul corespunzător din matricea B, procedăm asemănător:

```

» L=A<=B
L =
     1     0
     0     1

```

Operatorii logici, adică & pentru ȘI (AND), | pentru SAU (OR), ~ pentru NU (NOT), vor returna valoarea 1 pentru ADEVĂRAT și 0 pentru FALS. Exemplu:

```

» A&B
ans =
     1     1
     1     1
» ~A
ans =
     0     0
     0     0

```

3. MEDIUL DE LUCRU MATLAB

- ❑ MATLAB este, după cum s-a afirmat deja, atât un limbaj cât și un mediu de programare. Ca mediu de lucru, MATLAB include facilități pentru manipularea variabilelor în spațiul de lucru, pentru importul și exportul datelor, precum și instrumente pentru dezvoltarea și manipularea fișierelor (M-files) și a aplicațiilor MATLAB.
- ❑ Mediul de programare este utilizat în mod diferit în funcție de platforma pe care rulează MATLAB-ul (este vorba de sistemul de operare care poate fi de tip Windows sau Unix).

3.1. Startul și terminarea sesiunilor de lucru

- Pe platformele de tip Windows, programul de instalare creează un “shortcut” la programul executabil, shortcut care poate fi plasat pe desktop (pe display-ul de lucru al calculatorului). Prin efectuarea unui dublu click pe icon-ul care reprezintă acest shortcut se startează MATLAB-ul.
- Pentru startarea MATLAB-ului pe un sistem UNIX trebuie tastat `matlab` la promptul sistemului de operare.

Fișiere de pornire (Startup Files)

La pornire, MATLAB-ul execută automat fișierul master `matlabrc.m` și, dacă există, fișierul `startup.m`.

Fișierul `matlabrc.m` este rezervat pentru administratorul de sistem (rețea), în timp ce fișierul `startup.m` este destinat utilizatorilor. De aici se pot seta căile de acces, se pot defini setările implicite pentru instrumentele *Handle Graphics* și se pot predefini variabile în spațiul de lucru.

De exemplu, dacă în `startup.m` se introduce linia

```
addpath /home/me/mytools
```

se adaugă un director de instrumente proprii pentru calea implicită de căutare.

Pe platformele Windows, fișierul `startup.m` se plasează în directorul `local` din directorul `toolbox`.

Opțiuni de pornire

Se pot specifica opțiuni de pornire, aceste opțiuni fiind adăugate pe calea shortcut-ului MATLAB.

În continuare sunt prezentate câteva posibile opțiuni.

Opțiunea de Startup	Descriere
Automation	Startează MATLAB-ul ca un server automat, minimizat, fără “splash screen”.
Logfile logfilename	Scrie în mod automat ieșirile din MATLAB într-un “log file” specificat.
Minimize	Startează MATLAB-ul minimizat și fără “splash screen”-ul MATLAB.
Nosplash	Startează MATLAB-ul fără afișarea “splash screen”-ului MATLAB.
r M_file	Rulează automat fișierul .m specificat imediat după pornirea MATLAB-ului.
Regserver	Modifică regiștrii Windows cu intrări adecvate tip ActiveX pentru MATLAB.
Unregserver	Modifică regiștrii Windows pentru ștergerea intrărilor ActiveX pentru MATLAB. Se utilizează pentru resetarea regiștrilor.

De exemplu, pentru a porni MATLAB-ul și a rula imediat în mod automat un fișier cum ar fi de exemplu `rezultate.m`, se poate utiliza următoarea cale pentru shortcut:

```
D:\bin\nt\matlab.exe /r rezultate
```

Terminarea unei sesiuni de lucru

- Pentru a părăsi mediul MATLAB, se tastează `quit` la promptul MATLAB.
- În cazul platformelor Windows, se poate termina sesiunea prin selectarea opțiunii `exit` din meniul File sau se poate utiliza butonul `close` vizibil în colțul din dreapta sus al ferestrei de comandă MATLAB.
- Comanda `quit` rulează fișierul `finish.m`, dacă acesta există în căile de căutare ale MATLAB-ului. Se pot include comenzi de tipul `save` în acest fișier, pentru a salva automat variabilele din spațiul de lucru la încheierea sesiunii. O altă opțiune este de a cere afișarea unei casete de dialog pentru confirmarea terminării sesiunii. Aceste două tipuri de opțiuni se pot găsi în directorul `/toolbox/local`:

`finishsav.m`: permite salvarea variabilelor curente

`finishdlg.m`: afișează o casetă de dialog pentru confirmarea opririi

Fereastra de comandă (fereastra principală)

Fereastra de comandă este fereastra principală prin intermediul căreia se poate comunica cu MATLAB-ul.



Pe platformele Windows, MATLAB-ul furnizează o fereastră specială, cu facilități de tip Windows.



Pe sistemele UNIX, fereastra de comandă este fereastra terminal din care este lansat MATLAB-ul.

Interpreterul MATLAB afișează un prompter (`>>`) indicând faptul că este gata să accepte comenzile utilizatorului. De exemplu, pentru introducerea unei matrici 3 x 3 se poate tasta:

```
>> A = [1 2 3; 4 5 6; 7 8 10]
```

și la apăsarea tastelor **Enter** sau **Return**, MATLAB-ul răspunde cu:



```
A =
     1     2     3
     4     5     6
     7     8    10
```

3.2.1. Editarea liniilor de comandă în fereastra principală







Tastele de tip săgeată și tasta **Ctrl** permit apelarea, editarea și eventual reutilizarea comenzilor editate anterior. De exemplu:

```
» rho = (1+ sqrt(5))/2
```

```
Undefined function or variable 'sqrt'.
```

Pentru eliminarea greșelii de editare a numelui funcției radical (`sqrt`) nu se mai editează din nou toată linia, ci se folosește tasta , apare din nou linia de comandă greșită și apoi cu tasta  se deplasează cursorul pe linie și se repară greșeala.

Lista completă a săgeților și tastelor care permit controlul asupra operațiunilor cu linia de comandă este prezentată în tabelul următor:

Arrow Key	Control Key	Operation
	Ctrl-p	Recall <i>previous</i> line.
	Ctrl-n	Recall <i>next</i> line.
	Ctrl-b	Move <i>back</i> one character.
	Ctrl-f	Move <i>forward</i> one character.
ctrl- 	Ctrl-r	Move <i>right</i> one word.
ctrl- 	Ctrl-l	Move <i>left</i> one word.
home	Ctrl-a	Move to beginning of line.
end	Ctrl-e	Move to <i>end</i> of line.
esc	Ctrl-u	Clear line.
del	Ctrl-d	Delete character at cursor.
backspace	Ctrl-h	Delete character before cursor.
	Ctrl-k	Delete (<i>kill</i>) to end of line.

Ștergerea ferestrei de comandă

Pentru ștergerea conținutului (afișajul) ferestrei principale se poate folosi comanda `clc`, care însă nu are ca efect ștergerea variabilelor curente din spațiul de lucru.


Controlul afișării paginilor ecran în fereastra de comandă

Pentru a controla afișarea paginilor în fereastra de comandă se utilizează comanda `more`. Setarea implicită este `more off`. Atunci când setăm `more on`, o pagină ecran este afișată. Apoi se poate utiliza:

Return	To advance to the next line
Space Bar	To advance to the next page
q	To stop displaying the output

Întreruperea unui program care rulează

Se poate întrerupe un program care rulează prin apăsarea pe tastele **Ctrl-c**.

 Pe sistemele Windows se va aștepta terminarea execuției funcțiilor de tip built-in sau a fișierelor de tip MEX-file.



Pe sistemele UNIX, execuția programului se va încheia imediat.

Comanda `format`

Comanda `format` controlează formatul numeric al valorilor afișate pe ecran și a fost deja discutată într-un paragraf anterior. Comanda are efect asupra afișării numerelor, și nu asupra modului intern de reprezentare a acestora.

Pe sistemele Windows, se poate schimba setarea implicită a formatului prin selectarea meniului **Preferences** din meniul **File** și selectarea formatului dorit din **General**.

Suprimarea afișării rezultatelor unei linii comandă

Deoarece la apăsarea tastelor **Return** sau **Enter** MATLAB-ul afișează automat rezultatele pe ecran, dacă încheiem linia de comandă cu punct și virgulă, va fi realizat calculul, dar nu va mai fi afișat. Această facilitate este importantă atunci când avem de lucrat cu matrici mari sau cu multe date. Exemplu:

```
A = magic(100);
```

Linii de comandă lungi

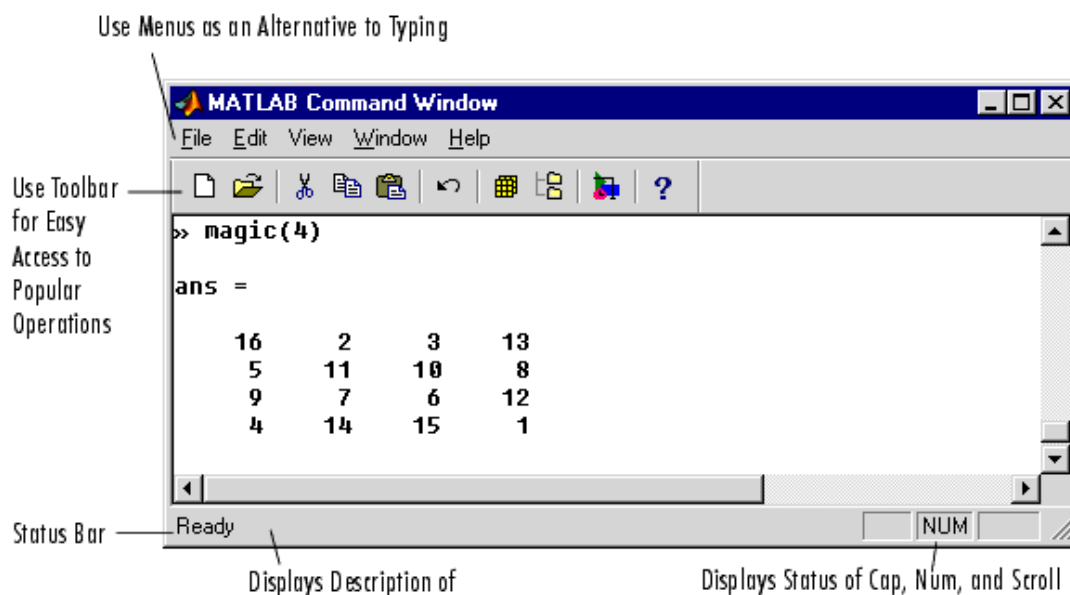
Dacă o declarație nu încapă într-o linie de comandă, se pot utiliza trei puncte urmate de **Return** sau **Enter** pentru a indica faptul că expresia continuă pe linia următoare. Exemplu:

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...  
      - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Spațiile albe din jurul semnelor `=`, `+`, `-` sunt opționale și pot îmbunătăți citirea liniilor. Maximul numărului de caractere pe o singură linie este de 4096.

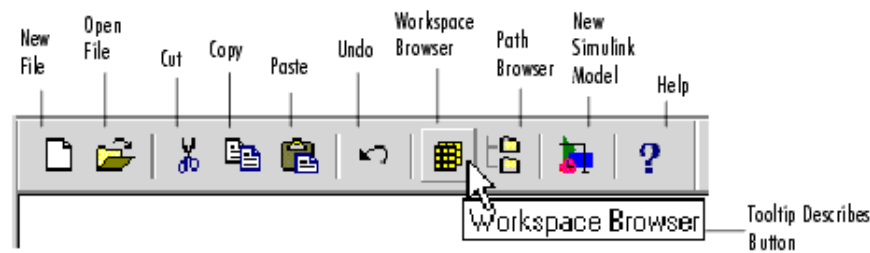
3.2.2. Descrierea ferestrei de comandă

Fereastra de comandă permite rularea comenzilor MATLAB, lansarea unor instrumente cum ar fi Editor/Debugger și permite startarea toolbox-urilor.



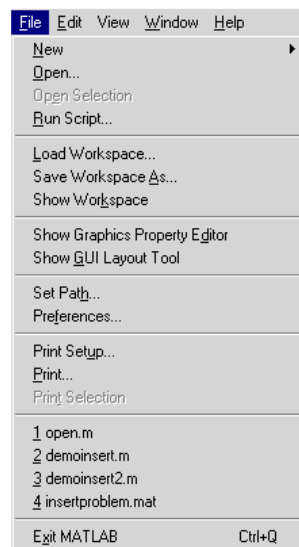
Toolbar (bara de instrumente)

Toolbar-ul din fereastra de comandă permite accesul la operații simple:



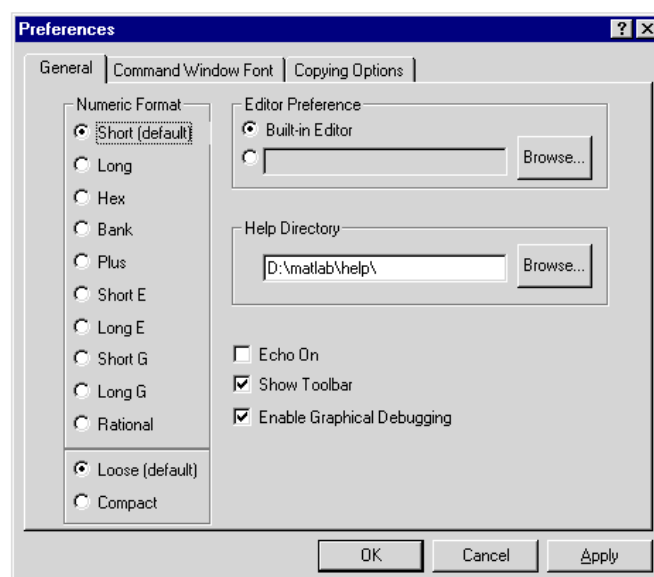
Meniuri

Meniurile fereastrei de comandă furnizează accesul la operații care nu sunt accesibile din toolbar.



Preferințe

Sunt utilizate pentru controlarea operațiilor și a modului de prezentare din fereastra de comandă. Pentru aceasta se selectează **Preferences** din meniul **File**, iar după apariția casetei de dialog **Preferences** se pot seta preferințele pentru **General**, **Command Window Font** și **Copying Options**.



General Preferences:

- **Numeric Format** – Specifică formatul numeric implicit.
- **Editor Preference** – Permite folosirea editorului MATLAB sau specificarea altui editor.
- **Help Directory** – Specifică directorul în care se află fișierele de tip help.
- **Echo On** – Setează facilitatea de afișare a liniilor program în timp ce un program MATLAB este rulat.
- **Show Toolbar** – Arată sau ascunde toolbar-ul.
- **Enable Graphical Debugging** – Permite depanarea (Debugger) în mod automat la fiecare breakpoint.

Command Window Font:

Specifică caracteristicile fontului pentru textul afișat în fereastra de comandă.

Copying Options:

Specifică opțiunile utilizate la copierea unor obiecte din MATLAB în clipboard pentru rescrierea în alte aplicații.

Spațiul de lucru al MATLAB-ului (workspace)

Spațiul de lucru conține un set de variabile (numite tablouri sau matrice) care pot fi manevrate din linia de comandă. Se pot folosi comenzile `who` și `whos` (deja prezentate) pentru a vedea care sunt variabilele curente din workspace. Pentru ștergerea variabilelor din workspace se utilizează comanda `clear`.

Încărcarea și salvarea din workspace

Comenzile `save` și `load` descrise într-un subcapitol anterior au rolul de a salva variabilele din spațiul de lucru și respectiv de a le reîncărca într-o sesiune ulterioară. Aceste comenzi se pot folosi și pentru a importa și exporta date.

 Pe platformele Windows, operațiile `save`, `load` sunt disponibile și prin selectarea opțiunilor **Save Workspace As**, respectiv **Load Workspace** din meniul **File**.

Citirea sau scrierea unor fișiere `.mat` din programe externe în C sau Fortran se poate face cu Interfața de Aplicații Program (API).

Formatul în care comanda `save` stochează datele poate fi controlat prin adăugarea unor flag-uri în lista de nume de fișiere sau variabile:

<code>-mat</code>	Utilizează formatul binar tip <code>.MAT</code> (implicit).
<code>-ascii</code>	Utilizează formatul ASCII pe 8 digiți.
<code>-ascii -double</code>	Utilizează formatul ASCII pe 16 digiți.
<code>-ascii -double -tabs</code>	Delimitează elementele tablourilor cu tab-uri.
<code>-v4</code>	Salvează într-un format pe care versiunea 4 MATLAB îl poate deschide.
<code>-append</code>	Adaugă datele într-un fișier <code>.MAT</code> existent.

Observație: atunci când se salvează conținutul spațiului de lucru în format ASCII trebuie salvată câte o variabilă pentru a permite reconstituirea ulterioară a acesteia.

Încărcarea unor fișiere cu date ASCII

Comanda `load` permite importul de fișiere de date ASCII. Exemplu:

```
» load diverse.dat
```

crează o variabilă cu numele `diverse` în workspace. Dacă datele ASCII au m linii cu n valori pe fiecare linie, rezultatul va fi o matrice numerică $m \times n$.

Nume de fișiere ca șiruri de caractere

Dacă numele fișierelor sau variabilelor cu care se lucrează sunt stocate în variabile de tip șir de caractere, se poate folosi dualitatea comandă/funcție pentru a apela `load` și `save` ca funcții. De exemplu

```
» save('myfile', 'VAR1', 'VAR2')
» A = 'myfile';
» load(A)
```

au același efect ca

```
» save myfile VAR1 VAR2
» load myfile
```

Pentru încărcarea sau salvarea mai multor fișiere cu același prefix și cu sufixe numere întregi succesive se poate utiliza o buclă. Exemplu:

```
file = 'data';
for i = 1:10
    j = i.^2;
    save([file int2str(i)], 'j');
end
```

Wildcards

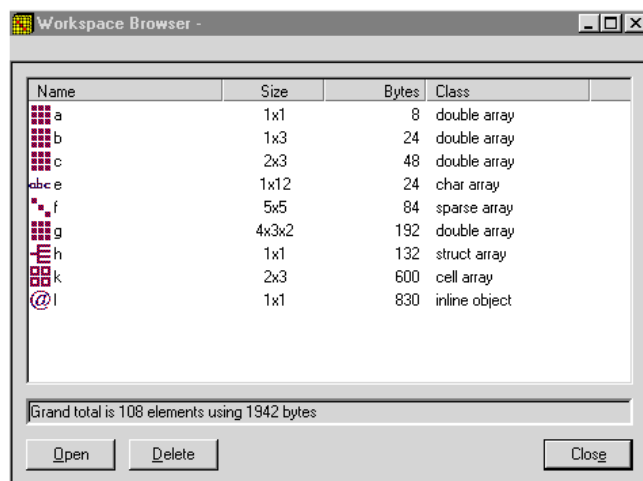
Comenzile `load` și `save` permit specificarea unui caracter special de tip wildcard (*). Exemplu:

```
» save rundata x*
```

salvează toate variabilele din workspace care încep cu `x` în fișierul `rundata.mat`.

Browser-ul Workspace

Browser-ul Workspace permite vizualizarea conținutului spațiului de lucru curent (este de fapt varianta grafică a comenzii `whos`). Pentru a deschide acest instrument, se selectează **Show Workspace** din meniul **File** și apoi se face click din mouse pe tasta **Workspace Browser** din toolbar.



3.2.3. Căi de căutare

MATLAB-ul utilizează o cale de căutare (*search path*) pentru a găsi fișierele .m . Aceste fișiere sunt organizate în directoare, din care unele sunt furnizate de MATLAB și altele sunt disponibile separat ca toolbox-uri.

Dacă de exemplu tastăm numele `fis` la promptul MATLAB, interpreterul MATLAB va face următoarea căutare:

1. Caută pe `fis` ca pe o variabilă.
2. Verifică dacă `fis` este o funcție tip built-in.
3. Caută în directorul curent fișierul numit `fis.m`.
4. Caută în directoarele aflate în calea de căutare fișierul `fis.m`.

Schimbarea căii de căutare

Calea de căutare poate fi afișată sau schimbată folosind funcțiile `path`, `addpath`, `rmpath`:

- `Path` determină reîntoarcerea la calea curentă.
- `path(s)`, unde `s` este un șir de caractere, setează calea la `s`.
- `addpath /home/lib` și `path(path, '/home/lib')` adaugă noi directoare la calea de căutare.
- `rmpath /home/lib` șterge calea `/home/lib`.

Calea de căutare implicită este definită în fișierul `pathdef.m` în directorul `local`.

Fișiere pe calea de căutare

Pentru a afișa calea de căutare se poate folosi `path`. Dacă dorim să vedem ce fișiere MATLAB sunt într-un director se utilizează comanda `what`. Exemplu:

```
» what matlab/elfun
```

Pentru a vedea conținutul unui fișier se folosește comanda `type` (deja descrisă într-un paragraf anterior). Exemplu:

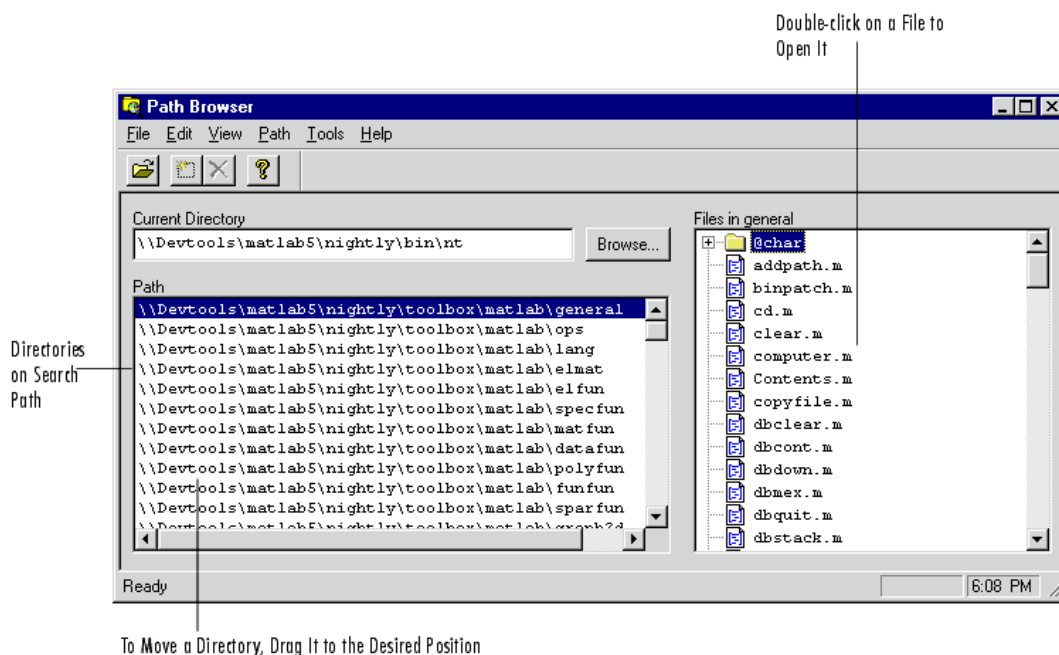
```
» type rank
```

Pentru editarea unui fișier M-file se poate utiliza `edit`. Exemplu:

```
» edit rank
```

Path Browser

MATLAB-ul furnizează și un browser al căii de căutare (**Path Browser**) cu o interfață grafică pentru vizualizarea și schimbarea căii de căutare. Pentru startarea acestui browser se utilizează `pathtool`, sau se selectează **Set Path** din meniul **File**, sau se face click pe butonul **Path Browser** din toolbar.



Meniurile din **Path Browser** pot fi folosite pentru:

- Adăugarea unui director pe calea de căutare.
- Ștergerea (îndepărtarea) unui director din cale.
- Salvarea setărilor în fișierul `pathdef.m`.
- Restabilirea setărilor implicite.

Directorul curent

MATLAB-ul menține un director curent pentru lucrul cu fișiere de tip `.m` și `.mat`.



Pe platformele Windows, directorul curent inițial este specificat în shortcut-ul utilizat pentru pornirea MATLAB-ului.

Pentru schimbarea setărilor implicite se poate face click cu butonul din dreapta al mouse-ului și se selectează meniul **Properties**.

Deschiderea fișierelor în MATLAB

Se pot deschide fișiere în funcție de extensiile lor prin folosirea funcției `open`, care este o funcție extensibilă de către utilizator. Se pot include și alte tipuri de fișiere în afara fișierelor MATLAB standard:

Nume	Acțiune
Figure file (*.fig)	Deschide o figură într-o fereastră tip figură.
M-file (name.m)	Deschide fișierul name de tip .m în Editor.
Model (name.mdl)	Deschide modelul name în Simulink.
P-file (name.p)	Deschide fișierul corespunzător name.m, dacă există, în Editor.

Variable	Deschide tabloul <code>name</code> în Array Editor (tabloul trebuie să fie numeric); <code>open</code> apelează <code>openvar</code> .
Alte extensii (<code>name.custom</code>)	Deschide <code>name.custom</code> prin apelarea funcției helper <code>opencustom</code> , unde <code>opencustom</code> este o funcție definită de utilizator.

3.3. Fereastra grafică (figure)

- MATLAB-ul direcționează ieșirile grafice spre o fereastră distinctă de fereastra de comandă. Această fereastră grafică este denumită *figură (figure)*.
- Funcțiile grafice creează în mod automat o nouă fereastră grafică dacă nu există una curentă. Dacă există o astfel de fereastră, MATLAB-ul o va utiliza.
- Dacă există mai multe ferestre de tip figură, atunci una dintre ele este asignată ca fiind fereastra curentă (în general este ultima fereastră figură utilizată).

Funcția `figure` generează ferestre grafice. De exemplu,

```
figure
```

generează o nouă fereastră și o face fereastra curentă.

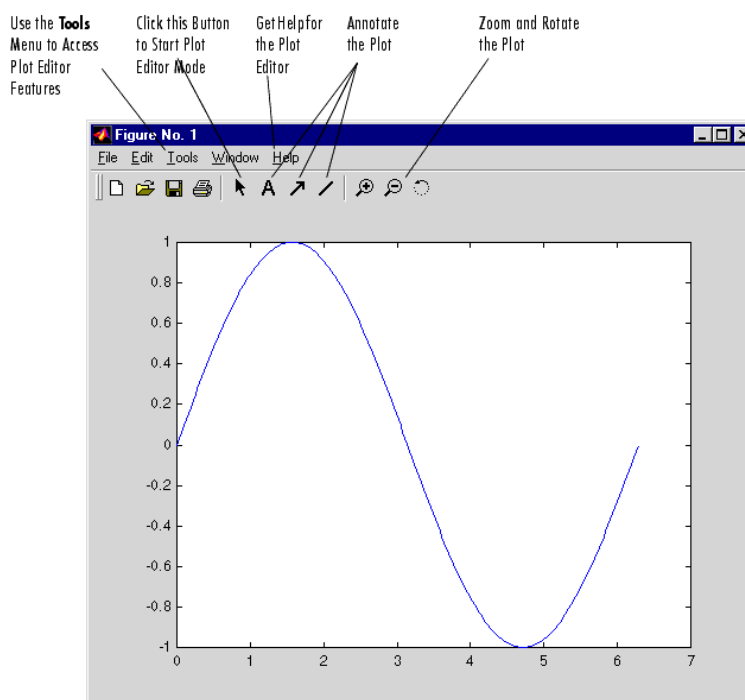
O funcție grafică, cum ar fi funcția `plot`, generează un grafic în fereastra de tip figură. De exemplu,

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
```

trasează graficul funcției sinus de la 0 la 2π în fereastra curentă de tip figură, dacă aceasta există, iar dacă nu într-una nou creată.

Prelucrarea graficelor cu Plot Editor

După generarea unui grafic (plot), se pot face schimbări și prelucrări ale graficului cu interfața grafică **Plot Editor**. Figura următoare ilustrează principalele facilități ale ferestrei grafice și ale interfeței **Plot Editor**.



Pentru salvarea unei figuri se selectează **Save** din meniul **File**. Pentru salvarea într-un format diferit, cum ar fi TIFF, necesar utilizării în alte aplicații se selectează **Export** din meniul **File**.

3.4. Importul și exportul de date

Sunt multe posibilități de a realiza importul și exportul de date între MATLAB și alte aplicații. În majoritatea cazurilor se pot utiliza facilitățile MATLAB de a citi sau scrie fișiere (pentru aplicații complicate trebuie scrise programe în C sau Fortran).

Importul de date

În tabelul următor sunt prezentate câteva metode de import date:

Metoda	Când trebuie utilizată metoda. Mod de utilizare	
Introducerea unei liste explicite de elemente	Atunci când cantitatea de date este mică. Se tastează pur și simplu datele utilizând parantezele drepte ([]).	
Crearea de date într-un fișier .m	Se utilizează un editor de text pentru generarea unui fișier .m. Metoda este utilă atunci când datele nu sunt deja în formă digitală. Este într-un fel similară cu prima metodă.	
Încărcarea datelor dintr-un fișier ASCII	Fișierele ASCII stochează datele pe linii cu un număr egal de elemente spațiate prin blanc-uri, linii încheiate cu Enter. Aceste fișiere se pot edita cu un editor de texte obișnuit. Datele sunt introduse în MATLAB cu funcția load. Se poate utiliza dlmread dacă este necesară specificarea altor delimitatori.	
Citirea datelor cu fopen, fread și cu funcțiile de intrare/ieșire	Metoda este folosită când se încarcă date de la alte aplicații, date care au propriul lor format.	
Funcții specializate de citire a fișierelor	Dlmread	Citește fișiere de date ASCII.
	Textread	Citește date de tip caracter sau numerice dintr-un fișier și le convertește în variabile MATLAB.
	Wklread	Citește fișiere de tip (WK1) (tip foaie de lucru)
	Imread	Citește din fișiere grafice.
	Auread	Citește fișiere de sunet tip (. au).
	Wavread	Citește fișiere de sunet Microsoft WAVE (. wav).
Crearea de fișiere tip MEX pentru citirea datelor	Este metoda potrivită dacă sunt deja disponibile rutine C sau Fortran pentru citirea datelor din alte aplicații.	
Dezvoltarea unor programe în Fortran sau C	Se utilizează în cazuri complexe pentru translatarea unor date în format .mat și apoi încărcarea cu comanda load.	

Exportul datelor

În tabelul următor sunt prezentate câteva metode de export date:

Metoda	Când trebuie utilizată metoda. Mod de utilizare
--------	---

Folosirea comenzii <code>diary</code>	Pentru tablouri de date de mică dimensiune se folosește comanda <code>diary</code> pentru crearea unui fișier de tip jurnal și afișarea variabilelor. Ieșirea <code>diary</code> include comenzile MATLAB folosite într-o sesiune de lucru.	
Salvarea datelor în format ASCII	Se utilizează comanda <code>save</code> cu opțiunea <code>-ascii</code> . Se poate folosi <code>dlmwrite</code> dacă este necesară specificarea altor delimitatori.	
Scrierea datelor în formate speciale	Se folosesc <code>fwrite</code> și alte funcții I/O de nivel scăzut. Este utilă la scrierea datelor în formate cerute de alte aplicații.	
Funcții specializate de scriere a fișierelor	<code>Dlmwrite</code>	Scrie fișiere în format ASCII.
	<code>wklwrite</code>	Scrie fișiere tip (WK1).
	<code>Imwrite</code>	Scrie imagini pentru fișiere grafice.
	<code>Auwrite</code>	Scrie fișiere de sunet tip (.au).
	<code>Wavwrite</code>	Scrie fișiere de sunet tip Microsoft WAVE (.wav).
Crearea unor fișiere tip MEX pentru scrierea datelor	Este metoda potrivită dacă sunt deja disponibile rutine C sau Fortran pentru scrierea datelor în formate cerute de alte aplicații.	
Scrierea datelor în fișiere tip .MAT	Se folosește comanda <code>save</code> și apoi se scrie un program în Fortran sau C pentru translatarea fișierului .mat în formatul dorit.	

Fișiere de tip text cu delimitatori

Funcțiile `dlmread` și `dlmwrite` amintite mai sus permit citirea și scrierea unor valori separate prin delimitatori într-un fișier de date ASCII. Un *delimiter* este orice caracter care separă valorile.

Exemplu: considerăm un fișier `fiz.dat` ale cărui componente sunt separate prin punct și virgulă:

```
7.2;8.5;6.2;6.6
5.4;9.2;8.1;7.2
```

Citirea și transcrierea componentelor într-un tablou (matrice) `A` se face astfel:

```
A = dlmread('fiz.dat', ';');
```

În mod similar se folosește `dmlwrite` pentru scrierea unui text cu delimitatori într-un fișier extern:

```
A =
    1     2     3
    4     5     6

dmlwrite('myfile',A, ';')
```

`myfile` va conține:

```
1;2;3
4;5;6
```

Citirea fișierelor cu format uniform

Funcția `textread` citește date de tip caracter sau numerice dintr-un fișier și le transcrie în variabile MATLAB folosind specificatorii de conversie care definesc lungimea câmpului de date și formatul acestora. Funcția `textread` este utilă pentru fișiere cu format uniform și cunoscut (de exemplu cu delimitatori de tip virgulă sau tab).

Exemplu: fie fișierul `mydata.dat` :

```
Sally      Type1 12.34 45 Yes
```

Pentru citirea fișierului `mydata.dat` ca fișier cu format liber se folosește formatul de conversie %:

```
[names,types,x,y,answer]=textread('mydata.dat','%s %s %f %d %s',1)
```

unde %s citește un șir de caractere separat prin spații albe, %f citește o valoare tip floating point, și %d citește un întreg cu semn.

MATLAB va răspunde:

```
names =  
      'Sally'  
types =  
      'Type1'  
x =  
    12.340000000000000  
y =  
     45  
answer =  
      'Yes'
```

Schimbarea de date între platforme (sisteme de operare)

În unele situații este necesar transferul de date și programe între utilizatori care lucrează cu sisteme de operare diferite. Aplicațiile MATLAB constau în fișiere .m cu funcții și script-uri și fișiere tip .mat cu date binare. Ambele tipuri de fișiere pot fi transportate direct între diferite computere:

- Fișierele .m conțin text simplu și sunt independente de “mașină”.
- Fișierele .mat sunt binare și dependente de “mașină” dar pot fi transportate între computere deoarece conțin semnătura de “mașină” în antetul fișierului.

Pentru utilizarea și transportul aplicațiilor MATLAB pe diverse computere (sisteme de operare) trebuie să ne asigurăm că fișierele .mat se transmit în *binary file mode* și fișierele .m în *ASCII file mode*.

Comanda diary

- Comanda `diary` generează o copie a sesiunii de lucru MATLAB într-un fișier disc (fără grafice). Se poate vizualiza și edita textul rezultat cu orice procesor de texte.
- De exemplu, pentru crearea unui fișier cu numele `octomb26.out` care conține comenzile și ieșirile (răspunsurile) MATLAB corespunzătoare, trebuie tastat:

```
diary octomb26.out
```

Pentru oprirea înregistrării sesiunii se folosește:

```
diary off
```

Utilizarea memoriei

- ❑ MATLAB-ul necesită o zonă continuă de memorie pentru stocarea datelor din fiecare matrice.
- ❑ De asemenea, imaginile și filmele (animația) cer o mare cantitate de memorie.
- ❑ În plus, harta de pixeli (pixmap) folosită pentru imagini cere o cantitate de memorie proporțională cu suprafața imaginii de pe ecran. O imagine color de 500x500 pixeli cere 1 Mb de memorie. Pentru limitarea memoriei necesare trebuie limitată dimensiunea imaginilor de pe ecran.

Rezolvarea erorilor de memorie

Dacă nu există memorie suficientă, în cazul unor matrici de dimensiuni mari este posibilă apariția unei erori de memorie de tip “out of memory”. Se poate încerca o defragmentare a memoriei cu comanda `pack`.

Dacă astfel de erori “out of memory” sunt dese se pot încerca și alte metode:



În cazul sistemelor Windows se crește memoria virtuală folosind **System Properties** pentru **Performance**, instrument accesibil din **Control Panel**.

X

Pentru sisteme UNIX trebuie cerut administratorului de sistem să crească spațiul swap.

Administrarea memoriei MATLAB

❖ MATLAB-ul utilizează funcțiile C standard `malloc` și `free` pentru alocarea memoriei dinamice. Aceste rutine mențin de regulă o rată relativ lentă de utilizare a memoriei alocată de sistemul de operare. Pentru MATLAB, `malloc` și `free` alocă memoria într-un ritm mai rapid. Pe măsură ce MATLAB-ul nu mai are nevoie de memorie, `malloc` și `free` nu returnează memoria adițională sistemului de operare (rutinele presupun că dacă a fost nevoie de o cantitate mare de memorie o dată, atunci este posibil să mai fie nevoie încă o dată).

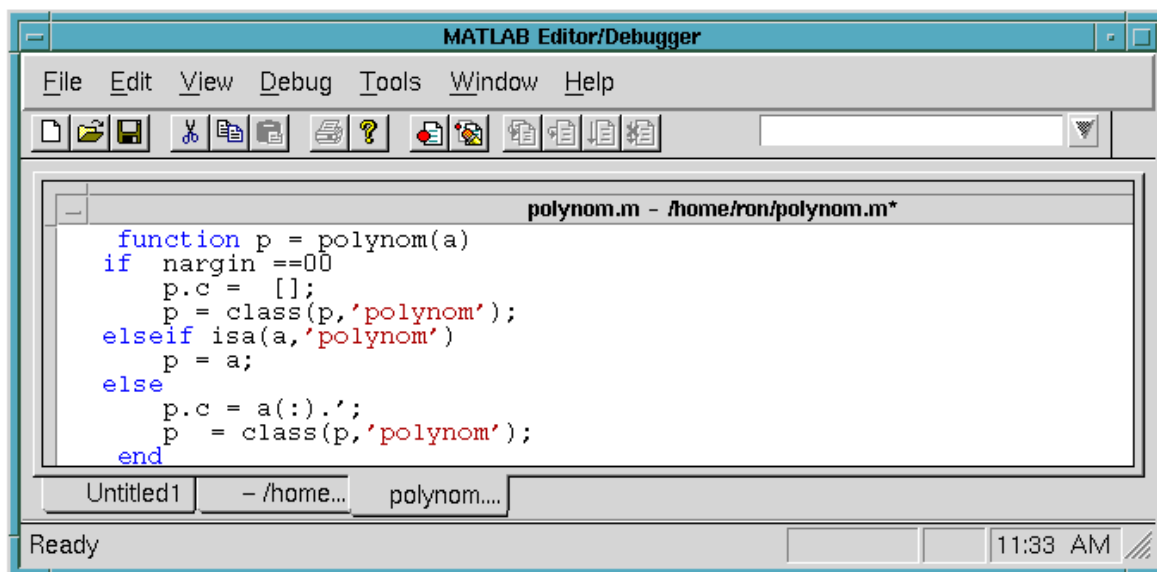
❖ Din acest motiv este posibil ca alte aplicații care rulează să nu poată dispune de memorie, chiar dacă MATLAB-ul nu mai are nevoie. Pentru a disponibiliza cantitatea de memorie trebuie terminată sesiunea MATLAB.

4. EDITORUL/DEBUGGER-UL ȘI PROFILER-UL MATLAB

4.1. Editorul/Debugger-ul MATLAB

MATLAB-ul dispune de un editor propriu, editor care este asociat și cu un program de depanare (debugger). Editorul/debugger-ul oferă posibilitatea de a efectua operațiunile de editare de bază precum și accesul la instrumente de depanare a fișierelor `.m`.

Pachetul Editor/Debugger oferă și o interfață grafică ușor de utilizat. Pentru lansarea editorului se tastează la prompterul MATLAB comanda `edit`. Se pot folosi și butoanele/meniurile accesibile din fereastra de comandă.



Setarea implicită a Editorului

Facilitățile de editare și depanare sunt setate să fie active în mod implicit atunci când MATLAB-ul este instalat.

Dacă se dorește instalarea altui editor sau nu se dorește apelarea la depanarea în regim grafic se pot dezactiva aceste facilități prin setarea corespunzătoare. De exemplu (în UNIX) se modifică `~home/.Xdefaults` file:

```
matlab*builtInEditor: Off
matlab*graphicalDebugger: Off
```

Trebuie rulat

```
xrdb -merge ~home/.Xdefaults
```

înainte de pornirea MATLAB-ului.

Debugger-ul MATLAB. Exemple de depanare a fișierelor MATLAB

➤ Debugger-ul permite identificarea erorilor de programare. Prin folosirea debugger-ului se poate vizualiza conținutul workspace-ului în orice moment în timpul execuției unei funcții și se poate executa programul (codul) MATLAB linie cu linie.

➤ Pentru folosirea acestui instrument de depanare se poate utiliza interfața grafică a debugger-ului sau se pot folosi linii de comandă.

Depanarea permite corectarea a două tipuri de erori:

- Erori de sintaxă, cum ar fi scrierea incorectă a numelui unei funcții sau omiterea unor paranteze. MATLAB-ul detectează majoritatea acestor erori și afișează un mesaj de eroare care descrie natura erorii și numărul liniei din programul `.m` în care a apărut eroarea respectivă.
- Erori de rulare (de calcul), care sunt mai mult de natură algoritmică. De exemplu este posibil să modificăm o altă variabilă decât trebuie sau să efectuăm un calcul incorect. Aceste erori sunt observate atunci când fișierul rulat furnizează rezultate necorespunzătoare.

În timp ce erorile de sintaxă se corectează relativ ușor pe baza mesajelor de eroare, erorile de rulare sunt mai greu de depanat. Se pot utiliza în acest caz mai multe tehnici de depanare:

- Se îndepărtează delimitatorii de tip punct și virgulă de la sfârșitul liniilor program. Astfel la rularea programului vor fi afișate și rezultatele intermediare corespunzătoare fiecărei linii.
- Se adaugă comanda `keyboard` în fișierele `.m` care sunt depanate. Această comandă oprește execuția programului respectiv, dă controlul la tastatură și dă posibilitatea examinării și schimbării unor funcții sau variabile. Acest mod de lucru este indicat printr-un prompter special: `"K>>."` Pentru a continua execuția, se tastează `return` și se apasă apoi tasta **Return**.
- Se folosește Debugger-ul MATLAB.

Debugger-ul este util deoarece permite accesul la funcțiile din workspace, examinarea și eventual modificarea conținutului acestora.

Debugger-ul permite setarea sau ștergerea unor puncte de oprire: *breakpoints*, care sunt linii speciale în programul MATLAB la întâlnirea cărora execuția se oprește și sunt posibile operații de schimbare și de execuție a liniilor de comandă una câte una.

Exemplu de depanare

Pentru a ilustra procedurile de depanare disponibile (îndeosebi pentru cazul erorilor de rulare) vom folosi un exemplu preluat din *MATLAB User Guide*. Vom scrie un fișier denumit `variance.m` care este o funcție având ca intrare un vector și ca ieșire un scalar. Fișierul apelează la o altă funcție, numită `sqsum`, care calculează o sumă medie pătratică a vectorului de intrare.

```
function y = variance(x)
mu = sum(x)/length(x);
tot = sqsum(x,mu);
```

```
y = tot/(length(x)-1);
```

În fișierul `sqsum.m` se strecoară intenționat o eroare.

```
function tot = sqsum(x,mu)
tot = 0;
for i = 1:length(mu)
    tot = tot + ((x(i)-mu).^2);
end
```

Pentru verificarea corectitudinii calculelor, folosim funcția MATLAB `std` (calculează deviația standard) care permite efectuarea unui calcul echivalent.

Se introduce mai întâi un vector de intrare de test:

```
» v = [1 2 3 4 5];
```

apoi se utilizează funcția `std`:

```
» var1 = std(v).^2
var1 =
    2.5000
```

Încercăm funcția `variance` care apelează funcția `sqsum` (scrisă eronat):

```
» myvar1 = variance(v)
myvar1 =
    1
```

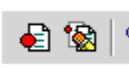






Răspunsul este greșit. Vom încerca cu debugger-ul să găsim și să corectăm greșeala.

Depanarea cu ajutorul interfeței grafice a Debugger-ului

A. Pentru startarea procedurii de depanare:

- Dacă fișierul `.m` (adică `variance.m`) a fost creat cu editorul MATLAB și suntem în fereastra **Editor/Debugger**, se continuă din acest punct.
- Dacă fișierul a fost creat cu un editor extern, se startează Editor/Debugger-ul și apoi se face click pe butonul **Open M-file** din toolbar.

Toolbar-ul Editor/Debugger conține o serie butoane descrise în continuare:

 Toolbar	Scop	Descriere	Comandă Echivalentă
	Setează/ Șterge Breakpoint	Setează sau șterge un breakpoint pe linia pe care este poziționat cursorul.	<code>Dbstop/ Dbclear</code>
	Șterge toate Breakpoint- urile	Șterge toate breakpoint-urile care sunt setate în mod curent.	<code>Dbclear all</code>
	Step In (Pas în)	Execută linia curentă a fișierului <code>.m</code> și dacă linia este o apelare la altă funcție sare (face un pas) în funcția respectivă.	<code>Dbstep in</code>
	Single Step (Un singur pas)	Execută linia curentă a fișierului <code>.m</code> .	<code>Dbstep</code>
	Continuă	Continuă execuția fișierului până la terminare sau până la alt breakpoint.	<code>Dbcont</code>
	Sfârșit depanare	Ieșirea din starea de depanare.	<code>dbquit</code>

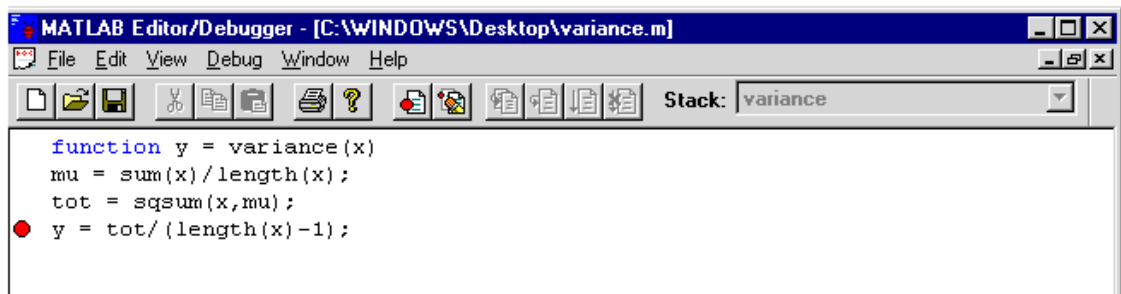
Prin apăsarea butonului din dreapta al mouse-ului în fereastra editorului se poate obține un meniu cu toate aceste opțiuni.

B. Setarea Breakpoint-urilor

Punctele de oprire (breakpoint-uri) determină oprirea execuției fișierului la linia specificată și permit evaluarea și schimbarea variabilelor din workspace înainte de reluarea execuției. Breakpoint-ul este indicat printr-un semn roșu de stop (●) înainte de linia respectivă.

Pentru exemplul considerat, la începutul depanării nu se știe unde ar putea fi eroarea, însă un loc logic de amplasare a unui breakpoint pentru a face verificări este în linia 4 a funcției `variance.m`:

```
y = tot/(length(x)-1);
```

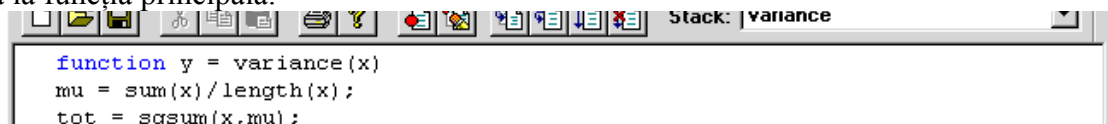


Pentru setarea breakpoint-ului se poziționează cursorul pe linia 4 și se face click pe butonul din toolbar sau se alege **Set Breakpoint** din meniul **Debug**.

C. Examinarea variabilelor

Pentru verificarea variabilelor, se execută mai întâi funcția din fereastra de comandă:
`variance(v)`

Atunci când execuția programului ajunge la breakpoint, o săgeată galbenă orizontală (\Rightarrow) arată următoarea linie care va fi executată. Dacă săgeata galbenă este verticală (\downarrow) atunci aceasta indică o pauză la sfârșitul unui script sau a unei funcții și permite examinarea variabilelor înainte de reîntoarcerea la funcția principală.



Acum putem verifica valorile variabilelor `mu` și `tot`. Se selectează textul care conține variabilele și se face click din butonul drept al mouse-ului după care se alege din meniu **Evaluate Selection**.

În fereastra de comandă va fi afișat atât textul selectat cât și rezultatul:

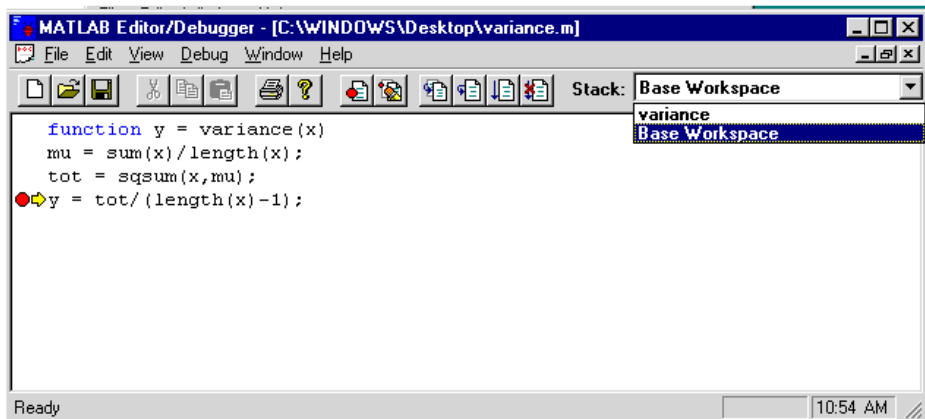
```
K>> mu
mu =
    3

K>> tot
tot =
    4
```

Din analiza acestor valori se observă că eroarea se află în `sqsum`.

D. Schimbarea contextului spațiului de lucru

Se poate folosi meniul **Stack** pentru schimbarea contextului spațiului de lucru, adică pentru ieșirea din funcția `variance` și vizualizarea conținutului workspace-ului, prin selectarea din meniu a opțiunii **Base Workspace**:

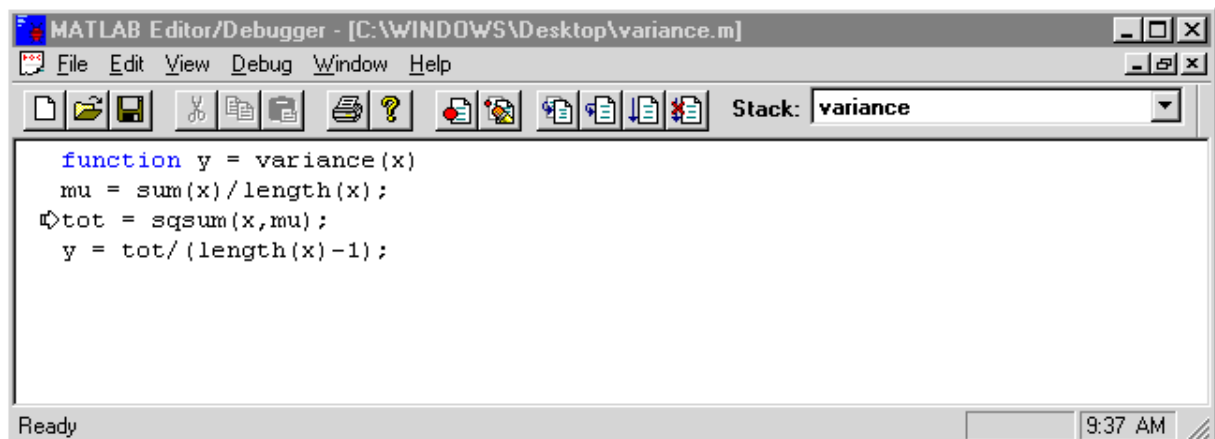


Prin utilizarea comenzii `whos` sau a Browserului Workspace se vor vizualiza variabilele `v` și `myvar1`, ca de altfel și celelalte variabile create. Pentru întoarcerea la contextul spațiului de lucru local al funcției `variance` se selectează **Variance** din meniul.

E. Executarea pas cu pas a programului și continuarea execuției

Se șterge breakpoint-ul din linia 4 din `variance.m` prin plasarea cursorului pe linie și selectarea opțiunii **Clear Breakpoint** din meniul **Debug**. Se continuă execuția programului cu **Continue** din meniul **Debug**.

Se deschide `sqsum.m` și se setează un breakpoint la linia 4 pentru verificarea buclei și a calculelor. Se rulează din nou `variance`. Execuția se va opri acum la linia 4 din `sqsum`.



Se poate acum evalua indicele buclei `i`:

```
K>> i
i =
    1
```

după care prin selectarea opțiunii **Single Step** din meniul **Debug** se execută linia următoare. Se evaluează variabila `tot`:

```
K>> tot
tot =
    4
```

Se selectează din nou **Single Step**:

```
for i = 1:length(mu)
```

Se observă că bucla este iterată numai până la lungimea lui `mu`, care este scalar, și nu până la lungimea lui `x`, vectorul de intrare (aceasta este de fapt greșeala).

O dată eroarea găsită se selectează **Quit Debugging** și se termină execuția programului. Se șterge breakpoint din `sqsum` și se pune un breakpoint la linia 4 din `variance.m`, după care rulăm din nou:

```
variance(v)
```

La oprirea execuției se setează valoarea lui `tot` la valoarea corectă (10):

```
K>> tot = 10
tot =
    10
```

Selectăm **Continue Execution** și obținem rezultatul corect.

F. Terminarea sesiunii de depanare

Se selectează **Exit Editor/Debugger** din meniul **File** și se termină sesiunea de depanare.

Pentru corectarea definitivă a erorii se folosește editorul și se rulează din nou programul pentru o ultimă verificare.

Depanarea din linia de comandă

Folosirea facilităților de depanare se poate realiza și direct din linia de comandă, prin intermediul unui set de comenzi. Aceste comenzi sunt prezentate în forma lor generală în tabelul următor.

Descriere	Sintaxă
Setarea unui breakpoint.	<code>dbstop at line_num in file_name</code>
Ștergerea unui breakpoint.	<code>dbclear at line_num in file_name</code>
Stop la atenționare, eroare sau generarea de NaN/Inf.	<code>dstop if warning error naninf infnan</code>
Reluarea execuției.	<code>Dbcont</code>
Listarea apelării de funcții.	<code>Dbstack</code>
Listarea tuturor breakpoint-urilor.	<code>dbstatus file_name</code>
Executarea a una sau mai multe linii.	<code>dbstep nlines</code>
Listarea fișierelor M-file cu liniile numerotate.	<code>dbtype file_name</code>
Schimbarea contextului spațiului de lucru local (down).	<code>dbdown</code>
Schimbarea contextului spațiului de lucru local (up).	<code>dbup</code>
Părăsirea modului de depanare.	<code>dbquit</code>

Pentru informații suplimentare privind utilizarea acestor funcții se poate apela la comanda `help` urmată de numele comenzii respective.

Exemplul de depanare a unui fișier cu erori prezentat anterior poate fi reluat, utilizându-se în locul interfeței grafice a debugger-ului comenzi corespunzătoare în linia de comandă.

4.2. Profiler-ul MATLAB

- Pentru îmbunătățirea performanțelor fișierelor MATLAB se utilizează un instrument MATLAB numit Profiler. Acest instrument furnizează informații utile privitoare la timpul alocat calculului efectuate de fiecare linie program.

➤ Cu ajutorul Profiler-ului se măsoară modul în care programul consumă timp, și o măsură este evident mai bună decât ghicitul rutinelor sau funcțiilor care consumă mult timp de calcul.

➤ Programarea eficientă presupune folosirea Profiler-ului pentru determinarea “strangulărilor” din programul creat și apoi modificarea programului pentru optimizarea timpului de calcul.

➤ Programele MATLAB au în general o structură multistrat generată de faptul că funcțiile utilizate apelează deseori alte funcții și așa mai departe. De aceea este important să fie identificate acele funcții consumatoare de timp și înlocuite dacă este posibil.

Profiler-ul permite:

- Evitarea calculelor inutile.
- Schimbarea algoritmilor pentru evitarea folosirii unor funcții consumatoare de timp.
- Evitarea recalculărilor prin stocarea unor rezultate ce pot fi utilizate ulterior.

Comanda `profile`

Pentru a crea un profil al programului (fișierului) MATLAB se folosește comanda `profile` pentru a genera și vizualiza statisticile despre programul respectiv. În tabelul următor sunt prezentate formele posibile ale acestei comenzi.

Sintaxă	Opțiuni	Descriere
<code>Profile on</code>		Startează profiler-ul și șterge statisticile înregistrate anterior.
	<code>-detail level</code>	Specifică nivelul funcției analizate.
	<code>-history</code>	Specifică secvența exactă a apelurilor făcute de funcția care va fi înregistrată.
<code>Profile report</code>		Suspendă activitatea profilerului după care generează un raport în format HTML pe care îl afișează în browserul Web.
	<code>Basename</code>	Salvează raportul în fișierul <code>basename</code> din directorul curent.
<code>Profile plot</code>		Suspendă activitatea profiler-ului după care afișează un grafic în fereastra figură cu funcțiile care consumă majoritatea timpului de execuție.
<code>Profile resume</code>		Restartează profiler-ul fără a șterge statisticile înregistrate anterior.
<code>Profile clear</code>		Șterge statisticile înregistrate.
<code>Profile off</code>		Termină activitatea profiler-ului.
<code>Profile status</code>		Afișează o structură care conține starea curentă a profiler-ului.
<code>stats = profile('info')</code>		Suspendă profiler-ul și afișează o structură cu rezultatele activității de analiză.

Exemplu de utilizare a Profiler-ului

1. Se startează profiler-ul:

```
profile on -detail builtin -history
```

Opțiunea `-detail builtin` determină profilerul să întocmească statistici și pentru funcțiile built-in.

2. Se execută un fișier .m . În exemplu este preluat programul care rulează modelul Lotka-Volterra pentru populații tip prădător-pradă (`lotkadem` pentru `demo`).

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```
3. Se generează un raport și se salvează rezultatele în fișierul `lotkaprof`.

```
profile report lotkaprof
```
4. Se restartează profiler-ul fără ștergerea statisticilor existente.

```
profile resume
```
5. Se oprește profiler-ul.

```
profile off
```

Vizualizarea rezultatelor

A. Rapoarte

Pentru afișarea unui raport cu rezultatele statistice obținute se tastează

```
profile report
```

Raportul care rezultă apare în fereastra browserului Web și începe cu un rezumat al raportului din care se pot accesa un raport detaliat și un raport al apelărilor de funcții (o cronică).

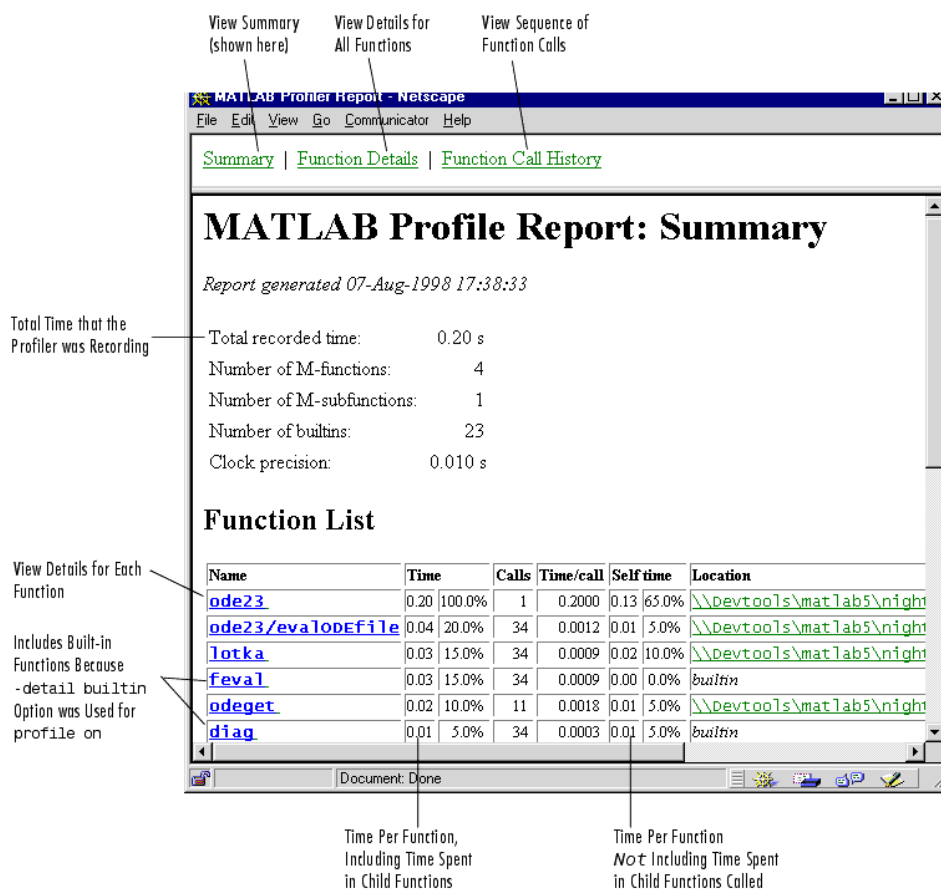
Raportul rezumat. În figura următoare este prezentat raporul rezumat pentru exemplul Lotka-Volterra.

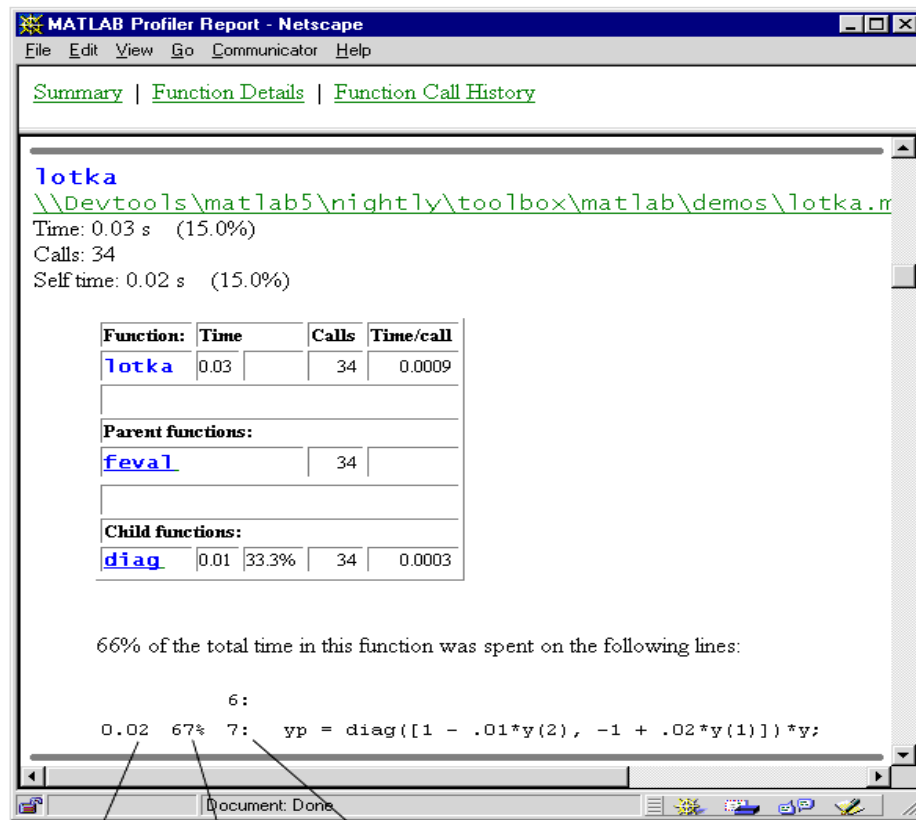
Raportul detaliat. Acest raport furnizează detalii despre funcțiile de tip “părinte” și “copil” ale unei funcții. Este prezentat raportul detaliat pentru funcția `lotka` din exemplul considerat.

Raportul apelărilor de funcții. Acest raport afișează secvența exactă a funcțiilor apelate. Pentru a vizualiza acest raport, trebuie startat profiler-ul cu opțiunea `-history`.

```
profile on -history
```

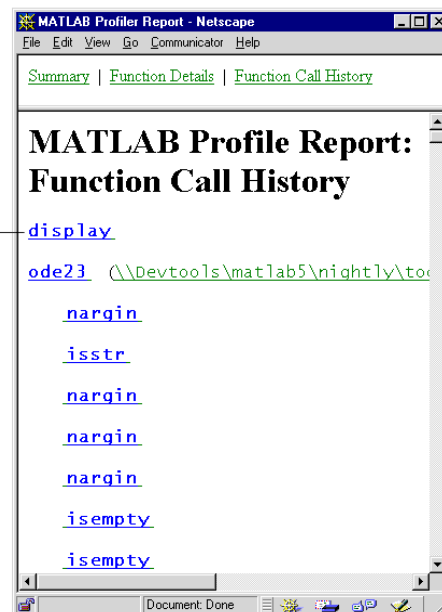
Este prezentat un exemplu de astfel de raport.





Time in Seconds Percentage of the Function's Time Spent on that Line Line Number

Exact Sequence of Calls

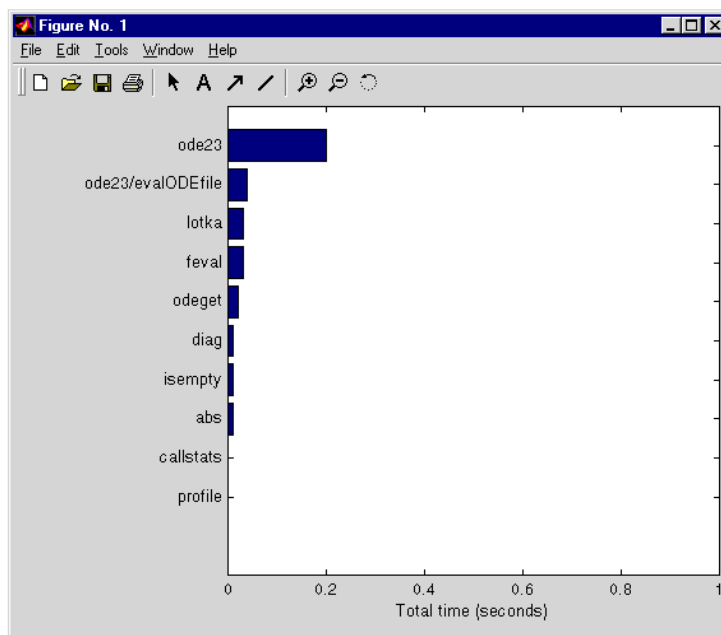


B. Reprezentarea grafică a rezultatelor Profiler-ului

Pentru a obține o reprezentare grafică trebuie să tastăm:

`profile plot`

În fereastra grafică va apare un grafic de forma din figura următoare:



5. MATRICI, ALGEBRĂ LINIARĂ, POLINOAME, TEHNICI DE INTERPOLARE

5.1. Matricile în MATLAB

- Prin matrice înțelegem un tablou bi-dimensional de numere reale sau complexe. În capitolul de Fundamente ale programării în MATLAB a fost prezentat modul de introducere a matricilor și au fost analizate câteva operații simple cu matrici.
- MATLAB-ul lucrează direct cu multe operații matriceale: aritmetica matricilor, ecuații liniare, valori proprii, factorizări etc.
- Funcțiile de algebră liniară sunt localizate în directorul `matfun`. În continuare sunt prezentate câteva din funcțiile care lucrează cu matrici.

Categoria	Funcția	Descriere
Analiza matriceală	<code>norm</code>	Norma unei matrice sau a unui vector.
	<code>normest</code>	Estimează norma-2 a matricei.
	<code>rank</code>	Rangul matricei.
	<code>det</code>	Determinant.
	<code>trace</code>	Suma elementelor de pe diagonală.
	<code>null</code>	Spațiul Nul.
	<code>orth</code>	Ortogonalizare.
	<code>subspace</code>	Unghiul dintre 2 subspații.
Ecuatii liniare	<code>\</code> și <code>/</code>	Utilizate la calculul soluțiilor ecuațiilor liniare.
	<code>inv</code>	Inversa matricei.
	<code>cond</code>	Numărul de condiție pentru inversare.

	chol	Factorizarea Cholesky.
	lu	Factorizarea LU.
	qr	Decompoziția ortogonal-triunghiulară.
	pinv	Pseudoinversa.
	lscov	Cele mai mici pătrate cu covarianță cunoscută.
Valori proprii și valori singulare	eig	Valori proprii și vectori proprii.
	svd	Decompoziția în valori singulare.
	poly	Polinomul caracteristic.
	hess	Forma Hessenberg.
	qz	Factorizarea QZ.
	schur	Decompoziția Schur.
Funcții de matrice	expm	Exponențiala de matrice.
	logm	Logaritmul de matrice.
	sqrtm	Rădăcina pătrată de matrice.
	funm	Evaluarea unei funcții generale de matrice.

Operațiunile elementare cu matrici au fost deja prezentate (adunarea matricilor, înmulțirea acestora, transpusa unei matrice, funcțiile sum, diag etc.).

În continuare sunt prezentate câteva matrici speciale utile în toate tipurile de reprezentări matematice:

- Matricea identitate (unitate)

Este o matrice cu elementele de pe diagonala principală egale cu 1 iar restul elementelor sunt nule. Notăția matematică I provine de la denumirea matricii și nu este folosită în MATLAB, pentru evitarea unor confuzii. Se utilizează sintaxa:

`eye(m,n)`

Această funcție returnează o matrice identitate $m \times n$. Dacă se folosește:

`eye(n)`

atunci este vorba de o matrice identitate pătratică $n \times n$.

- Matricea ones

Este o matrice care are toate elementele egale cu 1. Forme posibile:

`ones(n)` este o matrice $n \times n$ cu elemente de 1

`ones(m,n)` sau `ones([m,n])` sunt matrici $m \times n$ cu elemente de 1.

`ones(size(A))` are aceeași dimensiune cu o matrice A și are elemente de 1

- Matricea zeros

Este o matrice care are toate elementele egale cu 0. Forme posibile:

`zeros(n)` este o matrice $n \times n$ de zerouri

`zeros(m,n)` sau `zeros([m,n])` sunt matrici $m \times n$ de 0

`zeros(size(A))` are aceeași dimensiune cu o matrice A și are elemente de 0

5.2. Rezolvarea ecuațiilor liniare

Una din cele mai importante probleme ale calculului din domeniul tehnic este soluționarea sistemelor de ecuații liniare.

Definirea problemei este pe scurt următoarea:

➤ Dacă se dau două matrici A și B , există o matrice unică X astfel încât $AX = B$ sau $XA = B$?

MATLAB utilizează notația din cazul scalar și pentru descrierea soluției unui sistem de ecuații liniare. Cele două simboluri utilizate în cazul scalar al diviziunii (împărțirii) și anume *slash*, $/$, și *backslash*, \backslash , sunt folosite pentru definirea soluției:

$X = A \backslash B$ este soluția ecuației matriceale $AX = B$.

$X = B / A$ este soluția ecuației matriceale $XA = B$.

În practică, ecuațiile liniare de forma $AX = B$ sunt mai des întâlnite.

Deoarece matricea A , care conține de fapt coeficienții sistemului, poate să nu fie pătratică ci de tipul general $m \times n$, există trei cazuri posibile:

$m = n$.	Sistem pătratic. Se poate căuta o soluție exactă.
$m > n$.	Sistem supradeterminat (incompatibil). Se caută o soluție de tip cele mai mici pătrate.
$m < n$.	Sistem nedeterminat. Se poate căuta o soluție cu cel mult m componente nenule.

În multe cazuri MATLAB-ul dă un diagnostic (o soluție) automat prin examinarea coeficienților matricelor. Câteva din aceste cazuri:

- ◆ Permutarea matricilor triunghiulare
- ◆ Matrici simetrice, pozitiv definite
- ◆ Matrici pătratice nesingulare
- ◆ Sisteme rectangulare supradeterminate
- ◆ Sisteme rectangulare nedeterminate

Sisteme pătratice

Cel mai simplu caz este cel corespunzător unei matrice pătratice A și a unui vector coloană b . Soluția $x = A \backslash b$ are aceeași dimensiune ca b .

Dacă A și B sunt pătratice de aceeași dimensiuni atunci soluția $X = A \backslash B$ are aceeași dimensiune ca A sau B .

Observație: Dacă matricea A este singulară (determinant nul) atunci soluția ecuației $AX = B$ nu există sau nu este unică.

Sisteme supradeterminate (incompatibile)

Aceste tipuri de sisteme sunt des întâlnite în diverse situații, cum ar fi de exemplu aproximarea unor curbe din date experimentale.

Sisteme nedeterminate

Sistemele liniare nedeterminate au mai multe necunoscute decât ecuații. Dacă există și constrângeri (restricții) suplimentare, atunci este vorba de o problemă de programare liniară.

Operatorul *backslash* din MATLAB permite căutarea soluției în cazul fără restricții. Soluția nu este niciodată unică. MATLAB-ul găsește o soluție de bază (care are cel mult m componente nenule). Găsirea soluției particulare se bazează pe factorizarea QR (decompoziția ortogonal-triunghiulară).

Vom prezenta un exemplu (care utilizează funcția matriceală random – rand).

```
» R = fix(10*rand(2,4))
R =
     6     8     7     3
     3     5     4     1
```

```
» b = fix(10*rand(2,1))
b =
     1
     2
```

Sistemul linear $Rx = b$ implică două ecuații cu 4 necunoscute. Soluția se poate afișa în format rațional (coeficienții sunt numere întregi). Soluția particulară se obține astfel:

```
» format rat
» p = R\b
p =
     0
    5/7
     0
   -11/7
```

Soluția completă a sistemului nedeterminat se obține prin adăugarea unui vector arbitrar din spațiul nul folosind funcția null:

```
» Z = null(R,'r')
Z =
   -1/2    -7/6
   -1/2     1/2
     1         0
     0         1
```

Orice vector de forma $x=p+Z*q$ pentru q vector arbitrar satisface $R*x=b$.

5.3. Inverse și determinanți

- Dacă matricea A este pătratică și nesingulară, ecuațiile $AX = I$ și $XA = I$ au aceeași soluție X . Această soluție este chiar inversa lui A , notată matematic prin A^{-1} , și poate fi calculată cu funcția `inv`.
- Determinantul unei matrice se poate calcula cu funcția `det` (trebuie acordată atenție problemelor de scalare și rotunjire care apar în calcule).

Exemple:

```
» A=[1 1 1;1 2 3;1 3 6];

» d = det(A)
d =
     1

» X = inv(A)
X =
     3    -3     1
    -3     5    -2
     1    -2     1
```

Pseudoinverse

Matricile dreptunghiulare (rectangulare) nu au inverse sau determinanți. Atunci cel puțin una din ecuațiile $AX = I$ sau $XA = I$ nu are soluție. Se poate utiliza în acest caz o pseudoinversă care poate fi calculată cu funcția `pinv`:

```
» A1=[A; [1 3 5]]
A1 =
     1     1     1
     1     2     3
     1     3     6
     1     3     5
```

```

1      3      5
» X=pinv(A1)

X =
1.5000    -0.0000    1.0000   -1.5000
-0.8333    0.6667   -2.0000    2.1667
0.1667   -0.3333    1.0000   -0.8333

```

5.4. Funcții de matrice. Valori proprii

Puteri de matrice

Dacă A este o matrice pătratică și p este un număr întreg pozitiv, atunci A^p multiplică pe A cu ea însăși de p ori.

```

» X = A^2
X =
3      6      10
6      14      25
10     25      46

```

Dacă A este pătratică și nesară, atunci $A^{(-p)}$ multiplică pe $\text{inv}(A)$ cu ea însăși de p ori.

```

» Y=A^(-2)
Y =
19.0000   -26.0000    10.0000
-26.0000    38.0000   -15.0000
10.0000   -15.0000    6.0000

```

Ridicarea la putere element cu element se face utilizând operatorul (funcția) \cdot^{\wedge} . De exemplu:

```

» X = A.^2
A =
1      1      1
1      4      9
1      9     36

```

Rădăcina pătrată de matrice

Funcția $\text{sqrtm}(A)$ permite calculul lui $A^{(1/2)}$ printr-un algoritm mai precis decât utilizarea puterii de matrice.

Exponențiala de matrice

Un sistem de ecuații diferențiale ordinare cu coeficienți constanți poate fi scris:

$$\frac{dx}{dt} = Ax$$

unde $x = x(t)$ este un vector de funcții de timp și A este o matrice independentă de timp.

Soluția sistemului poate fi scrisă prin intermediul exponențialei de matrice

$$x(t) = e^{tA}x(0)$$

Funcția $\text{expm}(A)$ permite calculul exponențialei de matrice.

Valori proprii

O valoare proprie și un vector propriu ale unei matrice pătratice A sunt un scalar λ și un vector v care satisfac egalitatea

$$Av = \lambda v$$

Cu valorile proprii pe diagonala unei matrice de tip diagonal Λ și cu vectorii proprii corespunzători care formează coloanele unei matrice V vom avea

$$AV = V\Lambda$$

Dacă V este nesingulară obținem decompoziția (descompunerea) pe baza valorilor proprii:

$$A = V\Lambda V^{-1}$$

Exemplu:

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

```
A =
```

```

-1    -3     1
 2    -2    -1
 0     1    -3
```

```
>> lambda=eig(A)
```

```
lambda =
```

```

-1.7593 + 2.4847i
-1.7593 - 2.4847i
-2.4814
```

Lambda va fi un vector care conține valorile proprii ale matricei.

Dacă funcția `eig` este utilizată cu două argumente de ieșire vom obține vectorii proprii și valorile proprii (acestea sub forma diagonală):

```
>> [V,D]=eig(A)
```

```
V =
```

```

0.2233 + 0.6835i    0.2233 - 0.6835i    0.3160
0.6481 - 0.0862i    0.6481 + 0.0862i    0.4368
0.0765 - 0.2227i    0.0765 + 0.2227i    0.8422
```

```
D =
```

```

-1.7593 + 2.4847i    0    0
0    -1.7593 - 2.4847i    0
0    0    -2.4814
```

Observație: Toolbox-ul Symbolic Math extinde capacitatea MATLAB-ului prin conectarea la Maple, care este un sistem de calcul algebric performant. Una din funcțiile toolbox-ului permite calculul formei canonice Jordan.

```
>> [X,J]=jordan(A)
```

```
X =
```

```

0.1121    0.4440 + 0.1691i    0.4440 - 0.1691i
0.1549    -0.0775 + 0.4250i    -0.0775 - 0.4250i
0.2987    -0.1494 + 0.0434i    -0.1494 - 0.0434i
```

```
J =
```

```

-2.4814    0    0
0    -1.7593 - 2.4847i    0
0    0    -1.7593 + 2.4847i
```

Forma canonică Jordan este un concept teoretic important, dar nu este indicată folosirea în cazul matricilor mari sau pentru matricile cu elemente afectate de erori de rotunjire sau de alte incertitudini. MATLAB-ul poate folosi în astfel de cazuri descompunerea Schur (funcția `schur`).

5.5. Reprezentarea polinoamelor. Interpolarea

Polinoame

MATLAB-ul furnizează funcții pentru operații polinomiale standard cum ar fi calculul rădăcinilor, evaluarea polinoamelor, derivarea etc. O parte din aceste operații precum și modul de reprezentare a polinoamelor ca vectori au fost descrise în capitolul de Fundamente de programare.

Funcțiile polinomiale se află în directorul `polyfun`:

Funcție	Descriere
<code>conv</code>	Multiplică polinoamele.
<code>deconv</code>	Împarte polinoamele.

poly	Returnează coeficienții dacă se dau rădăcinile; Polinomul caracteristic.
polyder	Calcul derivatei unui polinom.
polyfit	Găsirea coeficienților unui polinom din aproximarea unui set de date.
polyval	Evaluarea unui polinom.
polyvalm	Evaluarea unui polinom cu argument matriceal.
residue	Descompunere în fracții simple.
roots	Găsirea rădăcinilor unui polinom.

- După cum s-a precizat deja, MATLAB-ul reprezintă polinoamele ca vectori linie care conțin coeficienții polinoamelor în ordinea descrescătoare a puterilor.
- Funcțiile uzuale care operează cu polinoame au fost prezentate (de exemplu `roots`). În continuare sunt parcurse alte câteva exemple utile.

❖ Funcția `poly` returnează coeficienții unui polinom dacă dispunem de rădăcinile acestuia (este o funcție inversă față de `roots`):

```

» p=[1 -1 2 4 1];
» r=roots(p)
r =
    1.0529 + 1.7248i
    1.0529 - 1.7248i
   -0.7995
   -0.3063

» coef=poly(r)
coef =
    1.0000   -1.0000    2.0000    4.0000    1.0000

```

O altă utilizare a funcției `poly` este aceea de calculare a coeficienților polinomului caracteristic al unei matrice:

```

» A
A =
    -1    -3     1
     2    -2    -1
     0     1    -3

» poly(A)
ans =
     1     6    18    23

```

Rădăcinile acestui polinom sunt chiar valorile proprii ale matricii A.

❖ Funcția `polyval` evaluează un polinom pentru o valoare specificată a argumentului.

Funcția `polyvalm` permite evaluarea unui polinom în sens matriceal. În acest caz polinomul p din exemplul anterior: $p(x) = x^4 - x^3 + 2x^2 + 4x + 1$ devine $p(X) = X^4 - X^3 + 2X^2 + 4X + I$, unde X este o matrice pătratică și I matricea unitate.

Exemplu:

```

» C=polyvalm(p,A)
C =
   -75   -61    81
    58  -130    75
    52   -23    49

```

❖ Funcțiile `conv` și `deconv` implementează operațiile de înmulțire și împărțire a polinoamelor.

Exemple:

Fie $a(x) = x^2 + 2x + 3$ și $b(x) = 4x^2 + 5x + 6$.

```
» a = [1 2 3]; b = [4 5 6];
» c = conv(a,b)
c =
     4     13     28     27     18

» [q,r] = deconv(c,a)
q =
     4     5     6
r =
     0     0     0     0     0
```

Funcția **polyder** permite calculul derivatei unui polinom.

Exemplu:

```
» p=[1 -1 2 4 1];
» pderivat=polyder(p)
pderivat =
     4     -3     4     4
```

Funcția **polyfit** găsește coeficienții unui polinom (o curbă) care aproximează un set de date în sensul algoritmului celor mai mici pătrate:

```
p = polyfit(x,y,n)
```

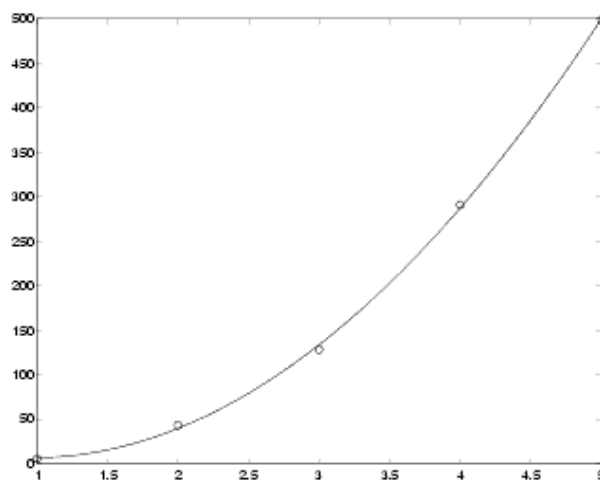
x și y sunt vectorii care conțin setul de date iar n este ordinul polinomului ai cărui coeficienți vor fi furnizați la apelarea funcției.

Exemplu:

```
» x = [1 2 3 4 5]; y = [5.5 43.1 128 290.7 498.4];
» p = polyfit(x,y,3)
p =
 -0.1917    31.5821  -60.3262    35.3400
```

Pentru plotarea rezultatului se utilizează mai întâi funcția **polyval** pentru o trasare cât mai exactă a graficului polinomului și apoi se plotează estimarea versus datele reale pentru comparații.

```
» x2 = 1:.1:5;
» y2 = polyval(p,x2);
» plot(x,y,'o',x2,y2)
```



Funcția **residue** se utilizează pentru descompunerea în fracții simple.

Se aplică în cazul raportului a două polinoame **b** și **a**,

$$\frac{b(s)}{a(s)} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \dots + \frac{r_n}{s-p_n} + k_s$$

unde r este un vector coloană, p tot un vector coloană care conține polii iar k un vector linie cu termenii direcți.

Exemplu:

$$\frac{-4 + 8s^{-1}}{1 + 6s^{-1} + 8s^{-2}}$$

```
» b = [-4 8];
» a = [1 6 8];
» [r,p,k] = residue(b,a)
r =
    -12
     8
p =
    -4
    -2
k =
     []
```

Dacă se folosesc trei argumente de intrare (r , p , și k), funcția **residue** asigură conversia înapoi în forma polinomială:

```
» [b2,a2] = residue(r,p,k)
b2 =
    -4     8
a2 =
     1     6     8
```

Interpolarea

Interpolarea este un proces de estimare a valorilor dintre date (puncte) cunoscute. Aplicațiile interpolării sunt numeroase în domenii cum ar fi procesarea numerică a semnalelor și imaginilor.

MATLAB-ul dispune de mai multe tehnici de interpolare, alegerea unei metode sau alteia făcându-se în funcție de acuratețea necesară, de viteza de execuție și de gradul de utilizare a memoriei.

Funcțiile de interpolare se află în directorul **polyfun**.

Funcție	Descriere
griddata	Interpolare pe suprafețe.
interp1	Interpolare monodimensională.
interp2	Interpolare bi-dimensională.
interp3	Interpolare tri-dimensională.
interpft	Interpolare mono utilizând metoda FFT.
spline	Interpolare spline (cubică).

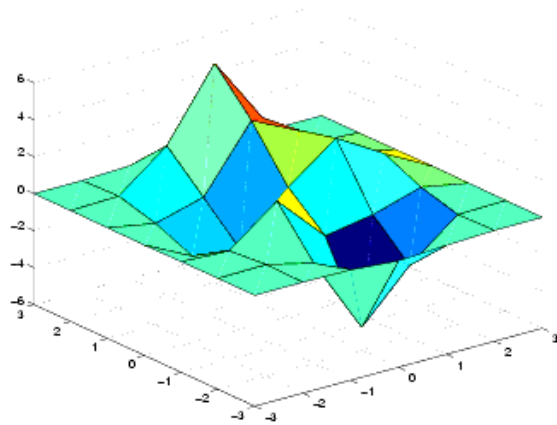
Compararea unor metode de interpolare bi-dimensională

În continuare este preluat și prezentat (informativ) un exemplu de folosire a unor metode de interpolare bi-dimensională pentru o matrice de date 7 x 7.

1. Generarea funcției **peaks** (cu rezoluție mică):

```
[x,y] = meshgrid(-3:1:3);
```

```
z = peaks(x,y);
surf(x,y,z)
```



2. Generarea unei suprafețe mesh fine pentru interpolare:

```
[xi,yi] = meshgrid(-3:0.25:3);
```

3. Interpolarea cu metoda celei mai apropiate vecinătăți:

```
zi1 = interp2(x,y,z,xi,yi,'nearest');
```

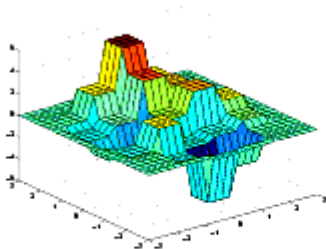
4. Interpolarea cu metoda biliniară:

```
zi2 = interp2(x,y,z,xi,yi,'bilinear');
```

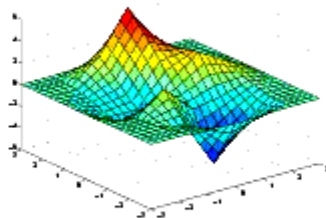
5. Interpolarea cu metoda bicubică:

```
zi3 = interp2(x,y,z,xi,yi,'bicubic');
```

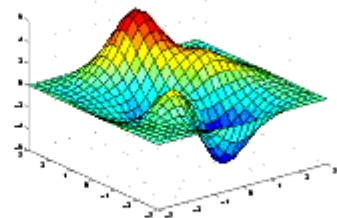
6. Compararea graficelor corespunzătoare diferitelor metode de interpolare:



```
surf(xi,yi,zi1)
% nearest
```

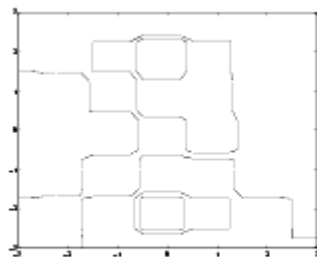


```
surf(xi,yi,zi2)
% bilinear
```

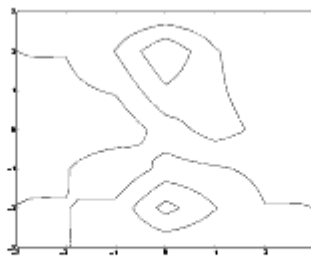


```
surf(xi,yi,zi3)
% bicubic
```

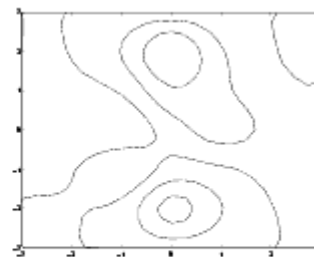
7. Compararea contururilor suprafețelor în cazul diferitelor metode de interpolare:



contour(xi,yi,zi1)
% nearest



contour(xi,yi,zi2)
% bilinear



contour(xi,yi,zi3)
% bicubic

Se observă că metoda bicubică produce cele mai netede contururi. O metodă cum ar fi cea a celor mai apropiate vecinătăți este preferată însă în anumite aplicații, cum ar fi cele medicale unde nu trebuie generate date noi.

6. REPREZENTAREA FUNCȚIILOR. ECUAȚII DIFERENȚIALE

6.1. Reprezentarea și plotarea funcțiilor matematice

Reprezentarea funcțiilor matematice

- Funcțiile matematice uzuale sunt furnizate de MATLAB ca funcții built-in (cum ar fi `sin`, `cos`, `log10`, `log`, `atan` etc.).
- Pentru reprezentarea altor funcții matematice se utilizează exprimarea în fișiere tip `.m`.

De exemplu, o funcție cum este următoarea:

$$f(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6$$

poate fi creată într-un fișier MATLAB de tip `function` și poate fi utilizată ulterior ca intrare în alte funcții (așa-numitele funcții de funcții – a se vedea paragraful 2.3).

Fișierul care descrie această funcție a mai fost prezentat în paragraful 2.3:

```
function y = humps(x)
y = 1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6;
```

O altă posibilitate este crearea la nivelul liniei de comandă a unui obiect `inline` prin folosirea unei expresii tip șir de caractere:

```
» f=inline('1./((x-.3).^2+.01)+1./((x-.9).^2+.04)-6');
```

Pentru a evalua această funcție `f` în 2.0 tastăm simplu:

```
» f(2.0)
ans =
-4.8552
```

Alt exemplu:

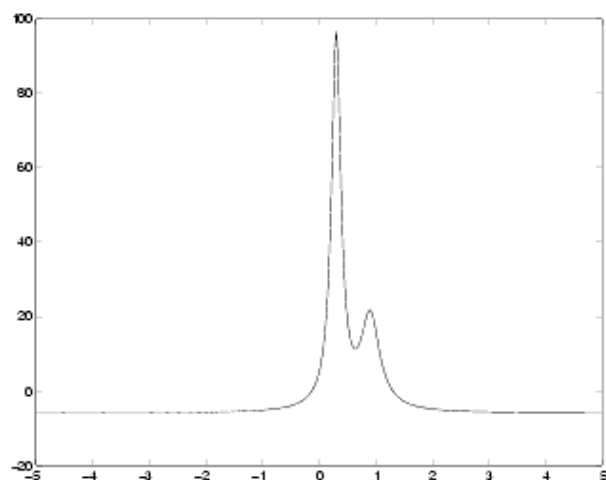
```
» f = inline('y*sin(x)+x*cos(y)', 'x', 'y')
» f(pi, 2*pi)
ans =
3.1416
```

Plotarea funcțiilor

Pentru reprezentarea grafică a funcțiilor se poate utiliza funcția `fplot`. Se pot controla limitele axelor de reprezentare grafică.

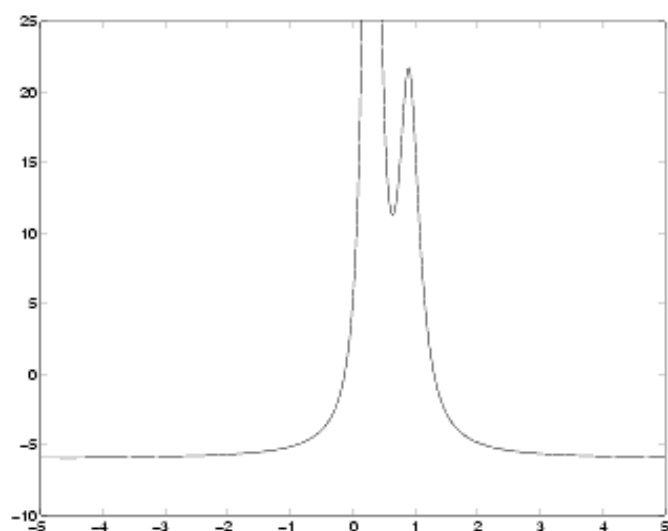
Exemplu: trasarea graficului funcției humps pentru limitele $[-5 \ 5]$ ale axei x:

```
fplot('humps', [-5 5])
```



Dacă dorim și precizarea limitelor de reprezentare pe axa y (realizarea unui zoom) folosim comanda:

```
fplot('humps', [-5 5 -10 25])
```

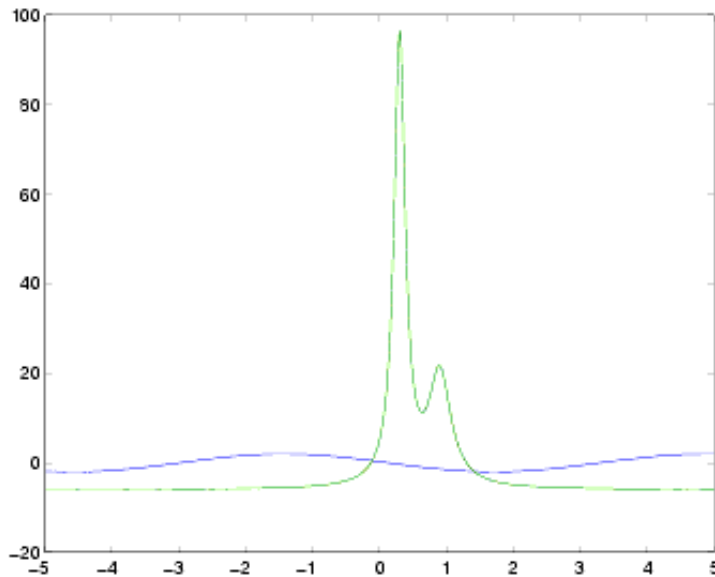


Un alt exemplu de folosire directă a funcției `fplot`:

```
fplot('2*sin(x+3)', [-1 1])
```

Se poate realiza și reprezentarea mai multor funcții pe același grafic:

```
fplot(' [2*sin(x+3), humps(x)] ', [-1 1])
```



6.2. Minimizarea funcțiilor și găsirea zerourilor

MATLAB-ul furnizează o serie de funcții de nivel înalt care realizează sarcini de optimizare a funcțiilor. Aceste funcții sunt grupate în principal pe următoarele domenii:

- Minimizarea funcțiilor de o variabilă
- Minimizarea funcțiilor de mai multe variabile
- Setarea opțiunilor de minimizare
- Găsirea zerourilor unor funcții

Pentru exemplificare vom considera minimizarea unei funcții de o singură variabilă și găsirea zerourilor pentru această funcție.

Minimizarea unei funcții

Pentru găsirea unui minim local al unei funcții scrise într-un fișier `function`, se utilizează funcția `fminbnd`.

Reluăm aici exemplul din paragraful 2.3: pentru găsirea minimului funcției `humps` în intervalul (0.3, 1) folosim instrucțiunea:

```
» x = fminbnd('humps',0.3,1)
x =
    0.6370
```

Dacă dorim o afișare detaliată a pașilor făcuți de procedura de minimizare se utilizează următoarea sintaxă:

```
» x = fminbnd('humps',0.3,1,optimset('Display','iter'))
```

Func-count	x	f(x)	Procedure
1	0.567376	12.9098	initial
2	0.732624	13.7746	golden
3	0.465248	25.1714	golden
4	0.644416	11.2693	parabolic
5	0.6413	11.2583	parabolic
6	0.637618	11.2529	parabolic
7	0.636985	11.2528	parabolic
8	0.637019	11.2528	parabolic
9	0.637052	11.2528	parabolic

```
x =
    0.6370
```

Găsirea zerourilor

Funcția `fzero` permite găsirea zerourilor unei funcții (este vorba de fapt de o ecuație cu o singură necunoscută).

- Dacă se dă un punct de plecare x_0 pentru procedura de căutare, `fzero` va căuta un interval în jurul acestui punct în care funcția schimbă semnul. Dacă un astfel de interval este găsit, `fzero` returnează valoarea pentru care funcția schimbă semnul (adică zeroul), iar dacă un astfel de interval nu este găsit returnează NaN.
- Altă variantă permite căutarea într-un interval specificat de utilizator.

Exemplu: găsirea unui zero al funcției `humps` în apropiere de -0.2 :

```
>> a = fzero('humps',-0.2)
a =
-0.1316
```

Pentru verificare evaluăm funcția în punctul -0.1316 și găsim într-adevăr o valoare foarte aproape de zero:

```
>> humps(a)
ans =
8.8818e -16
```

Se poate folosi și varianta cu interval de căutare precizat de utilizator. În continuare este prezentată această variantă plus folosirea unor opțiuni suplimentare pentru afișarea detaliată a informațiilor despre procedură și a pașilor:

```
>> options = optimset('Display','iter');
>> a = fzero('humps',[-1 1],options)
```

Func-count	x	f(x)	Procedure
1	-1	-5.13779	initial
2	1	16	initial
3	-0.513876	-4.02235	interpolation
4	0.243062	71.6382	bisection
5	-0.473635	-3.83767	interpolation
6	-0.115287	0.414441	bisection
7	-0.150214	-0.423446	interpolation
8	-0.132562	-0.0226907	interpolation
9	-0.131666	-0.0011492	interpolation
10	-0.131618	1.88371e-007	interpolation
11	-0.131618	-2.7935e-011	interpolation
12	-0.131618	8.88178e-016	interpolation
13	-0.131618	-9.76996e-015	interpolation

Zero found in the interval: [-1, 1].

```
a =
-0.1316
```

6.3. Integrarea numerică

Aria subgraficului unei funcții $F(x)$ poate fi determinată prin integrarea numerică, proces care se numește *quadratură* (*quadrature*).

Pentru rezolvarea integrării numerice în cazul monodimensional există două funcții MATLAB:

- `quad`, care folosește un algoritm de tip Simpson
- `quad8`, care utilizează un algoritm de tip Newton

Exemplu: pentru integrarea funcției `humps` între 0 și 1 folosim comanda

```
>> q = quad('humps',0,1)
q =
29.8583
```

Funcțiile `quad` sau `quad8` permit și alte argumente de intrare care specifică eroarea tolerată pentru integrare și alte opțiuni (a se vedea cu `help`).

Exemplu de integrare numerică: calculul lungimii unei curbe

Vom considera o curbă dată de ecuațiile:

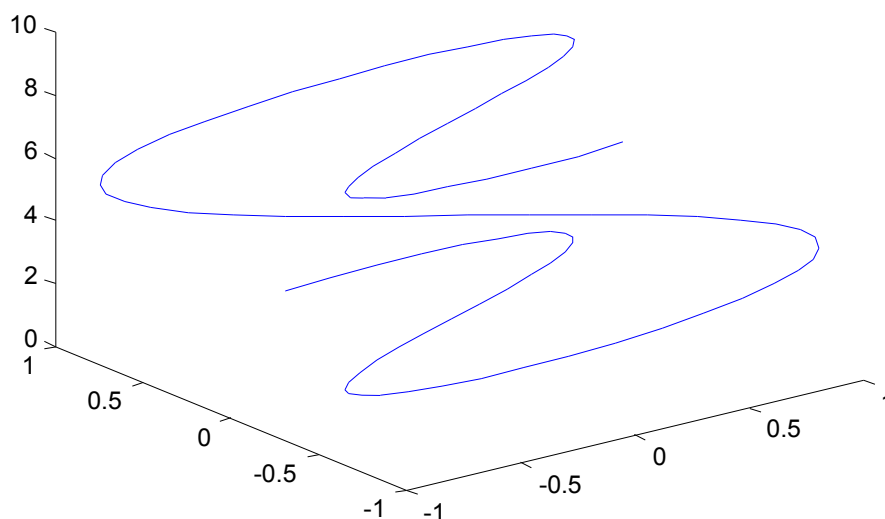
$$x(t) = \sin(2t), \quad y(t) = \cos(t), \quad z(t) = t$$

cu

$$t \in [0, 3\pi]$$

O plotare tri-dimensională a acestei curbe poate fi obținută cu

```
» t = 0:0.1:3*pi;
» plot3(sin(2*t), cos(t), t)
```



Lungimea acestei curbe este dată de formula următoare:

$$\int_0^{3\pi} \sqrt{4\cos(2t)^2 + \sin(t)^2 + 1} dt$$

Pentru calculul lungimii trebuie integrată numeric integrala de mai sus. Pentru aceasta se creează mai întâi o funcție MATLAB care descrie integrandul pe care o numim `fcurba`:

```
function f = fcurba(t)
f = sqrt(4*cos(2*t).^2 + sin(t).^2 + 1);
```

și apoi se integrează cu ajutorul funcției `quad`:

```
lungime = quad('fcurba', 0, 3*pi)
lungime =
    1.7222e+01
```

6.4. Rezolvarea ecuațiilor diferențiale

MATLAB-ul dispune de metode și funcții care pot rezolva problema condițiilor inițiale (Cauchy) ale sistemelor de ecuații diferențiale ordinare (ODE – Ordinary Differential Equations).

În tabelul următor sunt prezentate succint câteva din aceste funcții.

Categorie	Funcție	Descriere
Funcții care rezolvă ODE	ode45	Rezolvă ecuații diferențiale nonstiff, metodă de ordin mediu.
	ode23	Rezolvă ecuații diferențiale nonstiff, metodă de ordin scăzut.
	ode113	Rezolvă ecuații diferențiale nonstiff, metodă de ordin variabil.
	ode15s	Rezolvă ecuații diferențiale stiff și ecuații algebrice diferențiale, metodă de ordin variabil.
	ode23s	Rezolvă ecuații diferențiale stiff, metodă de ordin scăzut.
	ode23t	Ecuații diferențiale stiff și ecuații algebrice diferențiale, metoda trapezelor.
	ode23tb	Rezolvă ecuații diferențiale stiff, metodă de ordin scăzut.
	odeset	Creează sau schimbă opțiuni de structură ale ODE.
Opțiuni ODE	odeget	Permite obținerea parametrilor din opțiunile ODE.
	odeplot	Plotarea soluțiilor ODE (în funcție de timp).
Funcții de ieșire ODE	odephas2	Trasarea planului fazelor.
	odephas3	Trasarea spațiului fazelor (tri-dimensional).
	odeprint	Permite tipărirea soluției ODE în fereastra de comandă.

Observație: La ecuațiile diferențiale ordinare de tip stiff (rigide) soluțiile pot avea variații foarte rapide în timp în raport cu intervalul de timp de integrare și este necesară folosirea unor pași de integrare foarte mici, ceea ce nu mai este indicat la ecuațiile nonstiff.

Exemplu de rezolvare: ecuația van der Pol

Ecuația van der Pol este un exemplu clasic de ecuație diferențială:

$$y_1'' - \mu(1 - y_1^2)y_1' + y_1 = 0$$

unde $\mu > 0$ este un parametru scalar.

Pentru implementarea algoritmului de rezolvare este necesară rescrierea ecuației de ordinul 2 ca un sistem de două ecuații diferențiale de ordinul 1. Pentru aceasta se introduce variabila y_2 care este derivata în raport cu timpul a variabilei y_1 . Vom avea

$$y_1' = y_2$$

$$y_2' = \mu(1 - y_1^2)y_2 - y_1$$

Pentru a reprezenta în MATLAB acest sistem de ODE în scopul găsirii soluțiilor, trebuie scris în primul rând un fișier care descrie sistemul (un fișier de tip function). Un fișier ODE acceptă cel puțin două argumente, t și y .

Pentru ecuația van der Pol cu $\mu = 1$, fișierul este următorul (y_1 și y_2 devin $y(1)$ și $y(2)$):

```
function dy = vdp1(t,y)
dy = [y(2); (1-y(1)^2)*y(2)-y(1)];
```


La pasul următor, după ce sistemul de ecuații a fost scris, se poate utiliza una din metodele de rezolvare prezentate în tabelul anterior. Trebuie furnizat un interval de timp pentru care se dorește calculul soluțiilor și bineînțeles condițiile inițiale.

Pentru exemplul van der Pol, se poate apela la **ode45**. Dacă intervalul de timp este $[0 \ 20]$ iar condițiile inițiale $y(1)=2$ și $y(2)=0$ vom avea

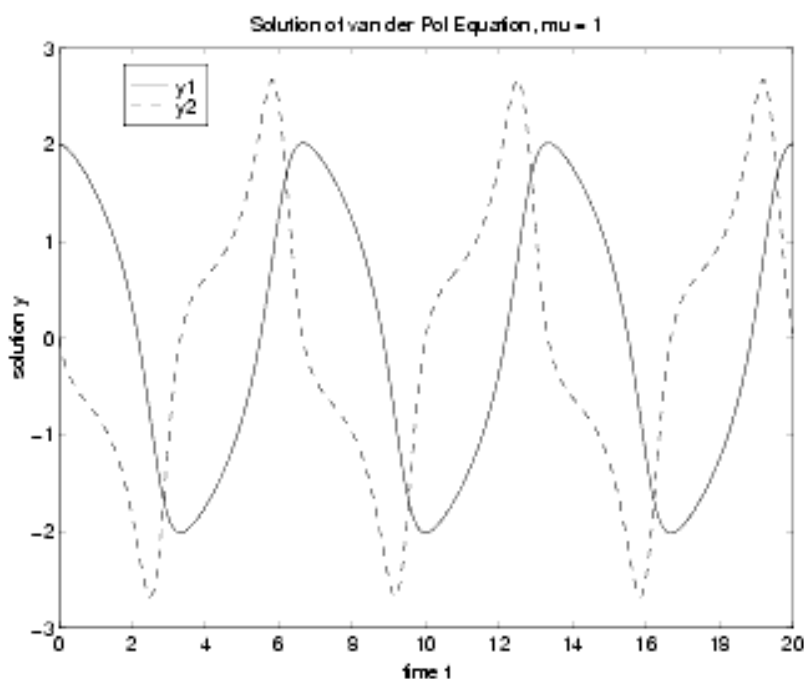
```
[t,y] = ode45('vdp1',[0 20],[2; 0]);
```

Ieșirea $[t,y]$ este un vector coloană care conține vectorul timp t și soluția de tip tablou y . Fiecare linie din y corespunde unui element (moment) din vectorul timp.

Pentru trasarea graficului cu soluția se folosește comanda `plot`:

```
plot(t,y(:,1),'-',t,y(:,2),'- -')
title('Solution of van der Pol Equation, mu = 1');
xlabel('time t');
ylabel('solution y');
legend('y1','y2')
```

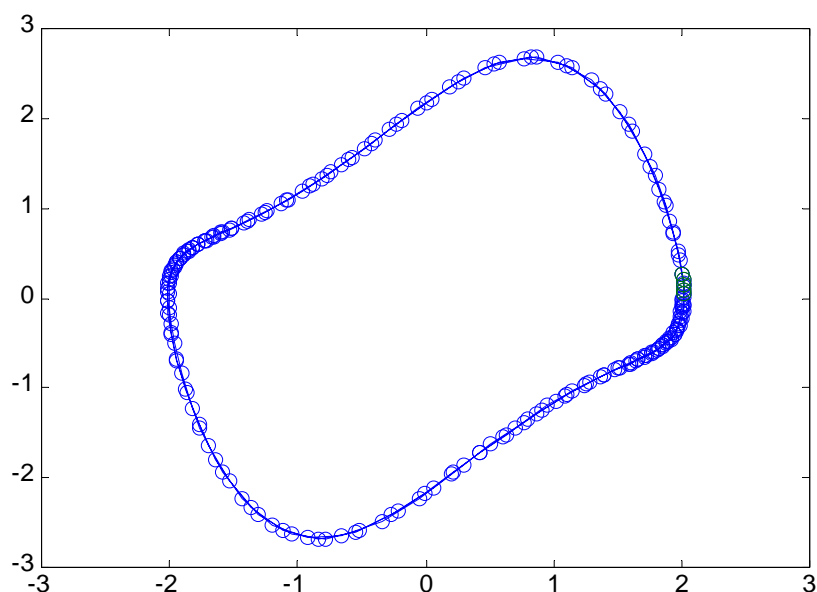
Se obține următorul grafic care conține evoluțiile celor două componente ale soluției în timp:



Dacă se dorește și trasarea planului fazelor se pot folosi liniile de comandă:

```
options=odeset('OutputFcn','odephas2');
[t,y] = ode45('vdp1',[0 20],[2; 0],options);
```

și obținem planul fazelor (este vorba de trasarea componentei $y(1)$ versus componenta $y(2)$):



7. PROGRAMAREA ÎN LIMBAJUL MATLAB

7.1. Fișiere MATLAB

Fișierele care conțin cod MATLAB sunt numite *M-files* sau fișiere *.m*. După cum s-a precizat în capitolul de Fundamente ale programării în MATLAB, aceste fișiere pot fi funcții (*functions*) care acceptă argumente de intrare și furnizează ieșiri, sau pot fi fișiere *script* care execută o serie de instrucțiuni MATLAB. Pentru ca MATLAB-ul să recunoască un fișier ca fișier M-file trebuie ca numele acestuia să se termine cu extensia *.m*.

Fișierul *.m* poate fi creat cu un editor de text și apoi poate fi folosit ca orice funcție sau comandă MATLAB:

1. Crearea unui fișier cu un editor de text.

```
function c = myfile(a,b)
c = sqrt((a.^2)+(b.^2))
```

2. Apelarea fișierului *.m* de la linia de comandă sau din alt fișier *.m*.

```
a = 7.5
b = 3.342
c = myfile(a,b)

c =
```

Caracteristicile celor două tipuri de fișiere sunt prezentate în tabelul următor:

Fișiere Script	Fișiere Function
Nu acceptă argumente de intrare și nu returnează ieșiri.	Acceptă argumente de intrare și returnează ieșiri.
Operează cu datele din workspace.	Variabilele interne ale funcției sunt locale (implicit).
Utile pentru automatizarea unei serii de pași care trebuie executați de multe ori.	Utile pentru extinderea limbajului MATLAB pentru diverse aplicații.

Script-uri

Fișierele script sunt cele mai simple fișiere MATLAB, nu au argumente de intrare sau de ieșire și sunt utile pentru executarea secvențială a unor calcule care altfel ar trebui executate în mod repetat de la linia de comandă. Script-urile operează cu datele din workspace sau pot crea date noi. Aceste date sunt disponibile după terminarea execuției fișierului.

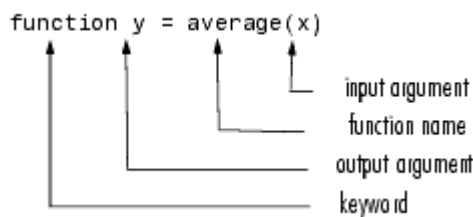
Părțile componente ale unui fișier de tip function

O funcție .m are următoarele părți componente:

- Linia de definire a funcției
- Linia de prim help H1
- Textul Help-ului
- Corpul funcției
- Comentarii

Linia de definire

Această linie informează MATLAB-ul că fișierul conține o funcție și specifică argumentele. Exemplu:



Linia H1

Linia H1 este o linie de comentariu care începe cu semnul "%" și furnizează prima linie text atunci când utilizatorul tastează `help function_name` la promptul MATLAB.

Textul Help-ului

Se poate crea un help online prin introducerea uneia sau mai multor linii de comentariu după linia H1, fiecare linie începând cu "%".

Corpul funcției

Corpul funcției conține toate instrucțiunile în cod MATLAB care permit efectuarea calculelor și asignează valori argumentelor de ieșire. Declarațiile din corp pot conține apelări de funcții, instrucțiuni de salt, intrări/ieșiri interactive, calcule etc.

Comentarii

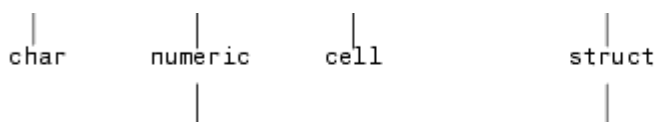
O linie de comentariu începe după cum s-a precizat cu semnul "%" și poate fi plasată oriunde într-un fișier.

Pot fi adăugate linii albe oriunde în fișier, acestea fiind ignorate.

7.2. Tipuri de date și operatori

Tipuri de date

MATLAB-ul are șase tipuri fundamentale de date (sau clase), fiecare putând fi considerată ca tablou multidimensional. Cele șase clase sunt: `double`, `char`, `sparse`, `storage`, `cell` și `struct`. Versiunile bi-dimensionale ale acestor tablouri sunt numite *matrici* și de aici provine și numele de MATLAB.



În tabelul următor sunt prezentate detaliat tipurile de date:

Clasa	Exemple	Descriere
Array		Tip de date virtual.
Cell	{17'hello'eye(2)}	Tablou tip celulă. Elementele celulei conțin alte tablouri.
Char	'Hello'	Tablou de tip caracter (sau șir de caractere–string); fiecare caracter are 16 biți lungime.
Double	[1 2;3 4] 5+6i	Tablou numeric în dublă precizie (cel mai obișnuit tip de variabilă MATLAB).
Numeric		Tip de date virtual.
sparse	Speye(5)	Matrice de tip “sparse” în dublă precizie (doar 2-D). Tablourile de tip “sparse” stochează matrici cu doar câteva elemente nenule într-o fracțiune din spațiul necesar unei matrici normale echivalente.
storage		Tip de date virtual.
struct	a.day = 12; a.color = 'Red'; a.mat = magic(3);	Tablou tip structură, care conține nume de câmpuri, câmpuri care conțin alte tablouri.
uint8	Uint8(magic(3))	Tablou de numere întregi fără semn pe 8 biți.
User Object	Inline('sin(x)')	Tip de date definit de utilizator.

Operatori

Operatorii MATLAB pot fi clasificați în trei categorii:

- Operatori aritmetici
- Operatori relaționali care compară operanzii cantitativ
- Operatori logici

Operatori aritmetici

+	Adunare	:	Operatorul două puncte
–	Scădere	. ^	Putere
. *	Înmulțire	. '	Transpusa
. /	Împărțire la dreapta	'	Transpusa complex conjugată
. \	Împărțire la stânga	*	Înmulțire de matrici
+	Plus unar	/	Împărțire matriceală la dreapta
–	Minus unar	\	Împărțire matriceală la stânga
		^	Putere de matrice

Cu excepția unor operatori matriceali, operatorii aritmetici lucrează cu elementele corespondente ale unor tablouri de dimensiuni egale. Pentru vectori și tablouri dreptunghiulare ambii operanzi trebuie să aibă aceeași dimensiune, cu excepția situației în care unul dintre ei este

scalar. În acest caz MATLAB-ul aplică scalarul fiecărui element al celui alt operand (proprietatea de expansiune scalară).

Operatori relaționali

<	Mai mic
<=	Mai mic sau egal
>	Mai mare
>=	Mai mare sau egal
==	Egal cu
~=	Diferit de

Operatorii relaționali compară elementele corespondente ale unor tablouri de dimensiune egală. Operatorii relaționali lucrează totdeauna element cu element. Exemplu:

```
» A = [2 7 6; 9 0 5; 3 0.5 6];
» B = [8 7 0; 3 2 5; 4 -1 7];

» A == B
ans =
    0     1     0
    0     0     1
    0     0     0
```

Operatori logici

&	AND (ȘI)
	OR (SAU)
~	NOT (NU)

- O expresie care utilizează operatorul & este adevărată dacă ambii operanzi sunt adevărați. În termeni numerici, expresia este adevărată dacă ambii operanzi sunt nenuli. Exemplu:

```
» u = [1 0 2 3 0 5];
» v = [5 6 1 0 0 7];
» u & v
ans =
    1     0     1     0     0     1
```

- O expresie care utilizează operatorul | este adevărată dacă unul dintre operanzi este logic adevărat sau dacă ambii operanzi sunt adevărați. În termeni numerici, expresia este falsă dacă ambii operanzi sunt nuli. Exemplu:

```
» u | v
ans =
    1     1     1     1     0     1
```

- O expresie care utilizează operatorul NOT, ~, neagă operandul. În termeni numerici, orice operand nenul devine nul și orice operand nul devine unu. Exemplu:

```
» ~u
ans =
    0     1     0     0     1     0
```

Operatorii logici lucrează cu elementele corespondente ale unor tablouri de dimensiuni egale. Pentru vectori și tablouri dreptunghiulare ambii operanzi trebuie să aibă aceeași dimensiune, cu excepția situației în care unul dintre ei este scalar. În acest caz, ca și la operatorii aritmetici, MATLAB-ul aplică scalarul fiecărui element al celui alt operand.

Funcții logice

În plus față de operatorii logici MATLAB-ul furnizează și funcții logice:

Funcție	Descriere	Exemple
xor	Realizează sau exclusiv. Returnează logic adevărat dacă unul din operanzi este adevărat și celălalt fals. În termeni numerici, returnează 1 dacă un operand este nenul și celălalt este zero.	<pre>» a = 1; » b = 1; » xor(a,b) ans = 0</pre>
all	Returnează 1 dacă toate elementele unui vector sunt adevărate sau nenule. Operează și cu matrici (pe coloane).	<pre>» u = [0 1 2 0]; » all(u) ans = 0 » A = [0 1 2; 3 5 0]; » all(A) ans = 0 1 0</pre>
any	Returnează 1 dacă oricare din elementele argumentului sunt adevărate sau nenule; în caz contrar returnează 0.	<pre>» v = [5 0 8]; » any(v) ans = 1</pre>

Alte funcții: `isnan`, `isinf`, `find` (a se folosi `help` pentru detalii).

Prioritatea operatorilor

Deoarece se pot construi expresii cu diverse tipuri de operatori, nivelurile de prioritate determină ordinea în care sunt evaluate expresiile. În cadrul fiecărui nivel, operatorii au prioritate egală și sunt evaluați de la stânga la dreapta.

Regulile de prioritate sunt prezentate în continuare, de la nivelul de prioritate cel mai mare spre cel mai mic.

Operator	Nivel de prioritate
()	Prioritate maximă
~ (negare)	
. ' . ^ ' ^ + (plus unar) - (minus unar)	
. * . / . \ * / \	
+ (adunare) - (scădere)	
: < <= > >= == ~=	
&	Prioritate minimă

7.3. Instrucțiuni de salt și bucle

În MATLAB există mai multe tipuri de instrucțiuni de control al buclelor:

- `if`, împreună cu `else` și `elseif` execută un grup de instrucțiuni pe baza unei condiții logice.
- `switch`, `case` și `otherwise` execută diverse grupuri de instrucțiuni în funcție de valoarea unei anumite condiții logice.
- `while` execută un grup de instrucțiuni de un număr nedefinit de ori, pe baza unei condiții logice.
- `for` execută un grup de instrucțiuni de un număr fixat de ori.

- `break` termină execuția pentru o buclă `for` sau `while`.
- `try...catch` schimbă controlul buclei dacă o eroare este detectată în timpul execuției.
- `return` provoacă întoarcerea la funcția care a apelat procedura.

Toate instrucțiunile de salt folosesc comanda `end` pentru a indica sfârșitul blocului respectiv.

Exemple de utilizare a unor instrucțiuni de salt:

➤ Instrucțiunile `if` și `elseif`:

```
if n < 0                                % Daca n este negativ afiseaza un mesaj de eroare.

    disp('Intrarea trebuie sa fie pozitiva');

elseif rem(n,2) == 0                    %Daca n este pozitiv si par, imparte-l la 2.
    A = n/2;

else

    A = (n+1)/2;                        %Daca n este pozitiv si impar incrementeaza si imparte la 2

end
```

➤ Instrucțiunea `for`:

```
for i = 1:m
    for j = 1:n
        A(i,j) = 1/(i + j - 1);
    end
end
```

7.4. Evaluarea datelor de tip caracter

Evaluarea datelor de tip caracter asigură putere și flexibilitate limbajului MATLAB.

Funcția `eval`

Funcția `eval` evaluează un șir de caractere care conține o expresie, o declarație sau un apel de funcție. În cea mai simplă formă, sintaxa este următoarea:

```
eval('string')
```

Exemplu: evaluarea unei expresii folosite la generarea unei matrice Hilbert de ordinul `n`:

```
t = '1/(i+j-1)';
for i = 1:n
    for j = 1:n
        a(i,j) = eval(t);
    end
end
```

Alt exemplu de utilizare a funcției `eval` pentru o declarație:

```
eval('t = clock')
```

Funcția `feval`

Funcția `feval` diferă de `eval` prin faptul că execută o funcție a cărui nume este într-un șir de caractere. Se poate folosi `feval` și funcția `input` pentru a alege din mai multe sarcini definite de fișiere `.m`. Exemplu:

```
fun = ['sin'; 'cos'; 'log'];
```

```
k = input('Choose function number: ');
x = input('Enter value: ');
feval(fun(k,:),x)
```

Este indicată folosirea funcției `feval` în locul funcției `eval`, deoarece execuția este mai rapidă.

Construirea șirurilor de caractere pentru evaluare

Se pot concatena șirurile de caractere pentru a crea expresii de intrare necesare funcției `eval`. În continuare este prezentat un exemplu în care funcția `eval` creează 10 variabile numite `P1`, `P2`, ..., `P10`, și setează fiecare variabilă la o anumită valoare:

```
for i=1:10
    eval(['P',int2str(i),'= i.^2'])
end
```

7.5. Reprezentarea și manipularea informațiilor despre dată și timp

MATLAB-ul furnizează funcții pentru manipularea informațiilor despre dată și timp, funcții grupate în directorul `timefun`.

Categorie	Funcție	Descriere
Data și timpul curent	<code>now</code>	Data și timpul curent ca număr serial.
	<code>date</code>	Data curentă ca șir de caractere.
	<code>clock</code>	Data și timpul curent ca vector.
Conversii	<code>datenum</code>	Conversia la număr serial al datei.
	<code>datestr</code>	Conversia la reprezentare de tip caracter.
	<code>datevec</code>	Componentele datei.
Utilitare	<code>calendar</code>	Calendar.
	<code>weekday</code>	Ziua din săptămână.
	<code>eomday</code>	Ultima zi din lună.
	<code>datetick</code>	Etichete formate de tip dată.
Timing	<code>cputime</code>	Timpul CPU în secunde.
	<code>tic, toc</code>	Start și oprire pentru timer.
	<code>etime</code>	Timp scurs.

7.6. Intrări utilizator

Pentru a obține o intrare de la utilizator în timpul execuției unui fișier există următoarele posibilități:

- Afișarea unui prompter prin intermediul unei funcții tip `input` și introducerea unor date de la tastatură.
- Oprirea execuției cu o comandă `pause` (reluarea execuției la apăsarea unei taste).
- Construirea unei interfețe grafice GUI completă.

Funcția `input` asigură afișarea unui prompter și așteaptă un răspuns de la utilizator. Sintaxa este:

```
n = input('prompt_string')
```

Funcția determină afișarea șirului de caractere `prompt_string`, așteaptă o intrare de la tastatură și returnează valoarea introdusă de la tastatură. Funcția este utilă pentru implementarea aplicațiilor de tip meniu. Această funcție poate să returneze intrarea de la utilizator sub formă de caracter. Exemplu:

```
name = input('Enter address: ','s');
```


Comanda `pause`, fără argumente, oprește execuția până la apăsarea unei taste. Pentru a avea o pauză de n secunde se folosește comanda:

```
pause(n)
```

8. GRAFICE ȘI INTERFEȚE GRAFICE ÎN MATLAB

8.1. Tehnici de plotare

În general, pentru a realiza o reprezentare grafică, trebuie parcurse etapele următoare:

Etapa	Instrucțiuni
1. Pregătirea datelor	<pre>x = 0:0.2:12; y1 = bessell(1,x); y2 = bessell(2,x); y3 = bessell(3,x);</pre>
2. Selectarea ferestrei grafice și poziționarea graficului în fereastră	<pre>figure(1) subplot(2,2,1)</pre>
3. Apelarea unei funcții elementare de plotare	<pre>h = plot(x,y1,x,y2,x,y3);</pre>
4. Selectarea caracteristicilor liniei și markerului.	<pre>set(h,'LineWidth',2,{ 'LineStyle'},{ '--'; ':'; '-.'}) set(h,{ 'Color'},{ 'r'; 'g'; 'b'})</pre>
5. Setarea limitelor axelor, gridare (caroieră)	<pre>axis([0 12 -0.5 1]) grid on</pre>
6. Completarea graficului cu etichete pe axe, legendă, text	<pre>xlabel('Time') ylabel('Amplitude') legend(h,'First','Second','Third') title('Bessel Functions') [y,ix] = min(y1); text(x(ix),y,'First Min \rightarrow',... 'HorizontalAlignment','right')</pre>
7. Export grafice	<pre>print -depsc -tiff -r200 myplot</pre>

Funcțiile de bază folosite la plotare sunt prezentate în tabelul următor:

Funcție	Utilizare
<code>Plot</code>	Generează grafice 2-D cu scalare liniară a axelor
<code>Plot3</code>	Generează grafice 3-D cu scalare liniară a axelor
<code>loglog</code>	Generează grafice cu scalare logaritmică a axelor
<code>semilogx</code>	Generează grafice cu scalare liniară a axei y și cu scalare logaritmică a axei x
<code>semilogy</code>	Generează grafice cu scalare liniară a axei x și cu scalare logaritmică a axei y
<code>plotyy</code>	Generează grafice cu dublă reprezentare a axei y (pe stânga și pe dreapta)

8.1.1. Plotări 2 D elementare

Generarea graficelor

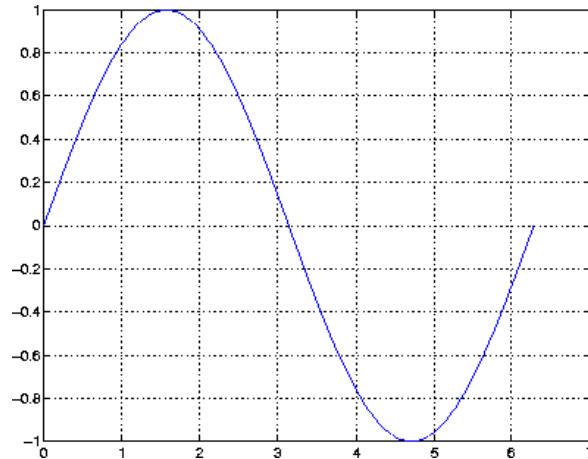
Funcția `plot` are diferite forme în funcție de argumentele de intrare.

- Dacă de exemplu y este un vector, `plot(y)` produce un grafic liniar al elementelor lui y versus indexul elementelor sale.

- Dacă se specifică doi vectori ca argumente, `plot(x,y)` produce graficul lui y versus x .

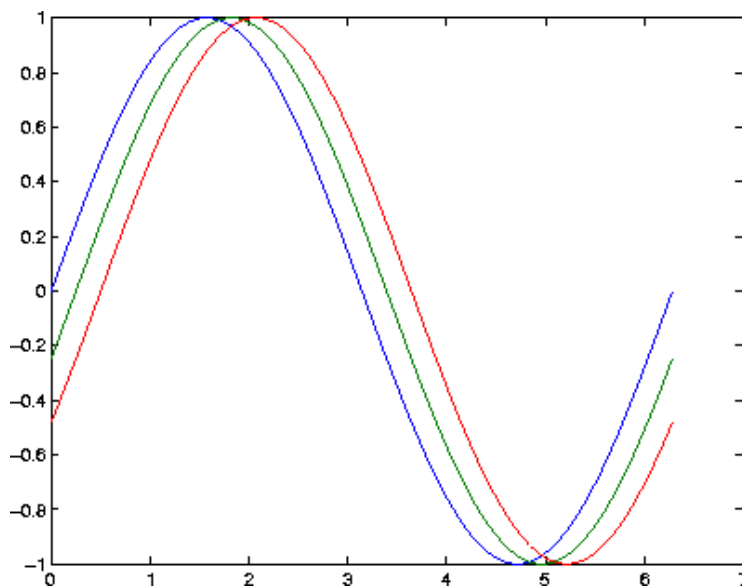
Exemplu:

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
grid on
```



Se pot realiza grafice multiple utilizând un singur apel al funcției `plot`. MATLAB-ul realizează automat o reprezentare cu culori diferite pentru a permite distingerea graficelor. Exemplu:

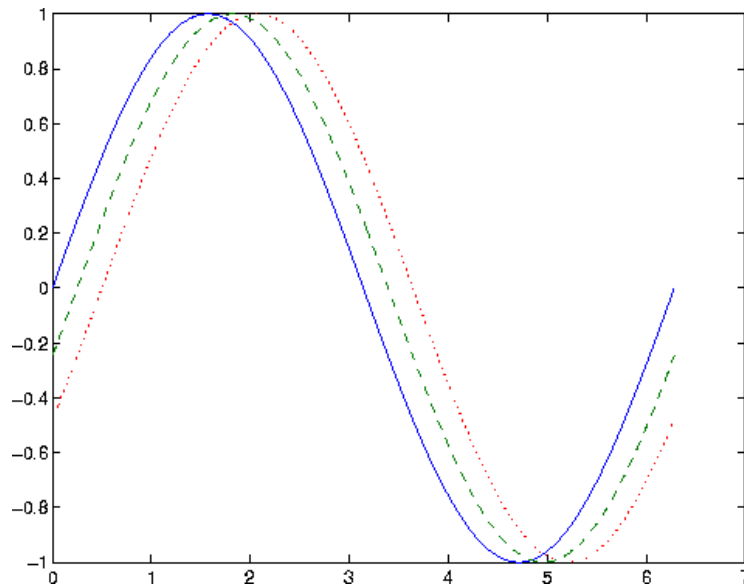
```
y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,t,y2,t,y3)
```



Specificarea stilului de linie

Se pot crea diferite tipuri de linii pentru fiecare set de date prin folosirea unor identificatori de tip string în funcția `plot`. Exemplu:

```
t = 0:pi/100:2*pi;
y = sin(t);
y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,'-',t,y2,'--',t,y3,':')
```



Funcțiile de plotare acceptă deci argumente de tip caracter care specifică stilul liniei, simbolurile utilizate pentru marker, culoarea etc. Forma generală este:

```
plot(x,y,'linestyle_marker_color')
```

unde *linestyle_marker_color* este un șir de caractere construit din:

- Un stil de linie (de exemplu linie punctată, plină etc.)
- Un tip de marker (de exemplu x, *, o, etc.)
- Un specificator de culoare (c, m, y, k, r, g, b, w)

Se poate folosi un specificator sau mai mulți, în orice ordine. De exemplu,

'go--'

definește o linie întreruptă, cu markere circulare, ambele colorate în verde.

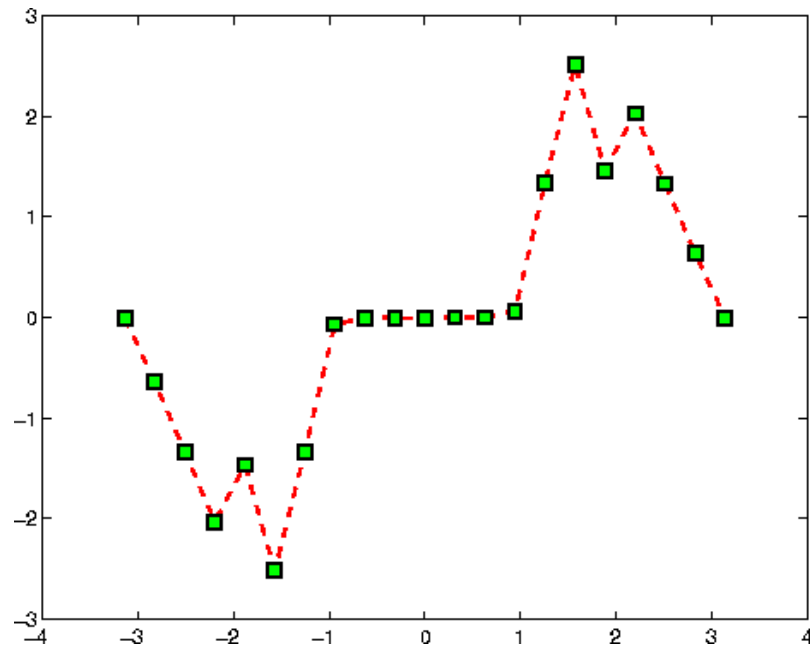
Specificarea culorii și dimensiunii liniilor

Caracteristicile liniilor se pot controla prin specificarea unor valori pentru proprietățile liniilor:

- *LineWidth* – specifică lățimea unei linii.
- *MarkerEdgeColor* – setează culoarea markerului sau culoarea marginilor markerului în cazul anumitor forme (cerc, pătrat etc.)
- *MarkerFaceColor* – setează culoarea interiorului markerelor.
- *MarkerSize* – specifică dimensiunea markerului.

Exemplu:

```
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
      'MarkerEdgeColor','k',...
      'MarkerFaceColor','g',...
      'MarkerSize',10)
```



Suprapunerea unor grafice peste un grafic existent

Se pot adăuga grafice peste unul existent cu comanda `hold`. Dacă se setează `hold on`, MATLAB-ul nu înlătură graficul existent, ci suprapune noul grafic în aceeași fereastră grafică.

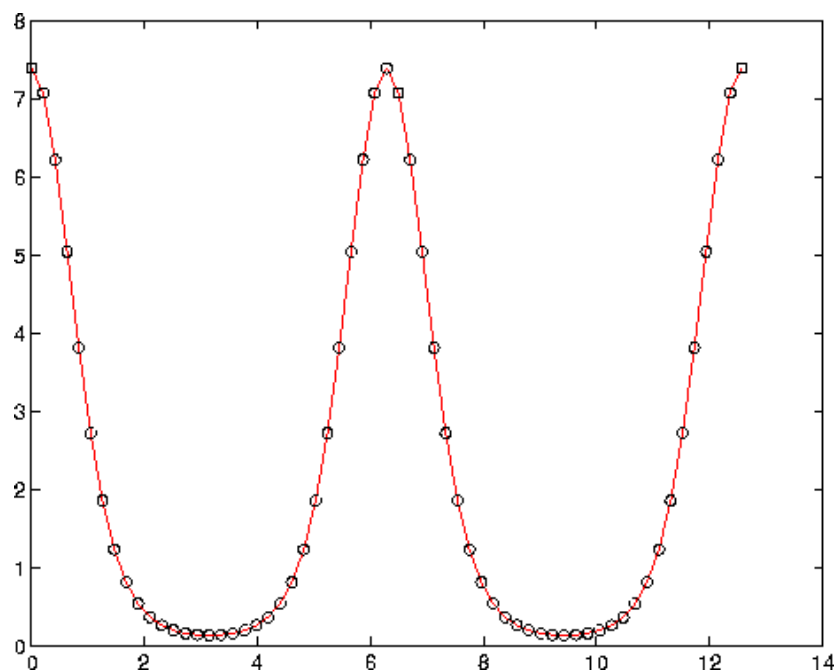
Exemplu:

```
semilogx(1:100, '+')
hold on
plot(1:3:300, 1:100, '--')
hold off
```

Plotarea simultană a markerelor și liniilor

Pentru plotarea markerelor (care indică punctele corespunzătoare datelor) și a liniilor (care unesc aceste date) se specifică atât tipul markerului cât și stilul liniei. Exemplu:

```
x = 0:pi/15:4*pi;
y = exp(2*cos(x));
plot(x, y, '-r', x, y, 'ok')
```



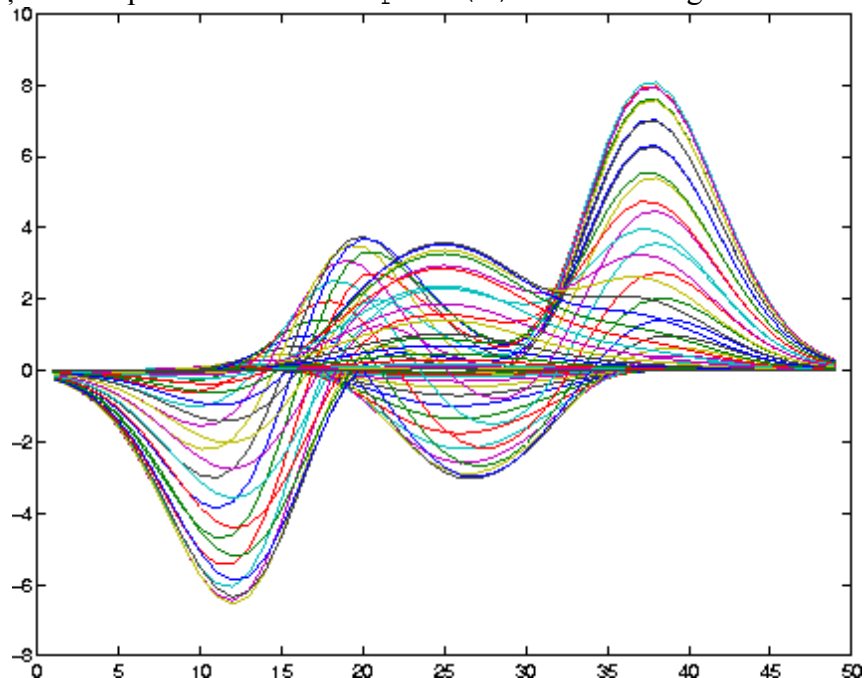
Plotarea datelor din matrici

Atunci când funcția `plot` este utilizată cu un singur argument de tip matrice:

```
plot(Y)
```

va fi realizat un grafic pentru fiecare coloană a matricii, cu axa x reprezentând indexul de linie $1:m$, cu m numărul liniilor din Y .

Exemplu: cu instrucțiunea `Z = peaks;` este creată o matrice 49×49 obținută printr-o evaluare de funcție. Dacă plotăm matricea cu `plot(Z)` vom avea un grafic cu 49 de linii.

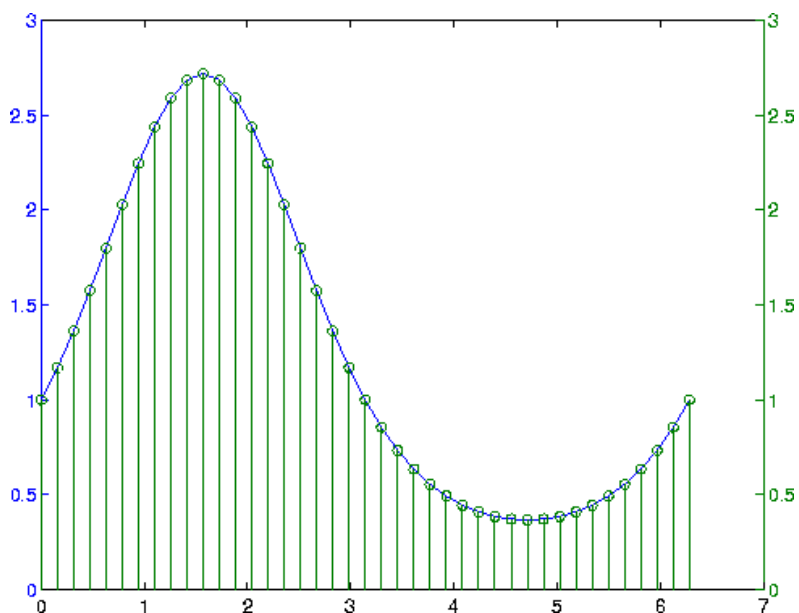


Plotarea cu axa Y dublă

Comanda `plotyy` permite crearea unor grafice pentru două seturi de date și cu reprezentare dublă a axei Y , pe partea stângă și pe partea dreaptă.

Exemplu:

```
t = 0:pi/20:2*pi;  
y = exp(sin(t));  
plotyy(t,y,t,y,'plot','stem')
```



Setarea parametrilor axelor

MATLAB-ul setează automat limitele axelor și gradarea acestora. Se pot însă folosi și setările utilizatorului, cu comenzile:

- `axis` – setează axele pentru fereastra grafică curentă.
- `axes` – creează axe noi cu caracteristici specificate.
- `get` și `set` – permit obținerea și setarea unor proprietăți ale axelor.
- `gca` – returnează identificatorul axelor curente.

Se pot parcurge în detaliu aceste comenzi prin apelarea la `help`.

Ferestre de tip figură

MATLAB-ul direcționează ieșirile grafice spre o fereastră distinctă de fereastra de comandă. Această fereastră grafică este denumită *figură* (*figure*). (a se vedea paragraful 3.3).

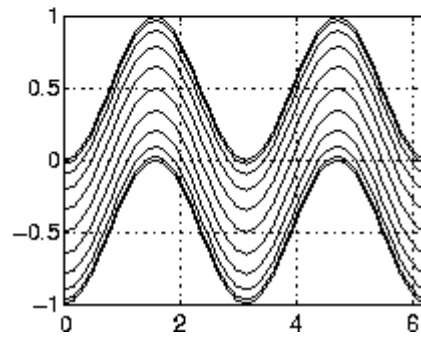
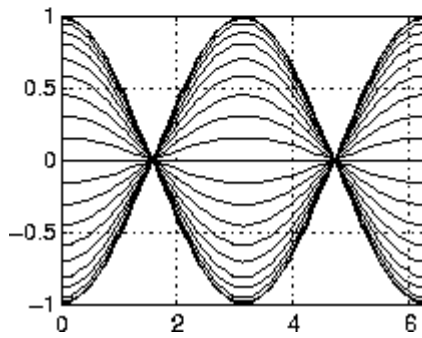
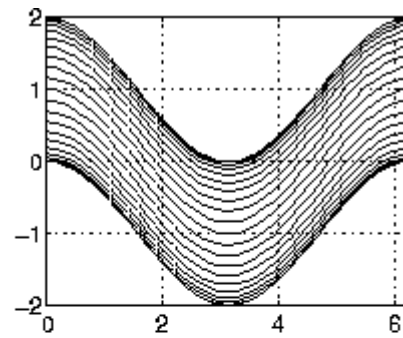
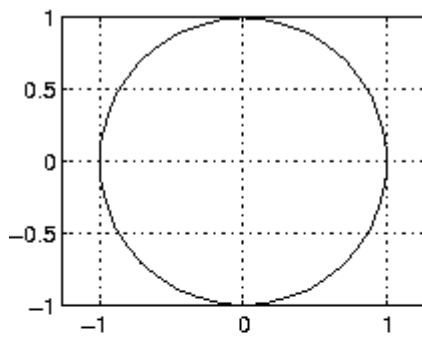
Funcția `figure` generează ferestre grafice. De exemplu,
`figure`
generează o nouă fereastră și o face fereastra curentă.

Afișarea unor grafice multiple în aceeași fereastră grafică

Se poate realiza o afișare a mai multor grafice în aceeași fereastră prin intermediul funcției `subplot`.

Funcția `subplot(m,n,i)` desparte fereastra de tip figură într-o matrice $m \times n$ de mici subploturi (subgrafice) și selectează subplotul i ca grafic curent. Exemplu:

```
t = 0:pi/20:2*pi;
[x,y] = meshgrid(t);
subplot(2,2,1)
plot(sin(t),cos(t))
axis equal
subplot(2,2,2)
z = sin(x)+cos(y);
plot(t,z)
axis([0 2*pi -2 2])
subplot(2,2,3)
z = sin(x).*cos(y);
plot(t,z)
axis([0 2*pi -1 1])
subplot(2,2,4)
z = (sin(x).^2)-(cos(y).^2);
plot(t,z)
axis([0 2*pi -1 1])
```



Comenzi de marcare, etichetare și gradare a graficelor

MATLAB-ul furnizează comenzi de etichetare a fiecărei axe și de plasare a unui text în orice loc din grafic. Comenzile sunt prezentate în tabelul următor.

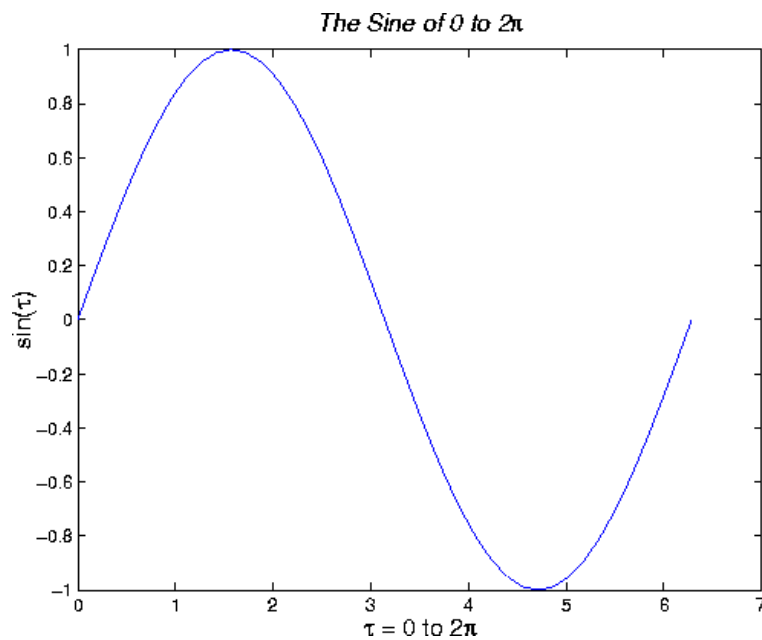
Comandă	Descriere
title	Adaugă un titlu
xlabel	Adaugă o etichetă pe axa x
ylabel	Adaugă o etichetă pe axa y
zlabel	Adaugă o etichetă pe axa z
legend	Adaugă o legendă
Text	Afișează un text la o locație specificată
Gtext	Plasează textul pe grafic utilizând mouse-ul

Etichetarea axelor

Se pot adăuga etichete pe axe cu comenzile xlabel, ylabel, zlabel.

Exemplu:

```
xlabel('t = 0 to 2\pi','FontSize',16)
ylabel('sin(t)','FontSize',16)
title('\it{Value of the Sine from Zero to Two
      Pi}','FontSize',16)
```



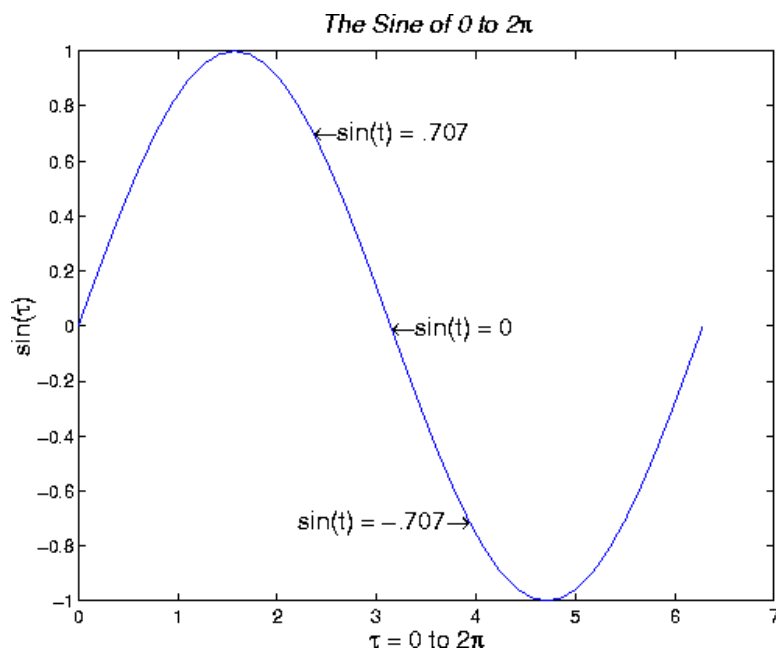
MATLAB-ul interpretează caracterele care urmează după backslash "\" ca și comenzi TeX. Aceste comenzi permit reprezentarea unor simboluri cum ar fi literele grecești sau săgețile.

Adăugarea textelor

Prin utilizarea funcției `text` se poate plasa un text (șir de caractere) oriunde pe grafic.

Exemplu:

```
text(3*pi/4, sin(3*pi/4), ...
    '\leftarrow sin(t) = .707', ...
    'FontSize', 16)
text(pi, sin(pi), '\leftarrow sin(t) = 0', ...
    'FontSize', 16)
text(5*pi/4, sin(5*pi/4), 'sin(t) = -.707 \rightarrow', ...
    'HorizontalAlignment', 'right', ...
    'FontSize', 16)
```



Plasarea textului în mod interactiv

Dacă utilizăm funcția `gtext` se poate plasa un text în mod interactiv, cu mouse-ul, oriunde pe grafic. Această funcție acceptă ca argument un șir de caractere și așteaptă până când utilizatorul selectează un loc pe grafic cu ajutorul mouse-ului.

Se poate utiliza și Plot Editor pentru plasarea textului (a se vedea paragraful 3.3).

8.1.2. Grafice 2 D specializate

MATLAB-ul permite lucrul cu o mare varietate de tipuri de grafice, astfel încât informațiile să poată fi prezentate eficient.

Tipul de grafic selectat depinde în mod esențial de natura datelor prelucrate.

- Graficele de tip bare sau arie (`bar`, `area`) sunt utile pentru vizualizarea unor rezultate, compararea lor și afișarea unei contribuții individuale din total.
- Graficele de tip statistic (`pie charts`) indică contribuțiile individuale dintr-un total.
- Histogramele (`histogram`) sunt utile pentru a indica distribuția valorilor datelor.
- Graficele de tip `stem` și `stairstep` sunt utile pentru date discrete.
- Graficele `compass`, `feather`, `quiver` sunt utile pentru plotarea vectorilor de tip direcție și viteză.
- Graficele de tip contur (`contour`) sunt utile la reprezentarea unor regiuni de valori egale ale datelor.
- Plotările interactive (`interactive`) permit selectarea unor puncte de plotare în mod interactiv.
- Graficele de tip animație (`animations`) adaugă date la grafice consecutive și creează o animație.

Grafice de tip Bar și Area

Funcție	Descriere
<code>Bar</code>	Afișează coloanele unor matrici $m \times n$ ca m grupe de n bare verticale
<code>Barh</code>	Afișează coloanele unor matrici $m \times n$ ca m grupe de n bare orizontale
<code>bar3</code>	Afișează coloanele unor matrici $m \times n$ ca m grupe de n bare verticale 3-D
<code>bar3h</code>	Afișează coloanele unor matrici $m \times n$ ca m grupe de n bare orizontale tridimensionale 3-D
<code>Area</code>	Afișează datele din vectori ca suprafețe

Grafice Bar grupate

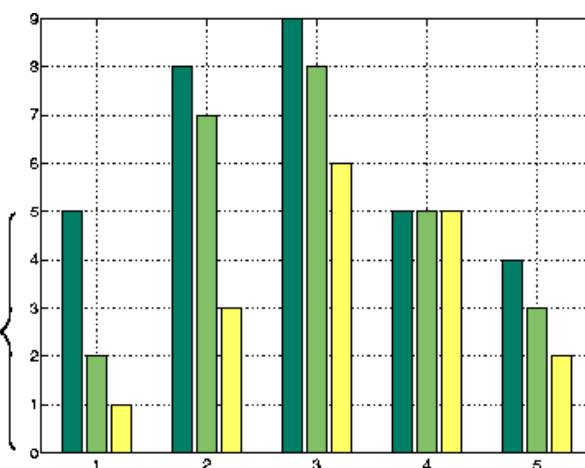
Ca setare implicită, un grafic cu bare reprezintă fiecare element dintr-o matrice cu o singură bară. În cazul unui grafic 2-D, barele create cu funcția `bar` sunt distribuite de-a lungul axei x , cu fiecare element dintr-o coloană desenat la altă locație. Toate elementele dintr-o linie sunt reprezentate grupat la aceeași locație pe axa x .

Exemplu:

```
Y = [5 2 1
      8 7 3
      9 8 6
      5 5 5
      4 3 2];
bar(Y)
```

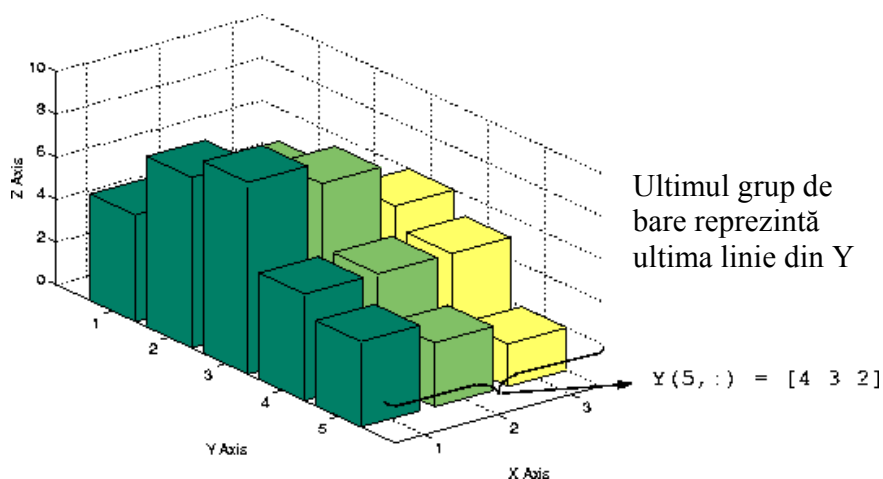
Grupele de coloane reprezintă linii din Y.

$Y(1, :) = [5 \ 2 \ 1]$



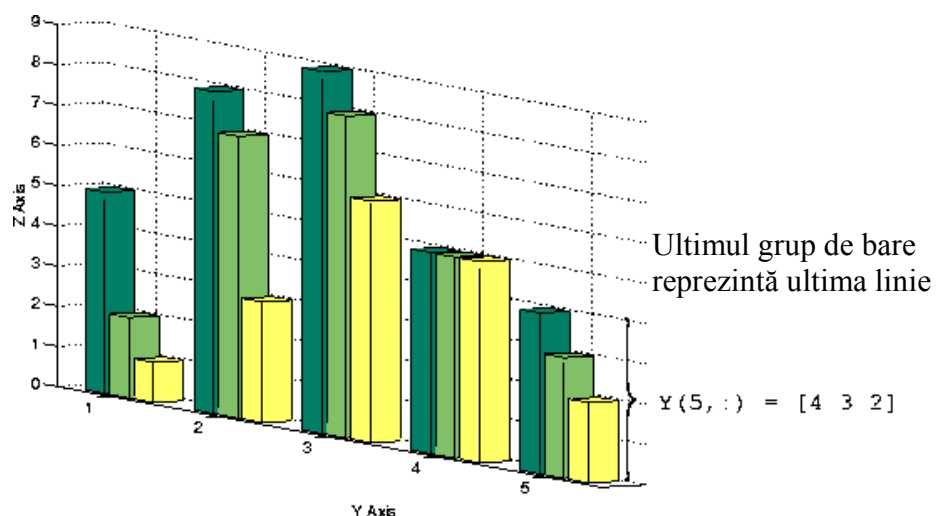
Grafice Bar 3-D separate

Funcția `bar3`, în cea mai simplă formă, trasează fiecare element ca un bloc separat de tip 3-D, cu elementele fiecărei coloane distribuite de-a lungul axei y . Barele care reprezintă elementele din prima coloană a unei matrice sunt centrate la 1 pe axa x ș.a.m.d. Barele care reprezintă elementele din ultima coloană sunt centrate la valoarea `size(Y, 2)` de pe axa x . Exemplu: `bar3(Y)`.



Grafice Bar 3-D grupate

Pentru a realiza un grafic de bare grupate 3 D se specifică argumentul `'group'` :
`bar3(Y, 'group')`



Grafice statistice - pie charts

Graficele `pie` afișează procentul cu care fiecare element al unui vector sau matrice contribuie la suma tuturor elementelor. Funcțiile `pie` și `pie3` creează grafice 2-D și 3-D.

În continuare prezentăm un exemplu de vizualizare a ponderii a trei produse din totalul vânzărilor. Se consideră o matrice `X`, pentru care fiecare coloană reprezintă vânzările anuale pentru câte un produs, pe o perioadă de înregistrări de 5 ani:

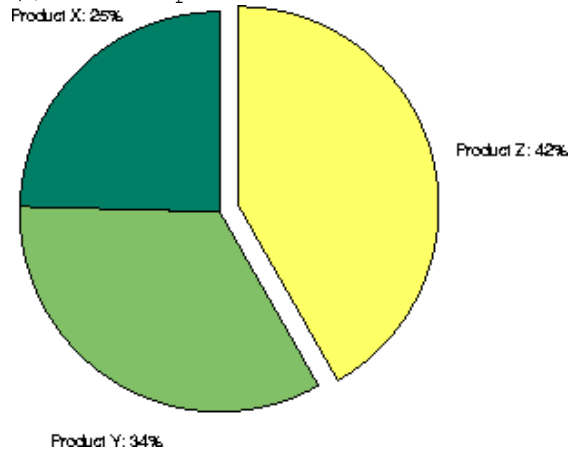
```
X = [19.3 22.1 51.6;  
     34.2 70.3 82.4;  
     61.4 82.9 90.8;  
     50.5 54.9 59.1;  
     29.4 36.3 47.0];
```

Se pot calcula vânzările pentru fiecare produs în cei 5 ani cu ajutorul funcției:

```
x = sum(X);
```

Dacă utilizăm argumentul de intrare `explode` putem reprezenta într-un mod explodat care dintre produse a avut o contribuție mai mare la vânzări (de exemplu). Programul are următoarea formă:

```
explode = zeros(size(x));  
[c,offset] = max(x);  
explode(offset) = 1;  
h = pie(x,explode); colormap summer
```



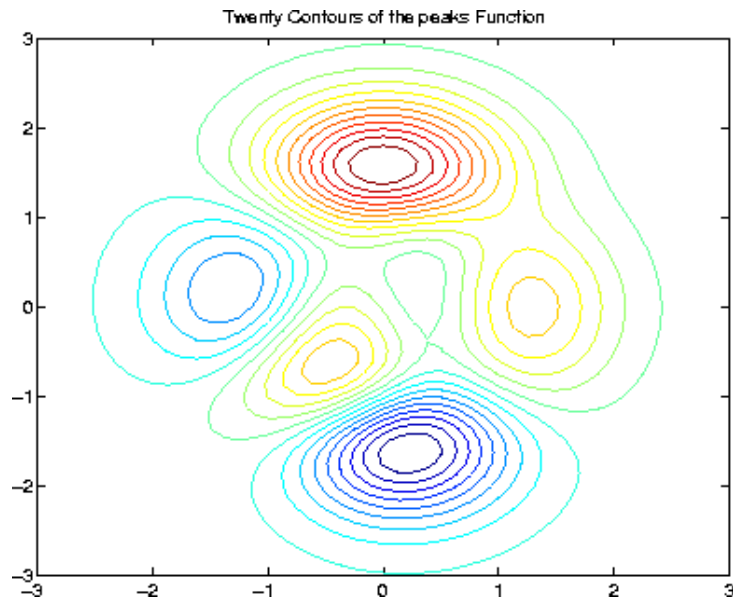
Crearea de grafice tip contur

Funcțiile `contour` și `contour3` afișează contururi 2-D și 3-D. Funcțiile cer un singur argument, și anume o matrice, ale cărei date sunt interpretate ca înălțimi față de un plan.

Pentru a seta numărul de niveluri de contur (implicit se realizează automat pe baza valorilor minime și maxime) se folosește un argument suplimentar opțional. De exemplu,

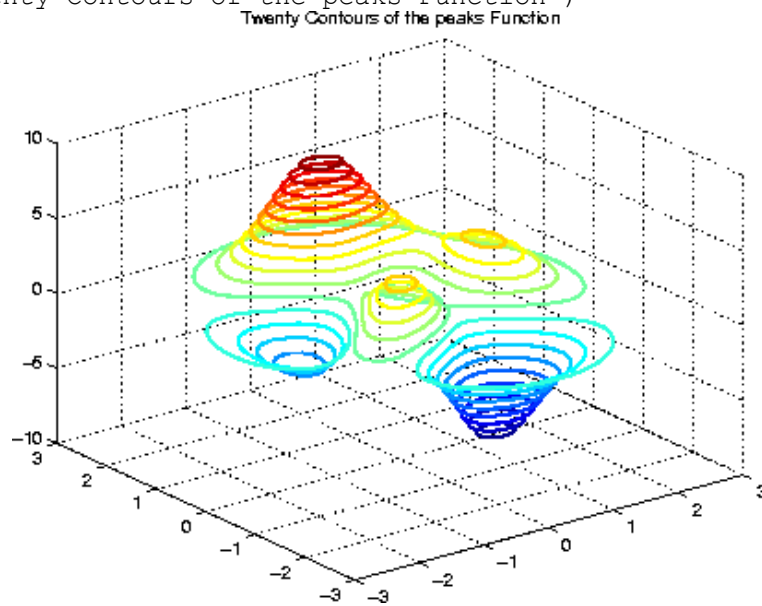
```
[X,Y,Z] = peaks;  
contour(X,Y,Z,20)
```

afișează 20 de contururi ale funcției `peaks` într-o vedere bidimensională.



Dacă dorim o reprezentare 3 D putem folosi comenzile:

```
[X,Y,Z] = peaks;
contour3(X,Y,Z,20)
h = findobj('Type','patch');
set(h,'LineWidth',2)
title('Twenty Contours of the peaks Function')
```



Animație

Se pot crea secvențe animate în MATLAB pe două căi:

- Salvarea unui număr de imagini și rularea lor ca pe un film.
- Ștergerea continuă și redesenarea unor obiecte pe ecran, făcând schimbări în mod incremental la fiecare redesenare.

Filme

Se parcurg trei etape:

- Se folosește `moviein` pentru inițializarea memoriei pentru o matrice suficient de mare.
- Se utilizează `getframe` pentru a genera fiecare cadru de film, care este returnat ca un vector coloană cu care se poate construi o matrice de tip film.

- Se folosește `movie` pentru rularea “filmului” de un număr specificat de ori cu o viteză specificată.

Ștergere și redesenare

Pot fi create diferite efecte prin selectarea unor moduri de ștergere. Pentru crearea unei animații sunt utile trei moduri de ștergere:

- `none` - MATLAB nu șterge obiectele.
- `background` - MATLAB șterge obiectul și îl redesenează în background. Acest mod șterge obiectul și tot ce este sub el (linii de grid etc.).
- `xor` – Acest mod șterge doar obiectul și este cel mai folosit la animație.

Pentru vizualizarea unor efecte de animație și construirea unor exemple proprii este indicată utilizarea facilității `demo` a MATLAB-ului.

8.1.3. Plotări tridimensionale (3 D)

Pașii tipici care trebuie parcurși pentru trasarea unor grafice tridimensionale sunt prezentați în continuare.

Etapa	Instrucțiuni
1. Pregătirea datelor	<code>Z = peaks(20);</code>
2. Selectarea ferestrei grafice și poziționarea graficului în fereastră	<code>figure(1)</code> <code>subplot(2,1,2)</code>
3. Apelarea unei funcții de plotare 3-D	<code>h = surf(Z);</code>
4a. Setarea unei hărți de culori și a unui algoritm de umbrire	<code>colormap hot</code> <code>shading interp</code> <code>set(h, 'EdgeColor', 'k')</code>
4b. Adăugarea unei iluminări	<code>light('Position', [-2, 2, 20])</code> <code>lighting phong</code> <code>material([0.4, 0.6, 0.5, 30])</code> <code>set(h, 'FaceColor', [0.7 0.7 0], ...</code> <code> 'BackFaceLighting', 'lit')</code>
5. Setarea unui punct de vizualizare	<code>view([30, 25])</code> <code>set(gca, 'CameraViewAngleMode', 'Manual')</code>
6. Setarea limitelor axelor și a marcajelor	<code>axis([5 15 5 15 -8 8])</code> <code>set(gca, 'ZTickLabel', 'Negative Positive')</code>
7. Setarea proporționalității	<code>set(gca, 'PlotBoxAspectRatio', ...</code> <code> [2.5 2.5 1])</code>
8. Completarea graficului cu etichete, legendă, text	<code>xlabel('X Axis')</code> <code>ylabel('Y Axis')</code> <code>zlabel('Function Value')</code> <code>title('Peaks')</code>
9. Operațiuni de tipărire	<code>set(gcf, 'PaperPositionMode', 'auto')</code> <code>print -dps2</code>

Reprezentarea unei matrice ca o suprafață

MATLAB-ul definește o suprafață prin coordonatele z ale punctelor de deasupra unui caroi aj dreptunghiular în planul x - y . Graficul este format prin unirea punctelor adiacente cu linii drepte. Plotările de suprafețe sunt utile pentru vizualizarea matricilor care sunt prea mari pentru a fi afișate în formă numerică și pentru trasarea graficelor funcțiilor de două variabile.

MATLAB-ul poate crea diferite forme de trasare a suprafețelor:

Funcție	Utilizare
<code>mesh, surf</code>	Trasare a unei suprafețe
<code>meshc, surfc</code>	Trasarea suprafeței, inclusiv conturul
<code>meshz</code>	Trasarea suprafeței, inclusiv planul de referință
<code>pcolor</code>	Plotare plană a suprafeței (valorile sunt proporționale doar cu culoarea)
<code>surfl</code>	Trasarea suprafeței luminată din direcția specificată
<code>surface</code>	Funcție de nivel scăzut pentru crearea unor obiecte tip grafice suprafață

Grafice realizate cu `mesh` și `surf`

Comenzile `mesh` și `surf` generează suprafețe 3-D din datele provenite de la matrici. Dacă Z este o matrice pentru elementele căreia $Z(i, j)$ se definește înălțimea unei suprafețe peste un caroi aj (i, j) atunci

```
mesh(Z)
```

generează o imagine colorată, caroiată a suprafeței și o afișează în vedere 3-D.

Similar,

```
surf(Z)
```

generează o imagine colorată, continuă a suprafeței și o afișează în vedere 3-D.

În cazul comenzii `mesh` se pot folosi comenzi de tipul `shading` pentru eliminarea liniilor de tip `mesh` (`shading flat`) sau pentru interpolarea umbririlor de-a lungul fațetelor suprafeței (`shading interp`).

Vizualizarea funcțiilor de două variabile

Primul pas care trebuie parcurs pentru trasarea graficului unei funcții de două variabile, $z = f(x, y)$, este de a genera matricile X și Y care definesc domeniul în care va fi vizualizată funcția. Apoi se utilizează aceste matrici pentru evaluare și trasarea graficului funcției.

Funcția `meshgrid` transformă domeniul specificat prin doi vectori, x și y , în matricile X și Y . Liniile matricei X sunt copii ale vectorului x și coloanele matricei Y sunt copii ale vectorului y .

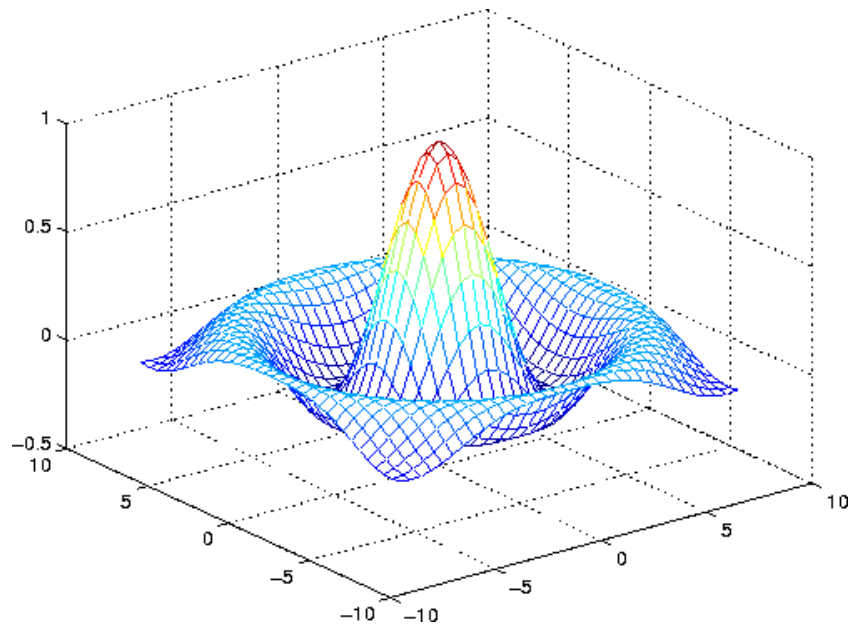
Pentru a vedea cum se folosește `meshgrid`, vom considera funcția $\sin(r)/r$ (numită funcția *sinc*). Pentru a evalua funcția între -8 și 8 și pentru x și pentru y , este necesar doar un argument de tip vector pentru `meshgrid`, care va fi utilizat în ambele direcții:

```
[X,Y] = meshgrid(-8:.5:8);  
R = sqrt(X.^2 + Y.^2) + eps;
```

Matricea R conține distanțele de la centru (originea), iar `eps` este adăugat pentru a evita împărțirea la zero.

Acum se poate forma funcția *sinc* și se poate realiza plotarea cu `mesh`.

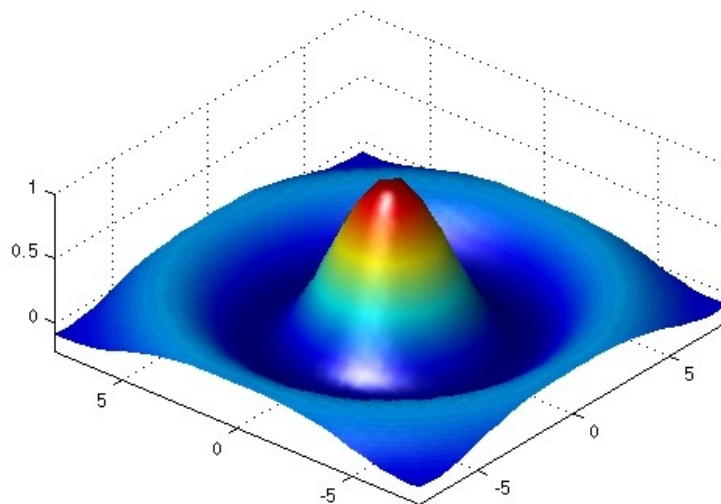
```
Z = sin(R) ./ R;
mesh(Z)
```



Se poate realiza o îmbunătățire a reprezentării grafice în condițiile utilizării aceluiași date, prin folosirea unor facilități de iluminare și ajustare a imaginii (`daspect`, `axis`, `camlight`, `view`).

Exemplu:

```
surf(X,Y,Z,'FaceColor','interp','EdgeColor','none',...
      'FaceLighting','phong')
daspect([5 5 1]);axis tight;view(-50,30);camlight left
```

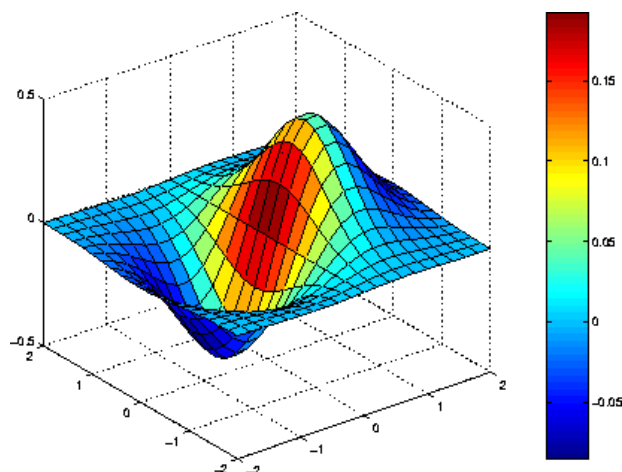


Harta culorilor

Fiecare fereastră grafică MATLAB are asociată o hartă a culorilor (`colormap`), care este o matrice cu trei coloane a căror lungime este egală cu numărul de culori definite. Fiecare linie a matricii definește o culoare particulară prin specificarea a trei valori în domeniul 0 – 1. Aceste valori definesc componentele RGB (red, green, blue) (adică intensitățile componentelor video roșu, verde și albastru). Funcția `colormap` fără argumente returnează harta figurii curente.

Funcția `colorbar` afișează în fereastra grafică harta curentă a culorilor, sub forma unei bare așezate lângă grafic. Exemplu:

```
[x,y] = meshgrid([-2:.2:2]);Z = x.*exp(-x.^2-y.^2);
surf(x,y,Z,gradient(Z));colorbar
```



8.2. *Handle Graphics* și Interfețe Grafice în MATLAB (GUI)

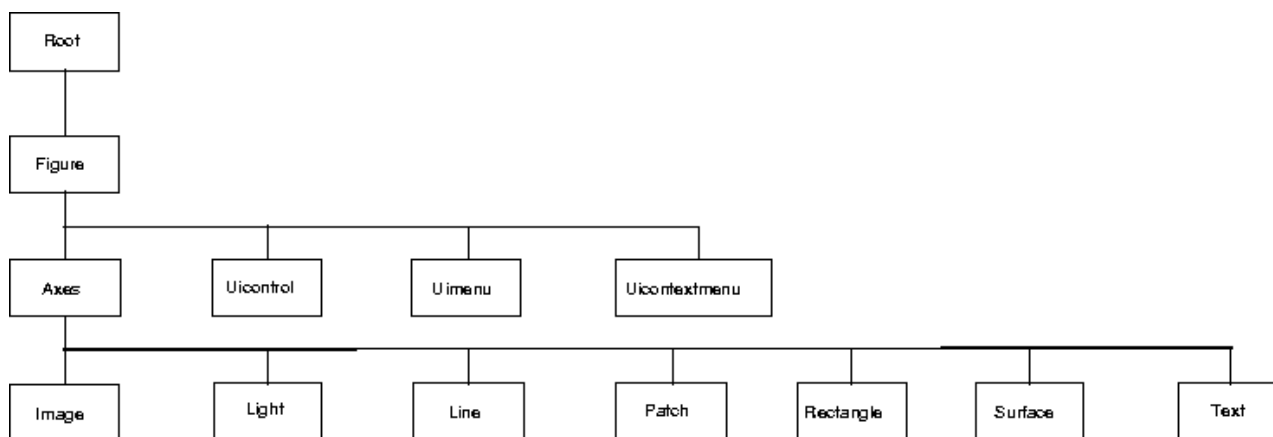
□ Crearea și manipularea graficelor în MATLAB se realizează cu ajutorul unui sistem de grafică orientată pe obiecte denumit ***Handle Graphics***. Acest sistem furnizează componentele necesare generării unor grafice: comenzi de trasare a liniilor, textelor, grafice 3-D, poligoane etc. precum și instrumente interactive de tipul meniurilor, butoanelor, ferestre de dialog etc.

□ Cu ***Handle Graphics*** se pot manipula direct elementele grafice (așa cum o fac funcțiile MATLAB de nivel înalt descrise în paragraful anterior) pe două căi: fie de la linia de comandă MATLAB fie cu ajutorul unor fișiere MATLAB create special.

8.2.1. Graficele privite ca obiecte. Ierarhia obiectelor

Obiectele grafice sunt de fapt elementele grafice de bază utilizate de MATLAB pentru afișarea datelor și pentru crearea Interfețelor Grafice Utilizator (Graphical User Interfaces - GUI). Fiecare stare a unui obiect este asociată unui identificator unic numit *handle*, care poate fi folosit pentru manipularea caracteristicilor obiectului respectiv (caracteristici care sunt numite ***proprietățile obiectului***).

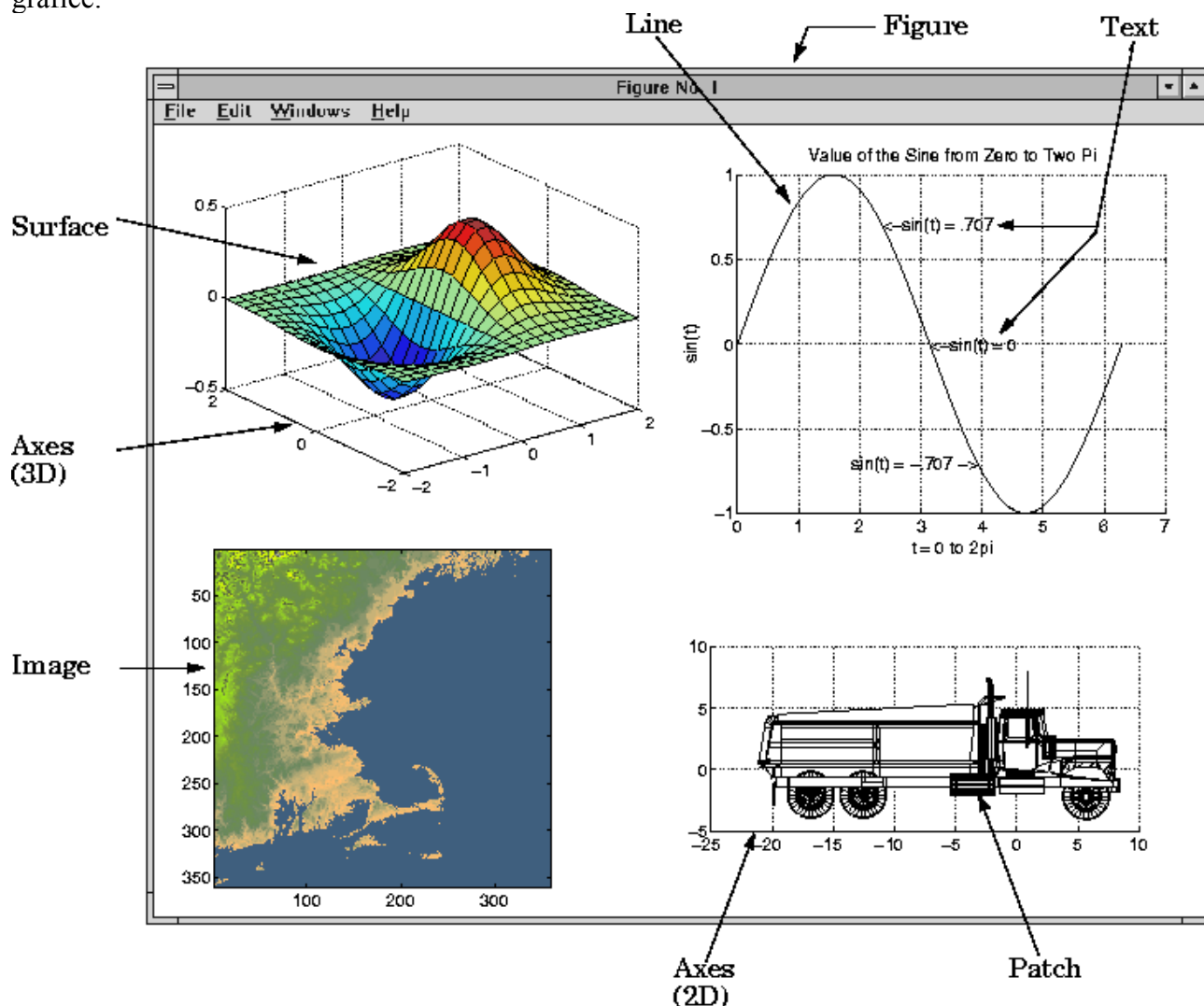
Obiectele grafice sunt structurate într-o ierarhie pe trei nivele:



Ierarhia este bazată pe interdependențele dintre diferitele obiecte grafice. De exemplu, pentru trasarea unui obiect linie, MATLAB utilizează un obiect de tip axe pentru orientarea și furnizarea unui sistem de referință liniei. Obiectul de tip axe are nevoie la rândul său de o fereastră grafică pentru afișarea liniei.

Obiectele grafice sunt interdependente și prin urmare un ecran grafic conține o mare varietate de obiecte care împreună furnizează o imagine sau un grafic care are o semnificație clară. Pentru

exemplificare se poate analiza următoarea fereastră grafică, fereastră care conține mai multe obiecte grafice.



Fiecare tip de obiect grafic are o funcție generatoare corespunzătoare, funcție care este utilizată pentru crearea unui obiect din clasa respectivă de obiecte. Funcțiile de generare a obiectelor au aceleași nume ca și obiectele pe care le creează (funcția `text` pentru obiecte de tip text, funcția `figure` pentru obiecte de tip figură etc.).

Tipurile de obiecte grafice sunt descrise pe scurt în continuare.

Rădăcina (Root)

În fruntea ierarhiei este obiectul rădăcină, care corespunde cu ecranul calculatorului. Acest obiect nu trebuie creat, el există, este unic și toate celelalte obiecte sunt descendenții acestuia. Se pot modifica anumite proprietăți ale obiectului rădăcină.

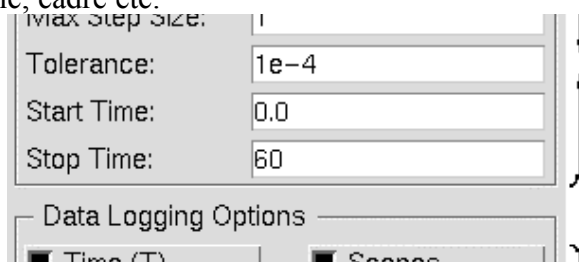
Obiectele figură (Figure)

Obiectele de tip figură sunt ferestre individuale pe ecranul rădăcină pe care MATLAB-ul afișează graficele. Nu există limită pentru numărul de ferestre grafice (decât cele datorate limitelor calculatorului). Toate obiectele figură sunt “copii” ai rădăcinii și celelalte obiecte grafice sunt descendenți ai figurilor.

Obiectele de tip Uicontrol

Obiectele `Uicontrol` sunt elemente de control ale interfeței utilizator care execută subrutine de apel atunci când utilizatorul activează un obiect. Există mai multe stiluri de control cum ar fi butoane, liste etc. Fiecare astfel de instrument este proiectat să accepte un anumit tip de informație de la utilizator. De exemplu, listele sunt de obicei folosite pentru furnizarea unei liste de nume, din care utilizatorul poate selecta unul sau mai multe articole.

Obiectele Uicontrol pot fi utilizate în diferite combinații pentru construirea unor ecrane de control și a unor ferestre de dialog. În exemplul următor sunt prezentate astfel de combinații: meniuri “pop-up”, ferestre de tip text editabile, ferestre de verificare (check boxes), butoane, text static, cadre etc.



Meniurile pop-up permit alegerea dintre mai multe articole predefinite

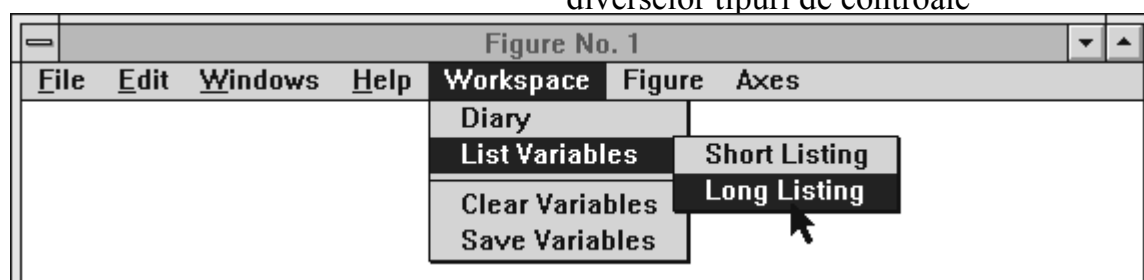
Utilizatorul poate introduce valori numerice în astfel de ferestre de tip text editabile

Obiectele Uicontrol sunt copii ale obiectelor de tip figură și deci sunt independente de tip axe.

Obiectele de tip Uimenu

Obiectele Uimenu sunt meniuri “pull-down” care execută rutine de apelare atunci când utilizatorul selectează un articol individual dintr-un meniu. MATLAB-ul plasează obiectele Uimenu pe bara de meniuri a ferestrei grafice, la dreapta meniului de comandă.

Imaginea următoare arată partea de sus a unei ferestre MATLAB. Se observă că pe bara de meniuri sunt definite trei meniuri de tip Uimenu (intitulate Workspace, Figure, și Axes). Două niveluri de submeniuri sunt vizibile în meniul Workspace.



Obiectele Uimenu sunt descendente directe ale obiectelor de tip figură și deci sunt independente de axe.

Obiectele de tip axe (Axes)

Obiectele de tip axe definesc o regiune într-o fereastră de tip figură și orientează descendenții lor spre această regiune. Obiectele de tip axe sunt copii ale obiectelor de tip figură și sunt părinții obiectelor de tip imagine, luminozitate, linie, patch, suprafață și text.

Toate funcțiile care trasează grafice (plot, surf, mesh, bar etc.) creează un obiect de tip axe dacă nu există deja unul.

Uicontrol și Uimenu nu sunt descendente ale obiectelor de tip axe.

Obiectele de tip Imagine (Image)

O imagine în MATLAB constă într-o matrice de date și o hartă a culorilor. Există trei tipuri de bază de imagini care diferă în funcție de modul în care elementele matricii de date sunt interpretate ca pixeli color – indexați, intensitate și “truecolor”.

Obiectele de tip lumină (luminozitate) (Light)

Aceste obiecte definesc surse de lumină care influențează toate obiectele de tip patch și suprafață dintre axe. Se pot seta proprietățile care determină tipul sursei de lumină, culoarea, localizarea și altele.

Obiectele de tip linie (Line)

Obiectele de tip linie sunt elemente grafice de bază care sunt folosite pentru a genera cele mai multe plotări 2-D și unele 3-D. Funcțiile de nivel înalt plot, plot3, loglog etc. generează

obiecte de tip linie. Sistemul de coordonate al obiectului părinte – obiectul de tip axe – poziționează și orientează linia.

Obiectele de tip Patch

Aceste obiecte sunt contururi poligonale cu muchii (laturi), umplute. Un singur obiect patch poate conține mai multe fețe, fiecare colorată independent. Funcțiile `fill`, `fill3`, `contour3` creează obiecte patch. Ca și în cazul liniei, sistemul de coordonate al obiectului părinte (axele) poziționează și orientează obiectul patch.

Obiectele de tip dreptunghi (Rectangle)

Aceste obiecte sunt arii 2-D umplute, cu o formă care poate varia de la un dreptunghi la o elipsă.

Obiectele de tip suprafață (Surface)

Obiectele de tip Surface sunt reprezentări 3-D ale matricilor de date create prin plotarea fiecărui element al matricii ca o înălțime deasupra planului x - y . MATLAB-ul poate trasa suprafețe pline, colorate sau doar o rețea de linii (mesh) care conectează punctele respective. Sistemul de coordonate al axelor poziționează și orientează obiectul de tip suprafață. Funcțiile de nivel înalt `pcolor`, `surf`, `mesh` generează obiecte de tip suprafață.

Obiectele de tip Text

Obiectele de tip text sunt de fapt șiruri de caractere. Sistemul de coordonate al axelor poziționează textul. Funcțiile de nivel înalt `title`, `xlabel`, `ylabel`, `zlabel`, `gtext` generează obiecte de tip text.

8.2.2. Proprietățile obiectelor grafice

◆ Proprietățile obiectelor grafice determină aspectul și comportamentul acestora. Proprietățile includ informații generale (tipul obiectului, părinte, copii, dacă obiectul este vizibil etc.) și informații specifice unei anumite clase particulare de obiecte.

◆ MATLAB-ul organizează informațiile într-o ierarhie și salvează aceste informații în proprietăți ale obiectelor. De exemplu, proprietățile rădăcinii conțin identificatorul (handle) figurii curente și locația curentă a pointerului (cursorului), proprietățile figurii conțin liste cu descendenții și evenimentele din fereastră, proprietățile axelor conțin informații despre cum fiecare din obiectele copil folosește harta culorilor etc.

◆ Valoarea curentă a oricărei proprietăți poate fi aflată, iar unele valori pot fi modificate. Valoarea unei proprietăți este aplicată numai unui obiect particular și nu întregii clase de obiecte. Se pot seta valori implicite care să fie valabile pentru toate obiectele create ulterior.

◆ Anumite proprietăți sunt comune tuturor obiectelor grafice:

Proprietate	Informații conținute
<code>BusyAction</code>	Controlează modul în care MATLAB-ul apelează rutinele de întreruperi definite pentru un anumit obiect.
<code>ButtonDownFcn</code>	Rutină executată la apăsarea unui buton.
<code>Children</code>	Manipulează toate obiectele copil ale obiectului.
<code>Clipping</code>	Activare/dezactivare mod tăiere.
<code>CreateFcn</code>	Rutină executată atunci când acest tip de obiect este creat.
<code>DeleteFcn</code>	Rutină executată atunci când se dă o comandă de distrugere (ștergere) a obiectului.
<code>HandleVisibility</code>	Permite controlul obiectului de la linia de comandă sau din rutine de apelare.
<code>Interruptible</code>	Determină când o rutină poate fi întreruptă printr-o rutină invocată ulterior.
<code>Parent</code>	Părintele obiectului.

Selected	Indică dacă obiectul este selectat.
SelectionHighlight	Specifică dacă este indicată starea de selectare.
Tag	Etichetă a unui obiect specificată de utilizator.
Type	Tipul obiectului (figură, linie, text etc.)
UserData	Orice dată care se dorește a fi asociată obiectului.
Visible	Determină dacă obiectul este vizibil sau nu.

8.2.3. Funcții de generare a obiectelor grafice

Fiecare obiect grafic, mai puțin rădăcina, are o funcție de generare corespondentă:

Funcție	Descriere obiect
axes	Sistem de coordonate carteziene care scalează și orientează obiectele copil: imagine, lumină, linie, patch, suprafață și text.
figure	Fereastră pentru afișare grafică.
image	Imagine 2-D definită prin indicarea hărții culorilor sau valori RGB. Datele pot fi pe 8 biți sau dublă precizie.
light	Sursă direcționată de lumină, localizată între axe, care influențează suprafețele și obiectele patch.
line	Linie formată prin conectarea coordonatelor prin segmente drepte într-o secvență specificată.
patch	Formă poligonală creată prin interpretarea fiecărei coloane din matricile de coordonate ca un poligon separat.
rectangle	Arie 2-D umplută (plină), cu formă de la dreptunghi la elipsă.
surface	Suprafață cu fețe dreptunghiulare, definite prin interpretarea elementelor matricei ca înălțimi deasupra planului.
text	Șir de caractere localizat în sistemul de coordonate al axelor.
uicontextmenu	Meniu context ce poate fi asociat cu alt obiect grafic.
uicontrol	Interfață utilizator programabilă (butoane, liste etc.).
uimenu	Meniu programabil care apare în partea superioară a figurii.

Toate funcțiile de generare a obiectelor au un format similar

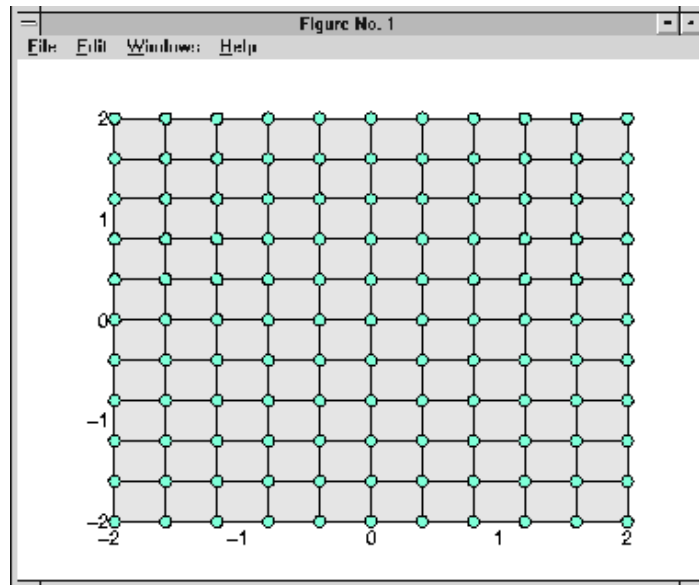
:

```
handle=function('propertyname',propertyvalue,...)
```

Exemplu de generare a obiectelor grafice

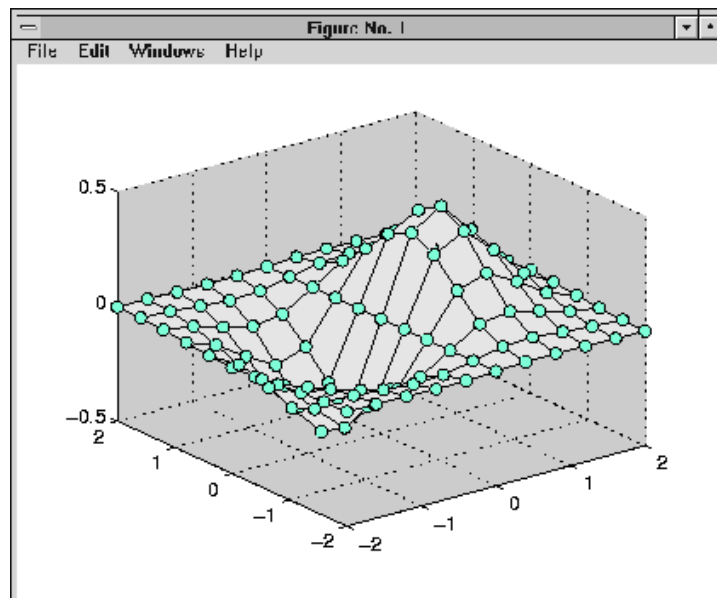
În exemplul următor este evaluată o funcție matematică și sunt create trei obiecte grafice folosind valorile proprietăților specificate ca argumente ale comenzilor `figure`, `axes` și `surface`, celelalte proprietăți având valori implicite.

```
[x,y] = meshgrid([-2:.4:2]);
Z = x.*exp(-x.^2-y.^2);
fh=figure('Position',[350 275 400 300],'Color','w');
ah=axes('Color',[.8 .8 .8],'XTick',[-2 -1 0 1 2],...
        'YTick',[-2 -1 0 1 2]);
sh = surface('XData',x,'YData',y,'ZData',Z,...
            'FaceColor',get(ah,'Color')+.1,...
            'EdgeColor','k','Marker','o',...
            'MarkerFaceColor',[.5 1 .85]);
```



Funcția `surface` nu folosește o vedere 3-D ca funcția de nivel înalt `surf`. Se poate schimba vederea într-una 3-D cu camera commands sau cu comanda `view`:

```
view(3)
```



9. PREZENTAREA TOOLBOX-URILOR MATLAB

MATLAB permite dezvoltarea unei familii de aplicații sub forma *toolbox*-urilor. Aceste *toolbox-uri* permit învățarea și aplicarea tehnologiilor specializate din diverse domenii. Sunt disponibile *toolbox-uri* pentru domenii cum ar fi: procesarea numerică a semnalelor, sisteme de conducere automată, rețele neurale, logică fuzzy, wavelet, simulare (**SIMULINK**), identificare, statistică, crearea de hărți, procesarea imaginilor, etc. În continuare sunt prezentate pe scurt facilitățile oferite de câteva din aceste *toolbox-uri*.

9.1. Toolbox-ul Comunicații (Communications Toolbox)

Communications Toolbox este o colecție de funcții de calcul și blocuri de simulare pentru cercetare, dezvoltare, proiectarea de sisteme și simulare în domeniul comunicațiilor. Toolboxul este proiectat atât pentru experți cât și pentru începători în domeniu, și se bazează pe MATLAB și Simulink.

Funcții disponibile:

- Surse de date
- Surse codare/decodare
- Controlul erorilor de codificare

- Modulare/demodulare
- Filtre transmisie/recepție
- Canale de transmisie
- Acces multiplu
- Sincronizare
- Utilitare

9.2. Toolbox-ul pentru Sisteme de Conducere Automată (Control System)

MATLAB-ul dispune de o colecție bogată de funcții utile atât inginerului automatist practician cât și teoreticianului din domeniul teoriei sistemelor. Printre facilitățile oferite enumerăm: aritmetică complexă, valori proprii, găsirea rădăcinilor, inversări de matrici, Transformarea Fourier Rapidă etc.

Toolbox-ul Control System folosește structurile matriceale ale MATLAB și reprezintă fundația MATLAB. Toolbox-ul este o colecție de algoritmi (în principal sub forma fișierelor .m), care implementează proiectarea sistemelor de conducere, precum și tehnici de analiză și modelare.

Sistemele pot fi modelate în MATLAB ca funcții de transfer, sub forma poli-zero-uri sau prin reprezentarea de stare, ceea ce permite aplicarea tehnicilor clasice și a celor moderne. Se poate lucra cu sisteme continue și cu sisteme discrete și pot fi efectuate conversii între diversele tipuri de modele. Toolbox-ul permite calculul și trasarea răspunsurilor în domeniul timp și domeniul frecvență, precum și a locului rădăcinilor. De asemenea, se pot realiza alocări de poli, se pot implementa conducerea optimală și estimatoare etc.

9.3. Toolbox-ul pentru Baze de Date (Database Toolbox)

Toolbox-ul Baze de Date permite importul și exportul de date între MATLAB și cele mai răspândite programe de baze de date. Cu acest toolbox se pot importa date din exterior, se utilizează capacitățile mari de calcul și prelucrare analitică ale MATLAB, și se exportă rezultatele înapoi în baza de date sau în altă bază de date.

Realizarea acestor operațiuni se face astfel: toolbox-ul Database conectează MATLAB la o bază de date utilizând funcțiile MATLAB; datele sunt preluate de la baza de date ca și caractere, sunt transformate în tipuri de date corespunzătoare și sunt stocate în tablouri de tip celulă. În acest moment se pot folosi instrumentele MATLAB de lucru cu date. Se pot include funcțiile toolbox-ului în fișiere M-files. Pentru exportul datelor se utilizează în final funcțiile specializate MATLAB.

Toolbox-ul Database este furnizat împreună cu interfața grafică Visual Query Builder (VQB).

9.4. Toolbox-ul de Procesare a Semnalelor (Signal Processing Toolbox)

Toolbox-ul de Procesare a Semnalelor este o colecție de instrumente construită în mediul de calcul numeric MATLAB. Toolbox-ul permite o mare varietate de operații de prelucrare a semnalelor, de la forme de undă la proiectarea filtrelor, modelare parametrică și la analiza spectrală. Toolbox-ul furnizează două categorii de instrumente:

- Funcții de prelucrare a semnalelor de la linia de comandă
- Intefete grafice utilizator pentru:
 - Proiectarea interactivă a filtrelor
 - Trasarea și analiza semnalelor
 - Analiză spectrală
 - Aplicarea de filtre semnalelor
 - Analiza filtrelor

9.5. Toolbox-ul DSP Blockset

Setul de blocuri de procesare numerică a semnalelor (DSP Blockset) este o colecție de biblioteci de blocuri care se utilizează cu pachetul Simulink.

Bibliotecile DSP Blockset sunt proiectate special pentru prelucrarea numerică a semnalelor și includ facilități cum ar fi filtrarea clasică, adaptivă, manipulări de matrici, algebră liniară, statistică, etc.

DSP Blockset extinde mediul Simulink prin furnizarea de componente și algoritmi pentru sistemele DSP. Facilități:

- Operațiuni bazate pe cadre
- Suport matriceal

- Filtrare adaptivă și cu eșantionare multiplă
- Operațiuni statistice
- Algebră liniară
- Estimarea parametrilor
- Facilități de timp real

9.6. Toolbox-ul Finanțe (Financial Toolbox)

MATLAB-ul împreună cu toolbox-ul de Finanțe furnizează un mediu de calcul integrat și complet pentru analiză și inginerie financiară. Toolbox-ul dispune de instrumente de analiză matematică și statistică a datelor financiare și instrumente de prezentare grafică a rezultatelor obținute.

Facilități:

- Calcul și analiză de preț și de producție
- Realizează analize venituri, prețuri etc. compatibile SIA (Securities Industry Association)
- Analiza și managementul portofoliilor
- Proiectarea și evaluarea de strategii financiare
- Identificarea, măsurarea și controlul riscului
- Analiza și calculul fluxului de cash
- Analiza și predicția activității economice
- Crearea de instrumente financiare structurate
- Cercetare academică

9.7. Toolbox-ul de Procesare a Imaginilor (Image Processing Toolbox)

Toolbox-ul Image Processing este o colecție de funcții care extind posibilitățile MATLAB din domeniul prelucrării imaginilor. Toolbox-ul dispune de o mare varietate de operațiuni de prelucrare a imaginilor, cum ar fi:

- Operații geometrice
- Operații de tip vecinătate
- Filtrare liniară și proiectarea filtrelor
- Transformate
- Analiza și îmbunătățirea imaginilor
- Operații binare
- Operații asupra regiunii de interes

Multe dintre funcțiile toolbox-ului sunt fișiere M-files care constau în instrucțiuni MATLAB care implementează algoritmi specializați de prelucrare a imaginilor. Aceste instrucțiuni pot fi vizualizate cu comanda cunoscută:

```
type function_name
```

Posibilitățile toolbox-ului pot fi extinse prin crearea de fișiere proprii prin utilizarea funcțiilor disponibile în combinație cu alte toolbox-uri cum ar fi Signal Processing Toolbox și Wavelet Toolbox.

9.8. Toolbox-ul Optimizare (Optimization Toolbox)

Acest toolbox este o colecție de funcții care includ rutine pentru o mare diversitate de optimizări:

- Minimizare neliniară fără constrângeri
- Minimizare neliniară cu constrângeri, inclusiv probleme de tip minimax și probleme de minimizare semi-infinită
- Programare liniară și pătratică
- Algoritmi neliniari de tipul celor mai mici pătrate
- Rezolvarea de sisteme de ecuații neliniare

Sunt disponibili și algoritmi specializați pentru sisteme mari (large-scale problems). Funcțiile toolbox-ului pot fi utilizate în combinație cu alte toolbox-uri sau cu Simulink.

9.9. Toolbox-ul pentru Sisteme de Putere (Power System Blockset)

Sistemele electrice de putere sunt combinații de circuite electrice și de aparate electro-mecanice cum ar fi motoarele și generatoarele. Inginerii care lucrează în acest domeniu trebuie să

îmbunătățească performanțele sistemelor de putere. Cerințele de creștere drastică a eficienței au determinat proiectanții să folosească aparatură electronică și sisteme sofisticate de conducere care necesită instrumente de analiză și proiectare corespunzătoare, fiind absolut necesară înțelegerea fenomenelor (neliniare) prin simulare.

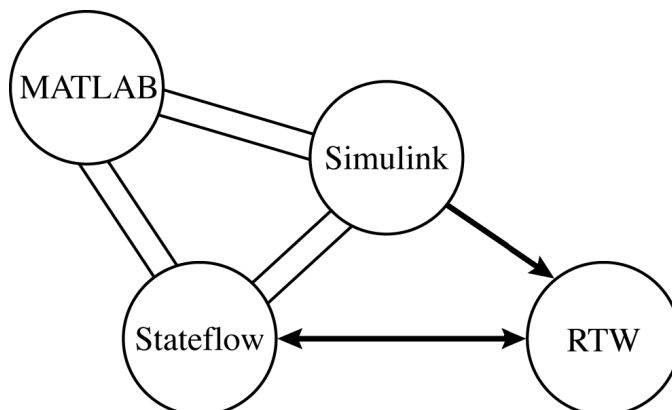
Power System Blockset a fost proiectat pentru furnizarea unor instrumente de proiectare care permit inginerilor să construiască rapid și ușor modele care simulează sistemele de putere. Blocurile utilizează mediul Simulink și permit construirea unui model cu proceduri simple de tip *click and drag*. Se poate trasa rapid topologia circuitelor electrice și de asemenea se poate face o analiză a circuitelor care include interacțiunile cu sistemele mecanice, termice, de control.

Bibliotecile conțin modele ale aparaturii tipice pentru sistemele de putere, cum ar fi transformatoare, linii electrice, motoare, electronică de putere etc. Posibilitățile toolbox-ului pot fi testate prin rularea fișierelor demonstrative.

9.10. Toolbox-ul Stateflow (Diagrame de stare)

Stateflow este un produs multiplatformă, care poate rula pe sisteme Microsoft Windows 95, Windows NT și UNIX, și care necesită MATLAB + Simulink.

Stateflow este un instrument grafic de proiectare și dezvoltare pentru sisteme de conducere complexe și pentru sisteme logice de supervizare. Cu ajutorul acestui toolbox se pot genera modele sub forma unor diagrame de stare dinamice ale unui sistem. Generarea de cod pentru elementele Simulink ale unui model Stateflow necesită pachetul suplimentar al Simulink-ului Real-Time Workshop.



9.11. Toolbox-ul de Statistică (Statistics Toolbox)

Toolbox-ul de Statistică dispune de instrumente care permit executarea de sarcini statistice uzuale, de la generarea de numere aleatoare până la proiectarea de experimente statistice și controlul proceselor statistice.

Există două categorii de instrumente:

- Construcția de funcții statistice și probabilistice
- Instrumente grafice interactive

Prima categorie constă în funcții care pot fi apelate din linia de comandă sau din aplicații proprii.

A doua categorie constă în instrumente interactive care permit accesul la funcții prin intermediul unei interfețe grafice utilizator (GUI), care furnizează un mediu adecvat pentru funcții de predicție, interpolare, probabilistice etc.

9.12. Toolbox-ul pentru Calcul Simbolic (Symbolic Math Toolbox)

Acest toolbox completează facilitățile grafice și numerice ale MATLAB-ului cu diverse alte tipuri de facilități matematice, care permit efectuarea de calcule simbolice.

Mașina de calcul utilizată este nucleul de la Maple. Există de fapt două toolbox-uri: unul de bază, care este o colecție de peste o sută de funcții MATLAB care permit accesul la Maple, și un toolbox extins care permite accesul extins la multiple pachete Maple (non-grafice, facilități de programare, proceduri utilizator etc.)

Facilități:

- Calcul matematic: Diferențieri, integrări, limite, sume, serii Taylor
- Algebră liniară: Inverse, determinanți, valori proprii, forme canonice ale matricilor
- Simplificări: Metode de simplificare a expresiilor algebrice
- Soluționarea ecuațiilor: Soluții simbolice și numerice ale ecuațiilor algebrice și diferențiale
- Aritmetică cu precizie variabilă: Evaluarea numerică a expresiilor matematice cu precizie specificată
- Transformări: Fourier, Laplace, Transformarea z și inversele lor
- Funcții matematice speciale: Funcții speciale ale matematicilor aplicate clasice

10. PACHETUL DE MODELARE ȘI SIMULARE SIMULINK

SIMULINK este un pachet software pentru *modelarea, simularea și analiza sistemelor dinamice*. Pot fi modelate sisteme liniare și neliniare, continue, discrete, hibride, cu mai multe perioade de eșantionare.

- **SIMULINK** furnizează o interfață grafică utilizator (GUI) pentru crearea modelelor sub forma unor diagrame construite din blocuri, pe baza unor tehnici de tip *click-and-drag* realizate cu mouse-ul. Astfel, trasarea diagramelor este simplă și intuitivă, aproape la fel de simplă ca trasarea acestor diagrame direct pe hârtie. În plus, se evită formularea matematică laborioasă (sistemele dinamice sunt de regulă descrise de ecuații diferențiale sau cu diferențe).
- **SIMULINK** dispune de o bibliotecă vastă de surse, receptoare, componente liniare și neliniare, conectori etc. pe baza cărora se pot trasa diagrame și construi blocuri proprii.
- Modelele realizate în **SIMULINK** sunt ierarhice. Se poate vizualiza modelul de nivel înalt, iar la efectuarea unui dublu click pe blocul respectiv se coboară nivel după nivel astfel încât se pot observa toate detaliile de construcție și de organizare ale modelului.
- După crearea unui model se pot realiza simulări apelând la diverse metode de integrare din meniurile **SIMULINK** și/sau utilizând comenzi **MATLAB**. Prin utilizarea unor blocuri de tip osciloscop sau diverse dispozitive de afișare se pot observa rezultatele chiar în timpul simulării. De asemenea se pot schimba valorile unor parametri și se poate observa imediat efectul acestor modificări. Rezultatele obținute se pot transporta în workspace-ul **MATLAB** pentru prelucrări și vizualizări ulterioare.

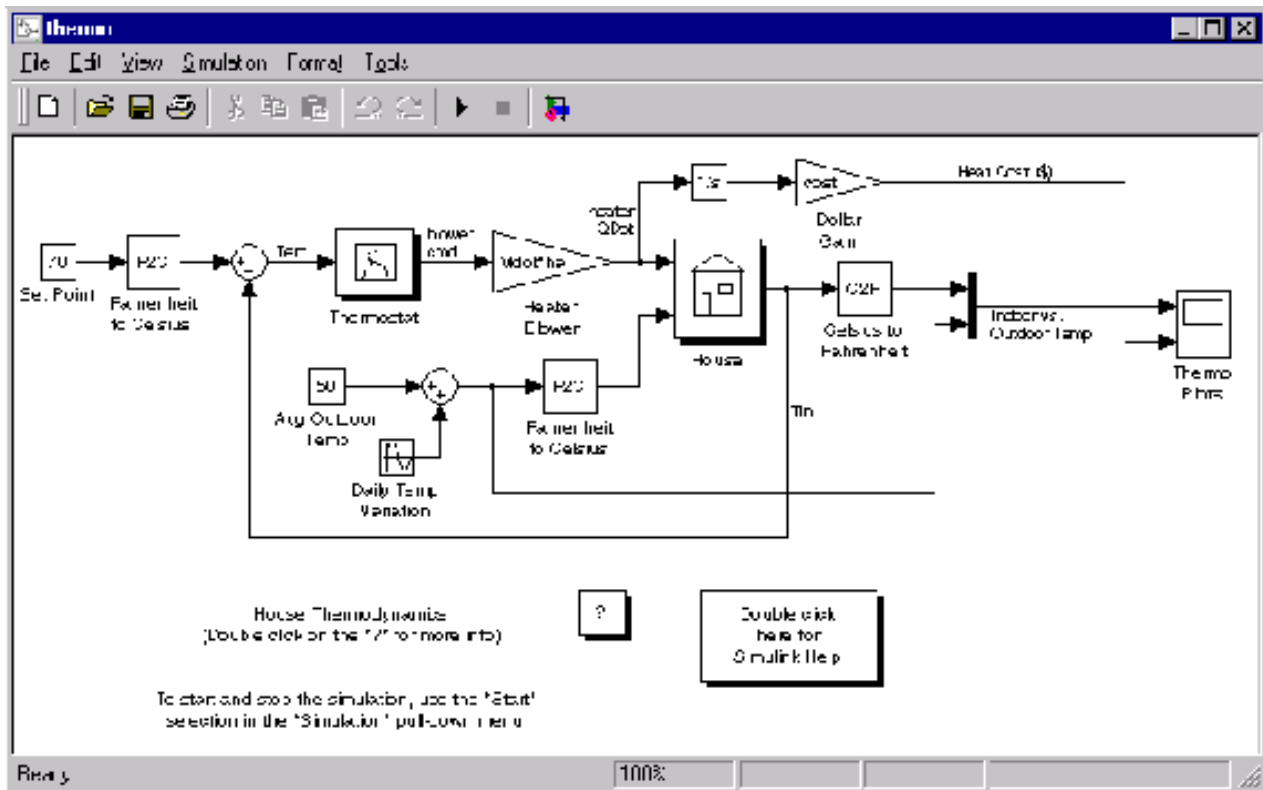
10.1. Rularea unui model SIMULINK demonstrativ Rularea modelului

Pentru a analiza modul de lucru cu **SIMULINK** se poate apela la rularea unor programe (modele) demonstrative.

Unul din programele demo este modelul termodinamic al unei case. Pentru rularea programului, trebuie parcurși următorii pași:

1. Se startează **MATLAB**.

2. Se rulează demonstrația tastând thermo în fereastra de comandă MATLAB sau se tastează comanda demo și se alege programul demonstrativ din meniul care apare. Aceste comenzi startează SIMULINK și creează o fereastră model care conține modelul respectiv.



La deschiderea modelului (extensiile fișierelor SIMULINK sunt .mdl) SIMULINK-ul deschide un bloc de tip osciloscop cu două ecrane (temperatură interioară/exterioară - Indoor vs. Outdoor Temp. și costul încălzirii - Heat Cost (\$)).

3. Pentru startarea simulării se activează meniul **Simulation** și se alege comanda **Start** command (sau se activează direct butonul **Start** din bara de instrumente). O dată cu startarea simulării sunt plotate evoluțiile temperaturii interioare și exterioare, ca și costul cumulativ al încălzirii.

4. Pentru oprirea simulării se alege comanda **Stop** din meniul **Simulation** (sau butonul **Pause** din bara de instrumente).

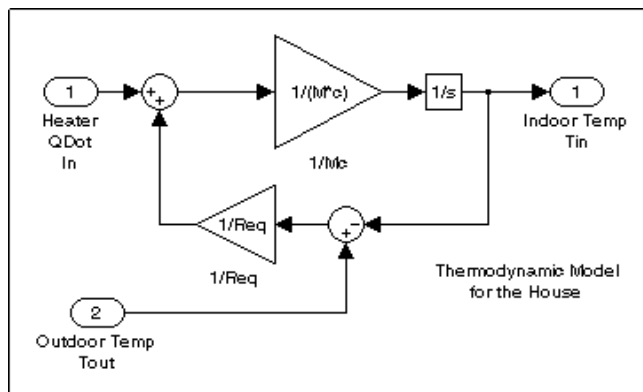
5. Atunci când se dorește terminarea rulării programului se închide modelul alegând **Close** din meniul **File**.

Descrierea modelului

Programul modelează sistemul termodinamic al unei case folosind o reprezentare simplă. Temperatura de referință este setată la 70 grade Fahrenheit (aprox. 21 grade Celsius). Temperatura din casă este influențată de temperatura exterioară, care poate fi variată sub formă sinusoidală (amplitudine 15 grade F, temperatura de bază 50 grade F), variație care simulează fluctuațiile temperaturii din exterior din timpul zilei.

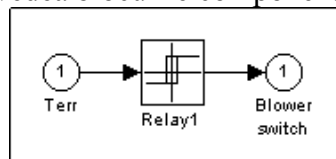
Sunt utilizate subsisteme care fac modelul simplu și configurabil (un subsistem este un bloc alcătuit dintr-un grup de blocuri conectate). Modelul conține 5 subsisteme: Thermostat, House și trei Convertoare de temperatură (Temp Convert), din care 2 convertesc Fahrenheit în Celsius și unul Celsius în Fahrenheit.

Efectuarea unui dublu click pe blocul House permite vizualizarea blocurilor componente ale subsistemului.



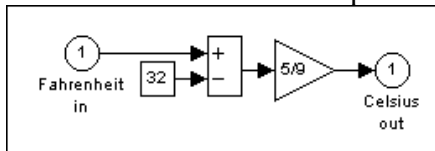
House subsystem

Subsistemul Termostat este de tip releu și determină pornirea sau oprirea sistemului de încălzire. Se pot vedea blocurile componente la efectuarea unui dublu click pe subsistem.



Thermostat subsystem

Subsistemele de conversie a temperaturii au o structură asemănătoare:



Fahrenheit to Celsius conversion (F2C)

Alte demonstrații

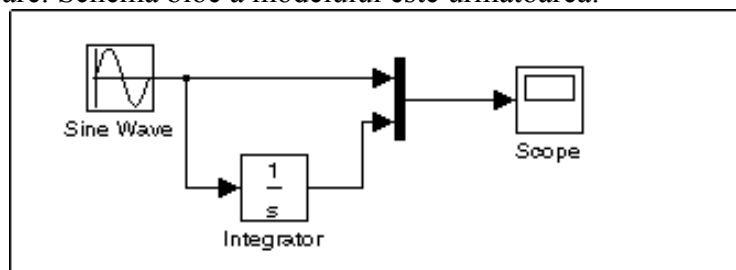
Din fereastra bibliotecilor SIMULINK pot fi rulate și alte demonstrații care pun în evidență diverse concepte și tehnici de modelare din diverse domenii. Pentru rularea din fereastra bibliotecilor SIMULINK se procedează astfel:

1. Se tastează `simulink3` în fereastra de comandă MATLAB; va apare fereastra bibliotecilor SIMULINK.
2. Se execută dublu click pe icon-ul Demos. Va apare fereastra demo a MATLAB-ului, care conține câteva modele SIMULINK interesante.

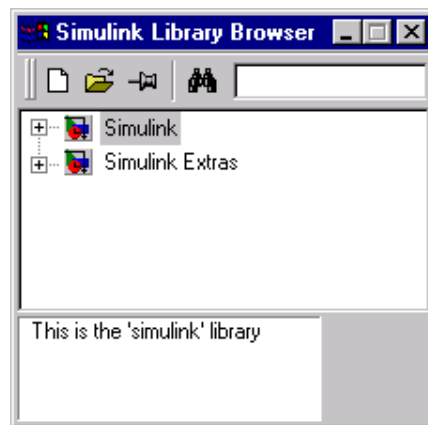
10.2. Crearea modelelor SIMULINK

Tehnica de creare a unor modele SIMULINK poate fi ilustrată cel mai bine prin exemple.

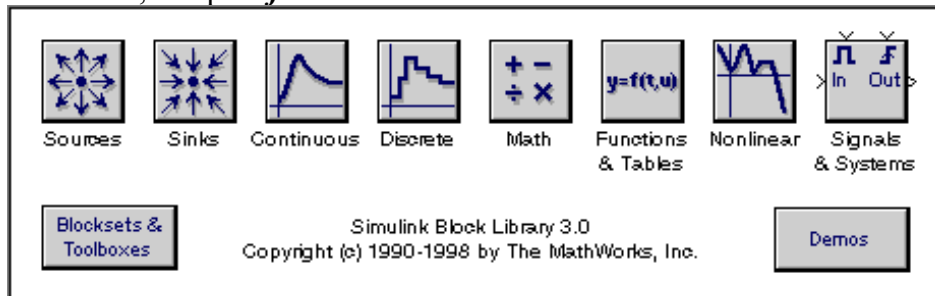
Modelul prezentat în continuare integrează un sinus și afișează atât rezultatul cât și unda sinusoidală de la intrare. Schema bloc a modelului este următoarea:



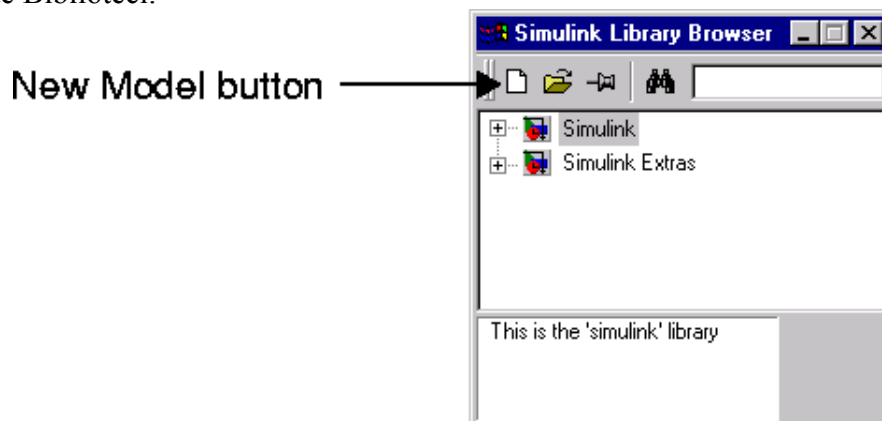
Pentru a genera modelul se tastează mai întâi `simulink` în fereastra de comandă MATLAB. Pe sistemele de operare de tip Windows va apare **Browser-ul bibliotecilor** SIMULINK.



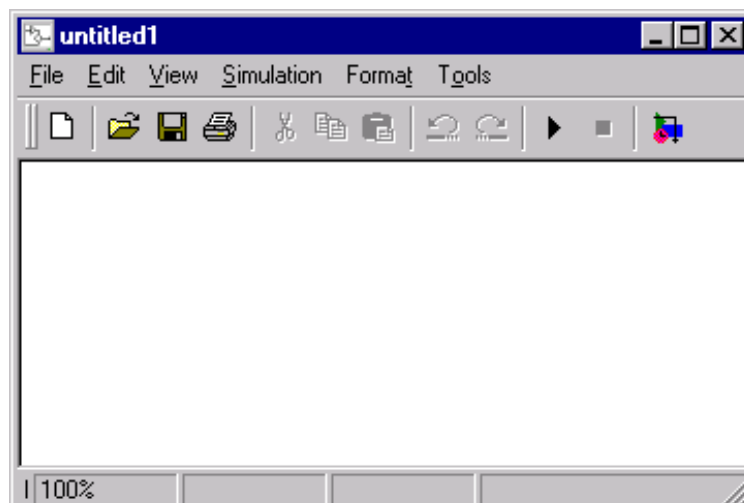
Pe sistemele UNIX, va apare *fereastra bibliotecilor* SIMULINK.



Pentru a genera un model nou pe sisteme UNIX se selectează **Model** din submeniul **New** al meniului **File**. Pe sisteme Windows se selectează butonul **New Model** din bara de instrumente a Browser-ului de Biblioteci.



Simulink va deschide o fereastră pentru un model nou.

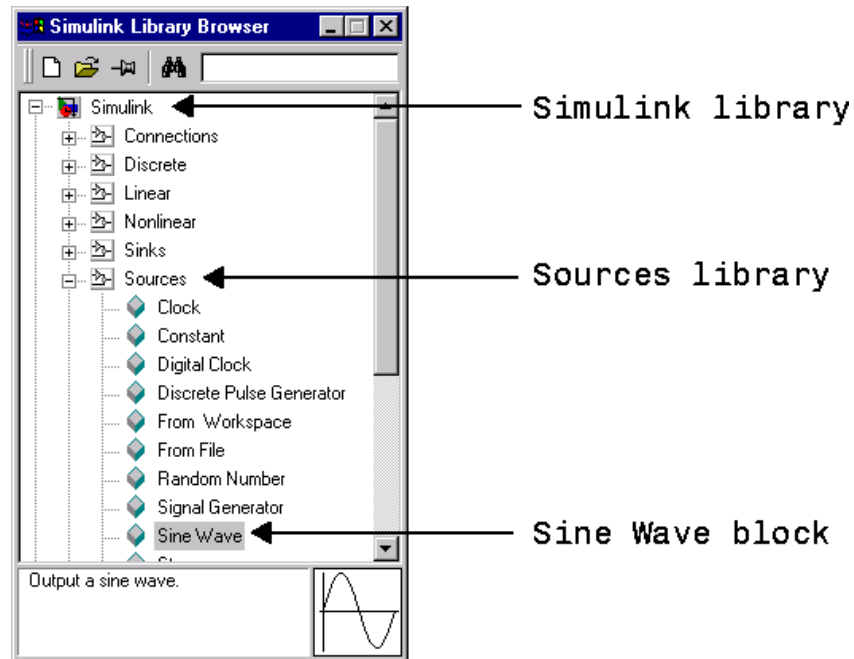


Pentru construcția modelului vor fi necesare blocuri din următoarele biblioteci Simulink:

- Biblioteca de surse (blocul Sine Wave)
- Biblioteca de receptoare (blocul Scope)
- Biblioteca de sisteme continue (blocul Integrator)
- Biblioteca Signals & Systems (blocul Mux)

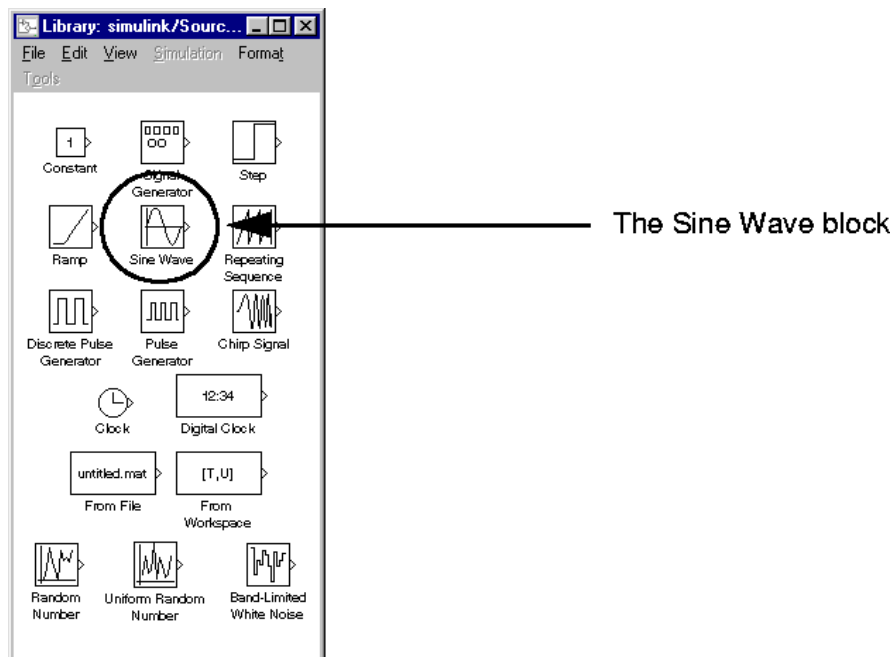
Pentru copierea blocului Sine Wave se utilizează Browser-ul de biblioteci: întâi se expandează arborele de biblioteci (prin click pe nodul Simulink și apoi click pe nodul surse) astfel încât să fie afișate blocurile din biblioteca de surse. Apoi se selectează blocul Sine Wave (click).

Fereastra Browser-ului de biblioteci va arăta astfel:

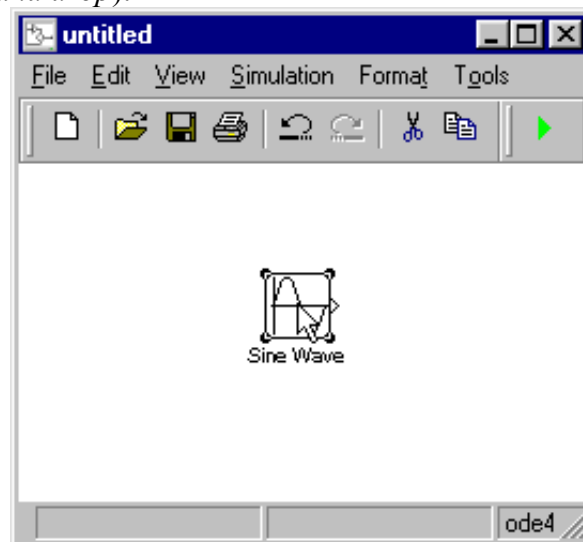


La pasul următor se trage (ținând apăsat butonul din stânga al mouse-ului) blocul Sine Wave din browser și i se dă drumul în fereastra modelului. Simulink va face o copie a blocului Sine Wave în punctul indicat.

Se poate proceda asemănător pentru copierea blocului Sine Wave din biblioteca de surse deschisă din *fereastra de biblioteci* Simulink (pe sisteme Windows se poate deschide fereastra de biblioteci din Browser prin click din butonul drept al mouse-ului și apoi click pe **Open Library**).

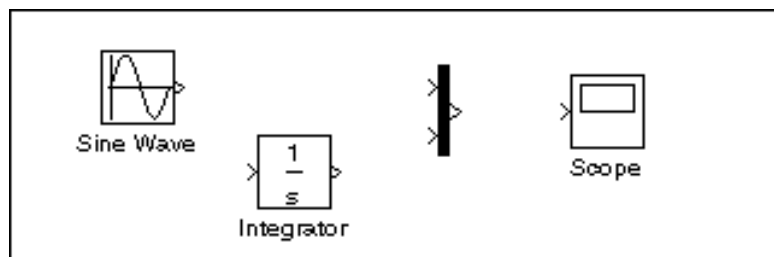


Ca și în cazul utilizării browser-ului se trage blocul Sine Wave din biblioteca de surse în fereastra modelului (*drag and drop*):

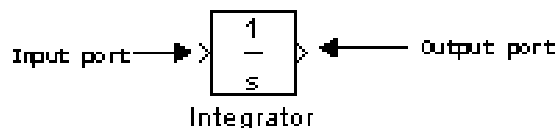


Se procedează în mod asemănător și cu copierea celorlalte blocuri din bibliotecile corespunzătoare în fereastra modelului. Se poate deplasa cu ușurință orice bloc prin tragerea cu mouse-ul sau prin selectare și deplasare cu tastele săgeți.

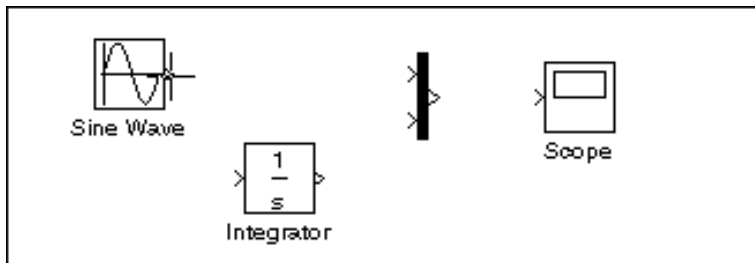
După copierea tuturor blocurilor necesare în fereastra de lucru, modelul trebuie să arate ca în figura următoare:



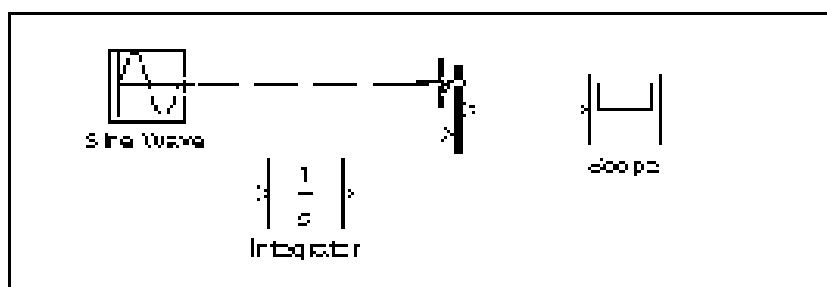
La o examinare atentă a simbolurilor de reprezentare a blocurilor se vor observa săgeți care indică intrările sau ieșirile din blocuri: dacă simbolul $>$ este orientat spre ieșirea blocului atunci este un port de ieșire (*output port*) iar dacă simbolul este îndreptat spre bloc este un port de intrare (*input port*). Un semnal circulă de la un port de ieșire al unui bloc spre un port de intrare al altui bloc printr-o linie de conectare, Atunci când blocurile sunt conectate, simbolurile porturilor dispar.



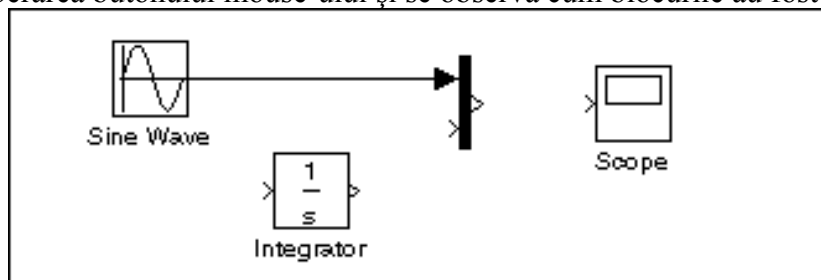
Pentru conectarea blocurilor se trece întâi la conectarea blocului Sine Wave la prima intrare (de sus) a blocului Mux. Pentru aceasta se poziționează pointerul mouse-ului deasupra portului de ieșire al blocului Sine Wave. În acest moment forma pointerului se schimbă și devine de tip cruce (cursor).



Se ține apăsat butonul stânga al mouse-ului și se deplasează cursorul până la intrarea de sus a blocului Mux.



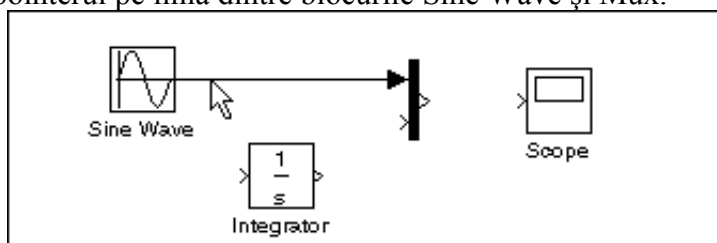
Urmează eliberarea butonului mouse-ului și se observă cum blocurile au fost conectate.



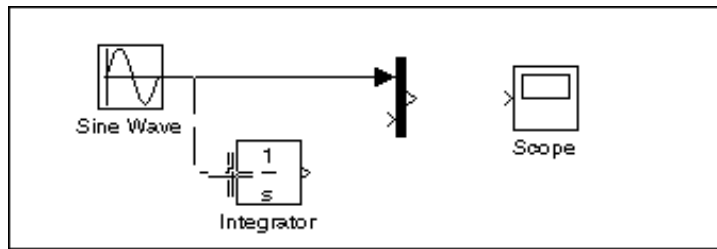
În afara liniilor care conectează ieșirile unor blocuri la intrările altora există și linii de bransare a unor linii la intrările unor blocuri (se poate observa în modelul prezentat inițial). O astfel de linie este utilizată pentru conectarea ieșirii din blocul Sine Wave și la blocul Integrator (există deja conexiunea la blocul Mux).

Pentru a efectua această conexiune se procedează astfel:

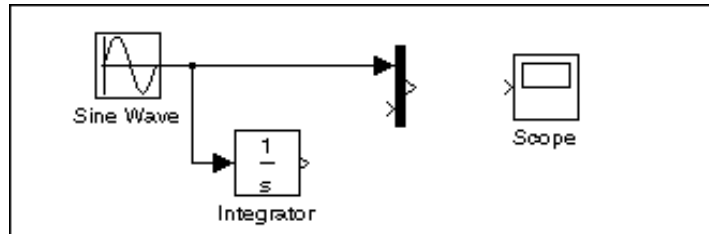
1. Se poziționează pointerul pe linia dintre blocurile Sine Wave și Mux.



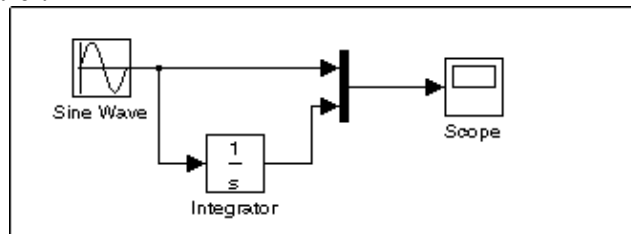
2. Se apasă și se ține apăsată tasta **Ctrl**. Se apasă butonul mouse-ului și apoi se trage până la intrarea în blocul Integrator sau până deasupra acestui bloc.



3. Se eliberează butonul mouse-ului și se observă cum apare o linie de branșare până la portul de intrare în blocul Integrator.



Se procedează conform indicațiilor și se efectuează toate conectările necesare. Modelul va trebui să arate în final astfel:

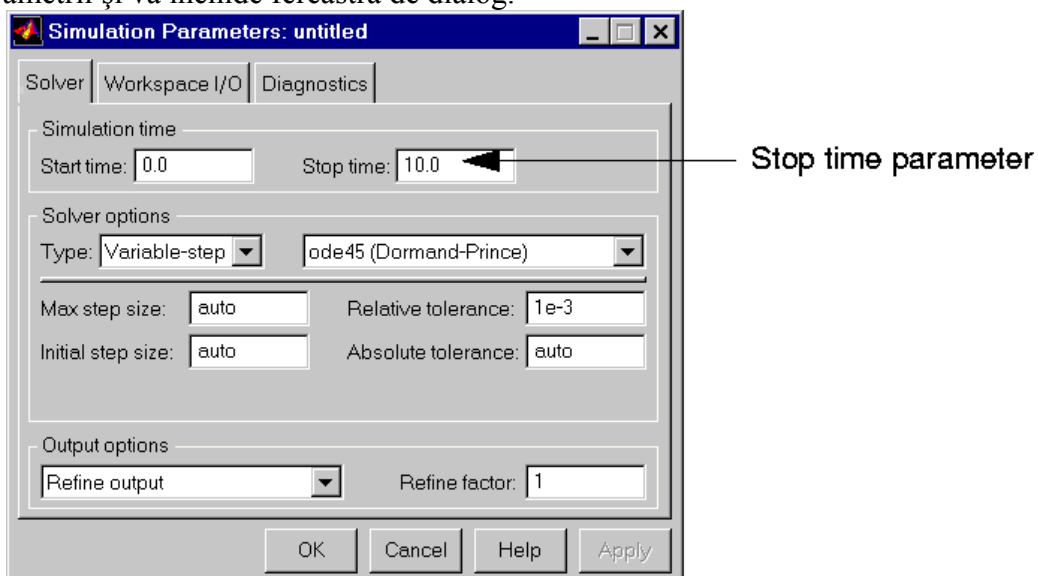


10.3. Rularea simulărilor în SIMULINK

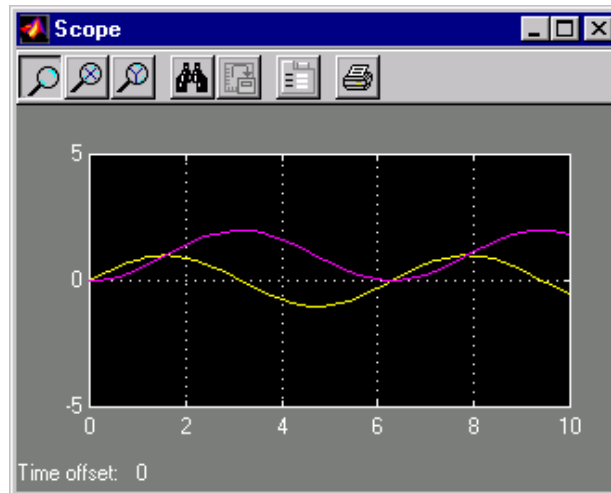
După încheierea procedurii de construcție a modelului, trebuie rulată o simulare pentru aprecierea corectitudinii modelului și pentru obținerea rezultatelor cerute.

Pentru aceasta se deschide mai întâi blocul osciloscopului (Scope), pentru a vizualiza evoluția mărimilor modelului. Păstrând fereastra osciloscopului deschisă se va seta Simulink pentru rularea unei simulări timp de 10 secunde. Pentru aceasta, parcurgem următorii pași:

1. Setăm parametrii simulării prin alegerea submeniului **Parameters** din meniul **Simulation**. În fereastra de dialog care apare vom seta parametrul **Stop time** la 10.0 (valoare implicită).
2. Închidem fereastra de dialog **Simulation Parameters** prin click pe butonul **Ok**. Simulink va aplica parametrii și va închide fereastra de dialog.



3. Se selectează **Start** din meniul **Simulation** și se observă curbele afișate în fereastra osciloscopului.



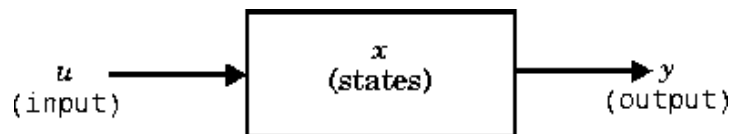
4. Simularea se va opri atunci când se ajunge la finalul timpului de rulare indicat în fereastra de dialog **Simulation Parameters** sau la selectarea opțiunii **Stop** din meniul **Simulation** (sau din bara de instrumente).

Pentru salvarea modelului se alege **Save** din meniul **File** și se introduce un nume de fișier și se alege directorul unde dorim să salvăm modelul (fișierul va avea automat extensia .mdl).

Pentru încheierea lucrului se selectează **Exit MATLAB** sau se tastează `quit` în fereastra de comandă a MATLAB-ului.

10.4. Modul de lucru al unui program SIMULINK

Fiecare bloc dintr-un model Simulink are următoarele caracteristici generale: un vector de intrare, u , un vector de ieșire, y , și un vector de stare, x :



Vectorul de stare poate consta din stări continue, stări discrete sau combinații ale acestora. Relațiile matematice dintre aceste mărimi (intrări, ieșiri, stări) sunt exprimate prin ecuații de tipul:

$$\begin{aligned}
 y &= f_0(t, x, u) && \text{iesirea} \\
 x_{d_{k+1}} &= f_u(t, x, u) && \text{actualizar e} \\
 x'_c &= f_d(t, x, u) && \text{derivata} \\
 \text{unde } x &= \begin{bmatrix} x_c \\ x_{d_k} \end{bmatrix}
 \end{aligned}$$

Simularea constă în două faze: inițializare și simulare propriu-zisă.

Faza de inițializare înseamnă parcurgerea următoarelor etape:

1. Blocul parametrilor este trecut în MATLAB pentru evaluare. Valorile numerice rezultate sunt folosite ca parametri actuali (curenți).

2. Este parcursă ierarhia modelului. Fiecare subsistem care nu este un subsistem executat condiționat este înlocuit prin blocurile componente.

3. Blocurile sunt sortate în ordinea în care este necesară actualizarea lor. Algoritmul de sortare întocmește o listă astfel încât orice bloc nu este actualizat până când blocurile care furnizează intrările acestuia nu sunt actualizate. În timpul derulării acestei etape sunt detectate buclele algebrice.

4. Conexiunile dintre blocuri sunt verificate pentru asigurarea compatibilității ieșire-intrare.

Urmează faza de simulare propriu-zisă. Modelul este simulat prin integrare numerică. Calculul derivatelor se face în doi pași. Prima dată ieșirea fiecărui bloc este calculată în ordinea determinată de algoritmul de sortare. La al doilea pas, pentru fiecare bloc se calculează derivatele în funcție de timp, intrări și stări. Vectorul derivatelor rezultat este returnat algoritmului de rezolvare de tip ODE, care îl utilizează pentru calculul noului vector de stare pentru momentul de timp următor. O dată ce noul vector de stare este calculat, blocurile sunt actualizate.

BIBLIOGRAFIE

1. Ionete C., Selișteanu D., Petrișor A. – *Proiectarea sistemică asistată de calculator în MATLAB*, Reprografia Universității din Craiova, 1995.
2. Leonard N.E., Levine W.S – *Using MATLAB to analyze and design Control Systems*, Addison-Wesley Publ., SUA, 1995.
3. Marchand P. – *Graphics and GUIs with MATLAB*, CRC Press, SUA, 1999.
4. *** – *MATLAB User's Guide*, The Mathworks Inc., SUA, 2000.