# SOFT MATEMATIC

## Note de curs

Ana-Maria MOŞNEAGU

Universitatea "Al. I. Cuza" Iaşi
Facultatea de Matematică

# Ce este **MATLAB**?

- **MATLAB** (R2021b) este un produs MathWorks®, fiind un pachet de programe de înaltă performanță conceput pentru calculul științific și reprezentări grafice în domeniul științelor și cel al ingineriei;
- numele programului este o abreviere de la **MATrix LABoratory**, elementul de bază cu care operează **MATLAB** fiind matricea;
- este folosit în mod uzual în mediile universitare, în activitatea de cercetare, dar și în practică în domenii ca matematici financiare, statistică, procesarea semnalelor digitale, biologie etc.
- **MATLAB** este un program flexibil și simplu de învățat/utilizat, dar care poate fi folosit și ca limbaj de programare de nivel înalt: utilizatorul poate folosi funcțiile deja existente în **MATLAB** și în plus, poate adăuga propriile sale programe, dezvoltând aplicații specifice domeniului în care activează;

# Ce este **MATLAB**?

- **MATLAB** conține un nucleu de bază (kernel), cu interpreter propriu, în jurul căruia sunt construite trusele de unelte (toolbox-urile) – colecții extinse de funcții **MATLAB** scrise pentru a rezolva probleme din diverse domenii specializate;
- folosind comanda **MATLAB** `demo` se poate urmări o demonstrație atât a principalelor facilități oferite de **MATLAB** cât și a toolbox-urilor existente;
- interfața grafică (GUI – Graphical User Interface) a **MATLAB** -ului este, de asemenea, foarte bună, iar funcționalitățile oferite sunt ușor de folosit;
- este posibilă scrierea de programe în alte limbaje de programare (de ex. în limbajul C) care să interacționeze cu un program **MATLAB** .

# Bibliografie


- S. Curteanu, *Inițiere în MATLAB*, Ed. Polirom, Iași, 2008.
- M. Ghinea, V. Firețeanu, *MATLAB. Calcul Numeric, Grafică. Aplicații*, Ed. Teora, București, 2001.
- S. Attaway, *MATLAB: A Practical Introduction to Programming and Problem Solving, Fourth Ed.*, Elsevier Academic Press, 2016.
- D. Higham, N. Higham, *MATLAB Guide (Third Edition)*, SIAM, Philadelphia, 2017.
- http://www.mathworks.com

# Lansarea și oprirea aplicației **MATLAB**

- sub SO Windows, **MATLAB** se lansează dând un click dublu pe iconul corespunzător sau selectând programul din meniul Start;
- fereastra aplicației conține o bară de titlu, bară de meniuri, bară de instrumente și ferestrele: Command Window, Current Folder și Workspace;
- în fereastra de comandă (Command Window) sunt tastate comenzile după prompterul **MATLAB** care este indicat prin $>>$;
- după ce o comandă a fost introdusă și a fost apăsată tasta ENTER, **MATLAB** execută comanda și rezultatul apare pe ecran;
- prin tastarea comenzii

```
>>version
```

  se afișează versiunea de **MATLAB** folosită;

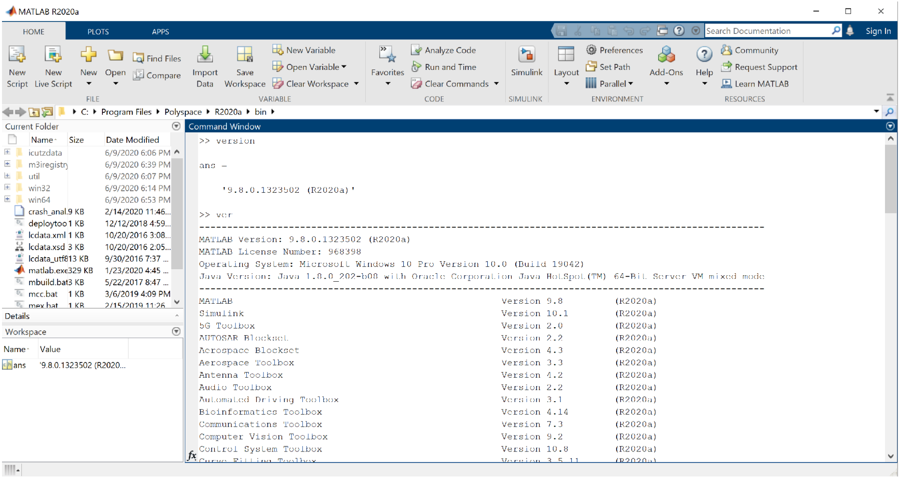
# Lansarea și oprirea aplicației **MATLAB**

- comanda

  ```
  >>ver
  ```

  indică, pe lângă versiunea curentă de **MATLAB** și toolbox-urile existente;
- fereastra Current Folder descrie fișierele din directorul curent;
- în fereastra Workspace sunt afișate toate variabilele declarate;
- oprirea aplicației **MATLAB** se poate face în diferite moduri: prin intermediul comenzii `quit` tastată în Command Window sau din meniul **File** −>**Exit** sau click pe butonul de închidere aflat în partea din dreapta sus a ferestrei aplicației.

# Fereastra aplicației



# Sistemul de **help**

Informațiile oferite de sistemul de help al aplicației **MATLAB** pot fi obținute astfel:

- din linia de comandă se utilizează comanda

```
>>help nume_functie
```

- de exemplu, prin

```
>>help exp
```

  este afișată o scurtă descriere a funcției, sintaxa și o listă a subiectelor înrudite;

- folosind instrumentul **Help** sau tastând comanda `doc`;
- utilizând comanda `lookfor`, urmată de un cuvant cheie, care caută în sistemul de fișiere help cuvântul cheie; de exemplu

```
>>lookfor interpolation
```

  oferă informații despre rutinele de interpolare;

- utilizând **MATLAB** helpdesk memorat pe hard sau CD;

# Sistemul de **help**

- comanda

  `>> help help`

  oferă o scurtă descriere a comenzii `help`;
- pentru a obține doar un ecran la un moment dat se poate folosi comanda

  `>> more on`
- pentru a trece la pagina următoare se apăsa tasta *spacebar*;
- **MATLAB** este case sensitive – face diferența între literele mici și literele mari.

# Variabile și comenzi în **MATLAB**

- numele unei variabile în **MATLAB** trebuie să înceapă cu o literă; acesta poate fi format dintr-un număr arbitrar de litere, cifre sau simboluri;
- tipul de bază al unei variabile simple este double;
- **MATLAB** folosește unele variabile speciale, interne, cum ar fi:

| **Variabila** | **Scop** |
|---|---|
| *ans* | păstrează valoarea ultimei expresii evaluate, rezultat ce nu a fost atribuit unei variabile |
| *eps* | valoarea acestei variabile este de aproximativ 2.2204e-016 și reprezintă precizia de calcul a **MATLAB** -ului |
| *pi* | numărul $\pi$ |
| *i* sau *j* | numărul complex $i$ cu proprietatea $i^2 = -1$ |
| *Inf* | desemnează $\infty$; rezultatul calculului $1/0$ |
| *NaN* | Not a Number; se obține ca rezultat al unei operații ilegale sau al unei nedeterminări din analiza matematică ($0/0, \infty/\infty, \infty - \infty$, etc.) |

# Variabile și comenzi în MATLAB

- valorile variabilelor interne **MATLAB** nu ar trebui modificate de către utilizator; dacă accidental, aceste valori implicite se modifică, se poate folosi comanda `clear` pentru a se reveni la valorile implicite;
- în **MATLAB**, variabilele nu se declară explicit – acestea sunt introduse prin atribuirea unei valori;
- valoarea unei variabile poate fi o valoare numerică, o matrice sau de alte tipuri;
- variabilele pot fi reinițializate;
- nu este posibil să se efectueze calcule cu variabile cărora nu li s-a atribuit o valoare;

```
>> a = 1 + 2 * 3 + 4
a =
    11
>> b = a^2 + a - 2
b =
   130
```

# Variabile și comenzi în **MATLAB**

- pentru a afișa/vizualiza valoarea atribuită unei variabile, se tastează comanda formată din numele variabilei sau se deschide fereastra Workspace;

```
>> a
a =
     11
```

- o comandă nu trebuie să înceapă neapărat cu o atribuire de forma: variabilă = valoare;
- rezultatul unei evaluări poate fi atribuit automat unei variabile numită *ans* – variabilă standard (answer), ce poate fi folosită în calculele ulterioare:

```
>> 1 + 2 * 3 + 4
ans =
     11
>> ans^2 + ans - 2
ans =
    130
```

- dacă o comandă depășeste o linie, se poate încheia linia cu "...", apoi se apasă tasta "ENTER"; se continuă comanda pe a doua linie:

```
>> suma = 1 + 2 + 3 + 4 +...
5 + 6 + 7 + 8 + 9
suma =
      45
```

- dacă se dorește suprimarea afișării ultimei expresii evaluate, se va pune caracterul ";" la sfârșitul expresiei;
- pe o linie de comandă se pot introduce mai multe expresii separate prin caracterele "," sau ";"
- dacă o expresie este urmată de virgulă, valoarea expresiei va fi afișată, iar dacă se termină cu ";", valoarea expresiei nu va fi afișată:

```
>> x = 5, y = 7 * x, z = y; t = x + y + z
```

# Variabile și comenzi în **MATLAB**

- lista variabilelor utilizate în sesiunea curentă se poate vizualiza folosind comanda `who`, iar folosirea comenzii `whos` oferă, pe lângă lista variabilelor în uz, informații suplimentare despre acestea (dimensiune, tip etc.);
- comanda `whos` urmată de numele unei variabile utilizate în sesiunea curentă întoarce informații referitoare la variabila respectivă:

```
>>whos a
```

- comanda `clear` șterge toate variabilele din spațiul de lucru, iar comanda

```
>>clear a b
```

  șterge doar variabilele a și b;
- un calcul **MATLAB** (prea lung, infinit chiar) poate fi întrerupt cu "Ctrl-C"; după această întrerupere, un nou prompter apare și pot fi introduse noi comenzi.

# Comanda **format**

- formatul de afișare al valorilor numerice în **MATLAB** este controlat de comanda `format`;
- formatul implicit este afișarea cu 4 cifre după punctul zecimal; dacă se dorește o afișare cu 15 cifre zecimale, atunci se folosește comanda

```
>>format long
```

| **Comanda** | **Rezultatul afișat pentru numărul $\pi$** |
|---|---|
| format short | 3.1416 |
| format long | 3.141592653589793 |
| format shorte | 3.1416e+000 |
| format longe | 3.141592653589793e+000 |
| format bank | 3.14 |
| format rat | 355/113 |
| format hex | 400921fb54442d18 |

- pentru o listă completă a formatelor de afișare tastați comanda

```
>>help format
```

# Operatori aritmetici și funcții elementare

- operațiile aritmetice de bază între scalari sunt: $+$, $-$, $*$, $/$ și ridicarea la putere ^;
- ordinea implicită a operațiilor în **MATLAB** este cea standard: ridicarea la putere, apoi înmulțirea și împărțirea, apoi adunarea și scăderea; ordinea implicită se poate schimba cu ajutorul parantezelor;
- **MATLAB** dispune de un set bogat de funcții elementare și funcții matematice speciale, dintre care amintim:

| **Funcția MATLAB** | **Efectul** |
|---|---|
| `sin(x), cos(x)` | $sin(x), cos(x)$ |
| `tan(x), cot(x)` | $tan(x), cot(x)$ |
| `asin(x), acos(x)` | $sin^{-1}(x), cos^{-1}(x)$ |
| `atan(x), acot(x)` | $tan^{-1}(x), cot^{-1}(x)$ |
| `log(x), log2(x), log10(x)` | $ln(x), log_2(x), log_{10}(x)$ |
| `exp(x), pow2(x)` | $e^x, 2^x$ |

# Funcții elementare și funcții matematice speciale

| Funcția MATLAB | Efectul |
|---|---|
| `ceil` | rotunjește la cel mai apropiat întreg spre $\infty$ |
| `fix` | rotunjește la cel mai apropiat întreg spre 0 |
| `floor` | rotunjește la cel mai mic întreg spre $-\infty$ |
| `round` | rotunjește la cel mai apropiat întreg |
| `abs, angle, conj, imag, real` | funcții pentru numere complexe |
| `mod, rem, sign` | rest, semn |
| `beta, erf, expint, gamma, legendre` | funcții matematice |
| `factor, gcd, isprime, lcm, primes, nchoosek, perms, rat, rats` | funcții din teoria numerelor |
| `cart2sph, cart2pol, pol2cart, sph2cart` | transformări de coordonate |

# Matrice

- elementul de bază în **MATLAB** este matricea;
- o variabilă simplă este considerată ca fiind o matrice $1 \times 1$;
- cel mai des utilizate sunt matricele $m \times n$, care sunt tablouri bidimensionale de elemente aranjate pe $m$ linii și $n$ coloane;
- vectorii linie $(m = 1)$ și vectorii coloană $(n = 1)$ sunt cazuri particulare de matrice;
- tablourile și dimensiunile lor nu sunt declarate în mod explicit;
- în **MATLAB** există mai multe modalități prin care se poate genera o matrice;
- modalitatea explicită necesită utilizarea parantezelor pătrate între care se precizează elementele matricei, pe linii; liniile se despart prin ";" sau apăsând tasta ENTER, iar elementele unei linii se separă prin spații libere sau prin ",":

# Matrice

```
>> A = [1 2; 3 4]
>> A = [1,2; 3,4]
>> A = [1 2
3 4]
```

- rezultatul este același, pentru fiecare dintre cele 3 comenzi și anume:

```
A =
     1     2
     3     4
```

- dimensiunea unei matrice se poate obține folosind comanda `size`;

```
>> C = [1 2; 3 4; 5 6];
>> size(C)
ans =
     3     2
```

- se returnează un vector cu două elemente: numărul de linii, respectiv numărul de coloane;

# Matrice

- elementele unui matrice pot fi numere reale, complexe sau orice expresii **MATLAB**

```
>> x = [0.7 1+9/4^2 sqrt(5) exp(2)]
x =
    0.7000    1.5625    2.2361    7.3891
```

- **MATLAB** are un set util de funcții pentru generarea unor matrice speciale: `zeros` – matricea nulă, `ones` – matrice formată doar din elemente 1, `eye` – matricea identitate, `repmat` – matrice obținută prin replicarea unei alte matrice, `rand` – matrice de numere pseudo-aleatoare uniform distribuite în $[0, 1]$, `randn` – matrice de numere pseudo-aleatoare normal distribuite, de medie 0 și dispersie 1, `linspace` – vector de elemente echidistante, `logspace` – vector de elemente spațiate logaritmic;

# Matrice

```
>>O1 = zeros(2,3)
%genereaza o matrice cu 2 linii si 3 coloane de zerouri
>>O2 = zeros(3)
%genereaza o matrice patratica de ordin 3 de zerouri
>>U = ones(4,2)
%genereaza o matrice cu 4 linii si 2 coloane
%cu elemente 1
>>I = eye(3) %genereaza matricea identitate de ordin 3
>>x = rand
%genereaza un numar aleator distribuit uniform pe [0,1]
>>A = rand(3,4)
%genereaza o matrice (3 linii, 4 coloane) de numere
%aleatoare distribuite uniform pe [0,1]
```

# Matrice

```
>>y =  randn
%genereaza un numar aleator distribuit normal standard
>>B = randn(6,5)
%genereaza o matrice (6 linii, 5 coloane) de numere
%aleatoare distribuite normal standard
>>x = linspace(0, 1, 50)
%genereaza un vector de 50 de elemente echidistante intre
%0 si 1
```

- funcția **logspace(X1, X2, N)** generează un vector linie de N elemente spațiate logaritmic între $10^{X1}$ și $10^{X2}$. Dacă X2 este numărul $\pi$, atunci elementele sunt între $10^{X1}$ și $\pi$.

# Matrice

- matricele pot fi construite și sub formă de bloc;
- dacă matricea $A$ este deja generată prin

  ```
  >>A = [1 2; 3 4]
  ```

  atunci putem genera o matrice $B$ astfel:

  ```
  >>B = [A zeros(2); eye(2) ones(2)]
  B =
       1     2     0     0
       3     4     0     0
       1     0     1     1
       0     1     1     1
  ```

# Matrice

- matricele diagonale pe blocuri se pot defini utilizând funcția `blkdiag`, comanda `Y = blkdiag(A,B,...,Z)` având ca efect generarea unei matrice $Y$ de componenete:

$$Y = \begin{pmatrix} A & 0 & \cdots & 0 \\ 0 & B & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Z \end{pmatrix}$$

- funcția `repmat` permite construirea unei matrice prin repetarea unei matrice deja generate

  `repmat(A,m,n)`,

  unde $A$ este matricea ce va fi replicată;
- astfel se crează o matrice de $m$ pe $n$ blocuri în care fiecare bloc este o copie a matricei $A$; dacă parametrul $n$ lipsește, valoarea sa implicită este $m$;
- folosind matricea $A$ generată anterior, construim matricea C astfel:

```
>>C = repmat(A,2,3)
```

# Matrice

- exemple de funcții folosite pentru manipularea matricelor: `reshape` – schimbarea dimensiunii, `diag` – matrice diagonale și diagonale ale matricelor, `blkdiag` – matrice diagonală pe blocuri, `tril` – extragerea părții triunghiulare inferior, `triu` – extragerea părții triunghiulare superior, `fliplr` – rotirea matricei în jurul axei de simetrie verticale, `flipud` – rotirea matricei în jurul axei de simetrie orizontale, `rot90` – rotirea unei matrice cu 90 de grade;
- funcția `reshape(A,m,n)` produce o matrice $m \times n$ ale cărei elemente sunt elementele matricei $A$, considerate pe coloană (în cazul în care matricea $A$ nu are $m * n$ elemente, rezultă o eroare)

```
>>A = [1 2 3; 4 5 6]; B = reshape(A,3,2)
B =
     1     5
     4     3
     2     6
```

# Matrice

- funcția `diag` poate avea ca argument o matrice sau un vector. Dacă $x$ este un vector, `diag(x)` generează o matrice cu diagonala principală $x$:

  `>>diag([1 2 3])`

- mai general, `diag(x,k)` pune vectorul $x$ pe diagonala $k$, unde $k = 0$ desemnează diagonala principală, $k > 0$ specifică diagonalele situate deasupra diagonalei principale, iar $k < 0$ diagonalele situate sub diagonala principală:

  `>>diag([1 2],3)`
  `>>diag([1 2],-1)`

- dacă argumentul funcției `diag` este o matrice pătratică $A$, atunci `diag(A)` returnează vectorul coloană format din elementele de pe diagonala principală a matricei $A$;

- analog cazului vectorial, `diag(A,k)` produce un vector coloană format din a $k$-a diagonală a matricei A;

- funcția `tril(A)` returnează o matrice ce conține partea triunghiulară inferior a matricei $A$ (elementele situate pe diagonală principală și cele situate dedesubtul ei) și în rest elementele zero;
- funcția `triu(A)` construește o matrice ce conține partea triunghiulară superior a matricei $A$, în rest elementele fiind zero;
- mai general, funcția `tril(A,k)` generează o matrice ce cuprinde elementele situate pe diagonala $k$ a matricei $A$ și sub aceasta (restul elementelor matricei vor fi 0), în timp ce funcția `triu(A,k)` crează o matrice ce conține elementele situate pe diagonala $k$ a matricei $A$ și deasupra ei (restul elementelor matricei vor fi 0);
- **MATLAB** deține un set de funcții pentru generarea unor matrice speciale, cum ar fi: `vander`, `hadamard`, `hilb`, `pascal` etc.

- indexarea tablourilor în **MATLAB** începe de la 1;
- pentru a accesa elementul de pe linia $i$, coloana $j$ a matricei $A$, scriem $A(i,j)$;
- în cazul unui vector $x$ (coloană sau linie), pentru a accesa elementul $i$, vom scrie $x(i)$

```
>> x = [-1.3 2*3^2 sqrt(5)];
>> y = x(2)
y =
     18
>> x(5) = abs(x(1))
x =
   -1.3000   18.0000    2.2361         0    1.3000
```

# Matrice

- pentru a permite accesul și atribuirea la nivel de submatrice, **MATLAB** folosește caracterul ":";
- `A(l1:l2,c1:c2)` desemnează submatricea constând din elementele matricei $A$, începând cu linia l1 până la linia l2 și de la coloana c1 la coloana c2;
- o scriere de forma `A(:,j)` desemnează coloana $j$ a matricei $A$, iar `A(i,:)` reperezintă linia $i$;
- poate fi folosit și cuvântul cheie `end` (de exemplu, `A(end,:)` selectează ultima linie a matricei $A$);
- o atribuire de forma `x = A(:)` generează un vector coloană ce conține toate elementele matricei $A$, așezate coloană după coloană, începând cu prima;
- notația [] semnifică matricea vidă, $0 \times 0$;

```
>>A = [1 2 3; 4 5 6; 7 8 9]
>>A(5,5) = 10
>>A(4,:) = [], A(:,4) = []
>>B = A(1:3,2:3)
```

- pentru a genera un vector cu componente de la *a* la *b* cu pasul 1, vom folosi comanda

```
>>x = a : b
```

iar un pas de echidistanță *h*, diferit de 1, este specificat prin comanda

```
>>x = a : h : b
```

- pentru a determina dimensiunea unui vector se poate folosi, ca și la matrice, funcția `size` sau funcția `length`:

```
>>x = 0:.01:1
>>length(x)
ans =
     101
>>y = 3:-1:-3
>>size(y)
ans =
     1     7
```

# Operații în sens matriceal și în sens tablou

- operațiile aritmetice de bază între scalari sunt $+, -, *, /$ și ^;
- în plus, **MATLAB** -ul oferă operatorul de împărțire la stânga, $\backslash$, care, pentru doi scalari $a$ și $b$, are semnificația de $b/a$;

```
>>a = 4; b = 2; c = a / b, d = a \ b
c =
     2
d =
   0.5000
```

- în ceea ce privește operațiile pe matrice, acestea sunt aceleași ca în cazul scalarilor;
- toate aceste operații pot fi realizate atât în sens matriceal, respectând regulile algebrei matriceale, cât și în sens tablou, adică element cu element ("array-smart operation");

# Operații în sens matriceal și în sens tablou

- operațiile de adunare și scădere sunt aceleași atât în sens matriceal, cât și în sens tablou;
- dacă $A$ și $B$ sunt două matrice, atunci o comandă de tipul $A + B$ sau $A - B$ adună, respectiv scade, cele două matrice ce trebuie să aibă aceeași dimensiune, cu excepția cazului când una dintre cele două matrice este de tip $1 \times 1$, adică este un scalar:

```
>> A = [1 2; 3 4]; B = eye(2); A + B, A - B
ans =
     2     2
     3     5
ans =
     0     2
     3     3
>> A = [1 2; 3 4]; B = 2; A + B, A - B
```

```
ans =
     3     4
     5     6
ans =
    -1     0
     1     2
```

- operația $A * B$ (operație în sens matriceal) reprezintă produsul uzual al matricelor $A$ și $B$ (numărul de coloane al matricei $A$ trebuie să coincidă cu numărul de linii al matricei $B$, cu excepția cazului când una dintre cele două matrice este de tip $1 \times 1$, adică este un scalar, caz în care fiecare element al matricei este înmulțit cu scalarul):

# Operații în sens matriceal și în sens tablou

```
>> A = [1 2 3; 4 5 6]; B = [1 2; 3 4; 5 6]; A * B, B * A
ans =
    22    28
    49    64
ans =
     9    12    15
    19    26    33
    29    40    51
>> A * 2, 4 * B
ans =
     2     4     6
     8    10    12
```

```
ans =
     4     8
    12    16
    20    24
```

- înmulțirea în sens tablou sau pe elemente se specifică plasând un punct înaintea operatorului de înmulțire "*";
- dacă $A = [a_{ij}]$ și $B = [b_{ij}]$ sunt matrice de aceeași dimensiune, atunci comanda

  ```
  >>C = A .* B
  ```

  construiește matricea $C = [c_{ij}]$ având aceeași dimensiune și componentele $c_{ij} = a_{ij}b_{ij}$:

  ```
  >> A = [1 2; 3 4]; B = [5 6;7 8]; A .* B
  ans =
       5    12
      21    32
  ```

# Operații în sens matriceal și în sens tablou

**Exercițiu**

*Ce se va afișa în urma execuției comenzii*

```
>>B .* A
```

*cu matricele $A$ și $B$ ca în exemplul de mai sus?*

# Operații în sens matriceal și în sens tablou

- operatorii de împărțire / (împărțirea la dreapta) și \(împărțirea la stânga) definesc soluții ale unor sisteme liniare;
- $A\backslash B$ este soluția $X$ a sistemului $A * X = B$ (dacă $A$ este o matrice $N \times N$, iar $B$ este o matrice $N \times M$, atunci soluția sistemului este determinată cu algoritmul lui Gauss de eliminare, dacă matricea $A$ este singulară, atunci va fi afișat un mesaj de eroare, iar dacă $A$ este o matrice $M \times N$, $(M < N$ sau $M > N)$ iar $B$ este o matrice $M \times P$, atunci soluția sistemului sub sau supradeterminat este în sensul celor mai mici pătrate);
- $A/B$ este soluția $X$ a sistemului $X * B = A$. Mai exact, $A/B = (B^t\backslash A^t)^t$;

# Operații în sens matriceal și în sens tablou

```
>>A = [1 2; 3 4]; B = ones(2); X = A \ B
X =
    -1    -1
     1     1
>> A = [1 4]; B = [1 2; 3 4]; X = A / B
X =
     4    -1
```

- dacă o matrice este împărțită la dreapta cu un scalar, operația se realizează element cu element:

```
>> A=[1 2; 3 4]; A / 2
ans =
    0.5000    1.0000
    1.5000    2.0000
```

- împărțirea în sens tablou sau pe elemente, se specifică plasând un punct înaintea operatorului de împărțire la dreapta "/" sau la stânga "\";

# Operații în sens matriceal și în sens tablou

- astfel, dacă $A = [a_{ij}]$ și $B = [b_{ij}]$ sunt matrice de aceeași dimensiune, atunci comanda

  ```
  >>C = A ./ B
  ```

  construiește matricea $C = [c_{ij}]$ având aceeași dimensiune și componentele $c_{ij} = a_{ij}/b_{ij}$;
- comanda

  ```
  >>C = A .\ B
  ```

  construiește matricea $C = [c_{ij}]$ având aceeași dimensiune și componentele $c_{ij} = b_{ij}/a_{ij}$;

  ```
  >> A = [1 2; 3 4]; B = [5 6; 7 8];
  >> A ./ B, A .\ B

  ans =
      0.2000    0.3333
      0.4286    0.5000
  ```

```
ans =
    5.0000    3.0000
    2.3333    2.0000
```

**Exercițiu**

*Ce se va afișa în urma execuției comenzii*

```
>>B ./ A, B .\ A
```

*cu matricele $A$ și $B$ ca în exemplul de mai sus?*

- oricare dintre cele două matrice $A$ sau $B$ poate fi de tipul $1 \times 1$;

- în cazul ridicării la putere ca și operație în sens matriceal, dacă $A$ este o matrice pătratică și $p$ este un întreg cu $p > 1$, atunci prin comanda `B=A^p` se construiește matricea $B$ prin înmulțirea repetată (de $p$ ori) a matricei $A$ cu ea însăși;
- dacă $p < 0$ este un întreg, atunci `A^p` este definit prin `inv(A)^(-p)`;
- pentru alte valori ale lui $p$, `A^p` este evaluată utilizând valorile proprii ale lui $A$; rezultatul poate fi incorect sau imprecis dacă A nu este diagonalizabilă sau atunci când $A$ este prost condiționată din punct de vedere al valorilor proprii;
- dacă $a$ este un scalar și $A$ este o matrice pătratică, atunci `a^A` este calculată folosind valorile proprii ale lui $A$;
- dacă $A$ și $B$ sunt matrice, atunci evaluarea `A^B` va genera o eroare;
- ridicarea la putere ca și operație în sens tablou se specifică plasând un punct înaintea operatorului de ridicare la putere și efectuează ridicarea la putere element cu element;

# Operații în sens matriceal și în sens tablou

```
>> A = [1 2; 3 4]; A .^ 2
ans =
     1     4
     9    16
```

- operația de ridicare la putere în sens tablou permite ca exponentul să fie un tablou când dimensiunile bazei și ale exponentului coincid, sau când baza este un scalar:

```
>> A = [1 2; 3 4]; B = [3 4; 5 6]; A .^ B, 2 .^ A
ans =
           1          16
         243        4096
ans =
     2     4
     8    16
```

# Operații în sens matriceal și în sens tablou

```
>> x = [1 2 3 4]; x .^ 3, 3 .^ x
ans =
     1     8    27    64
ans =
     3     9    27    81
```

- transpusa conjugată a matricei complexe $A$ se obține folosind comanda `A'`;
- dacă $A$ este o matrice reală, atunci `A'` calculează transpusa obișnuită;
- transpusa fără conjugare se obține utilizând comanda `A.'` (a se vedea și funcțiile `ctranspose(A)` și `transpose(A)`);

```
>> A = [1+2*i 1; 2 3-3*i]; A', A.'
ans =
   1.0000 - 2.0000i   2.0000
   1.0000             3.0000 + 3.0000i
ans =
   1.0000 + 2.0000i   2.0000
   1.0000             3.0000 - 3.0000i
```

- considerăm cazul particular al vectorilor coloană $x$ și $y$ de aceeași dimensiune; produsul scalar al acestora poate fi calculat folosind funcția `dot` sau folosind scrierea `x'*y`;
- produsul vectorial al doi vectori de dimensiune 3 se poate obține folosind funcția `cross`:

```
>> x = [1; 2; 3]; y = [-3; 4; -5]; x' * y
ans =
   -10
>> dot(x,y)
ans =
   -10
>>cross(x,y)
ans =
   -22
    -4
    10
```

# Operații în sens matriceal și în sens tablou

- inversa unei matrice pătratice nesingulare se obține folosind funcția `inv`, iar determinantul unei matrice pătratice cu funcția `det`;
- rangul unei matrice poate fi calculat cu funcția `rank`;

```
>> A = [1 2; 3 4]; invA = inv(A), detA = det(A)
invA =
   -2.0000    1.0000
    1.5000   -0.5000

detA =
    -2
>> I2 = A * invA
I2 =
    1.0000         0
    0.0000    1.0000
```

```
>> A = [0 0 2; 3 0 0; 9 0 0]; detA = det(A), rA = rank(A)
detA =
     0
rA =
     2

>> inv(A)
Warning: Matrix is singular to working precision.
```

# Operatori relaționali și logici, funcții logice

- operatorii relaționali în **MATLAB** sunt: $==$ (egal), $\tilde{}=$ (diferit), $<$ (mai mic), $>$ (mai mare), $<=$ (mai mic sau egal) și $>=$ (mai mare sau egal);
- un singur egal, $=$, desemnează o atribuire;
- rezultatul unei comparații între scalari este 1 dacă relația este adevărată și 0, în caz contrar;
- pentru a putea compara două matrice, acestea trebuie să aibă aceeași dimensiun, dar este posibilă și o comparație între o matrice și un scalar;
- în ambele cazuri, rezultatul este o matrice de 0 și 1 de aceeași dimensiune cu operandul (operanzii) de tip matrice;
- la comparația matrice–matrice se compară perechile corespunzătoare de elemente, iar la comparația matrice–scalar se compară scalarul cu fiecare element al matricei:

  ```
  >>A = [1 2; 3 4], B = eye(2), A == B
  ```

```
ans =
1     0
0     0
>>A <= 3
ans =
1     1
1     0
```

- exemple de funcții logice oferite de **MATLAB**: `ischar` – testează dacă argumentul este șir de caractere (string), `isempty` – testează dacă argumentul este vid, `isequal` – testează dacă tablourile sunt identice, `isfinite` – testează dacă elementele unui tablou sunt finite, `isinf` – testează dacă elementele unui tablou sunt Inf, `islogical` – testează dacă argumentul este un tablou logic, `isprime` – testează dacă argumentul este număr prim, `isnumeric` – testează dacă argumentul este numeric, `isreal` – testează dacă argumentul este tablou real;

```
>>isequal(A, B), isnumeric(A)
```

- în **MATLAB** operatorii logici sunt: & (și), | (sau), ˜(not), `xor` (sau exclusiv), la care se adaugă funcțiile `all` (aplicată unui vector, întoarce 1 dacă toate elementele vectorului sunt nenule) și `any` (aplicată unui vector, întoarce 1 dacă cel puțin un element al vectorului este nenul);
- precedența operatorilor:

| Nivel de precedență | Operator |
|---|---|
| 1 (cea mai mare) | `.'`, `.^`, `'`, `^` |
| 2 | + (unar), - (unar), ˜ |
| 3 | .*, ./, .\, *, /, \ |
| 4 | + (binar), - (binar) |
| 5 | : |
| 6 | <, <=, >, >=, ==, ˜= |
| 7 | & |
| 8 (cea mai mică) | \| |

# Operatori relaționali și logici, funcții logice

- **MATLAB** evaluează operatorii de precedență egală de la stânga la dreapta; precedența se poate modifica cu ajutorul parantezelor;
- pentru o matrice, `all` operează pe coloane, returnând un vector linie ce conține rezultatul aplicării lui `all` fiecărei coloane; această funcție poate fi făcută să opereze pe linii, specificând argumentul 2, după numele matricei;
- funcția `any` lucrează similar;

```
>>x = [0 1 2]; y = [-1 0 1]; x <= 0 & y+1 >= 0
ans =
1     0     0
>> xor(x, y)
ans =
1     1     0
>> all(y)
ans =
```

# Operatori relaționali și logici, funcții logice

```
0
>> any(y)
ans =
1
>> A = [1 0 3; 4 5 6]; all(A)
ans =
1     0     1
>> all(A, 2)
ans =
0
1
>> B = [1 -2; 3 4; -5 6]; any(B, 2)
ans = [1; 1; 1]
```

# Operatori relaționali și logici, funcții logice

- funcția logică `find`, aplicată unui vector, returnează indicii elementelor nenule ale vectorului:

```
>>y = [-1 0 1]; find(y)
ans =
1      3
```

- de asemenea, poate determina indicii elementelor unui vector care îndeplinesc o anumită condiție logică; de exemplu, următoarea secvență

```
>>y = 1:100, x = find(isprime(y))
```

  determină primele 25 de numerele prime;

- atunci când funcția `find` este aplicată unei matrice, vectorul de indici returnat corespunde matricei privită ca un vector coloană obținut din așezarea succesivă a coloanelor, începând cu prima; de exemplu, următoarea secvență

```
>>A = [-1 0 2; 3 -4 -5; 9 -1 -3]; A(find(A<0)) = 0
```

  setează elementele negative ale matricei $A$ la valoarea 0.

# Funcții pentru analiza datelor

- funcțiile de bază pentru analiza datelor sunt: `max` – calculează maximul, `min` – calculează minimul, `mean` – media, `median` – mediana, `std` – deviația standard, `var` – dispersia, `sort` – sortare, `sum` – suma elementelor, `prod` – produsul elementelor, `cumsum` – suma cumulată (daca $x$ este un vector cu $n$ elemente, $x = (x_i)_{i=\overline{1,n}}$, atunci `cumsum(x)` va genera un vector tot cu $n$ elemente, in care componenta $i$ este de forma $\sum_{j=1}^{i} x_j$, $i = \overline{1,n}$), `cumprod` – produsul cumulat, `diff` – diferența elementelor;
- acestea pot fi aplicate vectorilor linie sau coloană sau matricelor, caz în care funcțiile acționează pe coloană;

```
>>x = randn(1, 100);
>>min(x), max(x)
>>sort(x), -sort(-x)
>>sum(x), prod(x)
>>cumsum(x), cumprod(x)
```

# Funcții pentru analiza datelor

- funcția `diff`, aplicată unui vector $x$ de lungime $n$, va produce vectorul de lungime $n-1$ cu elementele $x(2)-x(1)$, $x(3)-x(2)$, ..., $x(n)-x(n-1)$:

```
>>diff(x)
```

- pentru o matrice, funcția `max` returnează un vector ce conține elementul maxim al fiecărei coloane, iar funcția `min` returnează un vector ce conține elementul minim al fiecărei coloane:

```
>>A = [1 -3 3 4; 0 -3 4 8; -7 9 5 8]
A =
     1    -3     3     4
     0    -3     4     8
    -7     9     5     8
```

# Funcții pentru analiza datelor

```
>>Mc = max(A), mc = min(A)
Mc =
1     9     5     8
mc =
-7    -3     3     4
```

- pentru a determina elementul maxim, respectiv minim, al matricei $A$, se aplică funcția `max`, respectiv `min`, de două ori succesiv:

```
>>M=max(max(A)), m=min(min(A))
M =
9
m =
-7
```

sau prin comenzile

```
>>M=max(A(:)), m=min(A(:))
```

# Funcții pentru analiza datelor

- funcțiile `max` și `min` pot returna un al doilea argument care specifică indicele elementului maxim, respectiv minim;
- dacă există mai multe elemente maxime, respectiv minime într-o coloană, se returnează indicele primului element maxim, respectiv minim:

```
>> [Mc, i]=max(A), [mc, i]=min(A)
Mc =
1     9     5     8
i =
1     3     3     2
mc =
-7    -3     3     4
i =
3     1     1     1
```

- funcțiile `max(X,Y)` sau `min(X,Y)` returnează un tablou de elemente ce conține elementele maxime, respectiv minime preluate din $X$ și $Y$;
- cele două argumente trebuie să aibă aceeași dimensiune, cu excepția cazului când unul dintre argumente este un scalar:

```
>>max(A, 0)
ans =
1       0       3       4
0       0       4       8
0       9       5       8
>>B = -1 * ones(3, 4); min(A, B)
ans =
-1      -3      -1      -1
-1      -3      -1      -1
-7      -1      -1      -1
```

- funcțiile `min(X,[],dim)` și `max(X,[],dim)` operează de-a lungul dimensiunii `dim` (implicit `dim=1`, corespunzător faptului că funcțiile operează pe coloană);
- pentru a obține maximul și minimul pe linii într-o matrice, vom scrie

```
>>min(A, [], 2), max(A, [], 2)
ans =
-3
-3
-7
ans =
4
8
9
```

# Funcții pentru analiza datelor

- sintaxa generală a funcției `sort` este

  `sort(X,dim,'mod')`,

  unde *X* reprezintă tabloul de sortat, *mod* poate fi *ascend* pentru sortare crescătoare și *descend* pentru sortare descrescătoare, iar *dim* reprezintă dimensiunea asupra căreia operează funcția;
- implicit, această funcție sortează crescător elementele fiecărei coloane a unei matrice (implicit, *dim*=1, *mod*=ascend), dar poate fi făcută să acționeze pe linii, setând argumentul *dim* la valoarea 2;
- funcția `sum`, aplicată unei matrice, returnează un vector ce conține sumele elementelor fiecărei coloane, dar poate să acționeze și asupra liniilor;

```
>>sort(A), sort(A, 'descend'), sort(A, 2)
>>sum(A), sum(A, 2)
```

- pentru executarea unui program scris într-un limbaj oarecare, există, în principiu, două abordări: compilare sau interpretare;
- prin compilare, compilatorul transformă programul sursă în totalitatea sa într-un program echivalent scris în limbaj mașină, care apoi este executat;
- prin interpretare, interpretorul selectează prima instrucțiune din programul sursă, o transformă în limbaj mașină și o execută; apoi trece la instrucțiunea a doua și repetă aceleași acțiuni ș.a.m.d.;
- **MATLAB** este un limbaj de programare de nivel înalt ce poate opera atât în regim de linie de comandă, prin execuția imediată a comenzii introduse de utilizator, cât și de interpretor, prin execuția unui program sursă;
- deoarece **MATLAB** citește, interpretează și execută linie cu linie programele, viteza de de execuție poate să scadă, dar această abordare este foarte convenabilă pentru detectarea erorilor;
- în limbaje de programare compilate (de exemplu, C/C++), viteza de execuție poate fi mai mare, dar erorile pot fi detectate uneori mai greu;

# Fișiere M

- fișierele **M** din **MATLAB** sunt echivalentele programelor, funcțiilor, subrutinelor și procedurilor din alte limbaje de programare;
- un fișier **M** este un fișier text cu extensia **.m** ce conține comenzi **MATLAB**;
- sunt de două tipuri:
  - fișiere **M** de tip **script** (fișiere sursă **MATLAB** ); acestea sunt fișiere care nu au nici un argument de intrare sau ieșire, operează asupra variabilelor din spațiul de lucru și permit memorarea unei secvențe de comenzi care este utilizată apoi în mod repetat sau care va fi necesară ulterior;
  - fișiere **M** de tip **funcție**; acestea sunt fișiere ce conțin o linie de definiție care începe cu cuvântul cheie **function** și care pot accepta parametri de intrare și returna parametri de ieșire. Variabilele interne ale unei funcții sunt locale (cu excepția cazului când sunt declarate folosind cuvântul cheie **global**).

# Fișiere M

- pentru a crea un fișier **M** nou: din tab-ul **HOME**, pentru fișierele script, se alege opțiunea **New Script** sau opțiunea **New** $\rightarrow$ **Script**, iar pentru fișierele funcție, se alege opțiunea **New** $\rightarrow$ **Function**;
- în același scop, se poate folosi comanda (în fereastra de comandă):

```
>>edit
```

- se deschide fereastra editorului de text al **MATLAB** -ului, în care se va introduce corpul programului sau al funcției;
- salvarea fișierului se realizează alegând opțiunea **Save** $\rightarrow$ **Save** sau **Save** $\rightarrow$ **Save as**, de pe tab-ul **EDITOR**;
- pentru a deschide un fișier **M** pentru vizualizare sau editare, din fereastra principală a aplicației, tab-ul **HOME**, se alege opțiunea **Open** sau se tastează în fereastra de comandă:

```
>>open nume_fisier
sau
>>edit nume_fisier
```

# Fișiere M

- construim un fișier script pentru a realiza diverse operații pe vectori:

```
% PrimulScript.m
% Operatii pe vectori
% Utilizarea functiei MATLAB norm, pentru a calcula
% norma unui vector:
%   norm(v, p) = sum(abs(v).^p)^(1/p)
%   norm(v) = norm(v,2)
%   norm(v, inf) = max(abs(v))
%   norm(v, -inf) = min(abs(v))

n = 10;
% generam doi vector de dimensiune 4n+1
v = -n : .5 : n;
w = 0 : .5 : 2*n;
```

```
% inmultirea in sens tablou, element cu element
y = v .* w

% produsul scalar
ps1 = v * w'
ps2 = sum(y)

% norme ale lui v
n = sqrt(sum(v.^2))
n1 = norm(v, 1)
n2 = norm(v)
ninf = norm(v, inf)
ninfm = norm(v, -inf)
```

- salvăm fișierul script cu numele *PrimulScript*; extensia **.m** se va adăuga automat;

# Fișiere M

- primele opt linii ale acestui fișier script încep cu simbolul % și sunt linii de comentariu; ori de câte ori **MATLAB** întâlnește un simbolul % va ignora restul liniei;
- sunt admise blocurile de comentarii (comentarii care se întind pe mai multe linii); ele sunt delimitate prin caracterele %{ și %}, ce trebuie să fie singure pe linie, ca în exemplul:
  %{
  Comentariu bloc
  pe doua linii
  %}
- pentru a pune în execuție fișierul script:
  - tastăm numele său în fereastra de comandă, la prompterul **MATLAB** (directorul curent listat pe bara de instrumente a ferestrei principale a **MATLAB** -ului trebuie să coincidă cu directorul în care a fost salvat fișierul);

# Fișiere M

- opțiunea **Run** de pe bara de instrumente **EDITOR** (se confirmă schimbarea directorului curent, dacă este cazul);

```
>> PrimulScript
```

- acesta este executat, iar **MATLAB** -ul afișează rezultatele în fereastra de comandă;
- prin execuția comenzii

```
>> help PrimulScript
```

toate liniile, de la prima linie de comentariu până la prima linie care nu este de comentariu sunt afișate pe ecran; deci, aceste linii ar trebui să conțină o scurtă descriere a scriptului;

# Fișiere **M**

- sintaxa generală prin care se definește o funcție de către utilizator într-un fișier **M** de tip funcție este:

```
function [param_de_iesire]=nume_functie(param_de_intrare)
%scurta descriere a functiei
corpul functiei
end
```

- parametrii de intrare sunt parametri formali și sunt separați prin virgule;
- dacă o funcție nu are parametri de intrare, atunci parantezele () lipsesc;
- dacă o funcție nu are parametri de ieșire, atunci parantezele [ ] și "=" lipsesc;
- dacă o funcție are doar un parametru de ieșire, atunci parantezele [ ] pot lipsi;
- numele fișierului funcție trebuie să fie același cu numele funcției, la care se adaugă extensia **.m**;

# Fișiere M

```
function z = F(x,y)
% F.m
% F(x,y) = x^2 + y^2
% argumentele functiei pot fi scalari sau tablouri

z = x.^2 + y.^2;
end
```

- o comandă de tipul

  ```
  >>help F
  ```

  va afișa comentariile de la începutul funcției;
- funcția astfel definită va putea fi apelată la fel ca orice funcție predefinită **MATLAB**, în fereastra de comandă a aplicației, într-un script sau într-o altă funcție;

# Fișiere M

```
>> F(2,3)
ans =
    13
>> F([1 2 3],[3 4 5])
ans =
    10     20     34
>> F([1 -2 3.5],5)
ans =
   26.0000   29.0000   37.2500
>> F([1 2; 3 4; 5 6],[1 0; -1 2; 4 -5])
ans =
     2      4
    10     20
    41     61
```

# Funcții anonime

- un **handle** pentru funcție ("**function handle**") este un tip de date **MATLAB** care stochează o asociere cu acea funcție (cu sau fără nume);
- prin crearea unui handle pentru o funcție, aceasta poate fi apelată indirect, putând fi transmisă ca argument unei alte funcții;
- o funcție anonimă (o funcție fără nume) este o funcție care este definită inline în loc să fie memorată într-un fișier funcție;
- poate fi construit un handle pentru o funcție definită de utilizator într-un fișier funcție, pentru o funcție predefinită **MATLAB** sau pentru o funcție anonimă;
- pentru a crea un handle pentru o funcție definită de utilizator într-un fișier funcție sau pentru o funcție predefinită, numele funcției va fi prcedat de caracterul `@`;

# Funcții anonime

- de exemplu, pentru a crea un handle pentru funcția definită anterior **F** sau pentru funcția **MATLAB** perdefinită **exp**, vom proceda astfel:

```
>> f = @F;
>> g = @exp;
```

- putem apela funcțiile **F** și **exp** prin intermediul lui **f**, respectiv **g**, în mod uzual:

```
>> f(2,3)
>> g(2)
```

- pentru a crea un handle pentru o funcție anonimă, se va descrie lista argumentelor, separate prin virgule, precedată de caracterul @ și succedată de expresia ce definește corpului funcției:

```
f = @(param) expresie_funcție_anonimă
```

# Funcții anonime

- de exemplu,

```
>> f = @(x,y) x.^2 + y.^2
f =
  function_handle with value:
    @(x,y)x.^2+y.^2
>> f(2,3)
ans =
    13
>> f([1 2; 3 4; 5 6],[1 0; -1 2; 4 -5])
ans =
     2     4
    10    20
    41    61
```

# Argumente de tip funcție

- există situații în care este necesar ca o funcție predefinită **MATLAB** sau una definită de utilizator să fie transmisă ca argument unei alte funcții;
- vom folosi în acest scop variabile de tip **function handle**;
- implementăm funcția `fderiv` care va aproxima derivata funcției $f$ (dată ca prim argument al funcției `fderiv`) în punctul $x$ cu ajutorul diferenței divizate

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

```
function y = fderiv(f,x,h)
% fderiv.m
% Aproximarea derivatei lui f in punctul x
% folosind diferenta divizata [x, x+h] a lui f;
% valoare implicita a lui h este sqrt(eps);

if nargin == 2
```

```
    h = sqrt(eps);
end
y = (f(x + h) - f(x)) / h;
end
```

- funcția `nargin` returnează numărul argumentelor de intrare cu care a fost apelată funcția;
- se pot atribui valori implicite argumentelor nespecificate la apel;
- numărul argumentelor de ieșire este returnat de funcția `nargout`;
- definim funcția `sqr` care calculează pătratul argumentului sau:

```
function out = sqr(x)
% sqr.m
% compute x^2

out = x.^2;
end
```

# Argumente de tip funcție

- apelăm funcția `fderiv` pentru a aproxima derivata unei funcții predefinite **MATLAB** , o funcție definită de utilizator într-un fișier funcție și o funcție anonimă, într-un punct dat ca al doilea argument:

```
>> fderiv(@exp, 1.2)
ans =
    3.3201
>> fderiv(@sqr, 1, 0.001)
ans =
    2.0010
>> f = @(x) x./(1 + x.^2);
>> fderiv(f, 2.4)
ans =
   -0.1042
```

# Variabile globale

- variabilele declarate în interiorul unei funcții sunt locale spațiului de lucru propriu acelei funcții;
- pentru ca o variabilă să existe în mai multe spații de lucru, eventual chiar în cel principal, se poate folosi instrucțiunea `global`;
- în corpul unei funcții, o variabilă globală trebuie declarată ca atare la începutul funcției, înaintea primei sale utilizări;
- verificarea faptului că o variabilă $X$ este globală se face prin apelarea funcției `isglobal(X)`, care returneaza 1, daca $X$ este de tip global și 0, altfel;

```
function y = g(x)
% g.m
% g(x) = x^2 + T

global T
y = x.^2 + T;
end
```

# Variabile globale

```
% exemplu.m
clear;
global T
T = 3;
x = 1 : 10;
g(x)
% sfarsitul scriptului exemplu

>> exemplu

ans =
     4     7    12    19    28    39    52    67    84   103
```

# Instrucțiuni de **I/O**

- comanda `input` permite utilizatorului să introducă de la tastatură o anumită dată de intrare ce va fi atribuită unei variabile;
- următoarea secvență

  ```
  >> R = input('Raza = ');
  ```

  afișează după prompter stringul "Raza = " și apoi așteaptă un input de la tastatură;
- data de intrare poate fi orice expresie **MATLAB**, care va fi evaluată utilizând variabilele din spațiul de lucru curent, iar rezultatul va fi întors în $R$;
- în cazul în care utilizatorul apasă tasta ENTER, fără a tasta nimic, `input` returnează o matrice vidă;
- următoarea secvență

  ```
  >> Nume = input('Numele: ','s');
  ```

afișează stringul "Numele: " și așteaptă un input șir de caractere care va fi asignat variabilei *Nume*;

- pentru a afișa valoarea unei variabile, putem folosi comanda reprezentată de numele variabilei;
- în același scop putem utiliza instrucțiunea `disp`, afișarea putând avea un aspect mai elegant (folosind această instrucțiune, numele variabilei nu mai este afișat);

```
>> R = input('Raza = ');
Raza = 4
>> R
R =
     4
>> disp(R)
     4
```

- scriind într-un script următoarele linii de program:

```
% Salut.m
Nume = input('Numele: ','s');
s = ['Salut ', Nume, '!'];
disp(s)
```

și punându-l în execuție obținem:

```
>> Salut
Numele: XYZ
Salut XYZ!
```

- pentru a plasa o valoare numerică într-un tablou de șiruri de caractere, trebuie să folosim funcția `num2str`;
- scriem un script care calculează aria unui cerc de rază $R$:

# Instrucțiuni de I/O

```
% Aria_cerc.m
R = input('Raza: ');
a = pi * R^2;
x = ['Aria= ', num2str(a)];
disp(x)
```

- sau afișăm direct,

```
disp(['Aria= ', num2str(a)]);
```

# Instrucțiunea `if`

- **MATLAB** are patru structuri de control: instrucțiunea `if`, instrucțiunile de ciclare `for` și `while` și instrucțiunea `switch`;
- forma generală a instrucțiunii `if` este

```
if expresie1
    instructiuni1
elseif expresie2
    instructiuni2
else
    instructiuni3
end
```

unde secvența de `instrucţiuni1` este executată dacă `expresie1` este evaluată la valoarea 1; dacă nu, se evaluează `expresie2` – dacă are valoarea 1, se execută `instrucţiuni2`, iar dacă nu se execută `instrucţiuni3`;

# Instrucțiunea `if`

- blocurile `elseif` și `else` sunt opționale;
- cea mai simplă formă a instrucțiunii `if` este

```
if expresie
    instructiuni
end
```

- secvența următoare generează un număr aleator $x$ din mulțimea $\{1, 2, \ldots, 10\}$ și, dacă numărul generat este par, atunci variabilei $y$ i se atribuie valoarea $x/2$:

```
clear;
x = fix(10 * rand) + 1 %fix trunchiaza
if mod(x, 2) == 0
    y = x / 2
end
```

- două variante ale deciziei se implementează cu `else`, ca în exemplul

```
clear;
x = fix(10 * rand) + 1
if mod(x, 2) == 0
    y = x / 2
else
    y = 0
end
```

- pentru teste suplimentare folosim `elseif`, ca în exemplul

# Instrucțiunea if

```
clear;
x = fix(10 * rand) + 1
if mod(x, 2) == 0
    y = x / 2
elseif isprime(x)
    y = x
elseif x^2 <= 25
    y = x^2
else
    y = 0
end
```

# Instrucțiunea `for`

- sintaxa generală a instrucțiunii `for` este

```
for variabila = expresie
    instructiuni
end
```

- de obicei, `expresie` este un vector de forma $i : h : j$, iar secvența de `instrucţiuni` este executată pentru `variabila` egală cu fiecare element al `expresiei` în parte;
- un alt mod de a defini `expresie` este utilizarea construcției un vector folosind parantezele pătrate;
- ciclurile `for` pot fi imbricate;

- codul următor construiește o matrice pătratică de ordin 10, cu elementele de pe diagonala principală egale cu 2, iar elementele de sub și de deasupra diagonalei principale egale cu $-1$;

```
clear;
N = 10;
for i = 1 : N
    for j = 1 : N
        if i == j
            A(i,j) = 2;
        elseif abs(i-j) == 1
            A(i,j) = -1;
        else
            A(i,j) = 0;
        end
    end
end
A
```

# Instrucțiunea `while`

- forma generală a instrucțiunii `while` este

```
while expresie
    instructiuni
end
```

  unde secvența de `instrucţiuni` se execută atât timp cât `expresie` este adevărată;

- în exemplul următor aproximăm epsilon-ul mașinii (cel mai mare număr $u$ cu proprietatea că $1 + u$ nu poate fi distins de $1$), care nu este altceva decât valoarea variabilei interne **MATLAB** *eps*:

```
clear;
u = 1;
while 1 + u > 1
    ueps = u;
    u = u / 2;
end
ueps
```

# Instrucțiunea `switch`

- instrucțiunea `switch` are următoarea sintaxă generală:

```
switch expresie_switch
    case expresie_case1
        instructiuni
    case {expresie_case2, expresie_case3,...}
        instructiuni
    ...
    otherwise
        instructiuni
end
```

- dacă `expresie_switch` se potrivește cu `expresie_case1`, atunci se execută blocul de instrucțiuni de după primul `case`; altfel, se testează `expresie_case` a următorului `case`, până se gasește o potrivire;
- dacă o `expresie_case` este formată din mai multe expresii, atunci dacă măcar una dintre acestea se potrivește cu `expresie_switch`, se va executa blocul de instrucțiuni de după `case`-ul corespunzător;

# Instrucțiunea switch

- dacă nicio expresie_case nu se potrivește cu expresie_switch, atunci se va executa blocul de instrucțiuni de după otherwise (dacă există);
- doar un singur case sau otherwise este executat, execuția programului continuând cu instrucțiunea de după end;
- instrucțiunea switch din **MATLAB** se comportă diferit de cea din C sau C++: odată ce **MATLAB** a selectat un case și blocul său de instrucțiuni a fost executat, se dă controlul primei instrucțiuni de după switch, fără a fi nevoie de instrucțiunea break;
- expresie_switch poate fi un scalar sau un string;
- în exemplul ce urmează, calculăm și afișăm norma $p$ a vectorului $v$, pentru 4 valori posibile ale lui $p$:

```
clear;
n = input('n= ');
v = -n : .5 : n;
p = input('Introduceti p (1, 2, inf, -inf): ');
```

# Instrucțiunea switch

```
switch p
  case 1
    norma = norm(v, 1);
    disp(['Norma 1 a vectorului este: ', num2str(norma)]);
  case 2
    norma = norm(v);
    disp(['Norma 2 a vectorului este: ', num2str(norma)]);
  case inf
    norma = norm(v, inf);
    disp(['Norma inf a vectorului este: ', num2str(norma)]);
  case -inf
    norma = norm(v, -inf);
    disp(['Norma -inf a vectorului este: ', num2str(norma)]);
  otherwise
    disp('Ati introdus o valoare diferita de cea ceruta!');
end
```

# Instrucțiunile break și continue

- instrucțiunea `break` termină execuția unei bucle `for` sau `while`;
- într-un ciclu imbricat un `break` iese în ciclul de pe nivelul anterior;
- instrucțiunea `break` nu este definită în afara unei bucle `for` sau `while`; în afara acestor instrucțiuni se poate folosi instrucțiunea `return`;

```
for i = 1 : 10
    if i > 5,  break; end
    disp(i)
end
```

- într-o buclă `for` sau `while`, instrucțiunea `continue` forțează trecerea controlului la execuția următoarei iterații, sărind instrucțiunile rămase din buclă;

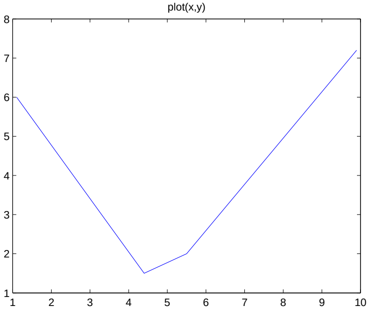
```
for i = 1 : 10
    if i <= 5, continue; end
    disp(i)
end
```

- **MATLAB** oferă facilități grafice puternice: se pot genera grafice și figuri relativ ușor, iar atributele acestora se pot modifica la fel de ușor folosind utilitarul **Plot Editor** (`help plotedit`) sau folosind meniurile și bara de instrumente din ferestrele figurilor;
- cele mai utilizate funcții **MATLAB** pentru grafice 2D sunt: `plot` – grafic x-y simplu, `loglog` – grafic cu scară logaritmică pe ambele axe, `semilogx` – grafic cu scară logaritmică pe axa x, `semilogy` – grafic cu scară logaritmică pe axa y, `plotyy` – grafic x-y cu axe y și la stânga și la dreapta, `polar` – grafic polar, `fplot` – reprezentare grafică a unei funcții, `fimplicit` – reprezentare grafică unei funcții implicite, `ezpolar` – versiune ușor de utilizat (easy-to-use) a lui polar, `fill` – umplere poligon, `area` – grafic de tip arie plină, `bar` – grafic de tip bară, `barh` – grafic de tip bară orizontală, `hist` – histogramă, `pie` – grafic cu sectoare de cerc, `comet` – grafic x-y animat, `errorbar` – grafic cu bare de eroare, `quiver` – câmp de vectori bidimensional, `scatter` – grafic dispersat (nor de puncte).

- funcția **MATLAB** `plot` este o funcție grafică de bază ce realizează grafice bidimensionale simple prin unirea punctelor vecine;
- dacă $x = (x_1, x_2, \cdots, x_n)$ și $y = (y_1, y_2, \cdots, y_n)$ sunt doi vectori de dimensiune $n$, atunci printr-o instrucțiune de tipul `plot(x,y)` se reprezintă grafic linia poligonală ce trece prin punctele de coordonate $(x_i, y_i)$, $i = \overline{1, n}$;
- **MATLAB** deschide o fereastră pentru figură în care desenează imaginea;
- spre exemplu, următoarea secvență

```
>> x = [1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9];
>> y = [6.0 4.5 3.0 1.5 2.0 3.3 4.6 5.9 7.2];
>> plot(x,y)
```

produce următorul grafic:




- o comandă de tipul `plot(x)` reprezintă grafic linia poligonală ce trece prin punctele de coordonate $(i, x_i)$, $i = \overline{1, n}$;
- în exemplul precedent se utilizează valorile implicite ale unor facilități cum ar fi domeniul pentru axele x și y, spațiile dintre diviziunile de pe axe, culoarea și tipul liniei de desen;

- dacă dorim să precizăm valori diferite de cele implicite pentru culoare, marcaj și stilul de linie al desenului, în loc de `plot(x,y)`, putem utiliza `plot(x,y,'s')`, unde `s` este un șir de caractere prin care vom controla toate aceste aspecte;
- culoarea poate fi: `r` (roșu), `g` (verde), `b` (albastru), `c` (cian), `m` (magenta), `y` (galben), `k` (negru), `w` (alb);
- tipul de marcaj poate fi: `o` (cerc), `*` (asterisc), `.` (punct), `+` (plus), `x` (ori), `s` (pătrat), `d` (romb), `^` (triunghi în sus), `v` (triunghi în jos), $>$ (triunghi dreapta), $<$ (triunghi stânga), `p` (pentagramă (stea cu 5 colțuri)), `h` (hexagramă (stea cu 6 colțuri));
- stilul liniei de desen poate fi: `-` (linie continuă – stilul implicit), `- -` (linie întreruptă), `:` (linie punctată), `-.` (linie – punct);
- cele trei elemente care pot fi plasate în șirul de caractere `s` pot fi date în orice ordine;

- de exemplu:

```
>> plot(x,y,'g*-.')
>> plot(x,y,'ro')
>> plot(x,y,'k')
```

- în graficul din primul exemplu, se va plasa un asterisc verde în fiecare punct $(x_i, y_i)$ , punctele fiind unite cu o linie verde de tip linie – punct;
- în cel de-al doilea exemplu, se marchează fiecare punct de coordonate $(x_i, y_i)$ cu un cerc roșu, fără a le uni cu vreo linie;
- în cel de-al treilea exemplu, singura schimbare față de setările implicite este culoarea, aceasta fiind setată la negru;
- instrucțiunea `plot` poate reprezenta grafic, pe aceeași figură, mai multe seturi de date; secvența

```
>> v = x+1; w = y-1;
>> plot(x,y,'k-.',v,w,'r:')
```

desenează în aceeași figură graficele pentru punctele $(x_i, y_i)$ și $(v_i, w_i)$, primul cu linie de tip linie – punct, neagră, iar al doilea cu linie punctată, roșie;

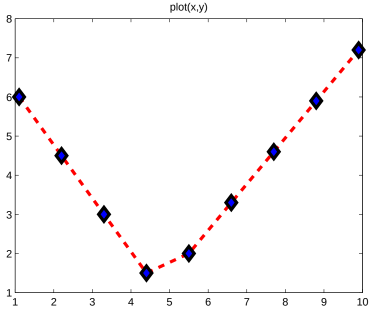
- comanda `plot` acceptă și argumente de tip matrice;
- dacă $x$ este un vector de dimensiune $m$ și $A$ este o matrice $m \times n$, atunci instrucțiunea `plot(x,A)` suprapune graficele obținute din $x$ și fiecare coloană a matricei $A$;
- dacă $A$ și $B$ sunt două matrice de aceeași dimensiune, atunci `plot(A,B)` suprapune graficele obținute din coloanele corespunzătoare ale lui $A$ și $B$;
- dacă argumentele lui `plot` nu sunt reale, atunci părțile imaginare sunt, în general, ignorate, excepție făcând cazul când `plot` este apelat cu un singur argument; dacă $A$ este complex, atunci `plot(A)` este echivalent cu `plot(real(A),imag(A))`;

- atributele unei figuri se pot controla furnizând argumente suplimentare instrucțiunii `plot`;
- proprietățile `LineWidth` (grosimea liniei de desen – implicit este de 0.5 puncte) și `MarkerSize` (dimensiunea marcajului – implicit este de 6 puncte) pot fi specificate în puncte (un punct $= 1/72$ inch);
- culoarea conturului și a interiorului marcajului se poate seta cu proprietățile `MarkerEdgeColor` și `MarkerFaceColor`;
- de exemplu, comanda

```
>> plot(x,y,'rd--','LineWidth',3,'MarkerEdgeColor','k',...
'MarkerFaceColor','b','MarkerSize',10)
```

  produce un grafic cu linie întreruptă, roșie, cu grosimea 3 puncte, marcaje în formă de romb cu dimensiunea 10 puncte, conturul romburilor fiind negre, iar interiorul albastru;

# Grafică 2D în **MATLAB**



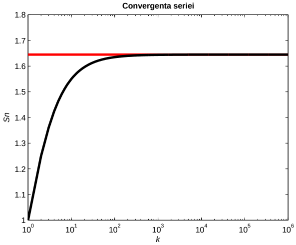

- o altă funcție **MATLAB** pentru grafică este `loglog`, care, spre deosebire de `plot`, scalează ambele axe logaritmic utilizând logaritmul în baza 10;
- funcțiile `semilogx` și `semilogy`, scalează doar una dintre axe;

- în exemplul care urmează am verificat grafic că suma seriei $\sum_{k=1}^{\infty} \frac{1}{k^2}$ este $\frac{\pi^2}{6}$.

```
% serie.m
clear;
n = input('n= ');
k = 1 : n;
sn = cumsum(1./k.^2);
semilogx([1 n],[pi^2/6 pi^2/6],'r',k,sn,'k','LineWidth',3)
err = abs(sn(n)-pi^2/6);
disp(['Err= ', num2str(err)]);
title('\bf{Convergenta seriei}')
xlabel('\it{k}')
ylabel('\it{Sn}')
```

# Grafică 2D în **MATLAB**

- puneți în execuție scriptul `serie` pentru $n = 10, 100, 1000, 10000, 100000, 1000000$. Ce puteți spune despre eroarea de aproximare a sumei seriei?
- pentru $n = 1000000$ obținem următoarea figură:

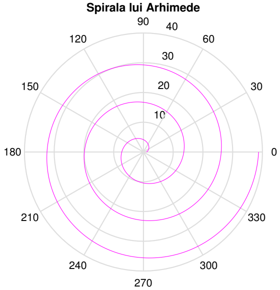



- funcțiile **MATLAB** `title`, `xlabel` și `ylabel` afișează șirul de caractere primit ca argument deasupra imaginii, ca titlu al graficului, sub axa $x$ și respectiv la stânga axei $y$ (etichetarea axelor);

- se pot reprezenta curbe în coordonate polare cu ajutorul comenzii `polar(t,r)`, unde `t` este unghiul polar, iar `r` este raza polară;
- se poate folosi și un parametru suplimentar s, șir de caractere, cu aceeași semnificație ca la `plot`;
- graficul unei curbe în coordonate polare, numită spirala lui Arhimede, de ecuație $r(t) = a + bt, t \in [0, 6\pi]$, cu $a$ și $b$ numere reale, se obține cu secvența:

```
%arh.m
%spirala lui Arhimede
t = 0:pi/1000:6*pi;
a = input('a= ');
b = input('b= ');
r = a + b*t;
polar(t,r);
title('Spirala lui Arhimede')
```

și are forma




- funcția `fill(x,y,'c')` reprezintă grafic poligonul cu vârfurile de coordonate $(x_i, y_i)$ și îl hașurează în culoarea `c`;
- punctele se consideră în ordine, iar ultimul punct se unește cu primul;

- culoarea c se poate da și sub forma unui triplet `rgb` (nu va mai fi plasat între apostrofuri), în forma [r g b] – 3 numere sub forma roșu, verde, albastru, nuanța fiecărei culori fiind reprezentată de un număr între 0 și 1;

  ```
  >> fill(x,y,'r')
  >> fill(x,y,[0.23 0.45 0.97])
  >> fill(x,y,rand(1,3))
  ```

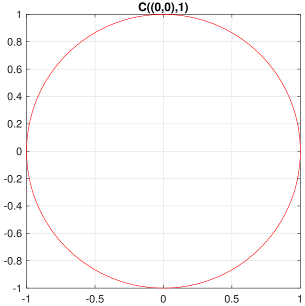
  unde `x` și `y` sunt vectori de aceeași lungime;
- **MATLAB** setează automat axele pentru o reprezentare grafică, în funcție de datele care urmează a fi reprezentate;
- comanda `axis([xmin xmax ymin ymax])` setează limitele pentru axa `x` și respectiv `y` la valorile date în vectorul transmis funcției, ca argument (pentru a reveni la setările implicite, se utilizează comanda `axis auto`);

- dacă se dorește ca una dintre limite să fie aleasă automat de către **MATLAB** , aceasta va fi setată la -inf sau inf (de exemplu, `axis([-1,1,-inf,0])`;
- limitele pe axa `x` sau `y` se pot seta individual folosind comenzile `xlim([xmin xmax])` și `ylim([ymin ymax])`;
- axele pot fi eliminate utilizând comanda `axis off`;
- raportul dintre unitatea pe `x` și cea pe `y` (aspect ratio) poate fi făcut egal cu unu, cu `axis equal`;
- comanda `axis square` face caseta axelor pătrată, iar `axis tight` setează limitele axelor egale cu limitele;
- în exemplul ce urmează vom reperezenta grafic un cerc de rază $R$ și centru $(x_0, y_0)$:

```
%cerc.m
clear all;
R = input('Raza= ');
disp('Centrul cercului: ');
x0 = input('x0= ');
y0 = input('y0= ');
theta = 0:pi/1000:2*pi;
x = x0 + R*cos(theta);
y = y0 + R*sin(theta);
plot(x,y,'r'), axis equal
axis([x0-R x0+R y0-R y0+R]), grid on;
title(['C((',num2str(x0),',',num2str(y0),'),',...
num2str(R),')'])
```

- comanda `grid on` produce o grilă de linii orizontale și verticale care pornesc din diviziunile axelor;

- se obține figura:

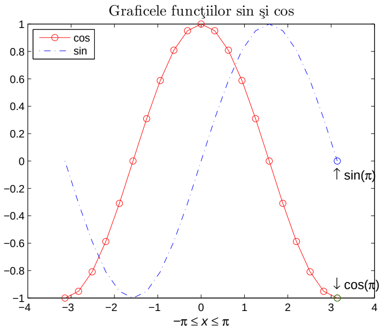



- comanda `legend('s1','s2',...,'sn','Location','p')` atașează unui grafic o legendă care pune stringul 'si' după informația reprezentată de culoare/marcaj/stil pentru graficul corespunzător;

- parametrii opționali `Location` și `p` indică poziția legendei (vezi `help legend`);
- pentru a introduce text într-un grafic se poate folosi comanda `text(x,y,'s')`, unde `x` și `y` sunt coordonatele punctului unde va apărea textul, iar s este un șir de caractere;
- **MATLAB** permite introducerea într-un șir de caractere construcții TEX, de exemplu _ pentru indice, ^ pentru exponent, sau litere grecești ( `\alpha`, `\beta`, `\gamma`, etc.); de asemenea, pot fi setate anumite atribute ale textului, cum ar fi tipul fontului, dimensiunea ș.a.;
- aceste facilități se pot utiliza și în titluri, legende sau etichete ale axelor;
- începând cu **MATLAB** 7, poate fi folosită proprietatea `Interpreter` cu valorile `TeX`, `LaTeX` sau `none`;

```
%graf.m
clear;
x = -pi:pi/10:pi;
plot(x,cos(x),'-ro',x,sin(x),'-.b',pi,0,'o',pi,-1,'o')
text(pi-0.1,-0.1,'\uparrow sin(\pi)','FontSize',12)
text(pi-0.1,-0.9,'\downarrow cos(\pi)','FontSize',12)
xlabel('-\pi \leq x \leq \pi','FontSize',12,...
    'FontAngle','italic');
title('Graficele func\c tiilor sin \c si cos',...
    'FontSize',14,'Interpreter','LaTeX')
legend('cos','sin','Location','NorthWest');
```

- rezultatul este:




- multe dintre atributele unei figuri pot fi modificate interactiv, după afișarea figurii, utilizând meniul **Tool** al ferestrei figurii sau bara de instrumente;

- dacă o comandă grafică este urmată de o alta, atunci noua imagine o va înlocui pe cea veche sau se va suprapune peste ea, acest lucru depinzând de starea `hold` curentă;
- comanda `hold on` face ca toate imaginile care urmează să se suprapună peste cea curentă, în timp ce `hold off`, care este starea implicită, va face ca fiecare imagine nouă să o înlocuiască pe cea precedentă;

```
%graf1.m
clear;
x = -pi:pi/10:pi;
plot(x,cos(x),'-ro');
hold on
plot(x,sin(x),'-.b');
plot(pi,0,'o');
plot(pi,-1,'o');
hold off
%...continuarea din graf.m
```

- comanda `clf` șterge figura curentă, iar comanda `close` o închide;
- este posibil să avem mai multe ferestre figuri pe ecran; cel mai simplu mod de a crea o nouă figură este prin comanda `figure`;
- a n-a fereastră figură poate fi făcută figura curentă folosind comanda `figure(n)`;
- comanda `close all` va închide toate ferestrele figuri;

```
%graf2.m
close all;
x = -pi:pi/10:pi;
figure;
plot(x,cos(x),'-ro');
xlabel('-\pi \leq x \leq \pi','FontSize',12,...
    'FontAngle','italic');
title('Graficul func\c tiei cos',...
    'FontSize',14,'FontAngle','italic',...
```

```
     'Interpreter','LaTeX')
figure;
plot(x,sin(x),'-.b');
xlabel('-\pi \leq x \leq \pi','FontSize',12,...
     'FontAngle','italic');
title('Graficul func\c tiei sin',...
     'FontSize',14,'FontAngle','italic',...
     'Interpreter','LaTeX')
```

- funcția **MATLAB** `subplot` permite plasarea mai multor imagini în aceeași figură;
- utilizând comanda `subplot(m,n,p)` fereastra figurii este împărțită în $m \times n$ regiuni, fiecare având propriile ei axe; comanda de desenare curentă se va aplica celei de-a p-a dintre aceste regiuni, regiunile fiind numerotate pe linii;

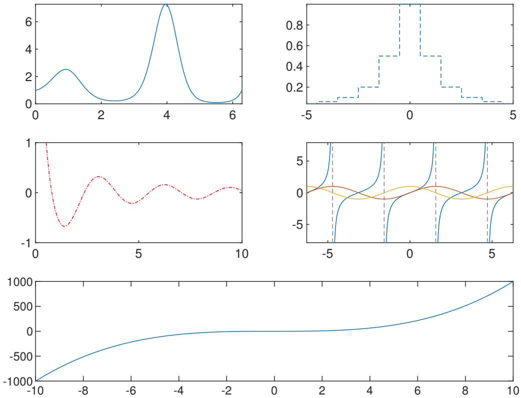
- sintaxa generală a funcției **MATLAB** `fplot` este

  `fplot(fun,lims,'LineSpec')`

  unde: `fun` este funcția de reprezentat grafic, `lims` setează limitele pe axa x (implicit, `lims = [-5,5]`), iar prin `LineSpec` se indică specificațiile pentru linia de desen;

```
%graf3.m
subplot(3,2,1)
fplot(@(x)exp(sqrt(x).*sin(2*x)),[0 2*pi])
subplot(3,2,2)
fplot(@(x)1./(1+round(x).^2),'--')
subplot(3,2,3)
fplot(@(x)cos(2*x)./x,[0.01 10],'r-.'), ylim([-1 1])
subplot(3,2,4)
fplot(@(x)[tan(x),sin(x),cos(x)], 2*pi*[-1 1])
subplot(3,2,5:6)
```

```
f = @(x,n)x.^n;
fplot(@(x)f(x,3),[-10 10])
```



# Grafică 2D în MATLAB

- `fplot(x,y,lims)` reprezintă grafic o curbă parametrizată, unde `x`, `y` sunt function handle;

```
x = @(t) cos(3*t);
y = @(t) sin(4*t);
fplot(x,y,[-pi pi])
```

reprezintă grafic curba dată prin ecuațiile parametrice:

$$x(t) = \cos(3t)$$
$$y(t) = \sin(4t), \quad t \in [-\pi, \pi].$$

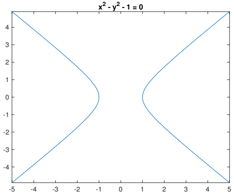
- pentru a reprezenta grafic o funcție implicită definită prin $f(x,y) = 0$, se folosește funcția `fimplicit`, cu sintaxa

  `fimplicit(f,lims,'LineSpec')`

  unde `lims` poate fi `[xmin xmax ymin ymax]` sau `[xymin xymax]` cu `xymin <= x, y <= xymax` (implicit, `lims = [-5,5]`), iar `LineSpec` sunt specificațiile pentru linia de desen;

  ```
  >> fimplicit(@(x,y) x.^2 - y.^2 - 1)
  ```

  reprezintă grafic curba dată prin ecuația implicită $x^2 - y^2 - 1 = 0$, cu $x, y \in [-5, 5]$;

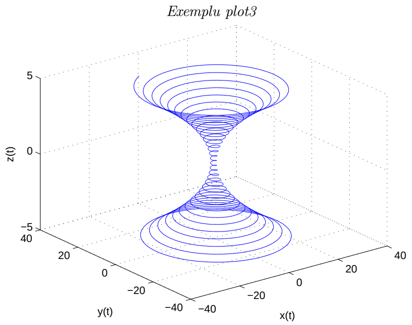


Cele mai folosite funcții pentru grafice 3D sunt: `plot3` – grafic simplu x-y-z, `fplot3` – graficul unei curbe dată parametric, `contour` și `fcontour` – contur, `contourf` – contur plin, `contour3` – contur 3D, `mesh` și `fmesh` – reprezentare wire-frame, `meshc` – reprezentare wire-frame plus contururi, `meshz` – suprafață wire-frame cu cortină, `surf` și `fsurf` – suprafață plină, `surfc` – suprafață plină plus contururi, `waterfall` – wire-frame unidirecțional, `fimplicit3` – suprafață implicită, `bar3` – bare 3D, `bar3h` – bare 3D orizontale, `pie3` – grafice sector 3D, `fill3` – poligon plin 3D, `comet3` – grafic 3D animat, `scatter3` – nor de puncte 3D.

- funcția plot3 este un analog tridimensional al lui plot;
- prin execuția următorului script se va reprezenta grafic curba 3D dată prin ecuațiile parametrice: $x(t) = (1+t^2)sin(20t)$, $y(t) = (1+t^2)cos(20t)$, $z(t) = t, t \in [-5, 5]$;

```
%graf4.m
t = -5:0.005:5;
x = (1+t.^2).*sin(20*t);
y = (1+t.^2).*cos(20*t);
z = t;
plot3(x,y,z)
grid on
xlabel('x(t)'), ylabel('y(t)'), zlabel('z(t)')
title('\textit{Exemplu plot3}','FontSize',14,...
'Interpreter','LaTeX')
```

# Grafică 3D în **MATLAB**




- funcțiile `xlabel`, `ylabel` și `title` au același efect ca în cazul lui `plot`;
- prin utilizarea funcției `zlabel` se poate eticheta axa `z`;
- culoarea, marcajul și stilul de linie pentru `plot3` se controlează la fel ca pentru `plot`;

- limitele de axe în 3D se determină automat, dar acestea pot fi schimbate cu instrucțiunea `axis([xmin xmax ymin ymax zmin zmax])`; pe lângă `xlim` și `ylim`, există și `zlim`, prin care se pot schimba limitele pe axa `z`;
- aceeași figură poate fi obținută prin:

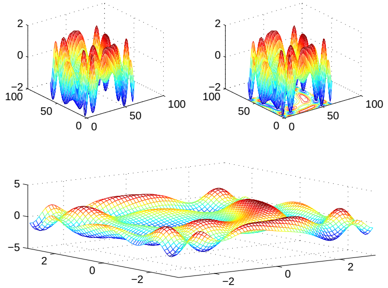
```
fplot3(@(t)(1+t.^2).*sin(20*t),@(t)(1+t.^2).*cos(20*t),...
       @(t)t)
```

- vom reprezenta grafic suprafața definită prin $f(x, y) = sin(x + y^2) - cos(x^2 - y)$, cu $-\pi \leq x, y \leq \pi$ ;
- funcția `meshgrid(x,y)` este foarte utilă în pregătirea datelor pentru multe funcții **MATLAB** de grafică 3D;
- astfel, se obțin matricele `X` și `Y` a.î. fiecare linie a lui `X` să fie o copie a vectorului `x` și fiecare coloană a lui `Y` să fie o copie a vectorului `y`;

- matricea `Z` este apoi generată prin operații în sens tablou a lui `X` și `Y`; `Z(i,j)` memorează valoarea funcției corespunzând lui `x(j)` și `y(i)` (aceasta este forma cerută de funcția `mesh`);
- funcția `mesh` produce o reprezentare de suprafață de tip cadru de sârmă (wire-frame);
- funcția `meshc` adaugă un contur sub suprafață;
- un prim grafic este generat cu `mesh(Z)`; deoarece nu se dă nicio informație pentru abscisă și ordonată, `mesh` utilizează în locul lor indicii de linie și de coloană;
- un al doilea grafic este realizat cu `meshc(Z)`;
- pentru cel de-al treilea, s-a utilizat `mesh(x,y,Z)`, și deci gradațiile de pe axele `x` și `y` corespund valorilor `x` și `y`;
- limitele pe axe s-au specificat cu funcția `axis`;

```
%graf5.m
x = -pi:0.1:pi; y = -pi:0.1:pi;
[X,Y] = meshgrid(x,y);
Z = sin(X + Y.^2) - cos(X.^2 -Y);
subplot(2,2,1)
mesh(Z)
subplot(2,2,2)
meshc(Z)
subplot(2,2,3:4)
mesh(x,y,Z)
axis([-pi pi -pi pi -5 5])
```

# Grafică 3D în **MATLAB**




- în locul funcțiilor `mesh` și `meshc`, putem folosi funcția`fmesh`:

  ```
  fmesh(@(x,y) sin(x+y.^2)-cos(x.^2-y),[-pi pi -pi pi]);
  ```

  respectiv,

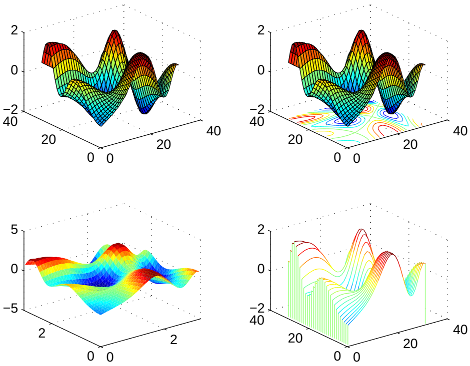
  ```
  fmesh(@(x,y) sin(x+y.^2)-cos(x.^2-y),[-pi pi -pi pi],...
  'ShowContours','on');
  ```

- funcția `surf` produce grafice cu celulele umplute (colorate) ale unor suprafațe, iar `surfc` adaugă contururi dedesubt;
- în exemplul următor am folosit aceeși suprafață ca la exemplul anterior, iar domeniul este dat de $0 \leq x, y \leq \pi$;

```
%graf8.m
x = 0:0.1:pi; y = 0:0.1:pi;
[X,Y] = meshgrid(x,y);
Z = sin(X + Y.^2) - cos(X.^2 -Y);
subplot(2,2,1), surf(Z)
subplot(2,2,2), surfc(Z)
subplot(2,2,3), surf(x,y,Z), shading flat
axis([0 pi 0 pi -5 5])
subplot(2,2,4), waterfall(Z)
```

- funcția `waterfall` este similară funcției `mesh`, dar fără cadrul de sârmă pe direcția coloanelor;

# Grafică 3D în **MATLAB**




- similar funcției `fmesh`, există și funcția `fsurf`, cu același mod de utilizare;
- funcțiile `mesh`, `surf`, `fmesh` și `fsurf` sunt utilizate și pentru reprezentarea suprafețelor date prin ecuații parametrice;

- reprezentăm grafic suprafața dată prin ecuațiile parametrice:

$$\begin{aligned} x(u,v) &= R \sin u \cos v \\ y(u,v) &= R \sin u \sin v \\ z(u,v) &= R \cos u, \quad R > 0, \;\; u, v \in [0, 2\pi] \end{aligned}$$

```
clear;
close all;
R = 1;
u = 0:0.1:2*pi;
v = 0:0.1:2*pi;
[U V] = meshgrid(u,v);
x = R*sin(U).*cos(V);
y = R*sin(U).*sin(V);
z = R*cos(U);
figure;
```

```
mesh(x,y,z)
%surf(x,y,z)
figure;
f1 = @(u,v)R*sin(u).*cos(v);
f2 = @(u,v)R*sin(u).*sin(v);
f3 = @(u,v)R*cos(u);
fmesh(f1,f2,f3,[0 2*pi 0 2*pi])
%fsurf(f1,f2,f3,[0 2*pi 0 2*pi])
```

- funcția `fimplicit3(f,lims,'LineSpec')` reprezintă grafic o suprafață dată prin ecuația implicită $f(x, y, z) = 0$, pentru `lims = [xmin xmax ymin ymax zmin zmax]` sau `lims = [xyzmin xyzmax]`, unde `xyzmin <= x,y,z <= xyzmax`;

```
fimplicit3(@(x,y,z) x.^2+y.^2+z.^2 - 9, [-3 3])
```

- graficele 3D realizate până acum utilizează unghiurile de vizualizare implicite ale **MATLAB** ; acestea pot fi modificate cu funcția `view`;
- apelul `view(a,b)` setează unghiul de rotație (în sens invers acelor de ceasornic) în jurul axei `z`, de `a` grade și unghiul față de planul xOy, de `b` grade (implicit este view(-37.5,30));
- instrumentul **rotate 3D** de pe bara de instrumente a ferestrei figurii permite utilizarea mouse-ului pentru schimbarea unghiurilor de vizualizare;
- este posibil să vedem un grafic 2D ca pe unul 3D, utilizând comanda `view` pentru a seta unghiurile de vizualizare, sau mai simplu utilizând view(3);
- toate funcțiile grafice interpretează valorile NaN ca "date lipsă" și nu sunt reprezentate;

# Salvarea și imprimarea graficelor

- comanda `print` permite listarea unui grafic la imprimantă sau salvarea lui pe disc într-un format grafic sau sub formă de fișier **M**;
- sintaxa comenzii `print` este:

```
print -periferice -optiuni numefisier
```

- opțiunile pot fi vizualizate cu `help print`;
- dintre tipurile de periferice admise amintim: `dps` – Postscript pentru imprimante alb-negru, `dpsc` – Postscript pentru imprimante color, `deps` – Encapsulated PostScript pentru imprimante alb-negru, `depsc` – Encapsulated PostScript pentru imprimante color, `djpeg - <nn>` – imagine JPEG la nivelul de calitate nn (implicit nn=75) ș.a. ( `help print`);
- comanda

```
print -deps2 figura.eps
```

creează un fișier Postscript încapsulat alb și negru, nivelul 2, numit figura.eps, care poate fi listat pe o imprimantă sau inclus într-un document;

- comanda `print` se poate utiliza și în formă funcțională:

```
clear;
x = linspace(-2*pi,2*pi);
for i=1:5
    subplot(1,5,i)
    plot(x,sin(i*x))
    print('-deps2',['figura',int2str(i),'.eps'])
end
```

- exemplul anterior generează o secvență de cinci figuri și le salvează în fișierele figura1.eps,..., figura5.eps.
- funcția `saveas` salvează o figură într-un fișier care apoi poate fi încărcat de către **MATLAB** ;

# Salvarea și imprimarea graficelor

```
saveas(gcf,numefigura,fig)
```

salvează figura curentă în format binar FIG, care poate fi încărcat în **MATLAB** folosind funcția

```
open('numefigura.fig')
```

- se pot salva și imprima figuri din meniul **File** al ferestrei figurii.

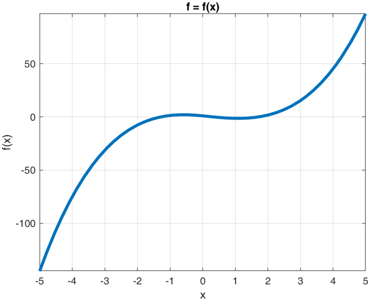
## Zerouri și puncte de minim pentru funcții de o variabilă reală

- pentru a aproxima soluțiile ecuației $f(x) = 0$, cu $f : \mathbb{R} \longrightarrow \mathbb{R}$ se poate folosi funcția **MATLAB** `fzero`;
- funcția $f$ trebuie definită în prealabil;
- forma generală de utilizare a funcției **MATLAB** `fzero` este

  `x = fzero(f,x0)`
- dacă `x0` este un vector cu două componente, atunci se presupune că acesta reprezintă un interval în care semnele lui $f(x0(1))$ și $f(x0(2))$ sunt diferite; în cazul în care acest lucru nu este adevărat apare o eroare;
- dacă `x0` este un scalar, atunci această valoare se utilizează ca punct de start; `fzero` caută un interval care conține o schimbare de semn pentru $f$ și care conține, de asemenea, punctul de start `x0`; dacă nu se găsește un astfel de interval, este returnată valoarea NaN;

- fie ecuația $f(x) = 0$, cu $f(x) = x^3 - x^2 - 3arctg(x) + 1$; aceasta are 3 zerouri, primul fiind pe intervalul $[-2, -1]$, așa cum putem identifica din reprezentarea sa grafică, semnele funcției în punctele $-2$ și $-1$ fiind diferite;

```
%reprezentarea grafica a functiei f = f(x)
fplot(@(x) x.^3 - x.^2 - 3*atan(x) + 1,...
 'LineWidth', 3), grid on
title('f = f(x)')
xlabel('x')
ylabel('y')
```




- mai întâi, definim funcția $f$, de exemplu într-un fișier funcție f.m:

```
function y = f(x)
y = x^3 - x^2 - 3*atan(x) + 1;
end
```

- apoi, primul zero se aproximează prin

```
>> fzero(@f,[-2,-1])
ans =
-1.2780
```

- dacă apelăm funcția `fzero` prin

  ```
  >> [x fval] = fzero(@f,[-2,-1])
  ```

  atunci, în `x` se returnează zeroul, iar `fval` reprezintă valoarea funcției `f` în punctul `x`, `f(x)`:

  ```
  x =
  -1.2780
  fval =
  4.4409e-16
  ```

- putem defini funcția $f$ ca funcție anonimă, accesând-o prin intermediul unui handle pentru aceasta:

```
>> h = @(x) x^3 - x^2 - 3*atan(x) + 1;
>> x = fzero(h,[-2,-1])
ans =
-1.2780
>> fzero(h,-1.5)
ans =
-1.2780
```

- pentru celelate două zerouri ale funcției $f$, procedăm în mod similar, examinând graficul funcției:

  ```
  >> fzero(h, 0.5), fzero(h, 1.5)
  ```

  sau

  ```
  >> fzero(h, [0,0.5]), fzero(h, [1,2])
  ```

- spre deosbire de funcția `fzero` care se apelează succesiv pentru a găsi fiecare soluție numerică a unei ecuației, funcția **MATLAB** `fsolve` permite determinarea tuturor soluțiilor, printr-un singur apel al acesteia;

- punctele de start ale soluțiilor trebuie furnizate ca argumente ale funcției:

  ```
  >> fsolve(h, [-1.5, 0.5, 1.5])
  ```

  sau

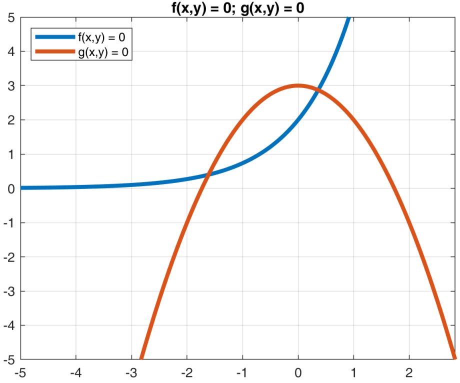
  ```
  >> [x fval] = fsolve(f, [-1.5, 0.5, 1.5])
  x =
     -1.2780    0.3204    1.7205
  fval =
     1.0e-08 *
     -0.0002         0    0.1438
  ```

- funcția `fsolve` poate fi utilizată și pentru rezolvarea numerică a sistemelor de ecuații neliniare;
- să rezolvăm numeric sistemul

$$\begin{cases} y = 2e^x \\ y = 3 - x^2, \; x, y \in \mathbb{R} \end{cases}$$

- punctele de start le detrminăm prin observarea graficelor celor două funcții;

```
% reprezentarea grafic\u a a curbelor din plan
% date prin ecua\c tiile implicite f(x,y) = 0, g(x,y) = 0
fimplicit(@(x,y) 2*exp(x)- y,'LineWidth',3)
hold on
fimplicit(@(x,y) 3-x.^2-y,'LineWidth',3)
hold off
title('f(x,y) = 0; g(x,y) = 0')
legend('f(x,y) = 0','g(x,y) = 0','Location','NorthWest')
grid on
```




- punctele de start vor fi aproximări ale punctelor de intersecție ale celor două grafice;

- observăm că sistemul poate fi rescris astfel:

$$\begin{cases} f(x_1, x_2) = 0 \\ g(x_1, x_2) = 0 \end{cases}$$

cu $x_1 = x,\ x_2 = y$ și $f(x_1, x_2) = 2e^{x_1} - x_2$, $g(x_1, x_2) = 3 - x_1^2 - x_2$;

- ecuațiile se definesc ca linii separate ale unui vector având atâtea elemente câte ecuații există în sistemul de rezolvat (într-un fișier funcție sau ca funcție anonimă);

```
>> h = @(x)[2*exp(x(1))-x(2); 3-x(1)^2 - x(2)];
>> start = [-1.5 0.3; 0.3 3];
>> fsolve(h, start(1,:)) %prima solutie
>> fsolve(h, start(2,:)) %a doua solutie
```

- dacă funcția $f$ este polinomială, atunci zerourile acesteia pot fi determinate folosind funcția **MATLAB** `roots`; sintaxa generală a acestei funcții este

  `roots(V)`

  unde `V` reprezintă vectorul ce conține coeficienții polinomului;
- dacă `V` are `N + 1` componente, atunci polinomul este $V(1) * X^N + ... + V(N) * X + V(N+1)$.
- să determinăm soluțiile ecuației $x^3 - 2x^2 + 3x - 1 = 0$;

```
>> V = [1 -2 3 -1];
>> roots(V)
ans =
0.7849 + 1.3071i
0.7849 - 1.3071i
0.4302
```

- pentru a calcula valoarea funcției polinomiale într-un punct se folosește funcția `polyval(V,P)`, unde `P` (nu este neapărat scalar, poate fi și vector) este punctul în care se calculează valoarea, iar `V` este vectorul coeficienților:

```
>> polyval(V,0.4302)
```

- pentru a aproxima punctele de minim pentru o funcție $f : \mathbb{R} \longrightarrow \mathbb{R}$ se poate folosi funcția **MATLAB** `fminbnd`;
- secvența următoare

```
x = fminbnd(f,x1,x2)
```

aproximează punctul de minim al lui $f$ pe intervalul $[x1, x2]$;

```
>> x = fminbnd(@cos,3,4)
x =
3.1416
>> x = fminbnd(@(x) x^2-2*x+1,0,2)
x =
1.0000
>> f1 = @(x) sin(x)-pi/2;
>> x = fminbnd(f1,3,5)
x = 4.7124
```

- considerăm o funcție de o variabilă reală, dar care depinde și de un parametru *a*; definim această funcție într-un fișier funcție fun.m

```
function out = fun(x,a)
out = (x - a)^2 + 2*a -1;
end
```

- pentru a aproxima punctul de minim, setăm mai întâi parametrul *a*:

```
>> a = 2;
>> x = fminbnd(@(x) fun(x,a),0,3)
x =
2.0000
```

- altfel, folosind o funcție anonimă

```
>> fun1 = @(x,a) (x - a)^2 + 2*a -1;
>> x = fminbnd(@(x) fun1(x,2),0,3)
```

- putem folosi varianta de apel

```
>> [x fval] = fminbnd(@(x) fun1(x,2),0,3)
x =
  2.0000
fval =
  3
```

dacă dorim să afișăm valoarea funcției obiectiv în punctul de minim returnat de `fminbnd`;

- pentru a aproxima punctul de maxim local al unei funcții reale $f$, vom apela funcția `fminbnd` pentru $-f$.

**Integrarea numerică**

- în **MATLAB** există mai multe funcții pentru calculul aproximativ al unei integrale definite;
- funcția `trapz(x,y)` aproximează integrala lui $y$ în raport cu $x$ folosind metoda trapezelor;
- să calculam $\displaystyle\int_0^{\pi} sin(x)dx$;
- în acest scop generăm doi vectori:

```
>> x = linspace(0,pi,50);
>> y = sin(x);
>> z = trapz(x,y)
z =
1.9993
```

- eroarea de aproximare este cu atât mai mică cu cât numărul de puncte considerate pe intervalul $[0, \pi]$ crește;

```
>> x = linspace(0,pi,200);
>> y = sin(x);
>> z = trapz(x,y)
z =
2.0000
```

- astfel, o aproximare bună necesită un număr mare de puncte pe intervalul de integrare, așadar un timp mare de execuție;
- din acest motiv, cele mai utilizate funcții **MATLAB** pentru calculul aproximativ al unei integrale definite sunt `quad` și `quadl`; pentru a putea fi folosite aceste funcții, intervalul de integrare $[a, b]$ trebuie să fie finit și integrandul să nu aibă nici o singularitate pe acest interval;
- forma uzuală de apel este

```
q = quad(f,a,b,tol)
```

(la fel pentru `quadl`), unde $f$ este funcția de integrat; $f$ trebuie să accepte argumente vectori și să returneze vectori; argumentul `tol` este eroarea absolută (implicit este de $10^{-6}$);

- funcția `quad` este o implementare a unei cuadraturi adaptive de tip Simpson, pe când `quadl` este mai precisă și se bazează pe o cuadratură de tip Gauss-Lobatto cu 4 puncte, cuadratura fiind adaptivă;
- ambele funcții returnează mesaje de avertisment dacă subintervalele devin prea mici sau dacă s-au făcut prea multe evaluări; astfel de mesaje indică posibile singularități;
- calculam $\displaystyle\int_0^2 \frac{1}{x^3 - 2x - 5} dx$:

```
>> f = @(x) 1./(x.^3-2*x-5);
>> q = quad(f,0,2)
q =
-0.4605
```

- calculăm integrala definită $\int_1^2 \left(x+1-\frac{1}{x}\right) e^{\left(x+\frac{1}{x}\right)} dx$:

```
>> f = @(x)(x+1-1./x).*exp(x+1./x)
>> quadl(f,1,2)
ans =
16.9759
```

- pentru a evalua numeric o integrală dublă pe un dreptunghi $[a, b] \times [c, d]$, avem la dispoziție funcția **MATLAB** `dblquad`; forma uzuală de apel este

```
dq = dblquad(f,a,b,c,d)
```

unde $f$ este funcția de integrat; aceasta este o funcție de două variabile $f = f(x, y)$, ce primește ca parametri de intrare un vector $x$ și un scalar $y$ și returnează un vector de valori ale integrandului;

- calculăm $\int_{-1}^1 \int_0^1 \sqrt{|y - x^2|} dxdy$:

```
>> fun = @(x,y) sqrt(abs(y-x.^2));
>> dblquad(fun,0,1,-1,1)
ans =
1.4380
```

- pentru a aproxima numeric o integrală triplă pe un domeniu $[a, b] \times [c, d] \times [e, f]$, avem la dispoziție funcția **MATLAB** `triplequad` cu sintaxa uzuală

  `tq = triplequad(f,a,b,c,d,e,f)`

  unde $f = f(x, y, z)$ este funcția de integrat (parametrii de intrare sunt: un vector $x$ și scalarii $y$ și $z$ și returnează un vector de valori ale lui $f$);
- calculăm $\int_0^1 \int_0^1 \int_0^1 \frac{xyz}{(1 + x^2 + y^2 + z^2)^4} dxdydz$

```
>> fun1 = @(x,y,z)(x*y*z)./((1 + x.^2 + y^2 + z^2).^4);
>> triplequad(fun1,0,1,0,1,0,1)
ans =
0.0052
```

- pentru alte tipuri de integrale (improprii, în coordonate polare etc.) sau pe domenii de alt tip decât cele specificate mai sus se pot utiliza funcțiile `integral`, `integral2` (pentru integrale duble), `integral3` (pentru integrale triple).

**Rezolvarea sistemelor algebrice de ecuații liniare**

- considerăm sistemul algebric de ecuații liniare $Ax = b$, cu $A$ matricea coeficienților și $b$ vectorul termenilor liberi;
- pentru a rezolva acest sistem se pot folosi operațiile matriceale (inv, *) cu observația că timpul de lucru poate fi destul de mare;
- instrumentul fundamental de rezolvare a sistemelor de ecuații liniare în **MATLAB** este operatorul de împărțire la stânga $\backslash$;

- în funcție de forma matricei sistemului, numărul de ecuații și numărul de necunoscute, operatorul \ se bazează pe algoritmi diferiți;
- dacă sistemul este pătratic, cu $A$ o matrice pătratică nesingulară de ordin $n$, atunci algoritmul folosit este cel al lui Gauss de eliminare;

```
>> x = A\b
```

- în cazul matricelor rău condiționate sau singulare va fi afișat un mesaj de atenționare;
- o altă metodă este utilizarea funcției **MATLAB** `rref`, care, aplicată unei matrice, o reduce la forma triunghiulară (cu operații gaussiene);
- în acest scop, se construiește matricea extinsă a sistemului, căreia i se aplică funcția `rref`;
- rezolvăm sistemul $\begin{cases} x_1 - 2x_2 + x_3 = 0 \\ 2x_1 + x_2 - x_3 = 1 \\ -3x_1 + x_2 + x_3 = 2 \end{cases}$

```
>> A = [1 -2 1; 2 1 -1; -3 1 1]; b = [0; 1; 2];
>> x = A\b
x =
1.0000
2.0000
3.0000
>> C = [A,b]
C =
 1     -2      1      0
 2      1     -1      1
-3      1      1      2
>> Crref = rref(C)
Crref =
1      0      0      1
0      1      0      2
```

0 0 1 3

- matricea obținută *Crref* ne spune dacă sistemul este compatibil determinat, compatibil nedeterminat sau incompatibil;
- dacă *Crref* are în partea stângă matricea identitate de aceeași dimensiune cu matricea sistemului $A$, atunci sistemul este compatibil determinat și soluția unică a acestuia este ultima coloană a matricei *Crref* (soluția sistemului din exemplul precedent este [1; 2; 3]);
- dacă *Crref* are ultima linie formată doar din 0, atunci sistemul este compatibil nedeterminat, iar dacă toate elementele de pe ultima linie a lui *Crref* sunt 0, cu excepția ultimului atunci sistemul este incompatibil;
- să rezolvăm sistemul $\begin{cases} x_1 + 2x_2 = 1 \\ 2x_1 + 4x_2 = 2 \end{cases}$

```
>> A = [1 2; 2 4]; b = [1; 2];
>> A\b
Warning: Matrix is singular to working precision.
ans =
NaN
NaN
>> C=[A,b]
>> Crref = rref(C)
Crref =
1      2      1
0      0      0
```

- deci sistemul este compatibil nedeterminat, cu $x_1 = 1 - 2x_2, x2 \in \mathbb{R}$.
- să rezolvăm sistemul $\begin{cases} x_1 + 2x_2 = 1 \\ 2x_1 + 4x_2 = 3 \end{cases}$

```
>> A = [1 2; 2 4]; b = [1; 3];
>> A\b
Warning: Matrix is singular to working precision.
ans =
-Inf
Inf
>> C=[A,b]
>> Crref = rref(C)
Crref =
1      2      0
0      0      1
```

- deci sistemul este incompatibil;
- o altă modalitate de rezolvare a unui sistem de ecuații liniare, este prin descompunerea LU a matricei sistemului;

- ideea descompunerii LU este de a găsi 2 matrice pătratice L și U, unde L – triunghiulară inferior, iar U – triunghiulară superior astfel încât $A = LU$;
- funcția **MATLAB** care realizează o astfel de descompunere este `lu`;
- aceasta acceptă la intrare doar matrice pătratice;
- după descompunere, sistemul revine la $L(Ux) = b$;
- astfel, notând $Ux = y$, rezolvarea sistemului $Ax = b$ prin comanda **MATLAB** $x = A\backslash b$, cu $A$ pătratică, este echivalentă cu secvența **MATLAB**

```
>> [L, U] = lu(A);
>> y = L\b;
>> x = U\y
```

**Rezolvarea numerică a problemelor cu valori inițiale pentru ecuații diferențiale ordinare**

- **MATLAB** dispune de facilități foarte puternice de rezolvare a problemelor cu valori inițiale (IVP) pentru ecuații diferențiale ordinare (ODE) (de ordinul I), de forma:

$$y'(t) = f(t, y(t)), \;\; y(t_0) = y_0,$$

  cu $y_0 \in \mathbb{R}$ și $f : D \longrightarrow \mathbb{R}$, $D = \{(t, y) \in \mathbb{R}^2;\ |t - t_0| \leq a,\ |y - y_0| \leq b\}$, $a, b > 0$;
- astfel, trebuie implementată o funcție care evaluează $f$ și apoi trebuie apelată una dintre rutinele de care dispune **MATLAB** pentru a integra IVP pentru ODE;
- această rutină trebuie să primească ca parametri funcția $f$, mulțimea valorilor lui $t$ pe care se determină soluția (intervalul de integrare) și valoarea inițială $y_0$;

- dintre cele mai folosite rutine **MATLAB** în acest sens sunt funcțiile `ode23` și `ode45`, care au la bază metoda Runge-Kutta de ordinul 2-3, respectiv 4-5, cu pas variabil:

  ```
  [t y] = rutina(fun, tspan, y0);
  ```

  unde `rutina` este ode23 sau ode45, iar `fun` este un handle pentru funcția $f$;
- argumentul de intrare `tspan` este un vector ce specifică intervalul de integrare; dacă acesta este un vector cu două elemente, `tspan=[t0,tf]`, rutina integrează de la `t0` la `tf`, iar dacă `tspan` are mai mult de două elemente rutina returnează soluțiile în acele puncte;
- rezultatele numerice obținute sunt returnate în doi vectori cu același număr de linii, $t$ – vectorul coloană al punctelor în care se calculează soluțiile și $y$ – vectorul soluțiilor;
- dacă `tspan` are 2 componente, lungimea tabloului $t$ nu este cunoscută a priori;

- să rezolvăm următoarea problemă: $\begin{cases} y'(t) = y - \dfrac{2t}{y}, & t > 0 \\ y(0) = 1 \end{cases}$
- mai întâi scriem o funcție care evaluează membrul drept al ODE în fișierul funcție rhs.m (sau ca funcție anonimă):

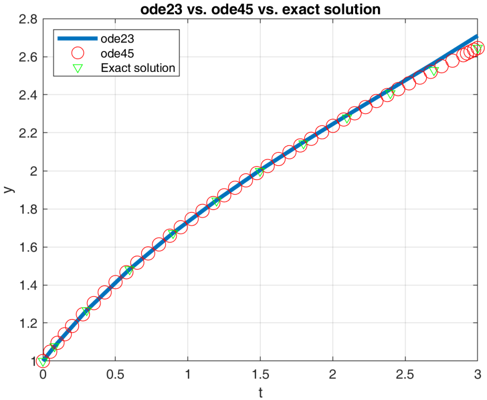
```
function out = rhs(t,y)
out = y - 2.0*t./y;
end
```

- urmează scriptul progr1.m, care compară rezultatele numerice obținute cu rutina ode23 versus ode45:

```
%progr1.m
clear
t0 = 0;
tf = input('tfinal: ');
y0 = 1;
[t y] = ode23(@rhs, [t0 tf], y0);
plot(t,y,'LineWidth',3); grid
title('ode23 vs. ode45 vs. exact solution')
xlabel('t');
ylabel('y');
[u z] = ode45(@rhs, [t0 tf], y0);
hold on
plot(u,z,'ro','MarkerSize',10)
hold off
```

- în acest caz, poate fi calculată soluția exactă a problemei; aceasta este $y(t) = \sqrt{2t + 1}$;
- completăm scriptul progr1.m cu următoarele linii:

```
hold on
w = sqrt(2.*t + 1);
plot(t,w,'gv')
hold off
legend('ode23','ode45','Exact solution','Location',...
'NorthWest');
hold off
```




- pentru a obține soluțiile în anumite puncte, vom da `tspan` astfel:

```
h = 0.01;
tspan = t0:h:tf;
```

- apelurile funcțiilor ode23 și ode45 se înlocuiesc cu

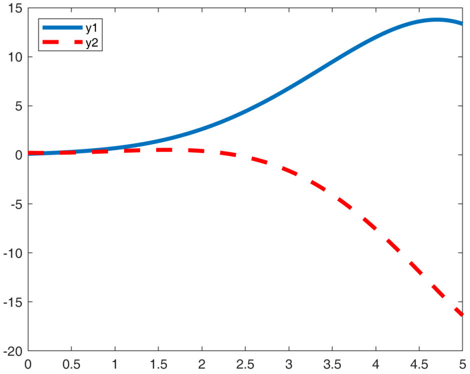
```
[t y] = ode23(@rhs, tspan, y0);
[u z] = ode45(@rhs, tspan, y0);
```

- să rezolvăm numeric sistemul de ecuații diferențiale:
$$\begin{cases} y_1'(x) = y_1(x) + y_2(x) \\ y_2'(x) = x - y_1(x), \ \ x > 0 \\ y_1(0) = 0.1, \ \ y_2(0) = 0.2 \end{cases}$$
- se consideră vectorul $y = [y_1, y_2]$; se definește vectorul expresiilor derivatelor într-un fișier funcție fsist.m (sau ca funcție anonimă):

```
function out = fsist(x,y)
out = [y(1)+y(2); x-y(1)];
end
```

- apoi se rezolvă sistemul de ecuații diferențiale prin execuția următorului script:

```
%progr2.m
clear;
xf = input('xf= ');
tspan = 0:0.01:xf;
[x y] = ode23(@fsist, tspan, [0.1 0.2]);
plot(x,y(:,1),x,y(:,2),'r--','LineWidth',3)
legend('y1','y2','Location','NorthWest')
```




- să se rezolve următoarea IVP pentru ODE de ordin II:

$$\begin{cases} y''(t) = -1.2y'(t) - y(t) + 10, \ \ t \in [0, T], \ \ T > 0 \\ y(0) = 2, \ \ y'(0) = 0 \end{cases}$$

- notăm cu $y_1 = y$ și $y_2 = y'$, iar problema devine:

$$\begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) = -1.2y_2(t) - y_1(t) + 10, \ \ t \in [0, T] \\ y_1(0) = 2, \ \ y_2(0) = 0 \end{cases}$$

și deci

$$\begin{cases} y_1'(t) = f(t, y_1(t), y_2(t)) \\ y_2'(t) = g(t, y_1(t), y_2(t)), \ \ t \in [0, T] \\ y_1(0) = 2, \ \ y_2(0) = 0 \end{cases}$$

cu

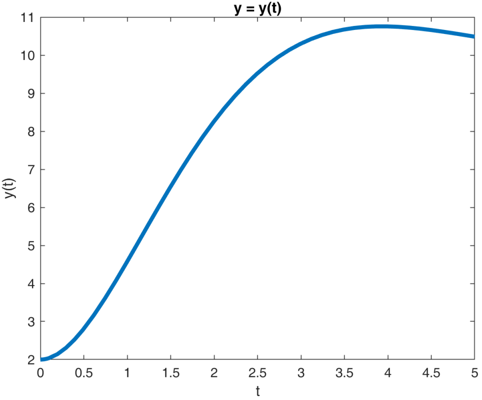
$$\begin{cases} f(t, y_1(t), y_2(t)) = y_2(t) \\ g(t, y_1(t), y_2(t)) = -1.2y_2(t) - y_1(t) + 10 \end{cases}$$

- problema revine, deci, la rezolvarea unui sistem de ODE de ordin 1.

```
%fun1.m
function out = fun1(t,y)
out=[y(2);-1.2*y(2)-y(1)+10];
end

%progr3.m
clear
T = input('T= ');
[t y] = ode45(@fun1, [0 T], [2 0]);
plot(t,y(:,1),'LineWidth',3)
xlabel('t')
ylabel('y(t)')
title('y = y(t)')
```

- $y$ returnat de funçtia **MATLAB** este un tablou cu 2 coloane: pe prima coloană se regăsesc valorile aproximative ale soluției problemei în punctele intermediare ale intervalului de integrare $[0\,T]$, iar pe cea de-a doua coloană sunt valorile derivatei în aceste puncte.



# Calcul simbolic cu Symbolic Math Toolbox

- Symbolic Math Toolbox încorporează facilități de calcul simbolic în mediul numeric al **MATLAB**;
- permite diferențierea, integrarea, simplificarea sau rezolvarea ecuațiilor din punct de vedere analitic;
- toolbox-ul Symbolic Math definește un nou tip de date **MATLAB** numit obiect simbolic sau `sym`;
- un obiect simbolic este o structură de date care memorează o reprezentare sub formă de șir a simbolului;
- toolbox-ul Symbolic Math utilizează obiectele simbolice: numere, variabile, matrice și expresii simbolice;
- aritmetica cu care se operează asupra obiectelor simbolice este exactă și implicit este cea rațională;
- obiectele simbolice se construiesc cu ajutorul funcției `sym`;

- instrucțiunea

  `>> x = sym('x')`

  produce o variabilă simbolică numită `x`;
- pentru a crea mai multe obiecte simbolice se folosește comanda `syms`:

  `>> syms x y z a b c`
- expresiile simbolice se definesc la fel ca expresiile aritmetice din **MATLAB**, variabilele numerice fiind înlocuite cu variabile simbolice;
- operațiile ce se pot realiza pe variabilele simbolice sunt aceleași cu cele ce se pot realiza pe variabilele numerice;
- de asemenea, se pot efectua calcule simbolice cu matrice având elemente expresii simbolice;
- funcțiile matematice uzuale (`sin`, `sqrt`, `exp`, `log` etc.) se pot utiliza în expresiile simbolice;

```
>> syms x t
>> f = cos(t*x) + t*exp(x)
```

- dacă o variabilă primește ca valoare o expresie simbolică atunci aceasta devine o variabilă simbolică:

```
>> f = x + y^a
```

- funcția `sym` se folosește în cazul în care vrem să transformăm o constantă într-un număr simbolic:

```
>> f = sym('1/2')
```

sau

```
>> f = sym(1/2)
```

- pentru a afișa expresiile simbolice într-un format apropiat de cel din matematică se utilizează funcția `pretty` ce are ca parametru expresia simbolică;

- pentru a defini o variabilă simbolică complexă, trebuie să definim mai întâi două variabile simbolice reale, corespunzând părții reale și celei imaginare a variabilei complexe, variabila *i* fiind predefinită;
- în acest caz se poate folosi funcția `sym` cu opțiunea `real` sau comanda `syms` cu aceeași opțiune:

```
>> x = sym('x', 'real');
>> y = sym('y', 'real');
>> z = x + i * y
```

sau

```
>> syms x y real
>> z = x + i * y
```

- în expresiile cu numere complexe se pot utiliza funcțiile matematice uzuale: `real`, `imag`, `conj` etc.

```
>> conj(z), imag(z)
```

- matricele simbolice au ca elemente obiecte simbolice: constante, variabile sau expresii simbolice;
- operațiile ce se pot realiza pe matrice simbolice sunt aceleași cu cele ce se pot realiza pe matrice cu elemente numerice;

```
>> clear
>> syms a b c d;
>> A = [a b; c d]
A =
 [ a, b]
 [ c, d]
>> det(A)
>> B = inv(A),
>> pretty(B) % pentru afisare in forma matematica a lui B
>> A^2
>> B.^2
```

```
>> A*B
>> simplify(A*B)
>> A.*B
```

- se pot defini funcții simbolice prin comenzi de tipul:

```
>> syms f(x,y)
>> f(x,y) = x + y
```

# Funcții pentru calcule simbolice

Toolbox-ul Symbolic Math deține funcții pentru:

- simplificarea expresiilor și substituire;
- calculul sumelor;
- dezvoltarea în serie Taylor;
- operații cu polinoame;
- algebra liniară;
- rezolvarea ecuațiilor algebrice;
- limite de funcții;
- derivare;
- integrare;
- rezolvarea ecuațiilor diferențiale ordinare.

- dacă dorim să atribuim anumite valori numerice variabilelor simbolice în scopul evaluării numerice a expresiei simbolice corespunzătoare, se folosește funcția `subs` cu forma generală

```
subs(expr_simbolica, {var_simbolice}, {valori_numerice})
```

- în cazul în care substituim o singură variabilă, atunci variabila simbolică și cea numerică nu mai trebuie plasate între acolade;
- de exemplu, considerăm funcția de două variabile $f(x,y) = ax^2 + by^2 - cxy$, cu $a, b, c$ parametri;

```
>> clear
>> syms x y a b c
>> f = a*x^2 + b*y^2 - c*x*y;
>> subs(f, x, 2)
ans=
b*y^2 - 2*c*y + 4*a
>> subs(f, {x,y}, {2,1})
```

```
ans=
4*a + b - 2*c
```

- în cazul în care substituim singura variabilă simbolică a unei expresii cu o valoare numerică, funcția `subs` poate avea următoarea formă mai simplă

```
subs(expr_simbolica, valoare_numerica)
```

- dacă expresia simbolică depinde de mai mult de o variabilă și variabila pentru care se face substituția nu este specificată, substituția se face pentru variabila simbolică implicită, care se alege după următoarea regulă: se alege din alfabet litera cea mai apropiată de $x$; dacă există două litere egal depărtate de $x$, se alege ultima din alfabet dintre cele două;
- dacă în exemplul de mai sus apelăm funcția `subs` astfel:

# Substituții ale variabilelor simbolice cu valori numerice

```
>> subs(f, 2)
ans=
 b*y^2 - 2*c*y + 4*a
```

atunci se substituie variabila simbolică $x$ cu valoarea 2, variabila simbolică fiind aleasă după regula de mai sus;

- pentru a determina variabilele simbolice dintr-o expresie se folosește funcția `symvar`:

```
>> symvar(f)
ans =
[ a, b, c, x, y]
```

- variabila simbolică implicită dintr-o expresie se determină astfel:

```
>> symvar(f,1)
ans=
x
```

- funcția `simplify` aplică diverse tipuri de identități pentru a aduce o expresie la o formă mai simplă;

```
>> clear
>> syms x a
>> f = (x^2 - a^2) / (x + a);
>> simplify(f)
ans=
x - a
```

```
>> clear
>> syms x y
>> f = cos(x)^2 + sin(x)^2;
>> simplify(f)
ans=
1
```

# Calculul sumelor

- pentru calculul simbolic al sumei $\sum\limits_{k=m}^{k=n} x_k$ se folosește funcția `symsum` cu formele

  `symsum(xk, k, m, n)`

  sau

  `symsum(xk, m, n)`

  unde *xk* este termenul general al sumei, *m* și *n* sunt limitele de sumare, care pot fi numere întregi sau $\infty$;
- în prima formă variabila simbolică după care se face sumarea este $k$, iar în a doua formă se folosește variabila simbolică implicită;
- să calculăm sumele

  a) $\sum\limits_{k=1}^{n} k$

```
>> clear
>> syms k n
>> s = symsum(k, 1, n)
s =
(n*(n + 1))/2
```

b) $\sum_{k=1}^{\infty} \frac{1}{k^2}$

```
>> ls = symsum(1/k^2, 1, inf)
ls =
pi^2/6
```

- considerăm o funcție $f(x)$ care are derivate până la ordinul $n$;
- funcția `taylor`, cu forma generală

  ```
  taylor(f, x, a, 'Order', n)
  ```

  oferă dezvoltarea în serie Taylor a funcției $f(x)$ până la ordinul $n-1$ în jurul punctului $a$;
- de exemplu, să calculăm dezvoltarea în serie Taylor a funcției $e^x$ până la ordinul 5 în jurul originii:

  ```
  >> clear
  >> syms x
  >> f = exp(x);
  >> T = taylor(f, x, 0, 'Order', 6)
  T =
  x^5/120 + x^4/24 + x^3/6 + x^2/2 + x + 1
  ```

# Polinoame

- în cele ce urmează se vor considera polinoame simbolice cu coeficienți numere raționale;
- fie polinomul $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$;
- acesta poate fi reprezentat într-una din următoarele forme: o combinație liniară a puterilor lui $x$ (ca mai sus), ca produs de factori ireductibili peste mulțimea numerelor raționale, sau sub forma Horner $a_0 + x(a_1 + x(a_2 + ... + x(a_{n-1} + x a_n)...))$;
- conversia între aceste trei forme de reprezentare a unui polinom se poate realiza cu următoarele funcții: `collect(p)` – transformă polinomul în prima formă, `factor(p)` – transformă polinomul în cea de-a doua formă (poate descompune în factori și numere întregi sau tablouri de întregi simbolici), `horner(p)` – transformă polinomul în forma a treia, argumentul $p$ fiind polinomul simbolic, dar putând fi și un vector sau o matrice cu elemente expresii simbolice;
- considerăm polinomul $x^4 - 10x^3 + 35x^2 - 50x + 24$;

```
>> clear
>> syms x
>> p = x^4 - 10*x^3 + 35*x^2 - 50*x + 24;
>> q = factor(p)
q =
[ x - 1, x - 2, x - 3, x - 4]
>> h = horner(p)
h =
x*(x*(x*(x - 10) + 35) - 50) + 24
>> s = collect(h)
s =
x^4 - 10*x^3 + 35*x^2 - 50*x + 24
```

- este posibilă conversia unui polinom cu coeficienți numerici într-un polinom simbolic;

# Polinoame

- coeficienții numerici sunt reținuți într-un vector $V$, în ordinea descrescătoare a puterilor lui $x$, aceștia fiind convertiți în numere raționale simbolice cu funcția `poly2sym(V)`;
- de asemenea, este posibilă și conversia inversă folosind funcția `sym2poly(p)`, unde $p$ este polinomul simbolic;
- rezultatul funcției este un vector numeric având coeficienții polinomului în ordinea descrescătoare a puterilor variabilei independente a polinomului simbolic.

# Rezolvarea simbolică a ecuațiilor

- fie $e$ o variabilă simbolică ce are ca valoare o expresie algebrică simbolică;
- rezolvarea ecuației $e = 0$ se realizează cu funcția `solve` ce admite următoarele forme:

  `solve(e, x)`

  atunci când se rezolvă ecuația după variabila simbolică $x$ și

  `solve(e)`

  când expresia simbolică $e$ este de o singură variabilă sau ecuația se rezolvă după variabila simbolică implicită;
- soluția obținută cu funcția `solve` este fie un vector cu soluții, dacă rezultatul este un vector cu un număr de componente egal cu numărul de variabile, fie o structură în care numele câmpurilor sunt numele variabilelor ecuațiilor, dacă rezultatul funcției `solve` este o variabilă;
- să determinăm soluțiile ecuației de gradul II, $ax^2 + bx + c = 0$:

```
>> clear, syms x a b c;
>> f = a*x^2+b*x+c;
>> x = solve(f)
x =
 -(b + (b^2 - 4*a*c)^(1/2))/(2*a)
 -(b - (b^2 - 4*a*c)^(1/2))/(2*a)
```

- dacă ecuația de rezolvat este dată sub forma $f(x) = g(x)$, se va folosi funcția `solve` cu forma

  `solve(f(x) == g(x), x)`

- funcția `solve` poate rezolva și sisteme de ecuații algebrice;
- considerăm sistemul de două ecuații algebrice

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0. \end{cases}$$

- dacă rezultatul funcției `solve` se dorește a fi un vector cu soluții, `solve` se apelează sub forma

  `[x, y] = solve(f, g)`
- dacă dorim ca rezultatul să fie o structură, în stânga semnului egal vom scrie o variabilă simbolică;
- să rezolvăm următoarele sisteme de ecuații:

  a) $\begin{cases} x - 2y + z = 0 \\ 2x + y - z = 1 \\ -3x + y + z = 2 \end{cases}$

```
>> clear, syms x y z;
>> f = x-2*y+z; g = 2*x+y-z-1; h = -3*x+y+z-2;
>> [x y z] = solve(f,g,h)
x =
1
```

```
y =
2
z =
3
>> s = solve(f,g,h)
>> s.x, s.y, s.z
```

- în acest caz soluția $s$ este o structură cu componentele x, y și z, iar rezultatele sunt s.x, s.y, s.z;

b) $\begin{cases} 2x^2 + y^2 = 0 \\ x - y = 1 \end{cases}$

```
>> clear, syms x y;
>> [x y] = solve(2*x^2+y^2 == 0, x-y == 1)
x =
 1/3 - (2^(1/2)*1i)/3
 (2^(1/2)*1i)/3 + 1/3
y =
 - (2^(1/2)*1i)/3 - 2/3
   (2^(1/2)*1i)/3 - 2/3
```

c) $\begin{cases} y = 2e^x \\ y = 3 - x^2 \end{cases}$

```
>> clear, syms x y;
>> f = y-2*exp(x); g = y-3+x^2;
>> [x y] = solve(f,g)
Warning: Unable to solve symbolically. Returning a
numeric solution using vpasolve.
x =
0.36104234240225080885012626307
y =
2.8696484269926958876157155521485
```

- dacă determinarea soluției simbolice nu este poibilă, **MATLAB** apelează automat funcția vpasolve pentru a returna soluțiile numerice, dând un mesaj de avertisment;
- din reprezentarea grafică se poate observa că sistemul are două soluții, ce pot fi obținute transmințând ca parametru intervalul în care se caută soluția:

```
>> clear, syms x y;
>> f = y-2*exp(x);
>> g = y-3+x^2;
>> [x y] = vpasolve(f,g,[0 1])
x =
0.36104234240225080885012626307
y =
2.8696484269926958876157155521485
>> [x y] = vpasolve(f,g,[-2 0])
x =
-1.6128773732205112107682419306365
y =
0.39862657895330378626951209209872
```

- presupunem că avem de calculat limita $\lim\limits_{x \to a} f(x)$;
- limita unei funcției se calculează cu funcția `limit` cu forma generală

  `limit(f, x, a)`

  unde $a$ este un număr sau inf (pentru infinit).
- funcția `limit` se poate apela și sub forma

  `limit(f, a)`

  caz în care variabila după care se calculează limita este variabila implicită;
- să calculăm limitele:

  a) $\lim\limits_{x \to \infty} \dfrac{x^2 - 3x + 2}{4x^2 - 7}$

```
>> clear, syms x;
>> f = (x^2-3*x+2)/(4*x^2-7);
>> limit(f,inf)
ans=
1/4
```

b) $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$

```
>> clear, syms n;
>> g = (1+1/n)^n;
>> limit(g,inf)
ans=
exp(1)
```

c) $\lim_{x\to 0}\frac{sin(x)}{x}$

```
>> clear, syms x;
>> h = sin(x)/x;
>> limit(h,0)
ans=
1
```

- dacă avem de calculat limitele laterale $\lim\limits_{\substack{x \to a \\ x > a}} f(x)$ și $\lim\limits_{\substack{x \to a \\ x < a}} f(x)$, putem folosi funcția `limit` cu un parametru suplimentar, astfel:
  `limit(f, x, a, 'right')`, pentru limita la dreapta și
  `limit(f, x, a, 'left')`, pentru limita la stânga;
- să calculăm limitele laterale ale funcției $\dfrac{|x|}{x}$, în punctul $x = 0$:

# Calculul limitelor de funcții

```
>> clear, syms x;
>> f = abs(x) / x;
>> limit(f, x, 0, 'right')
ans =  1
>> limit(f, x, 0, 'left')
ans = -1
```

- derivata simbolică se calculează cu funcția `diff`;
- se pot calcula derivatele funcțiilor de una sau mai multe variabile, derivatele de ordin 1 sau de ordin superior etc.
- funcția are următoarele forme:

```
diff(f) % pentru a calcula derivata unei functii
          in raport cu variabila implicita
diff(f, x) % pentru a calcula derivata de ordin 1 a unei
          functii in raport cu variabila x
diff(f, x, n) %  pentru a calcula derivata de ordin n a
          unei functii in raport cu variabila x
```

- de exemplu,

# Derivare

```
>> clear, syms a x
>> f = sin(a*x) + x*exp(a*x);
>> diff(f, x)
ans=
exp(a*x) + a*cos(a*x) + a*x*exp(a*x)
>> diff(f, x, 2)
ans=
2*a*exp(a*x) - a^2*sin(a*x) + a^2*x*exp(a*x)
>> clear, syms x y
>> g = x^2 + y^2 +x*y;
>> diff(g, x)
ans=
2*x + y
>> diff(g, y)
ans=
```

# Derivare

```
x + 2*y
```

- derivata unei matrice simbolice este matricea obținută derivând fiecare element:

```
>> clear, syms x y;
>> A = [ x^2 x*y; x*y y^2];
>> diff(A)
ans=
[2*x, y]
[  y, 0]
```

- Jacobianul unei transformări, $\frac{D(f,g,h,\cdots)}{D(x,y,z,\cdots)}$, se calculează cu funcția `jacobian`, cu forma generală:

```
jacobian([f; g; h;...], [x, y, z,...])
```

- de exemlu, fie transformarea dată prin: $\begin{cases} x = r\, cos(t) \\ y = r\, sin(t) \end{cases}$

- Jacobianul acestei transformării și determinantul matricei Jacobiene se calculează astfel:

```
>> clear, syms x y r t
>> x = r * cos(t)
>> y = r * sin(t)
>> J = jacobian([x; y], [r, t])
J =
[cos(t), -r*sin(t)]
[sin(t),  r*cos(t)]
>> detJ = det(J)
detJ=
r*cos(t)^2 + r*sin(t)^2
>> detJ = simplify(detJ)
detJ=
r
```

# Integrarea

- pentru calculul integralei (simbolice) nedefinite $\int f(x)dx$ se folosește funcția `int` cu formele:

  `int(f, x)`

  atunci când se specifică variabila de integrare, sau

  `int(f)`

  când variabila de integrare este variabila simbolică implicită;
- să calculăm primitiva funcției $f(x) = sin(ax) + xe^{ax}$

```
>> clear, syms x a
>> f = sin(a*x) + x*exp(a*x);
>> Int_f = int(f,x)
Int_f=
-(exp(a*x) + a*(cos(a*x) - x*exp(a*x)))/a^2
```

- prin comanda `diff(Int_f,x)` nu se obține neapărat $f$ ci o expresie echivalentă, dar după simplificare, folosind comanda `simplify(diff(Int_f,x))`, se obține expresia simbolică a lui $f$;
- calculul integralei definite a funcției $f(x)$, $\int_a^b f(x)dx$ se realizează tot cu funcția `int`, cu specificarea limitelor de integrare:
  `int(f, x, a, b)`
  atunci când se specifică variabila de integrare, sau
  `int(f, a, b)`
  atunci când variabila de integrare este variabila simbolică implicită;
- să calculăm $\int_0^\pi x sin(x)dx$:

```
>> clear, syms x
>> g = x*sin(x);
>> Int_g = int(g,x,0,pi)
Int_g =
pi
```

- să reprezentăm grafic funcția $f(x) = e^{-\frac{x^2}{2}}$ pentru $x \in \mathbb{R}$ și să calculăm $\int\limits_{-\infty}^{\infty} e^{-\frac{x^2}{2}}\,dx$:

```
>> clear, syms x;
>> f = exp(- x^2 / 2);
>> fplot(f); grid
>> int(f, x, -inf, inf)
ans =
2^(1/2)*pi^(1/2)
```

- fie o funcție $f(x)$ ce depinde de un parametru; dacă parametrul nu are o valoare specificată, el este implicit un număr complex; în cazul în care parametrul este un număr real sau un număr pozitiv, el trebuie declarat ca atare în comanda `syms`;
- de exemplu, integrala $\int\limits_0^\infty e^{-tx}\frac{sin(x)}{x}dx$ este uniform convergentă pe $[0,\infty)$;

```
>> clear, syms x t positive;
>> f = exp(-t*x)*(sin(x)/x)
>> I = int(f, x, 0, inf)
I=
acot(t)
```

- considerăm ecuația diferențială ordinară de ordinul întâi $y'(t) = f(y(t), t)$;
- soluția generală a acestei ecuații diferențiale depinde de o constantă arbitrară $C$;
- în cazul IVP pentru ODE, constanta $C$ se determină în funcție de condiția inițială;
- presupunem că soluția ecuației diferențiale există, în funcție de proprietățile funcției $f$, soluția putând fi unică sau nu;
- rezolvarea ecuațiilor diferențiale ordinare se face cu funcția `dsolve` care poate rezolva și ecuații diferențiale de ordin superior și sisteme de ecuații diferențiale ordinare;
- funcția `dsolve` are ca parametri: expresia simbolică a ecuației diferențiale sau, în cazul unui sistem, expresiile simbolice ale ecuațiilor sistemului; în aceste expresii, derivatele se introduc folosind `diff`;

- condițiile inițiale ale ecuației diferențiale, dacă există, sunt de forma $y(t0) = y0$, $Dy(t0) = y1$ etc., unde $Dy = diff(y, t)$;
- variabila independentă implicită este $t$, dar variabila independentă poate fi și altă variabilă simbolică, care va fi ultimul argument al funcției `dsolve`;
- soluția ecuației diferențiale poate fi: un vector cu un număr de componente egal cu cel al variabilelor dependente, dacă rezultatul funcției `dsolve` este un vector sau poate fi o structură în care numele câmpurilor sunt numele variabilelor dependente, dacă rezultatul funcției `dsolve` este o variabilă;
- să rezolvăm următoarele probleme:

a) $$\begin{cases} y'(t) = y(t) - \dfrac{2t}{y(t)}, \ \ t > 0 \\ y(0) = 1 \end{cases}$$

```
>> clear
>> syms y(t)
>> y(t) = dsolve(diff(y,t) == y-2*t/y, y(0) == 1)
y =
(2*t + 1)^(1/2)
```

b) $\begin{cases} x'(t) = x(t) + y(t) \\ y'(t) = t - x(t), \;\; t > 0 \\ x(0) = 0.1, y(0) = 0.2 \end{cases}$

- vom calcula soluția într-un vector și apoi într-o structură;

```
>> clear
>> syms x(t) y(t)
>> [x(t),y(t)] = dsolve(diff(x,t)==x+y, diff(y,t)==t-x,...
 x(0)==0.1, y(0)==0.2)
>> fplot([x y])
```

sau

```
>> clear
>> syms x(t) y(t)
>> s = dsolve(diff(x,t)==x+y, diff(y,t)==t-x,...
 x(0)==0.1, y(0)==0.2)
>> s.x, s.y
```

c)

$$\begin{cases} y''(t) = -1.2y'(t) - y(t) + 10, \ t > 0 \\ y(0) = 2, y'(0) = 0 \end{cases}$$

```
>> clear
>> syms y(t)
>> Dy = diff(y,t)
>> y(t)= dsolve(diff(y,t,2)==-1.2*diff(y,t)-y(t)+10,...
y(0)==2,Dy(0)==0)
>> fplot(y)
```

# Aritmetică cu precizie variabilă (vpa)

- există trei tipuri de operații aritmetice în toolbox-ul Symbolic Math: numerice – operațiile **MATLAB** în virgulă flotantă, raționale – aritmetica simbolică exactă, VPA – aritmetica cu precizie variabilă (variable precision arithmetic);
- aritmetica de precizie variabilă se realizează cu ajutorul funcției `vpa`;
- numărul de cifre este controlat de variabila Digits;
- funcția `digits` afișează valoarea lui Digits, iar `digits(n)`, unde *n* este un întreg, setează Digits la *n* cifre;
- comanda `vpa(e)` evaluează expresia *e* cu precizia Digits, iar `vpa(e,n)` evaluează *e* cu *n* cifre;
- rezultatul este de tip `sym`;
- cu toolbox-ul Symbolic Math, intrucțiunea

  ```
  >> sym(1/2+1/3)
  ```

  va produce, folosind calculul simbolic, răspunsul 5/6;

# Aritmetică cu precizie variabilă (vpa)

- tot în toolbox-ul Symbolic Math, cu aritmetica cu precizie variabilă, instrucțiunile

  ```
  >> digits(25)
  >> vpa(1/2+1/3)
  ```

  au ca rezultat `.8333333333333333333333333`