CURS 3

1. Instrucţiuni de control logic

Instrucţiunile de control logic în MATLAB sunt:

if Instrucţiune pentru execuţia condiţionată;

else Instrucţiune asociată cu „if”;

elseif Instrucţiune asociată cu „if”;

for Instrucţiune pentru crearea ciclurilor cu număr specificat de paşi;

while Instrucţiune pentru crearea ciclurilor cu condiţie logică;

break Instrucţiune pentru terminarea forţată într-un ciclu;

return Returnează execuţia la funcţia precizată;

error Instrucţiune pentru afişarea unui mesaj de eroare;

end Instrucţiune pentru încheierea ciclurilor „for", „while" si „if”.

Programele MATLAB sunt scrise pentru a realiza numai paşi secvenţiali, operaţiile fiind realizate una după alta. În multe cazuri, însă, este necesară repetarea unui set de instrucţiuni atâta timp cât o condiţie este realizată si a altui set de instrucţiuni, dacă condiţia nu mai este realizată. Alteori, este necesară repetarea unui grup de instrucţiuni de un anumit număr de ori. MATLAB pune la dispoziţia programatorului două instrucţiuni, ***for*** şi ***while***, cu ajutorul cărora un grup de instrucţiuni, numit corpul ciclului, sa poate repeta de mai multe ori.

Instrucţiunea ***for*** este folosită pentru repetarea unui grup de instrucţiuni de un anumit număr de ori, cunoscut iniţial. Un contor de ciclu ţine evidenţa numărului de repetări ale instrucţiunilor din corpul ciclului şi opreşte ciclul când contorul ajunge la valoarea precizată.

Instrucţiunea ***while*** este folosită atunci când calculele efectuate cu un grup de instrucţiuni trebuie repetate până când o anumită condiţie este îndeplinită. Numărul de iteraţii nu este cunoscut iniţial. Ieşirea din ciclu se realizează la îndeplinirea condiţiei impuse, care este testată la începutul fiecărei iteraţii.

1.1. Instrucţiunea condiţională „if”

În cadrul unui algoritm este deseori necesară o selecţie a grupului de instrucţiuni ce urmează a fi

executate, condiţionată de valoarea de adevăr a unei expresii. Instrucţiunile condiţionale utilizează operatorii relaţionali si operatorii logici.

1.1.1. *Operatori relaţionali*

MATLAB-ul are şase operatori relaţionali, care sunt utilizaţi pentru a compara două matrice de dimensiuni egale; lista este prezentată în tabelul 1.

Operatorii relaţionali compară două matrice sau două expresii matriceale, element cu element. Aceştia returnează o matrice de aceeaşi dimensiune cu a matricelor care se compară, cu elementele 1 când relaţia este ADEVĂRATĂ şi cu elementele zero când relaţia este FALSĂ. Primii patru operatori (< ,<= , >, >=) compară numai partea reală a operanzilor (partea imaginară este ignorată), iar ultimii doi operatori (== şi ~=) tratează atât partea reală cât şi cea imaginară.

Tabelul 1

Operatorii relaţionali	Semnificaţia
<	mai mic
<=	mai mic sau egal
>	mai mare
>=	mai mare sau egal
= =	identic
~ =	diferit

Forma generală de utilizare a operatorilor relaţionali este:

rezultat = expresie_ 1 op_relaţional expresie_2

unde: ***rezultat*** este o matrice de elemente 0 şi 1, în care sunt returnate rezultatele comparaţiei;

expresie_1, ***expresie_2*** - matricele sau expresiile matriceale care se compară ;

op_relaţional - unul dintre cei şase operatori relaţionali menţionaţi în tabelul 1.

Dacă A şi B sunt matricele care se compară şi C matricea în care este returnat rezultatul, expresia:

C=A op_relaţional B

returnează:

$$C(i,j)=\begin{cases}1\ daca\left(A(i,j)\,op\_relational\,B(i,j)\right)true\\0\ daca\left(A(i,j)\,op\_relational\,B(i,j)\right)false\end{cases}$$

Dacă unul dintre operanzi este un scalar şi celălalt o matrice, scalarul se „extinde" până la dimensiunea matricei (vezi exemplele din cursul anterior – operaţii cu matrici).

Exemplul 1. Următoarele două instrucţiuni:

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

X=5*ones(3,3)>=[1 2 3; 4 5 6; 7 8 10]

sunt echivalente, obţinându-se acelaşi rezultat:

$X =$

$$\begin{matrix}1 & 1 & 1\\1 & 1 & 0\\0 & 0 & 0\end{matrix}$$

Observaţie. Funcţia ***ones(3,3)*** returnează o matrice 3*3 cu toate elementele egale cu 1.

Exemplul 2. Fie matricele:

$$A=\begin{bmatrix}1 & -i & 0\\2-i & 0 & 1+i\end{bmatrix}\quad B=\begin{bmatrix}1 & -i & i\\2-i & 2 & 1-5i\end{bmatrix}$$

Să se realizeze comparaţiile:

C=A>=B şi **D=A= =B** cu A şi B definite mai sus, se obţin rezultatele

$C =$

$$\begin{matrix}1 & 1 & 1\\1 & 0 & 1\end{matrix}$$

$D =$

$$\begin{matrix}1 & 1 & 0\\1 & 0 & 0\end{matrix}$$

1.1.2. *Operatori logici*

Pentru combinarea a două sau mai multe expresii logice se utilizează operatorii logici din tabelul 2.

Tabelul 2.

Operatori logici	Simbol MATLAB	Prioritatea

NU	~	1
Şl	&	2
SAU	|	3

Operatorii & şi | compară doi scalari sau două matrice de dimensiuni egale. Pentru matrice operează element cu element. Operatorii logici retumează „1" (ADEVĂRAT) pentru orice valoare diferită de zero. Se va returna „1" pentru ADEVĂRAT şi „0" pentru FALS.

Operatorul logic NU (sau complementul logic), este operator unar. Expresia **~A** retumează „0" dacă A e diferit de zero si „1" dacă A este „0".

Operatorii logici au prioritate mai mică decât operatorii relaţionali şi aritmetici. Astfel, se evaluează întâi expresiile care conţin operatori relaţionali şi aritmetici, şi apoi cele cu operatori logici.

Operatorii logici se utilizează întotdeauna pentru compararea matricelor cu elemente 0 şi 1, calculate cu operatori relaţionali. Spre exemplu, expresia logică:

$$D(i,j)=\begin{cases} 1 & daca\ A(i,j)<B(i,j)\&\left(B(i,j)<C(i,j)\right) \\ 0 & in\ rest \end{cases}$$

are următoarea descriere în MATLAB:

D=(A<B)&(B<C)

Această operaţie este posibilă numai dacă matricele rezultate din operaţiile A< B şi B < C au aceeaşi dimensiune.

Operatorii logici se utilizează cu expresii logice complete. Spre exemplu, expresia **a<b & b<c** este corectă, însă expresia **a<b & c** este incompletă. Expresiile logice pot fi precedate de operatorul logic NU. Acest operator schimbă valorile expresiei în valorile opuse: dacă **a>b** este 1 (ADEVĂRAT), atunci **~(a>b)** este 0 (FALS). Expresiile logice pot conţine mai mulţi operatori logici, ca în exemplul: **~(b= =c | c = = 5)**

Ordinea de prioritate a operatorilor logici, de la superior la inferior, este: NU, Şl, SAU. Dacă b = 3 si c = 5, în exemplul anterior, prima expresie este 0 şi a doua 1. Se evaluează apoi expresia 0 | 1 a cărei valoare logică este 1. Aplicarea operatorului NU face ca rezultatul să fie 0.

Observaţie. Este preferabil, pentru siguranţă, ca atunci când nu se cunoaşte cu certitudine ordinea de prioritate a aplicării operatorilor logici şi relaţionali să se recurgă la folosirea parantezelor aşa cum o solicită ordinea operaţiilor din aplicaţia respectivă. Exemplu:

b=3; c=5; d=~(b= =c | c= =5) returnează d=0 pe când **b=3; c=5; d=~b= =c | c= =5** returnează **d=1.**

1.1.3. *Instrucţiunea „if" simplă*

Instrucţiunea „if" poate fi implementată ca instrucţiune „if" simplă, sau poate include clauzele „else" sau „elseif". Forma generală a unei instrucţiuni simple „if" este următoarea:

if expresie_logică

grup_de_instrucţiuni

end

Dacă **expresia_logică** este adevărată, se execută grupul de instrucţiuni dintre instrucţiunea **if** şi instrucţiunea **end**. Dacă expresia logică este falsă, se trece la prima instrucţiune care urmează după instrucţiunea **end**.

Pentru o citire mai uşoară, cât şi pentru o urmărire a grupurilor de instrucţiuni care se execută, se procedează la indentarea (deplasarea) acestora. Fie următoarea secvenţă MATLAB:

```
if   a<50

     k=k+l

     s=s+a

end
```

Dacă **a** este un scalar şi dacă **a< 50**, **k** se incrementează cu 1 şi apoi **a** este adunat cu **s**; altfel, cele două instrucţiuni sunt omise, **s** rămânând cu valoarea cu care a fost iniţializat înainte de a ajunge la instrucţiunea **if**. Dacă **a** este un vector sau o matrice, atunci **k** este incrementat cu 1 şi **a** este însumat cu **s**, numai dacă fiecare element al lui **a** este mai mic decât 50.

Instrucţiunea if poate fi inclusă în interiorul unei alte instrucţiuni **if**, ca în exemplul următor:

```
if expresia_logică_1

     grupul_de_instrucţiuni_A

if expresia_logică_2

     grupul_de_instrucţiuni_B

end

     grupul_de_instrucţiuni_C

end

     grupul_de_instrucţiuni_D
```

Dacă **expresia_logică_1** este 1 (ADEVĂRATĂ), se execută întotdeauna grupurile de instrucţiuni A şi C. Dacă si expresia_logică_2 este 1 (ADEVĂRATĂ), se execută grupul de instrucţiuni B, înaintea executării grupului de instrucţiuni C. Dacă expresia_logică_1 este 0 (FALSĂ), se trece la grupul de instrucţiuni D. Indentarea este o operaţie necesară la utilizarea instrucţiunilor **if**, deoarece pune în evidenţă grupurile de instrucţiuni care se execută funcţie de condiţiile pe care le îndeplinesc.

Fie următorul exemplu de utilizare a instrucţiunilor condiţionale **if** succesive:

```
if   a<50
     k=k+1
     s=s+a
     if  b>a
     b=0
     end
end
```

Dacă a şi b sunt scalari şi dacă a < 50, se incrementează k cu 1 si se adună a la s. în schimb, dacă b > a, se setează b la zero. Dacă a nu este mai mic decât 50, se trece la grupul de instrucţiuni care urmează după a doua instrucţiune **end**.

Dacă a este un vector sau o matrice, condiţia a<50 este adevărată numai dacă fiecare element al lui a este mai mic decât 50. Dacă a şi b sunt vectori sau matrice, atunci b>a numai dacă fiecare pereche (b, a) verifică această condiţie. Dacă a sau b este scalar, comparaţia se face între fiecare element al matricei cu scalarul extins la o matrice.

1.1.4. *Clauza „else”*

Clauza **else** este utilizată pentru a executa un set de instrucţiuni, dacă expresia logică este ADEVĂRATĂ şi un alt set de instrucţiuni, dacă expresia logică este FALSĂ. Forma generală a instrucţiunii **if** este combinată cu clauza **else** (numită uneori instrucţiunea **if-else**) ca în exemplul următor:

```
if expresie_logică
     grupul_de instrucţiuni_A
else
     grupul_de_instrucţiuni_B
end
```

Dacă expresia logică este adevărată se execută grupul de instrucţiuni A, iar dacă este falsă se execută grupul de instrucţiuni B.

Exemplul. Pentru următoarea funcţie:

$f(x)=\begin{cases}2x+8 & daca\ x\le 2\\ 3x^2 & daca\ x>2\end{cases}$ descrierea MATLAB este:

```
if  x <= 2

    f=2*x+8

else

f=3*x^2

end
```

1.1.5. *Clauza „elseif”*

Dacă funcţia de calculat are mai multe nivele de instrucţiuni **if-else**, este dificilă determinarea expresiei logice adevărate, care selectează grupul de instrucţiuni ce urmează a fi executat, în acest caz, se utilizează clauza **elseif**:

```
if expresia_logică_1

grupul_de_instrucţiuni_A

elseif expresia_logică_2

grupul_de_instrucţiuni_B

elseif expresia_logică_3

grupul_de_instruţiuni_C

end
```

- dacă expresia_logică_1 este adevărată, este executat numai grupul de instrucţiuni A;

- dacă expresia_logică_1 este falsă şi expresia_logică_2 este adevărată, se execută numai grupul_de_instrucţiuni_B;

- dacă expresiile logice 1 şi 2 sunt false, iar expresia logică 3 este adevărată, se execută numai grupul de instrucţiuni C;

- dacă mai multe expresii logice sunt adevărate, prima instrucţiune logică adevărată determină care grup de instrucţiuni este executat prima dată;

- dacă nici o expresie logică nu este adevărată, nu se execută nici un grup de instrucţiuni din structura **if**.

Clauza **elseif** poate fi combinată cu clauza **else** într-o structură generală de forma:

if expresia_logică_1

grupul_de,,instrucţiuni_A

elseif expresia_logică_2

grupul_de_instrucţiuni_B

elseif expresia_logică_3

grupul_de_instrucţiuni_C

else

grupu_de_instrucţiuni_D

end

Dacă nici o expresie logică dintre primele trei nu este adevărată, se va executa grupul_de_instrucţiuni_D.

1.1.6. *Instrucţiunea repetitivă „for”*

Instrucţiunea **for** permite repetarea unui grup de instrucţiuni din corpul buclei, de un anumit număr de ori; are următoarea structură generală:

for index = expresie

grupul_de_instrucţiuni

end

unde:

- index este numele contorului;

- expresie este o matrice, un vector sau un scalar;

- grup_de_instrucţiuni este orice expresie MATLAB. În aplicaţii, „expresie” este de cele mai multe ori de forma:

k = iniţial:pas:final unde:

iniţial - este prima valoare a lui k,

pas - pasul (dacă este omis, este considerat 1),

final - cea mai mare valoare pe care o poate lua k.

La fiecare pas de calcul „index" are valoarea unuia dintre elementele expresiei. Dacă „expresie" este o matrice, ciclarea se face pe coloane.

Pentru un ciclu **for** cu pasul negativ sau neîntreg se generează mai întâi un vector cu pasul şi limitele dorite si apoi se citesc valorile acestuia în cadrul buclei for.

La folosirea buclei **for** trebuie respectate următoarele reguli:

1. indexul buclei **for** trebuie să fie o variabilă;

2. dacă expresia este o matrice goală, bucla nu se execută. Se va trece la următoarea instrucţiune după instrucţiunea **end**;

3. dacă expresia este un scalar, bucla se execută o singură dată, cu indexul dat de valoarea scalarului;

4. dacă expresia este un vector linie, bucla se execută de atâtea ori câte elemente are vectorul, de fiecare dată indexul având valoarea egală cu următorul element din vector;

5. dacă expresia este o matrice, indexul va avea la fiecare iteraţie valorile conţinute în următoarea coloană a matricei;

6. la terminarea ciclurilor **for**, indexul are ultima valoare utilizată;

7. dacă se utilizează operatorul două puncte („:") pentru a defini expresia, ca în exemplul:

for k= iniţial:pas:final bucla se execută de:

$n = \left[\frac{final - initial}{pas} \right] + 1$ ori, dacă n este pozitiv, şi nu există dacă n este negativ. Prin [] s-a notat valoarea întreagă a numărului.

1.1.7. *Instrucţiunea repetitivă „while"*

Instrucţiunea **while** este o structură care se utilizează pentru repetarea unui set de instrucţiuni, atâta timp cât o condiţie specificată este adevărată. Formatul general al acestei instrucţiuni de control este următorul:

while expresie

grup_de_instrucţiuni

end

Grupul de instrucţiuni se execută cât timp „expresie” are toate elementele nenule. „expresie” este de obicei sub forma:

expresie_1 condiţie expresie_2

unde „condiţie” este unul dintre operatorii relaţionali: ==, <, >, <=, >=, ~=. Când condiţia este verificată (expresia este adevărată logic), se execută grupul de instrucţiuni. După ce se execută grupul de instrucţiuni, condiţia este retestată. Când condiţia este din nou adevărată, grupul de instrucţiuni se execută iar. Când condiţia este falsă, se trece la următoarea instrucţiune de după instrucţiunea **end**. Dacă expresia este totdeauna adevărată logic (valoarea acesteia este diferită de zero), bucla devine infinită; dintr-o buclă infinită se iese forţat prin apăsarea concomitentă a tastelor [Ctrl]+[C].

1.1.8. *Instrucţiunea „break”*

Instrucţiunea **break** se utilizează pentru a ieşi dintr-o buclă înainte ca aceasta să se fi terminat. Se recomandă a fi utilizată dacă o condiţie de eroare este detectată în interiorul unei bucle. Instrucţiunea **break** încetează execuţia ciclurilor **for** şi **while**. În cazul unor cicluri imbricate, **break** comandă ieşirea din ciclul cel mai interior. Se apelează cu sintaxa: **break.**

1.1.9. *Instrucţiunea „return”*

Instrucţiunea **return** comandă o ieşire normală din fişierul-M către funcţia care l-a apelat sau către tastatură. Se apelează cu sintaxa: **return.**

Exemple de implementat la laborator (utilizarea operatorilor de comparaţie şi logici şi a structurilor de control logic)

1. Să se genereze o matrice A, cu n linii şi n+1 coloane, ale cărei elemente sunt:

$$A=\begin{cases} 2 & daca\ i=j \\ -1 & daca\ |i-j|=1 \\ 0 & in\ rest \end{cases}$$

Cu secvenţa MATLAB:

```
n=4;
for i=1:n,
for j=1:n+1,
    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
```

Se obţine rezultatul:

$$A = \begin{matrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \end{matrix}$$

2. Să se genereze o matrice Hilbert de ordinul 4, ale cărei elemente sunt date de expresia:

$$H(i,j) = \frac{1}{i+j-1}$$

Secvenţa MATLAB:

```
n=4
for i=1:n,
    for j=1:n,
        H(i,j)=1/(i+j-1);
    end
end
H
```

VECTORIZAREA CALCULELOR

Deoarece operaţiile cu vectori şi matrici sunt executate în MATLAB mai repede cu un ordin de mărime decât operaţiile compilate/interpretate, se obţine o viteză de lucru mai mare dacă algoritmii înscrişi în fişierele-M sunt vectorizaţi. Oriunde este posibil, ciclurile ***for*** şi ***while*** trebuie convertite în operaţii cu vectori sau matrici.

Spre exemplu, programul care calculează sinusul în 1000 de puncte (de la 1 la 10, cu pasul .01), utilizând bucla ***for***.

```
t=0:0.01:10; N=length(t)

for i=1: N,

    y(i)=sin(t(i)) ;

end

plot (t, y)
```

are versiunea vectorizată:

```
t=0:0 .01:10;

Y=sin(t);

plot (t, Y)
```

Primul exemplu necesită un timp de 10-15 ori mai mare decât cel de-al doilea.

În cazurile în care nu se poate vectoriza o parte din program, pentru a face ca ciclurile să fie executate mai repede, se procedează la prealocarea unor vectori în care vor fi reţinute rezultatele. De exemplu, incluzând o primă instrucţiune de prealocare folosind funcţia ***zeros***, ciclul ***for*** următor se execută semnificativ mai repede:

```
y=zeros (1,100);

for i=1:100

    y(i)=det(x^i);

end
```

Explicaţia constă în faptul că dacă nu se face prealocarea, interpretorul MATLAB trebuie să redimensioneze vectorul y la o dimensiune mai mare, de fiecare dată când trece printr-o iteraţie a ciclului. Dacă vectorul este prealocat, acest pas este eliminat si execuţia este mai rapidă.

Pentru lucrul cu matrici mari pe computere cu memorie limitată, prealocarea are un al doilea avantaj: utilizarea memoriei mult mai eficient şi lansarea programelor fără a depăşi memoria. Prealocarea ajută inclusiv la reducerea fragmentării memoriei; o memorie fragmentată în cursul sesiunii MATLAB poate deveni insuficientă ca spaţiu continuu necesar memorării variabilelor mari.

FUNCŢII MATEMATICE UZUALE UZUALE

1. APROXIMAREA NUMERELOR

Funcţiile MATLAB folosite pentru aproximarea numerelor sunt:

ceil returnează un număr întreg, rotunjit la cel mai apropiat întreg spre plus infinit (+∞);

fix returnează un număr întreg, rotunjit la cel mai apropiat întreg spre zero;

floor returnează un număr întreg, rotunjit la cel mai apropiat întreg spre minus infinit (-∞);

round returnează un număr întreg, rotunjit la cel mai apropiat întreg;

rem returnează restul împărţirii argumentelor;

rat returnează aproximarea unui număr cu fracţii raţionale continue;

rats Returnează aproximarea unui număr cu numere raţionale;

sign returnează semnul argumentului.

1.1. Aproximarea cu numere întregi

Funcţiile MATLAB folosite pentru aproximarea cu numere întregi sunt: ***fix, floor, ceil, round***. Aceste funcţii operează asupra fiecărui element al unei matrice sau al unui vector şi se apelează cu sintaxa:

nume_funcţie(argument) unde:

- nume_funcţie este numele uneia dintre funcţiile de mai sus, iar

- argument poate fi un scalar, un vector sau o matrice, ale căror elemente se doresc a fi rotunjite.

Dacă elementele din argumentul funcţiei sunt numere complexe, funcţiile de mai sus operează independent asupra fiecărei părţi (reală şi imaginară).

Exemplul 10. Să se rotunjească elementele vectorului: V=[0 2 2.3 4.7 -5.2 -7.8]

- la cel mai apropiat întreg;

- la cel mai apropiat întreg spre 0;

- la cel mai apropiat întreg spre +∞;

- la cel mai apropiat întreg spre -∞.

Să se determine semnul elementelor vectorului V. Cu secvenţa de instrucţiuni:

V = [0 2 2.3 4,7 -5.2 -7.8] .; A =round (V); B = fix (V); C = ceil (V); D = floor(V); E = sign(V)

Exemplul 11. Acelaşi enunţ pentru o matrice cu elemente numere complexe [1,25+2,59i 7,3-5,3i; -4,2+1,8i -2,6-1,4i]

1.2. Aproximarea cu numere raţionale

Funcţia MATLAB ***rats*** realizează aproximarea cu numere raţionale; se apelează cu una dintre sintaxele: y=rats(x) y=rats(x, 's')

Argumentul de intrare 's' determină afişarea rezultatului simbolic y într-o matrice şir.

Exemplul 12. Să se aproximeze cu numere raţionale, numerele: 1.25, 0.25, π şi 1.2596. Cu secvenţa:

X=[1.25 0.25 pi 1.2596]; Y=rats(X)

1.3. Aproximarea cu fracţii continue

Funcţia MATLAB ***rat*** aproximează un număr cu fracţii continue; se apelează cu una dintre sintaxele:

y=rat(x) [a, b]=rat(x)

y=rat(x, tol) [a, b]=rat(x, tol)

unde:

x - este numărul care trebuie aproximat cu fracţii continue;

tol - este toleranţa care se acceptă între numărul x si numărul y (y-x<=tol); implicit, tol=10^{-6};

y - exprimarea lui x ca fracţie continuă;

a şi b - numărătorul si numitorul fracţiei care aproximează pe x cu toleranţa tol.

Funcţia **rat** aproximează fiecare element al vectorului x cu un număr de forma:

$$y = d_0 + \cfrac{1}{d_1 + \cfrac{1}{d_2 + ... + \cfrac{1}{d_k}}}$$

Exemplul 13. Să se aproximeze cu fracţii continue numerele: 0.25, 1.25, -2.25 si 1.343. Cu secvenţa MATLAB rat([0.25 1.25 -2.25 1.343]) se obţine rezultatul:

Exemplul 14. Să se aproximeze prin fracţii raţionale numerele: 2.25, 3.5, 6.57, 10. Se înscriu aceste numere într-un vector şi se aplică funcţia **rat**, ca în secvenţa de mai jos:

X = [2.25 3.5 6.57 10]; [A,B] =rat(X)

obţinându-se rezultatul:

X = [2.25 3.5 6.57 10] A = [9 7 657 10] B=[4 2 100 1]

1.4. Funcţia rest

Funcţia ***rem***(X,Y) calculează restul împărţirii lui X la Y, element cu element. Dacă elementele vectorului sau matricei sunt numere complexe, partea imaginară este ignorată. Argumentele X şi Y trebuie să fie matrice de aceeaşi dimensiune, sau unul dintre ele să fie scalar.

Exemplul 15. Să se calculeze restul împărţirii elementelor vectorului X la Y. Cu secvenţa de instrucţiuni:

X = [1 3 -6]; Y = [2 3 4]; Z = rem(X,Y) se obţine rezultatul:

Z = [1 0 -2]

Exemplul 16. Să se determine restul împărţirii unui vector la un scalar şi al unui scalar la un vector. Cu secvenţa MATLAB:

X= [2.5 6 -7]; Y=3;

Zl=rem(X,Y) Z2=rem(Y,X) se obţin rezultatele:

Z1=[2.5000 0 -1.0000] Z2 = [0.5000 3.0000 3.0000]

1.5. Funcţia semn

Funcţia **sign** asociază fiecărui element al vectorului X elementele -1, 0, 1, după următoarea regulă:

$$\operatorname{sgn} x = \begin{cases} 1, & daca\ x > 0 \\ 0, & daca\ x = 0 \\ -1, & daca\ x < 0 \end{cases}$$

1. DIVIZORI ŞI MULTIPLI COMUNI

Pentru calculul divizorilor şi al multiplilor comuni se folosesc funcţiile:

gcd calculează cel mai mare divizor comun a două numere;

lcm calculează cel mai mic multiplu comun a două numere.

1.1. Cel mai mare divizor comun

Funcţia MATLAB ***gcd*** calculează cel mai mare divizor comun a două numere întregi; se apelează cu sintaxa:

a=gcd(x,y)

Exemplul 1. Să se determine cel mai mare divizor comun al numerelor 30 şi 21. Cu secvenţa: a=gcd(30,21) rezultă: a=3

1.2. Cel mai mic multiplu comun

Funcţia MATLAB ***lcm*** returnează cel mai. mic multiplu comun a două numere întregi. Se apelează cu sintaxa:

a=lcm(x,y)

Exemplul 2. Să se determine cel mai mic multiplu comun al numerelor: 9 şi 30.

2. NUMERE COMPLEXE

Operaţiile cu numere complexe folosesc funcţiile:

abs Calculează valoarea absolută (modulul) a argumentului;

angle Calculează faza (unghiul) argumentului;

unwrap Calculează părţile reală şi imaginară ale numerelor complexe exprimate în forma polară

conj Calculează conjugata complexă a argumentului;

imag Extrage partea imaginară a argumentului;

real Extrage partea reală a argumentului.

2.1. Definirea numerelor complexe în MATLAB

Numerele complexe sunt permise în toate operaţiile şi funcţiile din MATLAB. Acestea sunt introduse utilizând variabilele speciale i si j, ca în exemplele: $z = 2 + 3i$ sau $z = 2 + 3j$. Pentru a defini matricea:

$$M = \begin{bmatrix} 2+3i & 1 \\ -i & 2-i \end{bmatrix}$$

există două metode:

- ca sumă a două matrice cu elemente numere reale, una reprezentând partea reală iar cealaltă partea imaginară;

M=[2 1; 0 2] +[3 0;-1 -1]*i

- ca o matrice cu elemente numere complexe.

M=[2+3*i 1; -i   2-i]

Prin al doilea procedeu trebuie evitat orice spaţiu liber (blanc) între părţile reală si imaginară ale aceluiaşi număr complex; astfel:

a=[2+3*i]

b=[2 +3*i] returnează:

a=[2.0000+3.0000i]

b=[2.0000   0+3.0000i], notaţia b reprezentând două numere separate.

Dacă variabilele i sau j au fost deja utilizate în alte scopuri, pentru calculul cu numere complexe poate fi declarată o nouă unitate imaginară, în modul următor: i1=sqrt(-1)

2.2. Modulul şi argumentul numerelor complexe

Un număr complex z se exprimă sub una dintre formele:

- carteziană: $z=x+iy$

- polară: $z=re^{i\varphi}$

unde x şi y sunt părţile reală şi imaginară ale numărului complex z, iar r şi φ sunt modulul şi argumentul numărului complex z.

Funcţia ***abs*** determină modulul elementelor unui vector sau unei matrice; se apelează cu sintaxa:

r=abs(z)

Funcţia ***angle*** calculează argumentul elementelor unui vector sau unei matrice, în radiani; se apelează cu sintaxa:

fi=angle(z)

Funcţia ***unwrap*** permite calculul părţilor reale, cu sintaxa: x=unwrap(z) sau x=unwrap(real(y)) respectiv a părţii imaginare, cu sintaxa: y= unwrap(imag(z)) Argumentul numărului complex trebuie să fie exprimat în radiani.

Exemplul 3. Să se scrie numărul complex z=1+i sub forma polară. Cu instrucţiunile:

x=l; y=l; z=x+i*y; r=abs(z); fi=angle(z) se obţine rezultatul: r= 1.4142; fi=0.7854

Exemplul 4. Să se scrie numărul complex $z=4e^{i\pi/4}$ sub forma carteziană. Cu instrucţiunile: r=sqrt(4); fi=pi/4; z=r.*exp(i*fi).

Exemplul 5. Fie dată matricea cu numerele complexe exprimate sub forma polară:

$$M=\begin{bmatrix} 1 & e^{i\frac{\pi}{4}} & e^{-i\frac{\pi}{4}} \\ e^{\frac{\pi}{2}} & e^{i2\pi} & e^{-i2\pi} \end{bmatrix}$$

Să se determine proiecţiile acestor numere pe axele reală şi imaginară, aplicând funcţia ***unwrap***

2.3. Părţile reală şi imaginară şi conjugatul numerelor complexe

Partea reală a unui număr complex poate fi determinată cu funcţia ***real***; se apelează cu sintaxa: x=real(z)

iar partea imaginară poate fi determinată cu funcţia ***imag*** ; se apelează cu sintaxa: y=imag(z)

Conjugatul z al unui număr complex se poate determina cu funcţia ***conj***; se apelează cu sintaxa: conjugata=conj(z)

Exemplul 6. Fie dată matricea

$M = \begin{bmatrix} 1 & 2+i \\ -i & 4e^{i\frac{\pi}{4}} \end{bmatrix}$. Să se determine partea reală, imaginară şi conjugatul elementelor acesteia.

3. FUNCŢIILE PUTERE, RADICAL, LOGARITM ŞI EXPONENŢIALĂ

Funcţiile MATLAB pentru ridicarea la putere, extragerea radicalului, calculul logaritmului şi al exponenţialei, sunt:

^ Ridică un număr a la puterea n (a^n);

exp Calculează exponenţiala (e^x);

log Calculează logaritmul natural (ln);

log2 Calculează logaritmul în baza 2 ($\log_2$);

log10 Calculează logaritmul zecimal ($\log_{10}$);

nextpow2 Determină puterea N a numărului 2 care majorează modulul argumentului P ($|P| \le 2^N$);

pow2 Calculează valoarea numărului 2 la puterea n (2^n);

sqrt Calculează radicalul de ordinul doi dintr-un număr.

Dacă argumentul acestor funcţii elementare sunt matrice, ele operează element cu element. Argumentele funcţiilor pot fi şi numere complexe.

3.1. Funcţia putere

MATLAB-ul dispune de două funcţii pentru ridicarea la putere:

- pow2 - pentru a ridica 2 la puterea n (2"),
- ^ - pentru a ridica un număr a la puterea n ($x=a^n$). Se apelează cu sintaxele:

y=pow2(x) - calculează numărul $y=2^x$. Dacă x este o matrice, y va fi o matrice de aceleaşi

dimensiuni cu elementele calculate după această regulă, funcţia acţionând element cu element y=pow2(m,n) - calculează numărul y=m*2";

x=a^n - calculează puterea n a numărului a, x=a". Exponentul n poate avea orice valoare, reală sau complexă. Pentru calculul radicalului de ordinul n dintr-un număr a, se utilizează funcţia putere sub forma: $x=a^{1/n}$. Funcţia ***nextpow2*** având ca argument scalarul P, se apelează cu sintaxa:

N=nextpow2(P) şi returnează cei mai mic număr natural N astfel încât $2^N \geq |P|$. Dacă P este vector, funcţia returnează scalarul N, astfel încât 2^N majorează numărul de elemente ale vectorului.

Exemplul 7. Să se calculeze: A = [2^3 2^5 $2^{13.5}$].

Exemplul 8. Să se efectueze aceleaşi calcule ca la punctul anterior, utilizând operatorul de ridicare la putere A. Cu secvenţa: A=[2^3 2^5 2^13.5] se obţin aceleaşi rezultate.

Exemplul 9. Să se calculeze: $x = \sqrt[3]{125}$.

Exemplul 10. Să se determine puterile N ale lui 2 care majorează elementele vectorului A=[4 -8 17]. Să se calculeze vectorul majorant $P=2^N$.

3.2. Funcţia radical

Calculul radicalului de ordinul 2 dintr-un număr, $x = \sqrt{a}$, poate utiliza funcţia putere, sau funcţia ***sqrt***, apelată cu sintaxa: x=sqrt(a)

Argumentul a poate fi orice număr real sau complex. Dacă numărul a este negativ sau complex, rezultatul calculului este un număr complex.

Exemplul 11. Să se calculeze radicalul fiecărui element al matricei: X =[1 2;4 -9]

3.3. Funcţia logaritm

Calculul logaritmului natural al logaritmului în baza 2 sau al logaritmului în baza 10 al unui număr a utilizează funcţiile ***log, log2*** şi respectiv ***log10,*** apelate cu sintaxele:

x=log(a) x=log2(a) x=log10(a)

Dacă argumentul funcţiilor log şi log10 este un număr negativ, sau complex, z=x+iy, rezultatul este calculat cu relaţiile:

log(z) = log(abs(z)) + i*atan2(y,x) şi log10(z) = log10(abs(z)) + i*atan2(y,x) unde ***atan2*** este funcţia MATLAB ce calculează arctangenta numărului complex.

Exemplul 12. Să se calculeze logaritmul natural şi zecimal din numerele e^2 şi 100.

Exemplul 13. Să se calculeze logaritmul în bază 2 al elementelor matricei: A = [4 2^3 8^2 10].

3.4. Funcţia exponenţială

Calculul exponenţialei: $x=e^a$, (unde e= 2.71828182845...) foloseşte funcţia ***exp***, apelată cu sintaxa: x=exp(a)

Dacă argumentul este numărul complex z=x+iy, rezultatul este calculat cu relaţia: $e^z = e^x (\cos(y) + i*\sin(y))$

Exemplul 14. Să se calculeze: e , e^2 şi e^{-3}. Cu secvenţa:

4. FUNCŢIILE TRIGONOMETRICE

Funcţiile trigonometrice se apelează cu sintaxa:

x=nume_funcţie(argument) unde:

- nume_funcţie este numele uneia dintre funcţiile trigonometrice de mai jos;

- argument este valoarea pentru care se evaluează funcţia;

- x este variabila în care se returnează rezultatul.

Dacă argumentul este o matrice, funcţiile trigonometrice operează asupra fiecărui element.

4.1. Funcţiile trigonometrice directe

Funcţiile trigonometrice directe în MATLAB sunt:

sin Calculează sinusul argumentului;

cos Calculează cosinusul argumentului;

tan Calculează tangenta argumentului;

cot Calculează cotangenta argumentului;

sec Calculează secanta argumentului;

cosec Calculează cosecanta argumentului.

Pentru argumente numere complexe, z=x+iy, relaţiile de calcul sunt: sin(z)=sin(x)cosh(y)+i* cos(x)sinh(y); cos(z) = cos(x)cosh(y) – i*sin(x)sinh(y); tan(z) = sin(z)/cos(z)

Exemplul 15. Să se calculeze funcţiile trigonometrice directe ale elementelor vectorului: $[\pi/4 \quad 3\pi/4 \quad 5\pi/4]$

4.2. Funcţiile trigonometrice inverse

Funcţiile trigonometrice inverse în MATLAB sunt:

asin Calculează arcsinusul argumentului;

acos Calculează arccosinusul argumentului;

atan Calculează arctangenta argumentului;

atan2 Calculează arctangenta unui argument complex;

acot Calculează arccotangenta argumentului;

asec Calculează arcsecanta argumentului;

acsc Calculează arccosecanta argumentului.

Exemplul 16. Să se calculeze funcţiile trigonometrice inverse pentru elementele vectorului: X = [0 1 $\sqrt{2}/2$]

4.3. Funcţiiie hiperbolice

Funcţiile hiperbolice se apelează cu sintaxa: x=nume_funcţie(argument), unde:

- nume_funcţie este numele uneia dintre funcţiile hiperbolice de mai jos;

- argument este valoarea pentru care se evaluează funcţia;

- x este variabila în care se returnează rezultatul.

Dacă argumentul este o matrice, funcţiile trigonometrice operează asupra fiecărui element.

4.3.1. Funcţiiie hiperbolice directe

Funcţiile hiperbolice directe în MATLAB sunt:

sinh Calculează sinusul hiperbolic al argumentului.

cosh Calculează cosinusul hiperbolic al argumentului.

tanh Calculează tangenta hiperbolică a argumentului.

coth Calculează cotangenta hiperbolică a argumentului.

sech Calculează secanta hiperbolică a argumentului.

csch Calculează cosecanta hiperbolică a argumentului.

4.3.2. Funcţiiie hiperbolice inverse

Funcţiile hiperbolice inverse în MATLAB sunt

asinh Calculează arcsinusul hiperbolic al argumentului;

acosh Calculează arccosinusul hiperbolic al argumentului;

atanh Calculează arctangenta hiperbolică a argumentului;

acoth Calculează arccotangenta hiperbolică a argumentului;

asech Calculează arcsecanta hiperbolică a argumentului;

acsch Calculează arccosecanta hiperbolică a argumentului.

Teme:

1) Se vor rula toate instrucţiunile şi exemplele din cadrul cursului.

2) Fie matricea:

$$A=\begin{bmatrix} |-1| & \sqrt{2} & \ln 8 \\ 2 & e^{-3} & 34 \\ 9 & 3 & \sqrt[3]{8} \end{bmatrix}$$

Sa se scrie un program Matlab care realizeaza:

a) introducerea matricei A;

b) calculul matricei $B=\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix}$

c) calculul determinantului matricei B.

3) Să se realizeze un program Matlab care calculează suma elementelor pare ale vectorului:
a=[1 2 4 6 7 9 10 11]
4) Să se realizeze un program Matlab care calculează suma elementelor impare ale vectorului:
a=[1 2 4 6 7 9 10 11]