Florina Maria ŞERDEAN

Iuliana Fabiola MOHOLEA

Radu Mircea MORARIU-GLIGOR

PROGRAMARE ÎN LIMBAJUL MATLAB CU APLICAŢII ÎN INGINERIE MECANICĂ

*

Volumul I

UTPRESS
Cluj-Napoca, 2021
ISBN 978-606-737-529-9

FLORINA MARIA ȘERDEAN

IULIANA FABIOLA MOHOLEA

RADU MIRCEA MORARIU-GLIGOR

PROGRAMARE ÎN LIMBAJUL MATLAB CU APLICAŢII ÎN INGINERIE MECANICĂ

*

Volumul I

UTPRESS
Cluj - Napoca, 2021
ISBN 978-606-737-529-9

Editura U.T.PRESS
Str. Observatorului nr. 34
400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: ing. Călin Câmpean

Recenzia: Prof.Dr.Ing. Tiberiu Alexandru Antal

Conf.Dr.Ing. Ovidiu Aurelian Deteșan

ISBN 978-606-737-529-9
Bun de tipar: 28.09.2021

Cuprins:

Prefață

„Toată lumea din această țară ar trebui să învețe să programeze un computer, pentru că te învață să gândești”.

Steve Jobs

Într-adevăr, învățarea unui limbaj de programare dezvoltă mult gândirea logică a unei persoane. Oamenii care pot gândi logic sunt capabili să analizeze problemele și să elaboreze soluții. Acest lucru nu este valoros doar atunci când se dezvoltă programe, ci este vital pentru orice situație care necesită o gândire rațională. Iar inginerii au nevoie de această aptitudine intelectuală pe lângă gândirea tehnică.

Cartea se adresează studenților de la specializările facultăților cu profil mecanic din cadrul Universității Tehnice din Cluj-Napoca și își propune să reprezinte un instrument util în activitatea de înțelegere și învățare a unor algoritmi de programare, precum și a limbajului MATLAB.

Această carte adoptă o abordare hibridă, introducând atât programarea, cât și utilizările eficiente ale acesteia. Provocarea pentru studenți constă în imposibilitatea de a prevedea dacă vor trebui, de fapt, să cunoască conceptele de programare mai târziu sau dacă un pachet software, cum ar fi MATLAB, va fi suficient pentru cariera lor. Prin urmare, considerăm că cea mai bună abordare este să le oferim studenților amândouă: atât conceptele de programare cât și funcțiile eficiente, dar specifice limbajului de programare. Deoarece MATLAB este foarte ușor de utilizat, este o platformă perfectă pentru această abordare a predării programării și rezolvării de probleme.

Prin structura sa, această carte prezintă în mod gradual atât noțiunile cât și problemele, pornind de la simplu spre complex. Lucrarea este structurată în opt capitole, urmate de bibliografie și două anexe care prezintă pașii de instalare ai mediilor de programare Octave, respectiv MATLAB.

Primul capitol include o prezentare compactă a limbajului de programare MATLAB, accentuând elementele necesare scrierii unor programe de complexitate scăzută sau medie în acest limbaj de programare.

Al doilea capitol prezintă elementele care stau la baza reprezentării unui algoritm, atât prin schemă logică, cât și prin limbaj pseudocod.

Al treilea capitol conține o serie de probleme de complexitate redusă, probleme care parcurse succesiv permit asimilarea treptată a cunoștințelor necesare studenților pentru a înțelege cum se abordează o problemă, cum se stabilesc datele de intrare, de ieşire, şi cele intermediare, respectiv cum se construieşte un algoritm şi cum se obţine soluția.

Capitolul patru este dedicat rezolvării problemelor cu funcţii, relaţia de definiţie a funcţiei depinzând de evaluarea unor condiţii. De asemenea, calculul valorilor funcţiei poate fi realizat pentru un interval cunoscut, parcurs cu un pas cunoscut.

Capitolul cinci tratează prelucrarea elementelor tablourilor unidimensionale, fiind prezentate o serie de probleme considerate clasice: sume, produse, numărare de elemente care îndeplinesc o anumită cerinţă, ordonarea elementelor, căutarea, inserarea sau eliminarea unor elemente, probleme cu cifrele unui număr natural. De asemenea, sunt prezentate probleme cu o complexitate mai ridicată, dar foarte utile, cum ar fi: verificarea CNP-ului, verificarea codului de pe card-ul bancar sau a codului ISBN.

În capitolul şase al lucrării sunt prezentate probleme legate de prelucrarea elementelor tablourilor bidimensionale (matrice): sume, produse, numărarea unor elemente care îndeplinesc o anumită cerinţă, determinarea elementului maxim sau minim, eliminarea sau inserarea unor linii sau coloane. De asemenea, sunt tratate matricele pătratice, precum şi generarea unor matrice particulare, ale căror elemente se obţin pe baza unor anumite reguli.

Capitolul şapte tratează probleme legate de şiruri de numere reale, serii de numere şi dezvoltări în serie de puteri.

Ultimul capitol conţine o colecţie de probleme tehnice din geometrie, mecanică tehnică (statică, cinematică şi dinamică), organe de maşini sau rezistenţa materialelor. Aceste probleme ilustrează în mod foarte sugestiv modul în care o serie de probleme tehnice pot fi rezolvate prin scrierea unor programe într-un anumit limbaj de programare.

Considerăm că programarea se învață în general prin exemple. De aceea, această lucrare abundă în exemple sugestive și probleme rezolvate pas cu pas, atât probleme clasice de programare cât și probleme de natură tehnică.

Autorii

Capitolul 1. Limbajul de programare MATLAB

1.1. Introducere

1.1.1. Etapele de rezolvare a unei probleme tehnice cu ajutorul limbajului de programare MATLAB

A. Stabilirea datelor inițiale (de pornire) și a modelului matematic pentru problema de rezolvat;
B. Întocmirea unui algoritm pentru rezolvarea problemei (eventual schema logică);
C. Scrierea programului sursă utilizând limbajul de programare MATLAB, cu ajutorul **editorului de texte** încorporat. Fişierul rezultat se numește **fișier script** și va avea extensia **.m**;
D. Lansarea în execuţie a programului (rularea programului).

MATLAB este de fapt un **mediu de dezvoltare** (engl. *development environment*, sau *integrated development environment* – „mediu integrat de dezvoltare"), asistând programatorul în scrierea altor programe. Practic toți pașii necesari creării unui program (ex.: editarea codului sursă, rularea, depanarea, generarea de documentație) sunt incluși în același soft, care oferă și o interfață grafică, prietenoasă cu utilizatorul.

Trebuie menționat faptul că există și probleme tehnice, sigur de o complexitate mai redusă, care se pot rezolva în fereastra de comenzi a mediului de programare MATLAB. Această fereastră are o linie de comandă în care se pot defini variabile sau constante, sau se pot introduce diferite comenzi pentru lucrul cu variabilele definite. Simbolul liniei de comandă, numit prompter, este **>>**.

1.1.2. Tipuri de limbaje de programare. Clasificare.

Limbajul de programare reprezintă *un sistem de convenții adoptate pentru realizarea unei comunicări între programator și calculator.*

Limbajele de programare au asemănări cu limbajele naturale, astfel ele sunt compuse din: cuvinte rezervate, punctuație, propoziții și fraze, reguli sintactice, etc. Așa cum, pentru învățarea unei limbi străine este necesar să se învețe cuvintele acesteia și regulile prin intermediul cărora acestea pot fi manevrate, pentru învăţarea unui limbaj de programare trebuie studiate cuvintele și semnele care îl compun, precum şi ansamblul de reguli pentru manevrarea acestora.

După modul în care este conceput ansamblul de reguli de comunicare, limbajele de programare se clasifică astfel:

- *Limbaje de nivel scăzut – nivel înalt:*
  Nivelul unui limbaj este dat de poziţia pe care acesta îl ocupă pe o scară de la nivelul cunoscut de microprocesor (limbaj maşină) şi până la limbajul natural al programatorului (limba engleză, limba română etc). Un limbaj de nivel scăzut este un limbaj foarte apropiat de maşină şi care lucrează cu elemente de nivel hardware, cum ar fi: regiştrii, microprocesor, locaţii de memorie, porturi de intrare/ieşire etc. Un limbaj de nivel înalt utilizează concepte apropiate de limbajul natural, concepte de nivel logic, cum ar fi: structuri de date, nume de operaţii, variabile, constante etc.
  O deosebire extrem de importantă între cele două tipuri de limbaje de programare o constituie portabilitatea acestora, adică posibilitatea transferării programelor pe un alt tip de platformă decât cea pe care au fost realizate. Limbajele de nivel scăzut sunt neportabile, deoarece sunt legate de tipul procesorului utilizat de mașină și de sistemul de operare. În ceea ce privește limbajele de nivel înalt, acestea permit transferul programelor deoarece între program şi calculator se interpune compilatorul care rezolvă transformarea fișierului sursă în fișier executabil, ținând cont de caracteristicile mașinii.
- *Limbaje procedurale – neprocedurale:*
  Cele două tipuri de limbaje se deosebesc prin nivelul de organizare (structurare) a programelor. În cazul limbajele neprocedurale programele sunt gândite la nivel de instrucţiune, pe când la cele procedurale programatorul

concepe programele la nivel de bloc de instrucţiuni. În limbajele procedurale programele sunt scrise instrucţiune cu instrucţiune, însă ele sunt organizate logic în blocuri (grupuri de instrucţiuni) care realizează acţiuni specifice.

- *Limbaje orientate sau de uz general:*
  Din acest punct de vedere, limbajele pot fi orientate pe o anumită problemă sau pot fi concepute pentru soluţionarea oricărui tip de problemă.
- *Limbaje concurente:*
  Un limbaj concurent permite definirea de procese (prelucrări) care se desfăşoară în paralel, la un anumit moment execuţia putându-se ramifica. În cazul limbajelor neconcurente (aşa cum sunt majoritatea limbajelor) execuţia are o desfăşurare liniară, la un moment dat fiind rulat un singur proces.

Un exemplu de limbaj de nivel scăzut este limbajul de asamblare. Limbaje de nivel înalt sunt: BASIC (**B**eginner's **A**llpurpose **S**ymbolic **I**nstruction **C**ode - Cod de instrucţiuni simbolice, de uz general, destinat începătorilor), FORTRAN (**FOR**mula **TRAN**slation), PASCAL (numele acestui limbaj provine de la matematicianul şi filosoful BLAISE PASCAL, în semn de recunoaştere a contribuţiilor sale în conceperea maşinilor de calcul), ADA, precum şi limbajul C și limbajul MATLAB.

1.1.3. Limbajul MATLAB. Scurt istoric.

Primul MATLAB nu era de fapt un limbaj de programare. Era un simplu calculator interactiv cu matrice. Nu permitea scrierea de programe, nu avea pachete de instrumente și nici nu permitea reprezentare grafică. Baza matematică pentru prima versiune de MATLAB a fost o serie de lucrări de cercetare realizate de JH Wilkinson și 18 dintre colegii săi, publicate între 1965 și 1970 și colectate ulterior în Handbook for Automatic Computation, Volumul II, Linear Algebra, editat de Wilkinson și C. Reinsch. Aceste lucrări prezintă algoritmi, implementați în Algol 60, pentru rezolvarea ecuațiilor liniare cu matrice și a problemelor cu valori proprii.

În 1970, un grup de cercetători de la Laboratorul Național Argonne a propus Fundației Naționale a Științei din SUA (NSF) să „exploreze metodologia, costurile și resursele necesare pentru a produce, testa și disemina software matematic de înaltă calitate și pentru a testa, certifica, difuza și susține pachete de programe matematice în anumite domenii cu probleme." Grupul a dezvoltat EISPACK (pachetul Matrix Eigensystem) prin traducerea procedurilor Algol pentru problemele legate de valoarea proprie din manual în Fortran și lucrând pe larg la testare și portabilitate. Prima versiune a EISPACK a fost lansată în 1971 și a doua în 1976.

Jack Dongarra, Pete Stewart, Jim Bunch și Cleve Moler au propus în 1975 către NSF un alt proiect de cercetare care să investigheze metodele de dezvoltare ale software-ului matematic. Un produs secundar a fost software-ul în sine, supranumit LINPACK, pentru Linear Equation Package. Acest proiect a fost, de asemenea, centrat la Argonne. LINPACK, așa cum a fost denumit inițial, își are originea în Fortran fără a fi implicat traducere din Algol. S-au conceput 44 de subrutine în fiecare din cele patru precizii numerice. Dintr-un punct de vedere se poate spune ca proiectele LINPACK și EISPACK au fost eșecuri. Aceste proiecte de cercetare propuse către NSF au avut ca si obiective de a „explora metodologia, costurile și resursele necesare pentru a produce, testa și disemina software matematic de înaltă calitate". Ceea ce s-a propus a fost doar software-ul in sine, fără a se scrie niciodată un raport sau o lucrare care să abordeze aceste obiective.

În anii 1970 și începutul anilor 1980, Cleve Moler, preda algebră liniară și analiză numerică la Universitatea din New Mexico. În aceasta perioadă a apărut dorința ca studenților săi să li se faciliteze accesul la LINPACK și EISPACK fără să scrie programe Fortran. Mai exact dorea ca ei sa nu treacă prin procesul repetat de editare-compilație-încărcare-executare care era de obicei necesar pe computerul central al campusului.

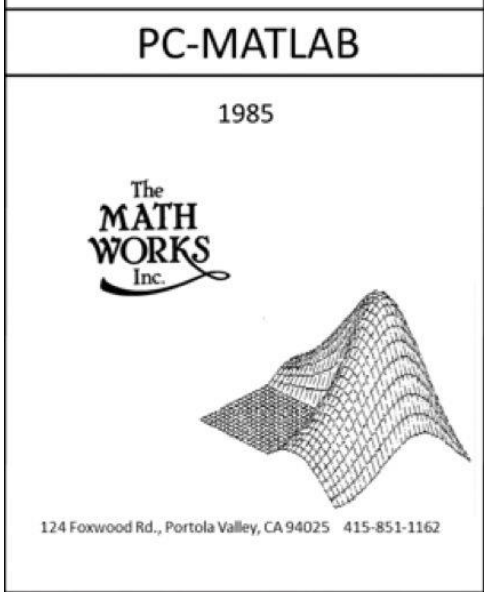
În încercarea de a atinge acest scop, acesta a studiat cartea lui Niklaus Wirth Algoritmi + Structuri de date = Programe și de asemenea a învățat în particular cum să analizeze limbajele de programare. Primul MATLAB - un acronim pentru Matrix Laboratory - a fost scris în Fortran, singurul tip de date fiind matricea. Sigur, acest proiect era un fel de hobby, profesorul având posibilitatea de a învăța programare și în același timp de a crea ceva care îi ajuta foarte mult pe studenții săi. Aceste începuturi au fost fără a concepe un plan de afaceri și fără vreun sprijin formal din afară.

```
        < M A T L A B >
     Version of 05/12/1981
<>

 The functions and commands are...
 ABS   ATAN  BASE  CHAR  CHOL  CHOP  COND  CONJ
 COS   DET   DIAG  DIAR  DISP  EIG   EPS   EXEC
 EXP   EYE   FLOP  HESS  HILB  IMAG  INV   KRON
 LINE  LOAD  LOG   LU    MAGI  NORM  ONES  ORTH
 PINV  PLOT  POLY  PRIN  PROD  QR    RAND  RANK
 RAT   RCON  REAL  ROOT  ROUN  RREF  SAVE  SCHU
 SIN   SIZE  SQRT  SUM   SVD   TRIL  TRIU  USER
 CLEA  ELSE  END   EXIT  FOR   HELP  IF    LONG
 RETU  SEMI  SHOR  WHAT  WHIL  WHO   WHY
```

Acest prim MATLAB a fost doar un calculator matricial interactiv. În ecranul de pornire acesta arăta toate cuvintele și funcțiile rezervate în număr de 71. Orice altă funcție putea fi adăugată obținând codul sursă de la profesor, scriind un subprogram Fortran, adăugând numele funcției în tabelul de analiză și recompilând MATLAB.

Profesorul a petrecut anul universitar 1979-80 la Stanford, unde a predat cursul postuniversitar de analiză numerică și a prezentat clasei acest computer matricial. În contextul în care se studiau materii precum teoria controlului și prelucrarea semnalului, matricele fiind esențiale pentru matematică la aceste discipline, MATLAB s-a dovedit util studenților.



Jack Little, un absolvent al universității Stanford, în particular al programului de inginerie a adoptat MATLAB pentru propria sa lucrare. Un prieten de-al său care a urmat cursul profesorului i-a arătat MATLAB, care i s-a dovedit a fi util. În 1983, Little a venit cu ideea de a crea un produs comercial bazat pe MATLAB. PC-ul IBM® a fost introdus doar cu doi ani mai devreme și abia s-a dovedit a fi suficient de puternic pentru a rula un program precum MATLAB, dar Little a anticipat evoluția acestuia. După ce și-a părăsit slujba, a cumpărat o clonă de computer Compaq® la Sears, s-a mutat pe dealurile din spatele universității Stanford și a scris o versiune nouă și extinsă a MATLAB în C. În tot acest timp a fost încurajat de profesor. Un prieten, Steve Bangert, de asemenea a lucrat la noul MATLAB în timpul lui liber.

PC-MATLAB a debutat în decembrie 1984 la Conferința IEEE privind decizie și control din Las Vegas. Pro-MATLAB, pentru stațiile de lucru Unix, a urmat un an mai târziu. Little și Bangert au făcut multe modificări și îmbunătățiri importante la MATLAB-ul istoric atunci când au creat versiunea nouă și extinsă. Cele mai semnificative au fost funcțiile, toolbox-urile și grafica.

În timp ce și-a păstrat rădăcinile în matematica matricială, MATLAB a continuat să evolueze pentru a satisface nevoile în schimbare ale inginerilor și oamenilor de știință. Soluția numerică a ecuațiilor diferențiale obișnuite a fost o parte vitală a MATLAB încă de la începuturile sale comerciale. ODE-urile sunt nucleul Simulink®, produsul însoțitor MATLAB pentru simulare și proiectare bazată pe modele. Timp de mulți ani, MATLAB a avut un singur tip de date numerice: IEEE standard 754 cu virgulă dublă de precizie, stocate în formatul pe 64 de biți. Pe măsură ce oamenii au început să folosească MATLAB pentru mai multe aplicații și seturi de date mai mari, am oferit mai multe modalități de reprezentare a datelor. Suportul pentru aritmetica cu precizie simplă a început la începutul anilor 2000 și a fost completat de MATLAB 7 în 2004. Necesitând doar 32 de biți de stocare, reprezentarea în precizie simplă reduce cerințele de memorie pentru matricele mari. MATLAB nu necesită declararea variabilelor, deci cele în simplă precizie sunt obținute prin funcții de conversie executabile. MATLAB 7 a introdus, de asemenea, trei tipuri de date întregi fără semn, uint8, uint16 și uint32; trei tipuri de date întregi cu semn, int8, int16 și int32; și un tip de date logice, logic.

Matricele rare au fost introduse cu MATLAB 4 în 1992. Acestea sunt o modalitate eficientă de folosire a memoriei pentru a reprezenta tablouri foarte mari, care au puține valori diferite de zero. Sunt stocate doar elementele diferite de zero, împreună cu indicii de rând și indicatorii către începutul coloanelor. Singura modificare a aspectului exterior al MATLAB este o pereche de funcții și aproape toate operațiile se aplică și matricelor rare. Șirurile de celule au fost introduse cu MATLAB 5 în 1996. Un șir de celule este o colecție indexată, posibil eterogenă de obiecte MATLAB, inclusiv alte șiruri de celule, care sunt create folosind acolade. Tot în 1996 sunt introduse și structurile și notațiile „punct” asociate.

În 2008 au fost făcute îmbunătățiri majore ale capacităților de programare orientată pe obiecte în MATLAB. Crearea de clase poate simplifica sarcinile de programare care implică structuri de date specializate sau un număr mare

de funcții care interacționează cu anumite tipuri de date. Clasele MATLAB acceptă „overloading” pentru funcții și operatori, accesul controlat la proprietăți și metode, semantică pentru referințe și valori, evenimente și „listeners”. Un exemplu mare și complex al abordării orientate pe obiecte a programării MATLAB este sistemul grafic.

Primele versiuni ale MATLAB au fost aplicații terminale simple. De-a lungul timpului s-au adăugat ferestre separate pentru grafică, editare și alte instrumente. Acestea au făcut treptat MATLAB mai ușor de utilizat, în special pentru utilizatorii fără experiență prealabilă în programare. Două caracteristici specifice care au avut cel mai mare impact sunt programul desktop și editorul live. Programul desktop MATLAB a fost introdus în 2000, iar editorul live a fost introdus în 2016 și încă evoluează rapid. În acesta din urmă textul descriptiv și datele de intrare, datele de ieșire și grafica MATLAB sunt combinate într-un singur document interactiv care poate fi exportat în HTML, PDF sau LaTeX.

Colecția de unelte de calcul paralel a fost introdusă la conferința SuperComputing din 2004. Anul următor, la SC05, Bill Gates a ținut discursul principal, folosind MATLAB pentru a demonstra intrarea Microsoft în calculul de înaltă performanță. Colecția de instrumente acceptă paralelismul memoriei distribuite, cu granulație grosieră, executând mai multe procese MATLAB pe mai multe mașini dintr-un cluster sau pe mai multe nuclee dintr-o singură mașină. Colecția de instrumente acceptă, de asemenea, paralelism de memorie partajată cu granulație fină în unitățile de procesare grafică atașate (GPU-uri).

O mare parte din puterea MATLAB-ului modern provine din seturile de instrumente disponibile pentru aplicații specializate. Începând cu versiunea 2018a, există 63 dintre ele:

- Implementare aplicație (3)
- Generarea codului (7)
- Biologie Computațională (2)
- Finanțe computaționale (8)
- Sisteme de control (8)
- Acces la baze de date și raportare (2)
- Prelucrarea imaginilor și viziunea computerizată (6)
- Matematică, statistică și optimizare (9)
- Calcul paralel (2)
- Prelucrare semnal și comunicații fără fir (11)
- Test și măsurare (5)

MATLAB a parcurs un drum lung de la calculatorul simplu care a început totul. Este un ecosistem viu care susține toate aspectele legate de calculul tehnic. Intenția dezvoltatorilor este de continua să consolideze caracteristicile existente pe măsură ce adaugă cu atenție altele noi. Obiectivele lor sunt întotdeauna ușurința utilizării, puterea și viteza.

1.1.4. Prezentare generală a mediului de programare MATLAB

În interfața aplicației MATLAB se disting o serie de elemente principale (Figura 1.1):

1 – meniul principal și bara cu unelte;

2 – browser-ul de fișiere: permite vizualizarea directoarelor și a fișierelor curente;

3 – fereastra de comenzi: permite introducerea comenzilor, definirea variabilelor sau a constantelor, precum și definirea și apelarea funcțiilor;

4 – fereastra de lucru: în această fereastră apar toate variabilele utilizate;

5 – fereastra de istoric: ilustrează în ordine cronologică toate comenzile introduse precum şi rezultatele executării acestora. Cele mai noi comenzi sunt cele din partea inferioară, iar cele mai vechi sunt cele din partea de jos.

Spre deosebire de alte limbaje de programare, limbajul MATLAB nu necesită definirea variabilelor înainte de folosirea lor, iar elementele de bază utilizate sunt matricele, operațiile fiind adaptate pentru acestea, fapt care face codul să fie ușor de citit și de executat. Instrucțiunile pot fi scrise direct în **linia de comandă**, în fișiere **script** sau în fișiere de tip **funcție**.

În general, problemele mai ușoare sau calculele rapide se pot efectua direct în linia de comandă din fereastra de comenzi. Se scriu comenzile dorite (atribuiri de valori, expresii matematice etc.) direct după simbolul **>>** al prompter-ului

liniei de comenzi și apoi se apasă tasta ENTER, rezultatul afișându-se imediat tot în linia de comandă (dacă afișarea acestuia nu este suprimată).

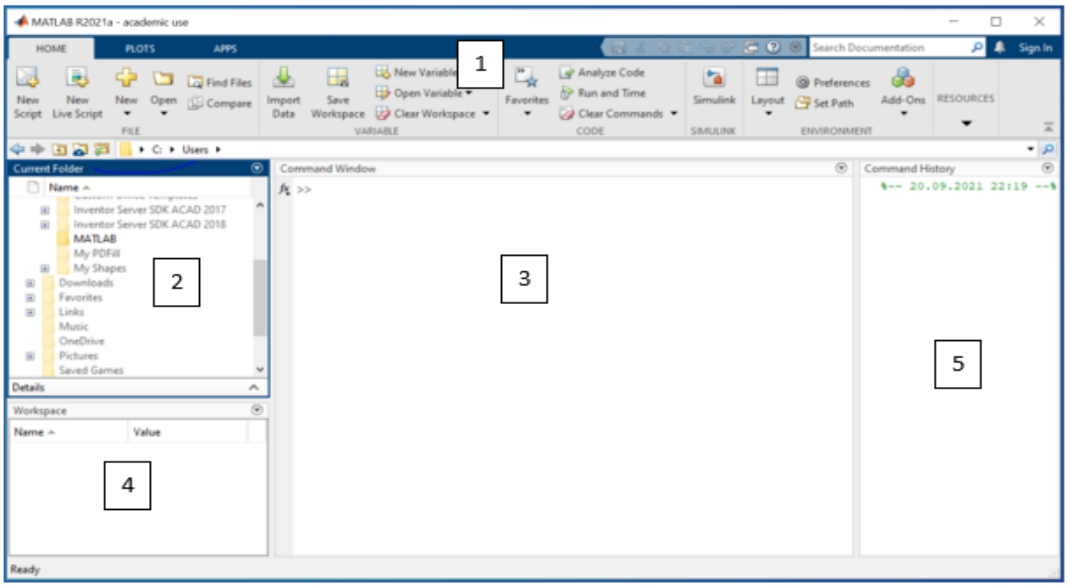


Figura 1.1.

Fișierele script sunt fișiere de tip text (cu extensia **.m**) care cuprind comenzi MATLAB. Atunci când un astfel de fișier este lansat în execuție MATLAB citește și execută comenzile ca și cum fiecare ar fi fost scrisă secvențial în linia de comandă. De aceea, variabilele declarate și/sau folosite în cadrul programelor scrise în fișiere script se regăsesc în spațiul de lucru ("Workspace") după execuția fișierului script.

Fișierele de tip funcție sunt fișiere text (tot cu extensia **.m**) și reprezintă un modul de program care realizează o anumită acţiune. Fiecare funcţie este definită folosind cuvântul cheie *function*, are un nume și poate returna un vector de valori. Numele unei funcţii este însoţit întotdeauna de paranteze rotunde între care se pot trece argumentele funcţiei separate prin virgulă (acestea pot lipsi). O astfel de funcţie nu poate fi rulată asemenea fișierelor script, ci poate doar să fie apelată într-un program sau în linia de comandă, precizând la apel: numele funcţiei și parametrii efectivi ai funcţiei (plasaţi între paranteze rotunde).

1.1.5. Declarații de variabile

Programele lucrează cu date, acestea fiind păstrate în memoria RAM (Randomize Access Memory). În general, în limbajele de programare, toate variabilele care apar în program trebuie declarate înainte de utilizarea acestora. În MATLAB însă declararea unei variabile se face implicit la crearea acesteia (prin atribuirea unei valori atunci când este folosită sau prin exprimarea ei, de exemplu, ca funcție de alte variabile). Variabilei i se alocă spațiu în memoria RAM, care este formată din mai multe locații de memorie consecutive, fiecare având o adresă distinctă. Dimensiunea unei locații este de 8 biți – 1 octet (byte). În fereastra de „Workspace" pe lângă numele și valoarea unei variabile, se poate observa și pe câți octeți este reprezentată aceasta.

Tipuri de date: În memoria RAM datele sunt reprezentate, indiferent de valoarea pe care o conțin, ca succesiuni de cifre 0 și 1, adică în formă binară. Pentru o interpretare corectă a informației stocate, trebuie precizate unde sunt localizate datele și care este reprezentarea acestora. Datele sunt caracterizate prin trei elemente principale:

Identificatorul (sau numele): reprezintă un nume asociat datei cu scopul de a fi identificată și de a o putea distinge de celelalte date.

Observaţii: *Numele trebuie să înceapă cu o literă, urmată de alte litere, cifre sau de caracterul de subliniere _.;*
Nu este admisă prezenţa spaţiului în numele unei variabile;
Limbajul MATLAB face distincţia între litere mari şi mici.

Limbajul MATLAB are 20 de cuvinte cheie care nu pot fi utilizate ca şi nume de variabile: ***break, case, catch, classdef, continue, else, elseif, end, for, function, global, if, otherwise, parfor, persistent, return, spmd, switch, try, while***. De asemenea, trebuie menționat faptul că în cazul creării unei variabile cu același nume cu al unei funcții (cum ar fi, de exemplu, *mode*, *size* sau *path*), numele variabilei are prioritate față de numele funcției.

Valoarea: reprezintă conţinutul datei. În funcţie de valoarea acestora, datele se deosebesc în: date variabile (numite pe scurt *variabile*) care pe parcursul rulării programului pot să-şi schimbe valoarea şi date constante (numite pe scurt *constante*) care-şi păstrează valoarea pe parcursul rulării programului;
Tipul: se referă la modul de reprezentare a datei în memoria calculatorului şi defineşte şi precizia asigurată de această reprezentare. MATLAB oferă posibilitatea de a lucra cu 11 tipuri de date: date numerice, date de tip caracter (alfanumerice), date de tip dată/timp, date de tip categorie, tabel, orar, structură, șiruri de celule, manipulator de funcție, obiecte de tip hartă și serii de timp.

Implicit, MATLAB stochează toate valorile numerice ca numerele reale în dublă precizie, dar utilizatorul poate alege dintre următoarele tipuri de date numerice predefinite ale limbajului MATLAB, tipuri care sunt specificate cu ajutorul următoarelor cuvinte cheie:

double – date de tip real, în dublă precizie, cu virgulă flotantă;
single – date de tip real, în simplă precizie, cu virgulă flotantă;
int8 – date de tip întreg, cu semn, pe 8 biți;
int16 – date de tip întreg, cu semn, pe 16 biți;
int32 – date de tip întreg, cu semn, pe 32 biți;
int64 – date de tip întreg, cu semn, pe 64 biți;
uint8 – date de tip întreg, fără semn, pe 8 biți;
uint16 – date de tip întreg, fără semn, pe 16 biți;
uint32 – date de tip întreg, fără semn, pe 32 biți;
uint64 – date de tip întreg, fără semn, pe 64 biți.

Pentru datele de tip caracter se pot folosi tipurile specificate folosind cuvintele cheie **char** (pentru șiruri de caractere, acestea putând fi adresate individual) și **string** (pentru șiruri de caractere privite ca un întreg).

MATLAB este un limbaj tastat dinamic, adică tipul variabilelor este interpretat/cunoscut abia la rulare. Astfel, **declararea variabilelor** se realizează prin simpla inițializare cu o anumită valoarea procesată implicit ca dată de tip *double* în cazul datelor numerice. Dacă se dorește ca o anumită variabilă să fie de un alt tip atunci asignarea valorii inițiale se face astfel:

```
var = tip (V) ;
```

unde: **var** este numele variabilei care se inițializează, **tip** este unul din tipurile de date predefinite ale limbajului MATLAB, respectiv **V** este valoarea atribuită sau o altă variabilă deja inițializată.

Inițializarea datelor de tip *char* se poate realiza și folosind forma

```
var = 'sir';
```

iar a celor de tip *string*, folosind

```
var = "sir";
```

unde **sir** este șirul de caractere care se asignează variabilei **var**.

Toate variabilele sunt privite ca tablouri în MATLAB: unidimensionale (şiruri), bidimensionale (matrice) sau cu mai multe dimensiuni, chiar numele programului provenind de la matrice: MATLAB (MATrix LABoratory). Nu este necesară specificarea dimensiunilor unui tablou la inițializare, acestea putând fi modificate pe parcursul programului. Declararea

unui tablou se face prin simpla asignare a unor valori scrise între simbolurile **[ ].** În locul valorilor pot fi variabile deja inițializate.

1.1.6. Constante

Pe lângă variabile, în programe adesea sunt folosite şi constantele, adică date care nu-şi modifică valoarea în cadrul programului. Spre deosebire de alte limbaje de programare, cum este limbajul C, MATLAB nu permite declararea de constante a căror valoare să nu poată fi schimbată de alți utilizatori sau programe. În schimb constantele sunt definite ca și funcții, deci există posibilitatea suprascrierii lor sau a declarării unor variabile cu aceeași nume. Totuși, atunci când se utilizează o dată într-o funcție, valoarea lor nu poate fi suprascrisă în acea funcție.

În MATLAB sunt definite următoarele constante ca funcții:

Denumire	Semnificație
eps	Precizie relativă în virgulă mobilă
flintmax	Cel mai mare număr întreg consecutiv în format cu virgulă mobilă
i	Unitate imaginară
j	Unitate imaginară
Inf	Reprezentarea scalară pentru “Infinit” (de ex. rezultatul pentru 1/0 sau log(0))
pi	Raportul dintre lungimea unui cerc și diametrul său
NaN	Reprezentarea scalară pentru “Not a Number” (de ex. rezultatul pentru 0/0 sau 0*Inf)
isfinite	Returnează 1 pentru valori finite și 0 pentru valori infinite sau NaN
isinf	Returnează 1 pentru Inf și -Inf și 0 pentru celelalte valori
isnan	Returnează 1 pentru NaN și 0 pentru celelalte valori

Pe lângă constantele din tabelul anterior, în MATLAB sunt definite, tot ca funcții, o serie de matrice de test:

Denumire	Semnificație
compan	Matricea companion/Frobenius
gallery	O familie de matrice de test
hadamard	Matricea Hadamard
hankel	Matricea Hankel
hilb	Matricea Hilbert
invhilb	Inversa matrice Hilbert
magic	Pătratul magic
pascal	Matricea Pascal
rosser	Matricea Rosser
toeplitz	Matricea Toeplitz
vander	Matricea Vandermonde
wilkinson	Matricea de testare a valorii proprii a lui Wilkinson

Programul MATLAB permite folosirea unor caractere speciale cu ajutorul cărora se poate realiza inițializarea variabilelor, suprimarea afișării datelor de ieșire sau calcularea unor expresii. Principalele caractere speciale sunt prezentate în tabelul de mai jos împreună cu semnificația lor.

Caracter	Semnificație
%	Se folosește pentru a scrie comentarii în program
=	Se folosește pentru asignare
:	Se folosește la generarea diviziunilor

( )	Se folosește la declararea sau apelarea unei funcții precum și la referirea unor elemente dintr-o matrice sau un vector
[ ]	Se folosește la declararea unei matrice sau a unui vector
[,]	Se folosește pentru a separa elementele dintr-un vector sau de pe aceeași linie dintr-o matrice (poate fi înlocuit cu "spațiu" prin apăsarea butonului SPACE)
[;]	Se folosește pentru a separa liniile unei matrice
,	Se folosește ca separator între instrucțiunile scrise pe aceeași linie cu ecou pe ecran
;	Se folosește ca separator între instrucțiunile fără ecou pe ecran
‘	Se folosește la calculul transpusei unei matrice
.	Se folosește înaintea operatorilor aritmetici pentru a realiza un calcul element cu element

1.2. Funcţii de intrare – ieşire

Limbajul MATLAB nu are instrucţiuni pentru introducerea şi extragerea datelor din program, astfel că acestea se realizează cu ajutorul unor funcţii. În limbajul MATLAB există o singură funcție pentru introducerea datelor (intrare) de la tastatură: **input()** și una pentru citirea datelor din fișier: **fscanf().** Funcţiile pentru extragerea datelor (ieşire) sunt: **disp()**, **display()** și **fprintf()**. Pentru citirea datelor sau afișarea lor într-un anumit format se pot folosi aceste funcții de intrare-ieșire complementar cu funcțiile **sscanf()** și **sprintf()**.

1.2.1. Funcții de intrare

Funcția care realizează efectiv citirea datelor de intrare de la tastatură **input().** Cu ajutorul ei se poate afișa un text în linia de comandă și citi valoarea introdusă de utilizator.

Formele de bază ale acestei funcții sunt:

```
v = input('text')
str = input('text','s')
```

Folosind prima formă se afișează mesajul **text**, se așteaptă până când utilizatorul introduce o valoare în linia de comandă și apasă tasta ENTER. Valoarea introdusă este atribuită variabilei ***v***. Dacă utilizatorul apasă tasta ENTER fără a introduce întâi o valoare, atunci funcția **input()** returnează o matrice vidă. Dacă utilizatorul introduce de la tastatură o expresie nevalidă, atunci MATLAB afișează un mesaj de eroare sugestiv și apoi reafișează mesajul **'text'** și așteaptă ca utilizatorul să introducă o nouă valoare.

La folosirea celei de-a doua forme datele introduse de utilizator nu sunt evaluate ca o expresie. După afișarea mesajului **'text'**, introducerea datelor și apăsarea tastei ENTER, datele introduse sunt convertite în variabilă de tip text și stocate în variabila **str**.

Complementar cu funcția de intrare, se poate folosi și funcția **sscanf()**. Această funcție permite citirea datelor într-un anumit format dintr-un șir de caractere. Cele mai des utilizate forme ale acestei funcții sunt:

```
A = sscanf(str, specFormat)
A = sscanf(str, specFormat, dimA)
[A, nr] = sscanf(...)
```

unde **'formatSpec'** este specificatorul de format ale cărui posibile valori se regăsesc în tabelul de mai jos.

Câmp de tip numeric sau caracter	Specificator de format	Descriere
Număr întreg cu semn	%d	În baza 10
	%i	Implicit, baza este 10. Dacă primele cifre sunt 0x sau 0X, atunci valorile sunt în baza 16. Dacă prima cifră este 0, atunci valorile sunt în baza 8.
Număr întreg fără semn	%u	În baza 10
	%o	În baza 8
	%x	În baza 16
Număr în virgulă flotantă	%f	Valori în virgulă flotantă. Câmpurile de intrare pot conține și *Inf, -Inf, NaN, -NaN*. De asemenea, pot conține numere scrise cu simbolurile '+'/'-' sau numere scrise folosind notația exponențială cu *e* sau *E*. Toți cei trei specificatori tratează câmpurile în același mod.
	%e	
	%g	
Șir de caractere	%s	Citește o succesiune de caractere, până la întâlnirea unui caracter alb, cum ar fi spațiu sau linie nouă.
	%c	Citește un singur caracter, inclusiv caracterele albe. Pentru a citi mai multe caractere deodată se specifică numărul maxim de caractere succesive. De exemplu, %10c citește 10 caractere o dată.

Variabila **str** poate fi șir de caractere ce pot fi adresate individual (**char** array) sau un șir de caractere privit ca un întreg (**string**). Folosind prima formă se citesc datele din șirul **str** și se convertesc în formatul dat de **specFormat**, iar rezultatul este returnat în vectorul **A**. Implicit variabila în care se salvează elementele este un vector coloană. Pentru a salva elementele într-un vector linie sau într-o matrice se poate folosi a doua formă. Aceasta setează întâi dimensiunea vectorului de ieșire ca fiind si **dimA** și apoi citește din **str** și le convertește după formatul specificat prin **specFormat**. Dacă elementele citite sunt mai puține decât dimensiunea vectorului/matricei dată prin **dimA**, atunci completează variabila **A** cu zerouri. A treia formă returnează în plus în variabila **nr** numărul de elemente pe care funcția **sscanf()** le citește cu succes în vectorul **A**.

Atunci când datele de intrare se află într-un fișier, acestea pot fi citite cu ajutorul funcției **fscanf()**. Formele de utilizare a acestei funcții sunt:

```
A = fscanf(IDfis, specFormat)
A = fscanf(IDfis, specFormat, dimA)
[A,nr] = fscanf(___)
```

unde **IDfis** este identificatorul fișierului ce se poate obține cu ajutorul funcției **fopen()** căreia i se transmite ca parametru numele fișierului. După finalizarea citirii fișierul se închide apelând **fclose(IDfis)**.

Prima formă permite citirea datelor dintr-un fișier text deschis în vectorul coloană **A** și interpretează valorile din fișier în conformitate cu formatul specificat de **specFormat**. Funcția **fscanf** reaplică formatul pe întregul fișier și poziționează indicatorul la sfârșitul fișierului. Dacă **fscanf** nu poate potrivi **specFormat** cu datele, citește doar porțiunea care se potrivește și încetează procesarea.

A doua formă citește datele fișierului într-o matrice, **A**, cu dimensiunile date de **dimA** și poziționează indicatorul fișierului după ultima valoare citită. Funcția populează matricea **A** în ordinea coloanelor. Parametrul **dimA** trebuie să fie un număr întreg pozitiv sau să aibă forma [m n], unde m și n sunt numere întregi pozitive.

A treia formă returnează în plus și numărul de câmpuri pe care **fscanf** le citește în **A** și îl stochează în variabila **nr**. Pentru date numerice, acesta este numărul de valori citite. Această sintaxă poate fi utilizată cu oricare dintre argumentele de intrare ale sintaxelor anterioare.

1.2.2. Funcții de ieșire

Afișarea rezultatelor fie în linia de comandă, fie într-un fișier script se poate realiza prin eliminarea caracterului '**;**' de după instrucțiunea de atribuie a unei valori unei variabile. Folosind această modalitate, se afișează și numele variabilei. Pe lângă aceasta, se pot folosi și funcțiile de ieșire **disp()** și **display()**.

Sintaxa funcției **disp()** este:

```
disp(X)
```

unde **X** este un vector.

Această funcție afișează în linia de comandă valoarea variabilei transmise ca parametru **X** indiferent de tipul ei, fără a afișa și numele variabilei. Dacă variabila este un șir sau o matrice vidă, atunci nu se afișează nimic. Dacă variabila **X** reprezintă codul HTML pentru un hyperlink, atunci funcția **disp()** va afișa spre pagina Web.

Funcția **display ()** este funcția apelată implicit atunci când se atribuie unei variabile o valoare fără ca atribuirea să fie urmată de caracterul '**;**'. Sintaxa acestei funcții este:

```
display(X)
```

unde **X** este vector.

Această funcție afișează în linia de comandă atât valoarea variabilei transmise ca parametru, cât și numele variabilei. Dacă se evaluează o expresie care nu este atribuită unei variabile și nu este urmată de '**;**', atunci MATLAB asignează rezultatul variabilei **ans** pe care funcția **display()** o și afișează în linia de comandă. Atunci când la inițializarea unei variabile se specifică și tipul variabilei, atunci funcția **display()** afișează și informații despre tipul valorilor care sunt rezultatul execuției unei instrucțiuni sau expresii.

Spre deosebire de **disp()**, funcția **display()** afișează în cazul variabilelor vide:

- [ ] — pentru tipuri numerice
- "0x0 struct array with no fields." — pentru structuri vide
- "0x0 empty cell array" — pentru șir vid de celule
- "0x0 empty char array" — pentru șir vid de caractere
- "0x0 empty string array" — pentru șir de tip string vid

Se pot afișa valorile mai multor variabile pe aceeași linie prin concatenarea în prealabil a variabilelor folosind operatorul '**[ ]**' și apoi folosind funcția **disp()** sau **display()**. O altă modalitate este folosirea funcției **sprintf()** pentru crearea textului într-un anume format, iar apoi afișarea lui efectivă folosind funcția **disp()** sau **display()**.

Această funcție permite formatarea datelor și convertirea lor într-un șir de caractere. Cea mai des utilizată formă a acestei funcții este:

```
str = sprintf(specFormat, A1,...,An)
```

unde '**specFormat'** este specificatorul de format ale cărui posibile valori se regăsesc în tabelul inclus în prezentarea funcției **sscanf()**. Practic se formatează datele din șirurile **A1,..,An** conform formatului ales '**specFormat'** și se returnează textul rezultat în variabila **str**. Afișarea efectivă se realizează prin folosirea funcțiilor **disp()** sau **display()** sau prin eliminarea caracterului '**;**' de după instrucțiune. Între specificatorii de format din **specFormat** şi parametrii **A1**,..., **An** trebuie să existe o concordanță de număr, ordine şi tip. În caz contrar poate apărea una din situațiile următoare: se face o conversie a datelor la ieșire, se afișează datele incomplet, se semnalează o eroare.

Atunci când datele de ieșire se doresc a fi afișate într-un fișier, se poate folosi funcția **fprintf()**. Formele de utilizare a acestei funcții sunt:

```
fprintf(IDfis,specFormat,A1,...,An)
   fprintf(specFormat,A1,...,An)
      nocteti = fprintf(___)
```

unde **IDfis** identificatorul fișierului în care se dorește să se facă afișarea și este returnat la apelarea funcției **fopen()**.

La folosirea primei forme se formatează datele din matricele **A1,..,An** în ordinea dată de coloane conform formatului ales **'specFormat'** (care se definește la fel ca la **sscanf()**) și se afișează în fișierul data de **IDfis**. Spre deosebire de prima formă, a doua formă a funcției **fprintf()** formatează datele din matricele **A1,..,An** conform formatului ales **'specFormat'** în ordinea dată de coloane și le afișează în linia de comandă. A treia formă returnează în variabila **nocteti** numărul de octeți pe care **fprintf()** îl scrie, folosind oricare dintre argumentele de intrare din sintaxele precedente.

1.3. Expresii, operanzi, operatori

1.3.1. Introducere

Orice **expresie** conține **operanzi** (date) şi **operatori**. În urma evaluării unei expresii rezultă o **valoare** şi un **tip**. Valoarea şi tipul expresiei depind de tipul operanzilor folosiți şi de valoarea acestora, precum şi de ordinea de efectuare a operaţiilor.

Expresiile se împart în trei tipuri principale: **matematice** (au ca rezultat date numerice de tip întreg sau real), **caracter** (au ca rezultat date de tip *char* sau *string*), respectiv **logice** (au ca rezultat **1**, care reprezintă valoarea de adevăr **adevărat** sau **0**, care reprezintă valoarea de adevăr **fals**).

Operanzii sunt date asupra cărora se efectuează **operaţii**. Pot fi:

- **constante** (numere);
- **variabile**;
- **elemente din tablouri sau structuri**;
- **apeluri de funcţii**;
- **subexpresii** (expresii cuprinse între paranteze).

Ca și în alte limbaje de programare, și în MATLAB, la evaluarea unei expresii trebuie avute în vedere următoarele aspecte: prioritatea operatorilor din clase de prioritate diferite și asocierea operatorilor de aceeași prioritate.

Prioritatea operatorilor indică modul în care se evaluează expresiile cu doi sau mai mulți operatori aplicați unuia sau mai mulți operanzi, în situația în care nu se folosesc paranteze care să impună o anumită ordine a operațiilor.

Asociativitatea ne arată cum se evaluează o expresie în care apare același operator de mai multe ori.

În cazul în care în expresii cu operatori aritmetici și cu operanzi numerici de tipuri diferite, unul din operanzi este de tip întreg, MATLAB returnează întotdeauna un rezultat de tip întreg, spre deosebire de alte limbaje unde rezultatul evaluării expresiei va avea tipul cel mai cuprinzător. De asemenea, trebuie menționat că în MATLAB asupra variabilelor de tip întreg nu pot fi aplicați operatori aritmetici dacă variabilele nu aparțin aceleași clase.

Evident, dacă într-o expresie apar numai operanzi de același tip, rezultatul va avea tipul comun al operanzilor. Deoarece fiecare tip de date are un anumit domeniu de valori, rezultatul aplicării unui operator este bine să se încadreze în limitele tipului obținut. În cazul în care una din limite este depășită, MATLAB rotunjește rezultatul obținut la valoarea maximă/minimă a tipului respectiv.

Regulile de prioritate pentru operatorii MATLAB sunt afișate în acest tabel, ordonate de la cel mai înalt nivel de prioritate la cel mai mic nivel de prioritate:

Operatori	Semnificație	Asociativitatea
( )	Apel de funcție	de la stânga la dreapta
.' ^ .^ '	Transpusa Ridicarea la putere Ridicarea la putere (element cu element) Transpusa conjugată complexă	de la stânga la dreapta
.^- .^+ .^~	Ridicarea la putere cu minus unar Ridicarea la putere cu plus unar Ridicarea la putere cu negație	de la dreapta la stânga

^- ^+ ^~	Ridicarea la putere cu minus unar (element cu element) Ridicarea la putere cu plus unar (element cu element) Ridicarea la putere cu negație (element cu element)	
+ - ~	Plus unar Minus unar Negație logică	de la stânga la dreapta
* / \ .* ./ .\	Înmulțire Împărțire la dreapta Împărțire la stânga Înmulțire (element cu element) Împărțire la dreapta (element cu element) Împărțire la stânga (element cu element)	de la stânga la dreapta
+ -	Adunare Scădere	de la stânga la dreapta
:	Operatorul două puncte	de la stânga la dreapta
< <= > >= == ~=	Mai mic Mai mic sau egal Mai mare Mai mare sau egal Egal Diferit	de la stânga la dreapta
&	Și (aplicat pe orice variabile)	de la stânga la dreapta
|	Sau (aplicat pe orice variabile)	de la stânga la dreapta
&&	Și (aplicat expresiilor boolene)	de la stânga la dreapta
||	Sau (aplicat expresiilor boolene)	de la stânga la dreapta

1.3.2. Operatorul de atribuire

Este format din semnul egal "**=**". Forma generală este:

```
v = expresie
```

unde: **v** este o variabilă simplă sau un element de tip tablou, structură etc.

expresie poate conține un singur număr sau operanzi și operatori sau poate fi un apel de funcție care returnează o valoare sau un tablou.

Deoarece operatorul de atribuire are prioritate mică (penultima clasă de prioritate) este posibil ca parantezele rotunde între care este încadrată expresia **expresie** să nu fie necesare.

Rezultatul atribuirii are valoarea expresiei **expresie** și tipul variabilei **v**. În cazul în care tipul expresiei diferă de tipul variabilei la atribuire se face o conversie de tip a valorii expresiei la tipul variabilei **v**.

Spre deosebire de alte limbaje, un se admit scrieri de forma: **v1 = v2 = v3 = v4 = expresie,** iar operatorul de atribuire nu poate fi combinat cu alți operatori.

1.3.3. Operatori aritmetici

Pot fi **unari** (adică se aplică unui singur operand) sau **binari** (adică se aplică la doi operanzi).

Operatorii aritmetici unari sunt: **+** și **–**, aceștia se utilizează pentru indicarea semnului unui operand. Operatorul unar **+** nu are nici un efect. Operatorul unar **–** are ca efect schimbarea semnului valorii operandului pe care îl precede.

Operatorii binari includ: **adunarea (+)**, **scăderea (–)**, **înmulțirea (*, .*)**, **împărțirea (/, \, ./, .\ ), ridicarea la putere (^,.^)** și **transpusa (.', ')**. După cum se observă în tabel, dintre operatorii aritmetici, transpusa și ridicarea la putere au prioritate maximă, fiind urmați de ridicarea la putere cu operatori unari, apoi de operatorii unari, mai apoi de înmulțire

și împărțire, iar în final de adunare și scădere. Trebuie menționat că în cazul operatorului împărțire, dacă cei doi operanzi sunt numere întregi, rezultatul împărțirii va fi tot un număr întreg.

Limbajul MATLAB nu are operator pentru a obține restul împărțirii întregi (modulo), dar pentru efectuarea acestei operații se utilizează funcția **mod()**. Sintaxa acestei funcții este:

```
b = mod(a,m)
```

Funcţia returnează restul după împărțirea lui **a** la **m**. Există o convenție în MALTAB că la apelul **mod(a,0)** se returnează valoarea **a**.

O altă funcție care returnează restul împărțirii este **rem()**. Sintaxa acestei funcții este:

```
r = rem(a,b)
```

Funcţia returnează restul după împărțirea lui **a** la **b**. Convenția în MALTAB pentru această funcție este că la apelul **rem(a,0)** se returnează **NaN**. Conceptul de rest după împărțire nu este definit în mod unic, iar cele două funcții **mod()** și **rem()** calculează fiecare o variație diferită. Folosind funcția **mod()** se obține un rezultat care este fie zero, fie are același semn cu deîmpărțitul, iar folosind funcția **rem()** se obține un rezultat care este fie zero, fie are același semn cu împărțitorul.

De asemenea, în MATLAB nu există operator pentru rotunjire, dar pentru efectuarea acestei operații se poate folosi una din funcțiile:

```
Y = ceil(X)
Y = fix(X)
Y = floor(X)
Y = round(X)
```

Funcția **ceil()** rotunjește fiecare element al lui **X** la cel mai apropiat număr întreg mai mare sau egal cu acel element și returnează rezultatul în variabila **Y**.

Funcția **fix()** rotunjește fiecare element al lui **X** la cel mai apropiat număr întreg spre zero. Această operație trunchiază efectiv numerele din **X** transformându-le în numere întregi prin eliminarea părții zecimale a fiecărui număr. Funcția **floor()** rotunjește fiecare element al lui **X** la cel mai apropiat număr întreg mai mic sau egal cu acel element. Pentru numerele pozitive, comportamentul funcției **fix()** este același cu cel al funcției **floor()**, iar pentru numerele negative, comportamentul funcției **fix()** este același cu al funcției **ceil()**.

Funcția **round()** rotunjește fiecare element al lui **X** la cel mai apropiat număr întreg. În caz de egalitate, atunci când un element are partea fracționară exact 0.5, funcția rotunjește de la zero înspre numărul întreg cu modul mai mare. Această funcție mai are o formă des folosită:

```
Y = round(X,N)
```

care rotunjește la **N** cifre:

- dacă **N** > 0 atunci rotunjește la N cifre la dreapta punctului care separă partea întreagă de cea zecimală;
- dacă **N** = 0 atunci rotunjește la cel mai apropiat număr întreg;
- dacă **N** < 0 atunci rotunjește **N** cifre la stânga punctului care separă partea întreagă de cea zecimală (practic rotunjește la cel mai apropiat multiplu de 10^{-N}).

1.3.4. Operatori relaționali

Folosind acest tip de operatori se pot compara două expresii logice, iar rezultatul obținut poate fi **adevărat (1)** sau **fals (0)**. Aceștia sunt: **<** mai mic, **<=** mai mic sau egal, **>** mai mare, **>=** mai mare sau egal, **==** egal cu și **~=** diferit de. Toți au aceeași prioritatea mai mică decât clasa operatorilor aritmetici. Asociativitatea acestei categorii de operatori este de la stânga la dreapta.

Pe lângă acești operatori, în MATLAB sunt implementate niște funcții specifice pentru lucrul cu tablouri, cum sunt:

```
af = isequal(A, B)
af = isequaln(A, B)
```

Prima funcție returnează **1 (adevărat)** dacă **A** și **B** sunt echivalente, iar în caz contrar, returnează **0 (fals)**. Parametrii de intrare **A** și **B** sunt tablouri, iar în unele cazuri tipul lor nu trebuie să coincidă pentru ca ele să fie echivalente:

- Datele de intrare numerice sunt echivalente dacă au aceeași dimensiune și conținutul lor are o valoare egală. Funcția compară atât partea reală, cât și partea imaginară a tablourilor numerice;
- Tabelele, orarele, structurile și tablourile de celule sunt echivalente numai atunci când toate elementele și proprietățile lor sunt egale;
- Șirurile de tip *string* și *char* care conțin aceeași secvență de caractere sunt echivalente.
- Elementele de tip NaN sunt considerate diferite de alte elemente, precum și de ele însele.

A doua funcție returnează **1 (adevărat)** dacă **A** și **B** sunt echivalente, iar în caz contrar, returnează **0 (fals**) la fel ca funcția **isequal(),** iar în plus elementele de tip NaN (Not a Number) sunt considerate a fi egale cu ele însele.

Aceste funcții sunt necesare, deoarece operatorul de egalitate (**==**) aplicat pe două tablouri compară element cu element și returnează un tablou de valori **1 (adevărat)** și/sau **0 (fals).**

1.3.5. Operatori logici

Se utilizează pentru formarea unor expresii logice complexe, prin combinarea a două sau mai multe teste logice.

Operatorii logici sunt: ŞI logic aplicat expresiilor logice (&&), SAU logic aplicat expresiilor logice (||), ŞI logic (&), SAU logic (|) și negație logică (~).

La evaluarea unei expresii logice, ordinea de prioritate a operatorilor este: ~, <, <=, >, >=, ==, ~=, &, |, &&, ||

a	b	~a	a&&b	a||b
1	1	0	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	0

Scurtcircuitarea operatorilor logici:

- pentru operatorul **&&**: dacă primul operand are valoarea egală cu 0 atunci al doilea operand nu se mai evaluează deoarece cu siguranță rezultatul aplicării operatorului este 0.
- pentru operatorul **||**: dacă primul operand are valoare diferită de 0 atunci al doilea operand nu se mai evaluează deoarece indiferent de valoarea lui, rezultatul va fi 1.

Operatorii logici **&** și **|** pot fi aplicați și pe tablouri și returnează un tablou de aceleași dimensiuni ca operanzii conținând doar valori **1 (adevărat)** și/sau **0 (fals).** În cazul operatorului **&**, un element din tabloul de ieșire este **1 (adevărat)** dacă ambii operanzi au un element nenul pe poziția dată de elementul din tabloul de ieșire. În caz contrar, elementul este setat la **0 (fals).** În cazul operatorului **|**, un element din tabloul de ieșire este **1 (adevărat)** dacă cel puțin un operand are un element nenul pe poziția dată de elementul din tabloul de ieșire. În caz contrar, elementul este setat la **0 (fals).**

Pe lângă acești operatori, în MATLAB sunt implementate și niște funcții, cum sunt:

```
C = xor(A, B)
B = all(A)
B = any(A)
```

Prima funcție efectuează SAU exclusiv pe elementele corespondente ale tablourilor **A** și **B** și returnează în **C** un tablou care conține elemente setate fie la **1 (adevărat)**, fie la **0 (fals)**. Un element al tabloului de ieșire este setat la valoarea 1 dacă A sau B, dar nu ambele, conține un element diferit de zero în aceeași locație a tabloului. În caz contrar, elementul matricei este setat la valoarea 0.

A doua funcție testează de-a lungul primei dimensiuni a tabloului **A,** a cărei valoare este diferită de 1 și determină dacă elementele sunt toate diferite de zero sau dacă toate sunt 1 (adevărat). În practică, este de fapt o extensie naturală a operatorului AND logic (&). A treia funcție procedează la fel doar că determină dacă există cel puțin un element nenul sau egal cu 1 (adevărat) și este de fapt o extensie naturală a operatorului OR logic (|).

Observații:

- *Dacă **A** este un vector (matrice coloană sau matrice linie), atunci **all(A)** returnează **1 (adevărat)** dacă toate elementele sunt diferite de zero și returnează **0 (fals)** dacă unul sau mai multe elemente sunt zero, iar **any(A)** returnează **1 (adevărat)** dacă există cel puțin un element diferit de zero în vectorul **A** și returnează **0 (fals)** dacă toate elementele sunt zero.*
- *Dacă **A** este o matrice nevidă și cu numărul de coloane și linii diferit de 1 (deci nu matrice coloană sau matrice linie), atunci atât **all(A)** cât și **any(A)** tratează coloanele lui **A** ca vectori și returnează un vector linie de valori logice **1 (adevărat)** și **0 (fals)**.*
- *Dacă **A** este o matrice vidă (0x0), atunci **all(A)** returnează **1 (adevărat)**, iar **any(A)** returnează **0 (fals)**.*
- *Dacă **A** este un tablou multidimensional, atunci atât **all(A)** cât și **any(A)** acționează de-a lungul primei dimensiuni a tabloului a cărei valoare nu este egală cu 1 și returnează un tablou de valori logice. Această dimensiune devine 1, în timp ce dimensiunile celelalte rămân aceleași.*

Atât funcția **all()**, cât și **any()** au o formă de apelare cu 2 parametri, unde primul este tabloul **A**, iar cel de-al doilea poate fi 'all', adică **B = all(A,'all'),** respectiv **B=any(A, 'all'),** atunci când se dorește testarea tuturor elementelor, un scalar pozitiv întreg **dim** care indică dimensiunea tabloului de-a lungul căreia să se facă testarea (**B = all(A, dim),** respectiv **B=any(A,dim)**) sau un vector de dimensiuni **vecdim** de-a lungul cărora să se facă testarea (**B = all(A, vecdim),** respectiv **B=any(A,vecdim)**).

1.3.6. Operatorul de forțare a conversiei la un anumit tip (cast)

Este un operator unar și se scrie sub forma:

```
c=tip (a);
```

unde: **c** este variabila de ieșire în care se stochează rezultatul, **tip** este un tip de date predefinit al limbajului MATLAB, iar **a** este un tablou sau o expresie. Utilizarea acestui operator are rolul de a modifica tipul rezultatului unei expresii sau tipul unei variabile. Cel mai adesea acest operator se utilizează atunci când funcțiile au parametri de un anumit tip, care trebuie respectat sau atunci când se dorește ca rezultatul obținut ca urmare a unui calcul să fie de un alt tip decât cel care ar rezulta în mod normal (implicit).

1.3.7. Operatorii ( ), [ ], { }

Operatorul ( ) este utilizat pentru delimitarea subexpresiilor și face parte din clasa I de prioritate. Aceste subexpresii sunt primele care se evaluează și împreună cu operatorul paranteză ( ) se comportă ca un operand. Este folosit și la funcții pentru a încadra argumentele. De asemenea, este operator de indexare, se întâlneşte la tablouri şi conţine valoarea indicelui/indicilor elementului unui tablou:

Exemple: s(5)se referă la elementul cinci al șirului unidimensional s (indicele primului termen este 1)

A(3,4) se referă la elementul de pe linia a treia și coloana a patra a matricei A

Operatorul [ ] se folosește pentru declararea unui tablou, concatenarea tablourilor, definirea matricelor vide, ștergerea unor șiruri de elemente și preluarea valorilor returnate de o funcție

Operatorul { } se utilizează pentru a construi o matrice de celule sau pentru a accesa conținutul unei anumite celule dintr-o matrice de celule.

1.3.8. Operatori pe mulțimi

Operațiile pe mulțimi sunt marea majoritate operatori binari, comparând elementele din două mulțimi pentru a găsi puncte comune sau diferențe. În MATLAB, mulțimile sunt tablouri de numere, date, timp sau text. Majoritatea operațiilor pe mulțimi compară seturile pentru egalitatea exactă, care poate fi problematică în contextul aritmeticii în virgulă mobilă. Din acest motiv, au fost implementate și funcții care efectuează comparații cu o toleranță. În MATLAB, toți operatorii pentru mulțimi sunt dați prin funcții și sunt prezentați în tabelul de mai jos.

Denumire	Descriere
intersect	Intersecție de două mulțimi date prin tablouri
ismember	Apartenența la o mulțime
setdiff	Diferență de două mulțimi date prin tablouri
setxor	SAU exclusiv pentru două mulțimi
union	Reuniune de două mulțimi date prin tablouri
unique	Valori unice dintr-o mulțime
ismembertol	Apartenența la o mulțime cu o anumită toleranță
uniquetol	Valori unice dintr-o mulțime cu o anumită toleranță
join	Combină rândurile a două tabele sau orare pe baza variabilelor cheie
innerjoin	Join intern a două tabele sau orare pe baza variabilelor cheie
outerjoin	Join extern a două tabele sau orare pe baza variabilelor cheie

1.3.9. Alți operatori

@ este un manipulator (handle) fie pentru funcția a cărei denumire urmează imediat după simbolul@, fie pentru funcția anonimă care urmează după simbolul @. De asemenea, se poate utiliza @ pentru a apela metodele superclaselor din subclase.

. se folosește împreună cu operatorii aritmetici pentru a aplica operația element cu element.

: este un operator care se utilizează pentru a crea vectori a căror valori sunt date cu ajutorul unui pas, pentru a defini indecșii unor tablouri și limitele unei bucle **for**.

%{ %} sunt operatori care încadrează blocuri de comentarii.

1.4. Instrucțiuni

1.4.1. Introducere

Execuția unui program este controlată prin intermediul instrucțiunilor scrise. Ca și alte limbaje moderne, și limbajul MATLAB este alcătuit pe principiile programării structurate, orice algoritm putând fi scris prin combinarea structurilor secvențiale, alternative (de decizie, de selecție) și repetitive (ciclice).

O instrucțiune poate fi alcătuită din expresii, cuvinte cheie și apeluri de funcții.

Instrucțiunile se pot clasifica în:

- instrucțiuni simple;
- instrucțiuni de control condițional (alternative);
- instrucțiuni de ciclare (bucle);
- instrucțiuni de întrerupere și salt.

1.4.2. Instrucțiuni simple

Instrucțiunea expresie este instrucțiunea cea mai des folosită în cadrul programelor. Această instrucțiune se compune dintr-o expresie care este eventual urmată de caracterul virgulă (**,**) sau de caracterul punct și virgulă (**;**).

Formele ei generale sunt:

```
expresie
expresie,
expresie;
```

unde **expresie** poate fi:

- o expresie de atribuire (situație în care poartă numele de instrucțiune de atribuire);
- un apel de funcție (situație în care poartă numele de instrucțiune de apel de funcție).

Caracterul virgulă (**,**) este folosit pentru a separa instrucțiunile scrise pe același rând, dar trebuie menționat că el permite afișarea în linia de comandă a rezultatului instrucțiunii. Rezultatul se afișează și la folosirea primei forme, dar în acest caz instrucțiunile trebuie scrise pe rânduri diferite. Caracterul punct și virgulă (**;**) poate fi folosit pentru a separa instrucțiunile scrise pe același rând, dar el va suprima afișarea rezultatului instrucțiunii pe care o succede.

Instrucțiunea vidă este instrucțiunea poate fi formată doar din caracterul spațiu ( ), virgulă (**,**) sau punct si virgulă (**;**) sau de nici un caracter cu condiția ca instrucțiunile următoarele să nu fie scrise pe același rând. Practic această instrucțiune nu are nici un efect și se folosește acolo unde sintaxa limbajului solicită utilizarea unei instrucțiuni, dar nu este necesară efectuarea unei operații concrete, cum ar fi, de exemplu, într-o instrucțiune de decizie.

1.4.3. Instrucțiuni de control condițional

Aceste instrucțiuni sunt utilizate atunci când executarea unor operații depinde de îndeplinirea unei condiţii. Limbajul MATLAB are patru astfel de instrucțiuni numite și instrucțiuni decizionale sau de decizie: **if ... end, if ... else ... end, if ... elseif ... else ... end** şi **switch ... case ... otherwise ... end**.

Instrucțiunea if ... end:

Sintaxa acestei instrucțiuni este:

```
...
if expresie
   instrucțiuni
end
...
```

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucțiuni** poate fi o singură instrucțiune sau un set de instrucțiuni.

Mod de lucru:

- se evaluează **expresie**;
- dacă are valoarea 1 (adevărat) se execută **instrucțiuni**, iar dacă are valoarea 0 (fals) nu se execută **instrucțiuni**;

Instrucțiunea if … else ... end:

Sintaxa acestei instrucțiuni este:

```
...
if expresie
    instrucţiuni_1
else
    instrucţiuni_2
end
...
```

unde: **expresie** reprezintă de cele mai multe ori o expresie logică, iar **instrucţiuni_1** și **instrucţiuni_2** pot fi alcătuite dintr-o singură instrucțiune sau dintr-un set de instrucțiuni.

Mod de lucru:

- se evaluează **expresie**;
- dacă are valoarea 1 (adevărat) se execută **instrucţiuni_1**, iar dacă are valoarea 0 (fals) se execută **instrucţiuni_2**.

După executarea uneia din cele două variante se trece la următoarea instrucțiune din program.

Instrucțiunea if … elseif ... else ... end:

Atunci când selecția trebuie să se realizeze din multiple variante se poate folosi această instrucțiune. În cazul general în care există „n" variante posibile selectate pe baza a „n-1" condiții, se recomandă folosirea acestei structuri. Sintaxa acestei instrucțiuni este:

```
...
     if expresie_1
          instrucțiuni_1
     elseif expresie_2
          instrucțiuni_2
     ...
     elseif expresie_n-1
          instrucțiuni_n-1
     else
          instrucțiuni_n
end
...
```

Întâi se evaluează **expresie_1** și dacă rezultatul are valoarea **adevărat** (diferit de 0) se execută setul de comenzi **instrucţiuni_1**. În caz contrar, dacă rezultatul are valoarea **0 (fals)**, se evaluează **expresie_2** și dacă rezultatul are valoarea **adevărat**, se execută comenzile **instrucţiuni_2**. În caz contrar, se continuă evaluarea expresiilor în același mod până la **expresie_n-1**. Dacă rezultatul evaluării ultimei expresii are valoarea **adevărat** se execută setul de comenzi **instrucţiuni_n-1**, iar în caz contrar se execută setul de comenzi **instrucțiuni_n**. Ultima ramură, cea de **else** urmată de setul de comenzi poate să lipsească.

Observație: Pentru toate instrucțiunile de control condițional prezentate mai sus, dacă rezultatul expresiei este un vector (nu doar un număr), atunci expresia este adevărată dacă vectorul conține doar elemente nenule. În caz contrar expresia este falsă.

Instrucțiunea switch ... case ... otherwise ... end

Sintaxa acestei instrucțiuni este:

```
...
switch expresie
     case expresie_case_1
          instrucţiuni_1
     case expresie_case_2
          instrucţiuni_2
          ...
otherwise
```

```
                instrucţiuni
            end
            ...
```

Această instrucțiune de selecție multiplă evaluează o expresie și alege să execute unul din mai multe seturi de instrucțiuni. Fiecare variantă este un caz (**case**). Blocul **switch** testează fiecare caz până când egalitatea dintre **expresie** și una din **expresie_case** este adevărată. Practic se testează:

- Pentru numere, expresie==expresie_case
- Pentru vectori de caractere, strcmp(expresie,expresie_case)==1, unde **strcmp** este o funcție care compară dacă două șiruri de caractere sunt egale sau nu.

Când **expresie** coincide cu **expresie_case** se execută instrucțiunile corespunzătoare și se iese din blocul **switch**. Ramura de **otherwise** este opțională. Instrucțiunile corespunzătoare acestei ramuri se execută doar când nici un caz nu a fost adevărat.

*Observație: Între instrucțiunile **switch** și **if ... elseif ...else ... end** există următoarea corespondență:*

```
if expresie == caz_1                          switch expresie
        instrucțiuni_1                        case caz1
elseif expresie == caz_2                              instrucțiuni_1
                                    ⇔         case caz_2
        instrucțiuni_2                                instrucțiuni_2
        …                                     …
elseif expresie == caz_n                      case caz_n
        instrucțiuni_n                                instrucțiuni_n
else                                          otherwise
        instrucțiuni                                  instrucțiuni
end
```

1.4.4. Instrucțiuni de ciclare

Acestea mai poartă denumirea și de instrucțiuni de iterație sau bucle. De multe ori există situații în care se dorește ca un bloc de instrucțiuni să se execute de un anume număr de ori pentru a putea modifica valorile unor variabile conform anumitor relații. În general, toate limbajele de programare au astfel de instrucțiuni cu ajutorul cărora se poate programa executarea unor cicluri. În limbajul MATLAB există două instrucțiuni de ciclare: **for ... end** și **while ... end.**

Instrucțiunea de ciclare for ... end:

Această instrucțiune execută un ciclu de un număr specific de ori și ține evidența fiecărei iterații folosind o variabilă de index incrementală. Sintaxa acestei instrucțiuni este:

```
...
for contor=val_init:pas:val_fin
     instrucţiuni
end
...
```

unde **contor** este o variabilă ce ia pe rând valori pornind de la valoarea inițială **val_init** și până la valoarea finală **val_fin** cu pasul de incrementare **pas** (poate fi și număr negativ). Trebuie menționat faptul că toate cele trei elemente **val_init**,

pas și **val_fin** pot fi date prin expresii. Pentru fiecare valoare a variabilei **contor** se execută **instrucțiuni**, care poate fi o singură instrucțiune sau un set de instrucțiuni și care reprezintă corpul ciclului. Dacă se dorește parcurgerea cu pasul 1, atunci **pas** poate să lipsească, declarându-se doar valoarea inițială și valoarea finală.
Observaţii: *Corpul ciclului for nu se execută niciodată dacă nu se poate face parcurgerea de la* ***val_init*** *spre* **val_fin** *cu pasul* ***pas****. Dacă se intră într-un ciclu for infinit, execuția acestuia se poate întrerupe folosind combinația de taste* **Ctrl+C***. Atât* **pas** *cât și* **val_fin** *pot lipsi, caz în care corpul ciclului se execută o singură dată pentru valoarea contorului egală cu* **val_init**.

Cicluri for imbricate:

Imbricarea este procesul de plasare al unei instrucțiuni (condiționale sau de ciclare) în interiorul alteia de aceeași natură. În cazul ciclurilor **for** imbricate ciclul conținut în corpul altui ciclu **for** se numește ciclu for interior, iar celălalt ciclu **for** exterior. Două cicluri **for** imbricate mai poartă denumirea de cicluri **for** suprapuse sau incluse.

Forma generală a ciclurilor imbricate este următoarea:

```
...
for contor_1=val_init_1:pas_1:val_fin_1
      instructiuni_1
      for contor_2=val_init_2:pas_2:val_fin_2
          instructiuni_2
      end
      instructiuni_3
end
...
```

Observație*: Ciclul interior trebuie să fie cuprins în întregime în corpul ciclului exterior.*

Pentru fiecare valoare a variabilei **contor_1** se va executa în întregime al doilea ciclu **for** (adică, variabila **contor_2** va lua pe rând valori pornind de la **val_init_2** și până la **val_fin_2** cu pasul de incrementare **pas_2**, și pentru fiecare valoare se va executa setul de **instructiuni_2**). Instrucțiunile sau seturile de instrucțiuni indicate prin **instructiuni_1**, **instructiuni_2** și **instructiuni_3** pot să și lipsească.

Instrucțiunea de ciclare while ... end:

Există situații în care se dorește ca un bloc de instrucțiuni să se execute de mai multe ori însă fără a se cunoaște numărul exact de repetiții. În acest caz, la implementare se folosește instrucțiunea de ciclare **while ... end**. Aceasta este o instrucțiune de ciclare *condiționată anterior*. Sintaxa ei este următoarea:

```
...
while expresie
      instrucțiuni
end
...
```

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

La execuția acestei instrucțiuni, se evaluează întâi valoarea de adevăr pentru **expresie** și se execută setul de **instrucțiuni** (poate fi și doar o instrucțiune) cât timp expresia este adevărată. O expresie este adevărată dacă rezultatul ei nu este o matrice vidă și dacă conține doar elemente nenule. În caz contrar expresia este falsă, iar atunci se trece la următoarea instrucțiune din program.

Cicluri while imbricate:

Asemenea ciclurilor **for**, și acest tip de instrucțiune de ciclare se poate folosi în formă imbricată. Sintaxa a două cicluri **while** imbricate este următoarea:

```
...
while expresie_1
    instructiuni_1
    while expresie_2
       instructiuni_2
    end
    instructiuni_3
end
...
```

Cât timp este adevărată **expresie_1** se vor executa următorii pași în această ordine:

- setul de **instructiuni_1**,
- cel de-al doilea ciclu **while** (adică, atât timp cât **expresie_2** este adevărată se va executa setul de **instructiuni_2**),
- setul de **instructiuni_3**.

Oricare din instrucțiunile sau seturile de instrucțiuni indicate prin **instrucțiuni_1**, **instrucțiuni_2** și **instrucțiuni_3** pot să și lipsească.

Observație: Între cele două instrucțiuni de ciclare există următoarea corespondență:

```
for contor=val_init:pas:val_fin          contor=val_init
    instrucțiuni                  ⇔      while contor<=val_fin
end                                          instrucțiuni
                                             contor=contor+pas
                                         end
```

1.4.5. Instrucțiunea break

Instrucțiunea **break** are sintaxa:

```
break
```

Această instrucțiune se utilizează în corpul unei instrucțiuni de ciclare (**for** sau **while**) pentru a întrerupe forțat ciclul și a ieși din acesta, execuția continuând cu instrucțiunea care urmează după ciclu. În cazul ciclurilor imbricate, această instrucțiune întrerupe doar execuția ciclului în care apare, iar execuția programului se continuă cu prima instrucțiune care urmează după instrucțiunea **end**, de la finalul ciclului.

Această instrucțiune nu poate fi folosită în cadrul instrucțiunii **switch** ca în alte limbaje de programare.

1.4.6. Instrucţiunea continue

Instrucţiunea **continue** are forma generală:

```
continue
```

Această instrucțiune se poate utiliza numai în corpul unui ciclu și întrerupe execuția iterației curente, înainte de a fi executate celelalte instrucțiuni de după ea din corpul ciclului. Practic, dacă această instrucțiune se regăsește:

- în corpul instrucțiunii de ciclare **for ... end** determină întreruperea iterației curente și trecerea la execuția actualizării contorului;
- în corpul instrucțiunii de ciclare **while ... end** determină întreruperea iterației curente și trecerea la evaluarea expresiei test care determină continuarea sau terminarea ciclului.

1.5. Fișiere script

1.5.1. Introducere

MATLAB nu este doar un limbaj de programare, ci și un mediu de programare. Pe lângă posibilitatea de folosire a acestuia drept calculator științific prin efectuarea diferitelor operații în linia de comandă, MATLAB-ul permite crearea de fișiere pentru realizarea sarcinilor repetitive ca orice alt mediu de programare.

O modalitate rapidă și eficientă de a rezolva problemele simple constă în scrierea comenzilor în linia de comandă. Dacă este necesar un număr mai mare de comenzi sau acestea trebuie rulate pentru mai multe valori ale variabilelor, scrierea comenzilor în mod repetat în linia de comandă devine incomodă. În aceste cazuri folosirea fișierelor script este extrem de utilă și chiar necesară.

Fișierele script sunt fișiere de tip text care cuprind comenzi MATLAB. Atunci când un astfel de fișier este lansat în execuție MATLAB citește și execută comenzile ca și cum fiecare ar fi fost scrise secvențial în linia de comandă. De aceea, variabilele declarate și/sau folosite în cadrul programelor scrise în fișiere script se regăsesc în spațiul de lucru (Workspace) după execuția fișierului script.

Toate fișierele script trebuie să aibă extensia *.m*. Dacă se creează un nou fișier script cu același nume cu unul deja existent, MATLAB îl va alege întotdeauna pe acela care apare primul în calea sa de căutare (path).

1.5.2. Crearea, deschiderea și afișarea conținutului fișierelor script

O primă modalitate de creare a fișierelor script este folosind comanda ***New*** din meniul ***File***.

O a doua modalitate este folosind comanda **edit**. Aceasta creează un fișier text nou sau editează unul deja existent. Formele acestei instrucțiuni sunt:

```
edit
edit fișier
edit fișier_1 ... fișier_n
```

Dacă se folosește prima formă, adică în linia de comandă se scrie doar comanda **edit** fără a fi urmată de numele unui fișier, atunci se deschide un fișier text nou în editorul de fișiere.

La folosirea celei de-a doua forme, adică în cazul în care comanda este urmată și de numele unui fișier care nu există atunci apare un mesaj care informează utilizatorul că fișierul nu există și îl întreabă dacă dorește crearea acestuia. Se poate bifa opțiunea 'Do not show this prompt again' care va suprima apariția acestui mesaj la crearea fișierelor viitoare. În cazul în care se alege "Yes" se creează un fișier nou cu numele și extensia specificată prin **fișier** și este ulterior deschis în editorul de comenzi. Dacă fișierul există deja, atunci el este deschis pentru a fi modificat:

Dacă se dorește crearea sau modificarea simultană a mai multor fișiere, atunci se folosește a treia formă, adică se specifică numele acestora după comanda **edit** separate de spațiu (**fișier_1 ... fișier_n)**. La folosirea acestei comenzi se poate observa în fereastra directorului curent că aceste fișiere au fost create (după alegerea variantei "Yes" la eventualele mesaje primite) și au fost deschise în editor. Dacă la folosirea comenzii **edit** nu se specifică extensia fișierului, atunci implicit aceasta se consideră a fi *.m*.

Deschiderea unui fișier din linia de comandă se poate realiza și cu comanda **open** urmată de numele fișierului. Formele acestei instrucțiuni sunt:

```
open nume
A = open('nume')
```

Prima formă deschide fișierul (sau variabila dacă există) specificat prin **nume**. A doua formă returnează în **A** o structură dacă fișierul dat prin **nume** este un fișier **.mat**, sau returnează un manipulator (handler) dacă **nume** este o figură. În caz contrar, deschide returnează în **A** o matrice goală.

Trebuie menționat faptul că aceste fișiere pot fi create sau editate nu doar folosind editorul MATLAB, ci și orice alt editor extern.

Afișarea în linia de comandă a conținutului fișierelor script și nu numai se realizează cu ajutorul comenzii **type**. Sintaxa acestei instrucțiuni este:

```
type nume
```

Practic, folosind această comandă se afișează conținutul fișierului specificat prin **nume** în fereastra de comandă a programului MATLAB.

1.5.3. Lansarea în execuție a fișierelor script

Principalul avantaj al fișierelor script este că acestea pot fi rulate cu ușurință ori de câte ori este nevoie fără a rescrie comenzile. Există trei moduri principale de a lansa în execuție fișierele script:

- se apasă butonul **Run** (săgeată verde) din editorul de fișiere;
- se apasă tasta funcțională **F5** atunci când fereastra cu fișierul script dorit este activă în editorul de fișiere;
- se tastează în linia de comandă numele fișierului script, iar apoi se apasă tasta **Enter** (numele fișierului script nu trebuie să conțină spații).

Observație: Comentariile (orice șir de caractere care începe cu '%') și blocurile de comentarii (textul cuprins între operatorii ***%{ %}****) nu reprezintă comenzi MATLAB care să se execute. Ele reprezintă doar informații utile pentru utilizatorul programului.*

1.6. Funcții utilizator

1.6.1. Introducere

Noţiunea de funcţie este o noțiune de bază în limbajul MATLAB alături de noțiunea de fișier script, ambele permițând reutilizarea secvențelor de comenzi prin stocarea lor în fișiere. Funcțiile oferă mai multă flexibilitate, în primul rând pentru că se pot transmite valorile de intrare și se pot returna valorile de ieșire. În plus, funcțiile evită stocarea variabilelor temporare în spațiul de lucru de bază și pot rula mai repede decât scripturile. Pe lângă acestea, se disting și avantajele comune tuturor limbajelor care permit utilizarea funcțiilor:

- funcțiile pot fi apelate din diferite părți ale unui program (în MATLAB chiar și din linia de comandă) și pentru valori diferite ale parametrilor acestora;
- funcțiile pot fi utilizate și de alți programatori;
- funcțiile pot fi utilizate și în alte programe.

Un utilizator al limbajului de programare MATLAB care doreşte să-şi scrie propriile funcţii trebuie să aibă în vedere trei elemente: **definiția** şi **apelul** funcțiilor.

1.6.2. Definiția funcției

Funcțiile definite în fișiere cu extensia **.m** sunt subrutine definite folosind cuvântul cheie ***function***, cu ajutorul căruia se declară numele, datele de intrare și datele de ieșire ale funcției. Forma generală a unei funcții este:

```
function  [y_1,...,y_n]  = nume_funcție(x_1,...,x_m)
```

Prin sintaxa de mai sus se declară o funcție cu numele **nume_funcție** care acceptă datele de intrare **x_1,...,x_m** și returnează datele de ieșire **y_1,...,y_n**. Această declarare folosind **function** trebuie să fie făcută pe prima linie executabilă din fișier, iar datele de intrare precum și cele de ieșire pot lipsi.

Pentru a fi valid, numele unei funcții trebuie să înceapă cu o literă și poate conține doar litere, numere și simbolul *underscore* ‚_' , fără a coincide cu un cuvânt cheie din MATLAB (cum ar fi de exemplu cuvintele cheie **if, else, elseif, end, while, for** etc.).

1.6.3. Apelul funcţiei

În general, fișierul **.m** în care se definește o funcție se salvează cu numele funcției. În cazul în care numele fișierului și cel al funcției nu coincid, apelarea funcției se va face folosind numele fișierului, nu pe cel al funcției.

Apelarea funcției se poate face

- din linia de comandă;
- din interiorul altei funcții;
- din interiorul unui fișier script

dar întotdeauna folosind numele fișierului de tip funcție urmat de valorile efective **v_1,...,v_m** pentru parametrii de intrare. Astfel, în caz general apelul funcției se face astfel **[y_1,...,y_n] = nume_funcție(v_1,...,v_m)** . Dacă parametrii de intrare lipsesc atunci funcția se apelează fără valori pentru aceștia, adică **[y_1,...,y_n] = nume_funcție()**, iar dacă și parametrii de ieșire lipsesc apelul funcției arată astfel: **nume_funcție().**

Observație: Între lista parametrilor efectivi de la apel **v_1,...,v_m** *și lista parametrilor formali de la declarare* **x_1,…,x_m** *trebuie să existe concordanță de* ***tip****,* ***număr*** *și* ***ordine****, altfel va apare eroare la executarea codului.*

1.6.4. Funcţii locale

Programele scrise în limbajul MATLAB pot conține mai multe funcții. Într-un fișier de tip funcție, prima funcție din fișier se numește funcția principală și, de obicei, ea va da numele fișierului. Această funcție este vizibilă pentru funcțiile din alte fișiere sau poate fi apelată din linia de comandă și din fișierele script.

Funcțiile suplimentare din fișier sunt numite funcții locale și pot apărea în orice ordine după funcția principală. Funcțiile locale sunt vizibile numai pentru alte funcții din același fișier. Ele sunt echivalente cu subrutinele din alte limbaje de programare și sunt uneori numite subfuncții. Sintaxa și modul de apelare sunt aceleași ca la funcțiile principale.

Începând cu varianta de MATLAB R2016b, se pot crea funcții locale și într-un fișier script, atâta timp cât acestea apar după ultima linie de cod script, iar ele pot fi apelate doar în cadrul fișierului unde au fost definite, la fel ca în cazul funcțiilor locale definite în fișiere de tip funcție.

Funcțiile locale din fișierul curent au prioritate față de funcțiile din alte fișiere. Adică, atunci când se apelează o funcție într-un fișier **.m**, MATLAB verifică dacă funcția este o funcție locală înainte de a căuta alte funcții principale. Prin urmare, se poate crea o variantă a unei anumite funcții, păstrând în același timp funcția originală într-un alt fișier.

Toate funcțiile, inclusiv funcțiile locale, au propriul spațiu de lucru separat de spațiul de lucru de bază. De aceea, funcțiile locale nu pot accesa variabilele utilizate de alte funcții decât dacă acestea sunt transmise ca argumente.

1.6.5. Funcţii imbricate

O funcție imbricată este o funcție care este complet conținută într-o altă funcție, numită și funcție părinte. Orice funcție dintr-un program MATLAB poate include o funcție imbricată.

Diferența principală dintre funcțiile imbricate și alte tipuri de funcții este că pot accesa și modifica variabilele definite în funcțiile lor părinte. Ca rezultat:

- Funcțiile imbricate pot utiliza variabile care nu le sunt transmise în mod explicit ca argumente de intrare;
- Într-o funcție părinte, se poate crea un manipulator (handle) pentru o funcție imbricată care conține datele necesare pentru a apela funcția imbricată.

După cum se poate observa în forma generală a funcției, de obicei, funcțiile nu necesită instrucțiunea de final **end**. Totuși, pentru a include o funcție într-un program script sau într-un fișier de tip funcție unde este definită cel puțin încă o funcție, toate funcțiile din acel fișier trebuie să utilizeze instrucțiunea de final **end**.

O funcție imbricată nu poate fi definită în oricare dintre instrucțiunile de control al programului MATLAB, cum ar fi **if ... elseif ... else ... end**, **switch ... case ... otherwise ... end**, **for ... end**, sau **while ... end**.

Apelul unei funcții imbricate se face fie folosind direct numele ei (fără a utiliza **feval**), fie folosind un manipulator (handle) de funcție care se poate crea folosind operatorul **@**.

Toate variabilele din funcțiile imbricate sau funcțiile care le conțin trebuie definite în mod explicit (inițializate înainte). Adică, nu se poate apela o funcție sau un script care atribuie valori variabilelor decât dacă aceste variabile există deja în spațiul de lucru al funcției. În general, variabilele dintr-un spațiu de lucru al unei funcții nu sunt disponibile pentru alte funcții. Cu toate acestea, funcțiile imbricate pot accesa și modifica variabile în spațiile de lucru ale funcțiilor care le conțin. Adică atât o funcție imbricată, cât și o funcție care o conține (funcția părinte) pot modifica aceeași variabilă fără a trece acea variabilă ca argument. Practic, funcțiile imbricate pot utiliza variabile din trei surse:

- Parametrii de intrare;
- Variabile definite în cadrul funcției imbricate;
- Variabile definite în funcția părinte.

Funcțiile imbricate care returnează argumente de ieșire au variabile pentru ele în spațiul lor de lucru. Cu toate acestea, funcțiile părinte au variabile pentru ieșirea funcțiilor imbricate numai dacă le solicită în mod explicit.

Fiecare funcție are un anumit domeniu, adică un set de alte funcții pentru care este vizibilă. O funcție imbricată poate fi apelată:

- De la nivelul imediat superior (adică în funcția părinte);
- De la o funcție imbricată la același nivel în cadrul aceleiași funcții părinte;
- De la o funcție la orice nivel inferior.

Cel mai simplu mod de a extinde domeniul unei funcții imbricate este de a crea un handle de funcție și de a-l returna ca argument de ieșire. Trebuie menționat însă că numai funcțiile care pot apela o funcție imbricată pot crea un handle pentru aceasta, iar atunci când se creează un handle pentru o funcție imbricată, acesta stochează nu numai numele funcției, ci și valorile variabilelor definite în funcția părinte.

1.6.6. Funcții anonime

O funcție anonimă este o funcție care nu este stocată într-un fișier **.m**, dar este asociată cu o variabilă al cărei tip de date este **function_handle**. Funcțiile anonime pot accepta mai mulți parametrii de intrare și pot returna un parametru de ieșire. Ele pot conține doar o singură instrucțiune executabilă. Forma ei generală este:

```
nume_funcție= @ (lista_param_intrare) expresie_funcție
```

Operatorul **@** creează manipulatorul (handler-ul), iar parantezele **( )** imediat după operatorul **@** includ lista parametrilor de intrare ai funcției **lista_param_intrare** (parametrii de intrare separați de virgulă). Legea funcției este dată prin **expresie_funcție**, iar rezultatul este returnat în **nume_funcție**.

Multe funcții MATLAB acceptă ca date de intrare astfel de variabile de tip **function_handle** pentru a putea realiza evaluarea funcțiilor pentru o diversitate de valori. Avantajul utilizării funcțiilor anonime este că nu este necesar un fișier pentru o funcție care poate fi definită printr-o expresie scurtă. Apelul unei astfel de funcții se face **nume_funcție (val_param_intrare)**, unde **val_param_intrare** sunt valorile efective ale parametrilor de intrare.

Variabilele de tip **function_handler** pot stoca nu numai o expresie, ci și variabile pe care expresia funcției le necesită pentru evaluare. Dacă funcția nu are date de intrare, atunci **lista_param_intrare** poate lipsi și se folosesc doar paranteze goale la definiția și apelul funcției anonime.

Capitolul 2. Reprezentarea algoritmilor

2.1. Introducere

Atunci când se urmărește rezolvarea unei probleme tehnice cu ajutorul calculatorului, este necesar să se parcurgă următoarele etape:
- analiza problemei: identificarea datelor de intrare / ieşire precum şi a modelului matematic de rezolvare a acesteia;
- descrierea algoritmului de rezolvare: fie prin reprezentare **grafică** (scheme logice), fie prin reprezentare **literală** (limbajul pseudocod);
- scrierea programului: într-un limbaj de programare, utilizând un editor de texte (în acest caz MATLAB, folosind editorul încorporat);
- rularea programului, pașii de compilare și asamblare a programului făcându-se automat.

Algoritmul (după numele matematicianului Abu Ja'far Mohammed ibn Musâ al- Khowârizmî) poate fi definit în matematică și informatică drept o metodă sau o procedură de calcul, alcătuită din pași elementari necesari pentru rezolvarea unei anumite probleme sau a unei categorii de probleme. Algoritmul este un concept de bază atât în matematică cât şi în informatică.

Noțiunea de algoritm poate fi definită și ca o succesiune finită de paşi ce trebuie să fie executați într-o anumită ordine, astfel încât pornind de la anumite date cunoscute (date de intrare) să se obțină rezultatele dorite (date de ieşire).

Indiferent de limbajul de programare în care vor fi implementați, algoritmii trebuie să fie caracterizați de:
- **generalitate**: un algoritm trebuie să rezolve o categorie (clasă) de probleme şi nu doar o problemă particulară a acelei categorii;
- **finitudine**: orice algoritm trebuie să conțină un număr finit de pași;
- **eficiență**: proprietatea unui algoritm de a se termina într-un număr cât mai mic de pași;
- **optimalitate**: un algoritm este optim atunci când se termină după un număr minim de paşi;
- **corectitudine**: proprietatea unui algoritm de a returna o soluție corectă;
- **caracter univoc**: orice algoritm trebuie să returneze întotdeauna același rezultat pentru același set de date inițiale;
- **claritate**: un algoritm trebuie să descrie cu exactitate și fără ambiguități pașii care trebuie parcurși pentru a rezolva problema;
- **verificabilitate**: se referă la posibilitatea de a verifica fiecare pas al algoritmului;

Scrierea algoritmilor trebuie să fie metodică și conform anumitor reguli pentru a duce la obținerea unor programe clare, uşor de înţeles şi depanat. În programarea structurată, algoritmii, și mai apoi programele, se elaborează pe baza unor reguli bine stabilite și a unui set redus de structuri de control. Conform teoremei lui Bohm-Jacopini, orice algoritm poate fi compus din numai trei structuri de control:
- **structura secvenţială**: instrucţiunile se derulează una după alta;
- **structura alternativă**: instrucţiunile se derulează după un criteriu de selecţie;
- **structura repetitivă**: instrucţiunile din cadrul structurii se repetă în funcţie de rezultatul unui test.

2.2. Tipuri de date şi expresii

Algoritmii lucrează cu date. Din punct de vedere logic, datele sunt definite prin trei elemente:
- **identificator**: reprezintă numele datei, format din unul sau mai multe caractere;
- **valoarea**: reprezintă conţinutul zonei de memorie în care este păstrată data;

- **tip**: descrie apartenenţa datei la o anumită clasă de date;

Clasificarea datelor:
A. În funcţie de momentul în care se introduc în fluxul de date:
- de intrare;
- de ieșire;
- de manevră;
B. În funcţie de valoare:
- constante;
- variabile;
C. În funcţie de tip:
- date numerice (reale sau întregi);
- date logice;
- date șiruri de caractere;
- alte tipuri specifice limbajului de programare utilizat.

O **expresie** conține **operanzi** şi **operatori**. **Operanzii** pot fi date constante, variabile sau alte expresii încadrate între paranteze rotunde. **Operatorii** desemnează operaţiile care se execută asupra operanzilor. Operatorii utilizaţi la întocmirea algoritmilor pot fi grupaţi astfel: operatori aritmetici, operatori relaţionali şi operatori logici.

Operatorii aritmetici definesc operaţiile aritmetice şi pot fi unari (adică se aplică unui singur operand) sau binari (acţionează asupra a doi operanzi). Operatorii aritmetici utilizați în general sunt **+** adunare, **-** scădere, ***** înmulțire, **/** împărțire. Pentru restul împărțirii întregi se va folosi funcția **mod()**.

Operatorii relaţionali sunt operatori binari şi se aplică operanzilor de tip numeric sau şir de caractere, rezultatul operaţiei fiind unul de tipul logic. Operatorii relaţionali sunt: **==** egal, **≠** diferit, **<** mai mic, **>** mai mare, **≤** mai mic sau egal, **≥** mai mare sau egal.

Operatorii logici definesc operaţiile logice şi acţionează doar asupra operanzilor logici, rezultatul fiind unul de tip logic. Operatorii logici sunt: **NOT** (**~**) – negare logică, **AND** (**&**) – ŞI logic, **OR** (**|**) – SAU logic. În tabelul 2.1. este prezentat modul de acţiune al operatorilor logici:

Tabelul 2.1. Operatori logici

a	b	~a	a | b	a & b
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

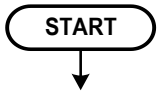
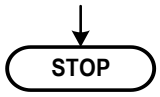
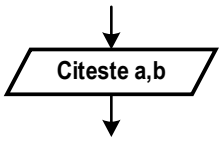
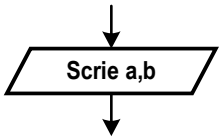
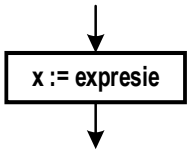
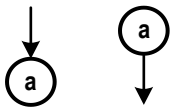
2.3. Reprezentarea algoritmilor prin scheme logice şi pseudocod

Modalităţile de reprezentare al algoritmilor sunt: scheme logice sau limbajul pseudocod.

Schema logică este o modalitate de reprezentare a algoritmilor sub formă grafică. Ea permite vizualizarea secvențelor de operații utilizând blocuri grafice specifice și a succesiunii acestora indicată prin săgeți. Un dezavantaj al utilizării schemelor logice constă în faptul că, în cazul unor probleme mai dificile schemele logice pot fi stufoase, deci mai greu de urmărit. Un avantaj al învăţării schemelor logice constă în faptul că acestea sunt utilizate şi în alte reprezentări, nu numai al algoritmilor de rezolvare al problemelor din domeniul informaticii.

Blocurile utilizate în reprezentarea grafică a algoritmilor sunt ilustrate în tabelul următor (tabelul 2.2):

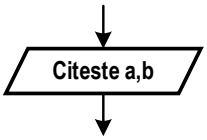
Tabelul 2.2. Blocurile utilizate în reprezentarea grafică a algoritmilor

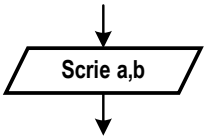
Simbol:	Denumire:	Semnificaţie:
	Bloc terminal	Marchează începutul algoritmului.
	Bloc terminal	Marchează sfârşitul algoritmului.
	Bloc de **intrare** (citire) a datelor de intrare	Se utilizează pentru transferul datelor de la utilizator către algoritm.
	Bloc de **ieşire** (scriere) a datelor de ieşire	Se utilizează pentru transferul datelor de la algoritm către utilizator.
	Bloc de **atribuire**	Se utilizează pentru atribuirea valorii expresiei către variabila **a.**
	Bloc de **decizie**	Se evaluează condiţia obţinându-se o valoare logică „Adevărat" sau „Fals". Dacă condiţia este adevărată se execută ramura **DA**, iar dacă condiţia este falsă se execută ramura **NU.**
	Bloc conector logic	Se utilizează pentru conectarea unor puncte din schema logică situate în aceeaşi pagină.
	Bloc conector de pagină	Se utilizează pentru conectarea părţilor din schema logică care sunt reprezentate pe pagini diferite.

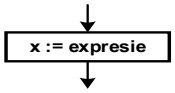
Limbajul pseudocod poate fi definit ca un ansamblu de codificări cu ajutorul cărora se definesc operaţiile (instrucţiunile) utilizate pentru reprezentarea algoritmilor. Limbajul pseudocod conţine cuvinte cheie cu anumite semnificaţii.

Aşa cum s-a menționat anterior, orice algoritm poate fi compus din trei structuri de control, corespondenţa dintre limbajul pseudocod şi reprezentarea grafică prin schema logică fiind ilustrată în continuare. De asemenea, sunt prezentate şi instrucţiunile aferente ale limbajului MATLAB.

I. Structura secvenţială: poate conţine o înşiruire de una sau mai multe instrucţiuni, ce se execută secvenţial (una după alta). Instrucţiunile pot fi: de citire / scriere de date, de atribuire sau combinații ale acestora.

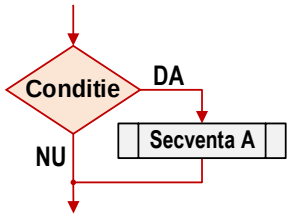
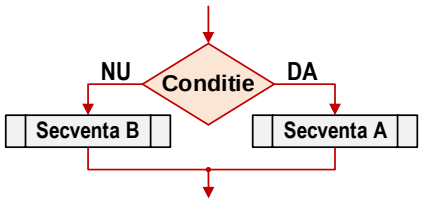
Schema logică	Limbajul pseudocod	Limbajul MATLAB
	**Citeşte** a,b	`a=input (' Introdu a = ');` `b=input (' Introdu b = ');`

	**Scrie** a,b	`s=sprintf('a = %d', a); disp(a);`
		`s=sprintf('b = %d', b); disp(b);`
		sau
		`s=sprintf('a = %f', a); disp(a);`
		`s=sprintf('b = %f', b); disp(b);`

	x ← a + b	`x = a + b;`

Observație: Instrucțiunea de atribuire poate fi scrisă în blocul schemei logice și x=expresie sau x←expresie, toate formele având aceeași semnificație.

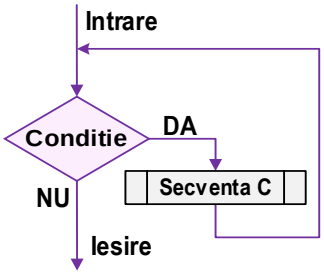
II. Structura alternativă (sau **decizia)**: reprezintă alegerea unei operaţii sau a unei secvenţe de operaţii dintre două alternative posibile. Structura alternativă poate fi întâlnită în două variante, conform tabelului de mai jos:

Schema logică	Limbajul pseudocod	Limbajul MATLAB
Varianta I:		
	**Dacă** *condiţie* **atunci** *Secventa A* **Sfârşit dacă**	**if** *condiţie* *Secventa A* **end**
Mod de lucru: - se evaluează **condiţie**, care de obicei este o expresie logică; - dacă aceasta este **adevărată** (ramura **DA**) se execută *Secvenţa A*, iar dacă este **falsă** (ramura **NU**) atunci nu se execută nimic;		
Varianta II:		
	**Dacă** *condiţie* **atunci** *Secventa A* **altfel** *Secventa B* **Sfârşit dacă**	**if** *condiţie* *Secventa A* **else** *Secventa B* **end**
Mod de lucru: - se evaluează **condiţie**, care de obicei este o expresie logică; - dacă aceasta este **adevărată** (ramura **DA**) se execută *Secvenţa A*, iar dacă este **falsă** (ramura **NU**) atunci se execută *Secvența B*;		

III. Structura repetitivă (sau **bucla)**: constă în executarea repetată a unei instrucțiuni sau a unei secvenţe de instrucțiuni, în funcţie de o anumită condiţie. Dacă condiţia este adevărată, se execută instrucțiunea sau secvența de instrucțiuni, iar dacă condiția este falsă se părăseşte structura repetitivă, continuând cu următoarea structură din schema logică sau din program. Sunt două tipuri de structuri repetitive: cu test inițial și cu contor.

III.1. Structura repetitivă cu test iniţial (condiţionată anterior)

Schema logică	Limbajul pseudocod	Limbajul MATLAB

	**Cât timp** *condiţie* **execută**
	Secventa C
	Sfârşit cât timp
		`while condiţie`
		`Secventa C`
		`end`

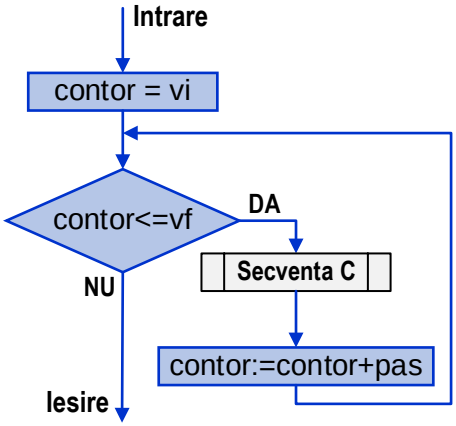
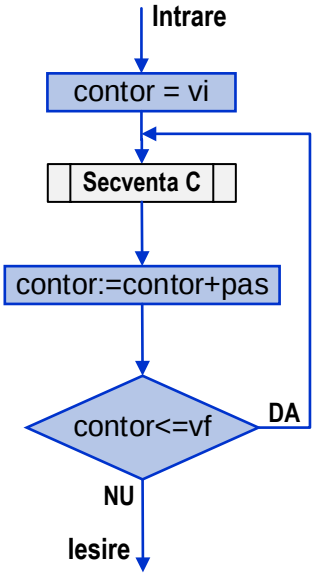
Mod de lucru:

- se evaluează **condiţie**, care de obicei este o expresie logică;
- dacă **condiţie** este **adevărată** (ramura **DA**) se execută *Secvenţa C* care reprezintă secvența de operații ce formează corul structurii repetitive, după care se revine la evaluarea condiţiei, ş.a.m.d.;
- dacă **condiţie** este **falsă** (ramura **NU**) se părăseşte structura repetitivă, algoritmul continuând cu structura imediat următoare;

Observaţii:

- structura repetitivă cu test iniţial se utilizează preponderent atunci când nu se cunoaşte numărul de repetări;
- dacă **condiţie** este **falsă** de la început, *Secvenţa C* nu se execută.

III.2. Structura repetitivă cu contor

Schema logică	
Varianta I:	**Varianta II:**
	

Limbajul pseudocod	Limbajul MATLAB

Pentru *contor ← vi, vf [, pas]*
Secventa C
Sfârşit pentru
	`for contor=vi : pas: vf`
	`Secventa C`
	`end`

Mod de lucru:
1. variabila **contor** primeşte valoarea iniţială, **vi**;
2. se verifică condiţia **contor <= vf**, iar dacă valoarea variabilei **contor** este mai mică sau egală decât valoarea finală, **vf**, (ramura **DA**) se execută *Secvenţa C*. Dacă condiţia este falsă (ramura **NU**) se părăseşte structura repetitivă
3. se modifică valoarea variabilei **contor**, cu pasul **pas**, astfel **contor := contor + pas**;
4. se revine la pasul 2, prin care se verifică condiţia **contor <= vf;**
Observaţii:
- structura repetitivă cu contor se utilizează atunci când se cunoaşte numărul de repetări;
- schema logică pentru structura repetitivă cu contor poate fi utilizată în două variante: condiţionată anterior sau condiţionată posterior;

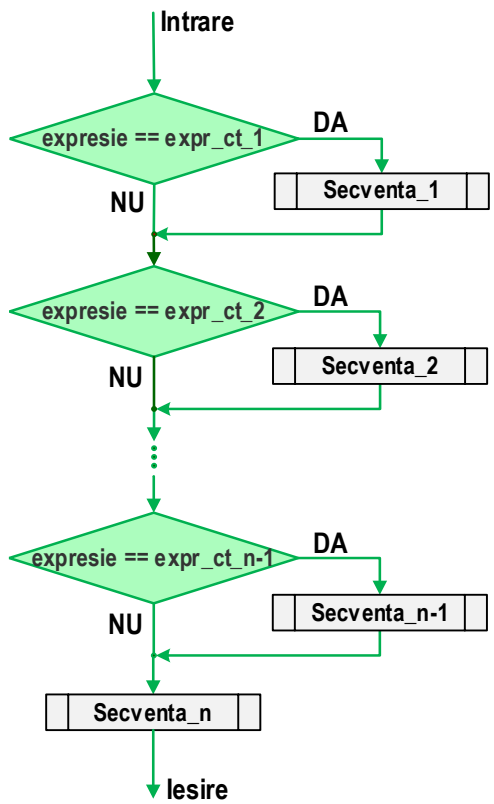
Structura repetitivă cu contor poate fi scrisă utilizând celelalte două structuri repetitive, astfel:

Structura repetitivă cu contor	Structura repetitivă cu test iniţial
Pentru *contor ← vi, vf [, pas]* *Secventa C* **Sfârşit pentru**	*contor ← vi* **Cât timp** *contor <= vf* **execută** *Secventa C* *contor = contor + pas* **Sfârşit cât timp**

for contor=vi:pas:vf
Secventa C;
end
	contor=vi;
	while contor<=vf
	Secventa C;
	contor=contor+pas;
	end

III.3. Structura de decizie cu ramuri multiple

Schema logică	Limbajul pseudocod	Limbajul MATLAB

	**Dacă** expresie == expr_ct_1
	Secvenţa_1;
	Altfel dacă expresie == expr_ct_2
	Secvenţa_2;
	...
	Altfel dacă expresie == expr_ct_n-1
	Secvenţa_n-1;
	Altfel
	Secvenţa_n;
	Sfarsit
		`switch expresie`
		`case expr_ct_1`
		`Secvenţa_1;`
		`case expr_ct_2`
		`Secvenţa_2;`
		`...`
		`case expr_ct_n-1`
		`Secvenţa_n-1;`
		`otherwise`
		`Secvenţa_n;`
		`end`

Mod de lucru:
- se evaluează **expresie**;
- se testează daca valoarea obţinută pentru **expresie** este egală cu constanta specificată **expr_ct_1**. Dacă cele două au valori egale se execută **Secvenţa_1** și se părăsește structura, execuția continuând cu următoarea instrucțiune care urmează după structura de decizie cu ramuri multiple. În caz contrar, execuţia continuă cu următoarea ramură din structură, adică ramura corespunzătoare **expr_ct_2**.
- se verifică dacă valoarea **expresie** este egală cu constanta specificată **expr_ct_2** procedându-se ca în cazul precedent;
- procedeul continuă până la epuizarea tuturor ramurilor, în final executându-se **Secvenţa_n,** dacă nici o egalitate nu a fost adevărată;

Observaţii:
- **Secventa_n** poate să lipsească;
- dacă **expresie** nu este egală cu niciuna dintre expresiile constante, atunci se execută **Secventa_n**, dacă aceasta există.

Capitolul 3. Probleme de complexitate redusă

3.1. Interschimbarea valorilor a două variabile

În anumite probleme este necesar să se schimbe între ele valorile a două variabile.

Să presupunem că sunt două variabile **a** şi **b**, de exemplu cu valorile, **a** = **5**, respectiv **b** = **11**. Se doreşte elaborarea unui algoritm cu ajutorul căruia să se realizeze interschimbarea valorilor celor două variabile, astfel încât să avem **a** = **11**, respectiv **b** = **5**.

La realizarea algoritmului, trebuie să se ţină cont de faptul că valoarea numerică a fiecărei variabile este reţinută într-o anumită zonă de memorie, iar în urma atribuirii unei alte valori, valoarea anterioară se pierde. O primă variantă de rezolvare soluționează această problemă prin utilizarea unei variabile auxiliare (variabila **aux**), algoritmul fiind următorul:

- se citesc cele două valori pentru **a** şi **b**;
- se atribuie variabilei **aux** valoarea variabilei **a**;
- se atribuie variabilei **a** valoarea variabilei **b**;
- se atribuie variabilei **b** valoarea variabilei **aux**;
- se afişează valorile celor două variabile **a** şi **b**;

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.1a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.1b.

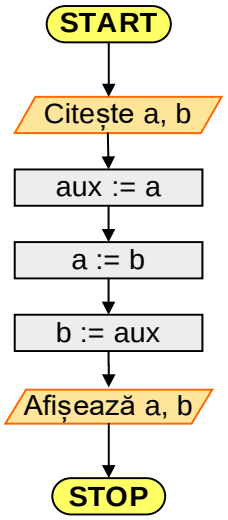
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** a,b aux ← a a ← b b ← aux **Scrie** a,b **Sfârşit**

Figura 3.1a. Reprezentarea algoritmului pentru interschimbarea valorilor a două variabile – varianta 1

Programul MATLAB	Execuția programului

`a=input('Introdu a=');`
`b=input('Introdu b=');`
`aux=a; a=b;`
`b=aux;`
`s=sprintf ('a=%d \t b=%d \n',a,b);`
`disp(s)`
	`Introdu a = 5`
	`Introdu b = 11`
	`a = 11  b = 5`

Figura 3.1b. Programul MATLAB și execuția acestuia pentru interschimbarea valorilor a două variabile – varianta 1

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

A doua variantă a algoritmului nu necesită utilizarea unei variabile auxiliare, iar pașii sunt următorii:

- se citesc cele două valori pentru **a** şi **b**;
- se atribuie variabilei **a** valoarea **a + b**;
- se atribuie variabilei **b** valoarea **a - b**;
- se atribuie variabilei **a** valoarea **a - b**;
- se afişează valorile celor două variabile **a** şi **b**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.1c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.1d.

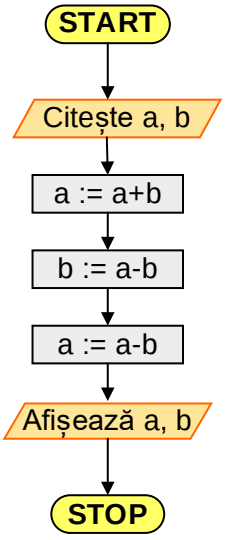
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** a,b a ← a+b b ← a-b a ← a-b **Scrie** a,b **Sfârşit**

Figura 3.1c. Reprezentarea algoritmului pentru interschimbarea valorilor a două variabile – varianta 2

Programul MATLAB	Execuția programului

`a=input('Introdu a=');`
`b=input('Introdu b=');`
`a=a+b;`
`b=a-b;`
`a=a-b;`
`s=sprintf ('a=%d \t b=%d \n',a,b);`
`disp(s)`
	`Introdu a = 5`
	`Introdu b = 11`
	`a = 11  b = 5`

Figura 3.1d. Programul MATLAB și execuția acestuia pentru interschimbarea valorilor a două variabile – varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.2. Conversia unghiului din grade în radiani

Radianul, simbolizat **rad**, reprezintă o unitate de măsură pentru unghiuri şi face parte din Sistemul Internaţional de Unităţi. Un radian reprezintă unghiul la centrul unui cerc care subîntinde un arc de lungime egală cu raza cercului. Un radian este egal cu 180° / π, sau aproximativ 57,2958° sau 57°17'45". În general în informatică, argumentele funcțiilor trigonometrice se exprimă în radiani. Pentru transformarea unui unghi din grade în radiani se utilizează relaţia de calcul:

$$ur = ug \cdot \frac{\pi}{180}$$

unde: **ug** – reprezintă unghiul în grade, **ur** – reprezintă unghiul în radiani;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.2a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.2b.

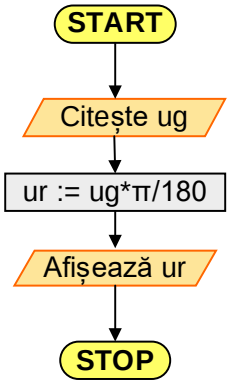
Schema logică	Execuția programului
	**Început** **Citeşte** ug ur ← ug*π/180 **Scrie** ur **Sfârşit**

Figura 3.2a. Reprezentarea algoritmului pentru conversia unghiului din grade în radiani

Programul MATLAB	Execuția programului

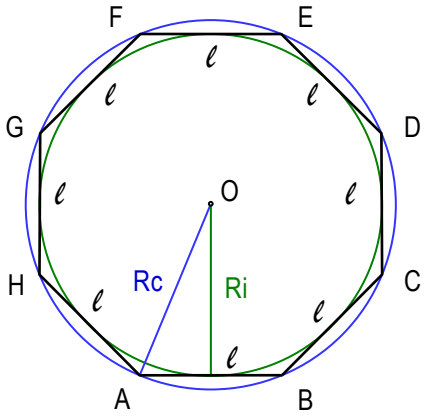
`ug=input('Introdu unghiul [grade]=');`
`ur=ug*pi/180;`
`s=sprintf('ug=%7.3f \t ur=%7.3f \n',ug,ur);`
`disp(s)`
	`Introdu unghiul [grade] = 35`
	`ug =  35.000  ur = 0.611`

Figura 3.2b. Programul MATLAB și execuția acestuia pentru conversia unghiului din grade în radiani

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.3. Calculul perimetrului şi ariei unui poligon regulat cu „n” laturi

În figura alăturată este reprezentat un poligon regulat cu „n” laturi pentru care se cunosc numărul de laturi „**n**” şi lungimea unei laturi, „**l**”. Pentru calculul ariei **A**, perimetrului **P**, razei cercului circumscris **Rc**, respectiv a razei cercului înscris **Ri** se utilizează relaţiile:



$$A := \frac{1}{4}\cdot n\cdot l^2\cdot ctg\left(\frac{\pi}{n}\right),\ P := n\cdot l,\ Rc := \frac{l}{2\cdot\sin\left(\frac{\pi}{n}\right)},\ Ri := \frac{l}{2\cdot tg\left(\frac{\pi}{n}\right)}$$

Algoritmul de calcul presupune parcurgerea următoarelor etape: citirea numărului de laturi **n** şi a lungimii unei laturi **l**, calculul pe baza relaţiilor de mai sus a ariei - **A**, perimetrului - **P**, razei cercului circumscris - **Rc**, respectiv a razei cercului înscris - **Ri** şi în final afişarea valorilor calculate.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.3a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.3b.

Schema logică	Limbajul pseudocod

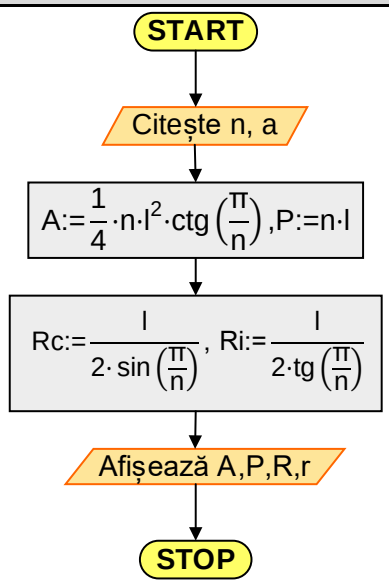
	**Început**
	  Citeşte n,a
	  A ← ¼*n*l*l*ctg(π/n)
	  P ← n*l
	  Rc ← l/2/sin(π/n)
	  Ri ← l/2/tg(π/n)
	  Scrie A,P,R,r
	Sfârşit

Figura 3.3a. Reprezentarea algoritmului pentru calculul ariei unui poligon regulat cu n laturi

Programul MATLAB	Execuția programului

`n=input('Nr. laturi, n=');`
`l=input('Lungimea laturii, lat=');`
`A=n*l*l/tan(pi/n)/4;P=n*l;`
`Rc = l/2/sin(pi/n);`
`Ri = l/2/tan(pi/n);`
`s=sprintf('Aria A=%7.3f',A);`
`disp(s)`
`s=sprintf('Perimetrul P=%7.3f ',P);`
`disp(s)`
`s=sprintf('Raza Rc=%7.3f',Rc);`
`disp(s)`
`s=sprintf ('Raza Ri=%7.3f ',Ri);`
`disp(s)`
	`Nr. laturi, n = 6`
	`Lungimea laturii, l = 10`
	`Aria A = 259.808`
	`Perimetrul P = 60.000`
	`Raza Rc = 10.000`
	`Raza Ri = 8.660`

Figura 3.3b. Programul MATLAB și execuția acestuia pentru calculul ariei unui poligon regulat cu n laturi

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

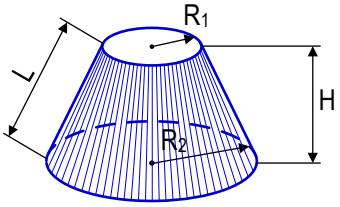
3.4. Calculul volumului, ariei laterale şi a ariei totale ale unui trunchi de con

Se consideră trunchiul de con din figura alăturată, pentru care se cunosc raza bazei mari – **R1**, raza bazei mici – **R2** şi înălţimea - **H**. Se cere să se determine volumul – **V**, aria laterală – **Alat** şi aria totală – **A**.

Pentru calcule se utilizează relaţiile de calcul:

$$L := \sqrt{H^2 + (R1 - R2)^2}, V := \frac{\pi H}{3} \cdot \left(R1^2 + R2^2 + R1 \cdot R2\right),$$

$$A_{lat} := \pi L(R1 + R2), A := \pi \cdot \left[L(R1 + R2) + R1^2 + R2^2\right]$$



Algoritmul de calcul presupune parcurgerea următoarelor etape:

- citirea datelor de intrare, adică a razei bazei mari - **R1**, a razei bazei mici - **R2** şi a înălţimii - **H**;

- calculul volumului - **V**, ariei laterale - **Alat** şi a ariei totale - **S** pe baza relaţiilor de mai sus;
- afişarea valorilor calculate;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.4a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.4b.

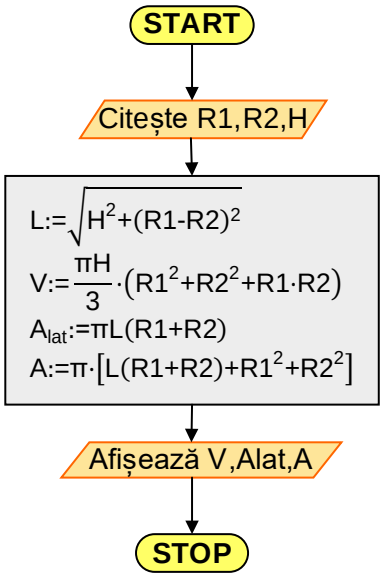
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** R1,R2,H L ← sqrt(H*H+(R1-R2)*(R1-R2)) V ← π*H/3*(R1*R1+R2*R2+R1*R2) Alat ← π*L*(R1+R2) A ← π *(L*(R1+R2)+R1*R1+R2*R2) **Scrie** V, Alat, A **Sfârşit**

Figura 3.4a. Reprezentarea algoritmului pentru calculul volumului, ariei laterale şi ariei totale ale unui trunchi de con

Programul MATLAB	Execuția programului

`R1 = input('Raza bazei mari, R1[cm]=');`
`R2 = input('Raza bazei mici, R2[cm]=');`
`H = input('Inaltimea, H[cm]=');`
`L = sqrt(H^2+(R1-R2)^2);`
`V = H*pi*(R1*R1+R2*R2+R1*R2)/3;`
`Alat = pi*L*(R1+R2);`
`A = pi*(L*(R1+R2)+R1*R1+R2*R2);`
`s=sprintf ('V=%7.3f [cm^3]',V);`
`disp(s)`
`s=sprintf ('Alat=%7.3f [cm^2]', Alat);`
`disp(s)`
`s=sprintf ('Atot=%7.3f [cm^2]',A);`
`disp(s)`
	`Raza bazei mari, R1 [cm] = 8`
	`Raza bazei mici, R2 [cm] = 5`
	`Inaltimea, H [cm] = 4`
	`V = 540.354 [cm^3]`
	`Alat = 204.204 [cm^2]`
	`Atot = 483.805 [cm^2]`

Figura 3.4b. Programul MATLAB și execuția acestuia pentru calculul volumului, ariei laterale şi ariei totale ale unui trunchi de con

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.5. Progresia aritmetică

O progresie aritmetică este un şir de numere reale $\mathbf{a_n}$ ($n \geq 1$) pentru care fiecare termen, începând cu al doilea, se obţine prin însumarea termenului precedent cu un număr **r**, numit raţia progresiei.

Se notează: $\mathbf{a_1}$ – primul termen al progresiei, **r** – raţia progresiei aritmetice, $\mathbf{a_n}$ – termenul general al progresiei aritmetice.

Termenul general al unei progresii aritmetice se obţine pe baza relaţiei:

$$a_n := a_1 + (n-1)\cdot r$$

Suma primilor **n** termeni ai progresiei aritmetice, notată $\mathbf{S_n}$ se obţine pe baza relaţiei:

$$S_n := \frac{(a_1 + a_n)\cdot n}{2}$$

În continuare, este prezentat un program pentru calculul termenului de rang **k**, precum şi suma primilor **n** termeni ai unei progresii aritmetice pentru care se cunosc primul termen $\mathbf{a_1}$ şi raţia **r**.

Algoritmul de calcul presupune parcurgerea următoarelor etape:

- citirea datelor de intrare: $\mathbf{a_1}$ primul termen, raţia **r**, numărul de termeni **n**, precum şi rangul **k** al termenului pentru care se doreşte determinarea valorii;
- calculul valorii elementului $\mathbf{a_k}$ şi suma primilor **n** termeni, pe baza relaţiilor de mai sus;
- afişarea valorilor calculate;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.5a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.5b.

Schema logică	Limbajul pseudocod

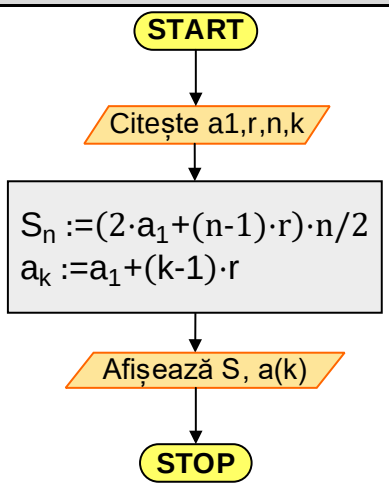
	**Început**
	  Citeşte a1,r,n,k
	  Sn ← (2*a1+(n-1)*r)*n/2
	  ak ← a1+(k-1)*r
	  Scrie Sn, ak
	Sfârşit

Figura 3.5a. Reprezentarea algoritmului pentru calculul termenului de rang k şi a sumei primilor n termeni dintr-o progresie aritmetică

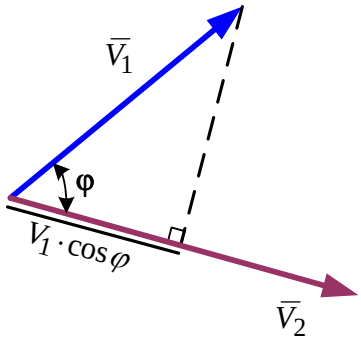
Programul MATLAB	Execuția programului

`a1 = input('Introdu a1:');`
`r = input('Introdu r:');`
`n = input('Introdu n:');`
`k = input('Introdu k:');`
`Sn = (2*a1+(n-1)*r)*n/2;`
`Ak = a1+(k-1)*r;`
`s = sprintf ('Suma(%d)=%d',n,Sn);`
`disp(s)`
`s = sprintf ('a(%d)=%d',k,ak);`
`disp(s)`
	`Introdu a1:2`
	`Introdu r:3`
	`Introdu n:10`
	`Introdu k:7`
	`Suma(10)=155`
	`a(7)=20`

Figura 3.5b. Programul MATLAB și execuția acestuia pentru calculul termenului de rang k şi a sumei primilor n termeni dintr-o progresie aritmetică

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.6. Produsul scalar a doi vectori



Produsul scalar a doi vectori $\bar{V}_1(V_{1x} \cdot \bar{\imath} + V_{1y} \cdot \bar{\jmath} + V_{1z} \cdot \bar{k})$, respectiv $\bar{V}_2(V_{2x} \cdot \bar{\imath} + V_{2y} \cdot \bar{\jmath} + V_{2z} \cdot \bar{k})$, este definit de relaţia:

$$\bar{V}_1 \cdot \bar{V}_2 = |\bar{V}_1| \cdot |\bar{V}_2| \cdot cos\varphi$$

unde φ reprezintă unghiul dintre cei doi vectori.

Rezultatul acestui produs este o mărime scalară, interpretarea geometrică fiind aceea că produsul scalar a doi vectori reprezintă produsul dintre mărimea unui vector şi proiecţia celuilalt pe direcţia primului vector.

Expresia analitică a produsului scalar este: $\bar{V}_1 \cdot \bar{V}_2 = V_{1x} \cdot V_{2x} + V_{1y} \cdot V_{2y} + V_{1z} \cdot V_{2z}$

Unghiul format de cei doi vectori este dat de relaţia: $cos\varphi = \frac{\bar{V}_1 \cdot \bar{V}_2}{|\bar{V}_1| \cdot |\bar{V}_2|} = \frac{V_{1x} \cdot V_{2x} + V_{1y} \cdot V_{2y} + V_{1z} \cdot V_{2z}}{\sqrt{V_{1x}^2 + V_{1y}^2 + V_{1z}^2} \cdot \sqrt{V_{2x}^2 + V_{2y}^2 + V_{2z}^2}}$

Algoritmul de calcul pentru produsul scalar, respectiv pentru unghiul dintre cei doi vectori presupune parcurgerea următorilor paşi: citirea datelor de intrare, adică a componentelor celor doi vectori: ax, ay, az, bx, by, bz; calculul produsului scalar şi a unghiului dintre cei doi vectori, pe baza relaţiilor de mai sus; afişarea valorilor calculate;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.6a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.6b.

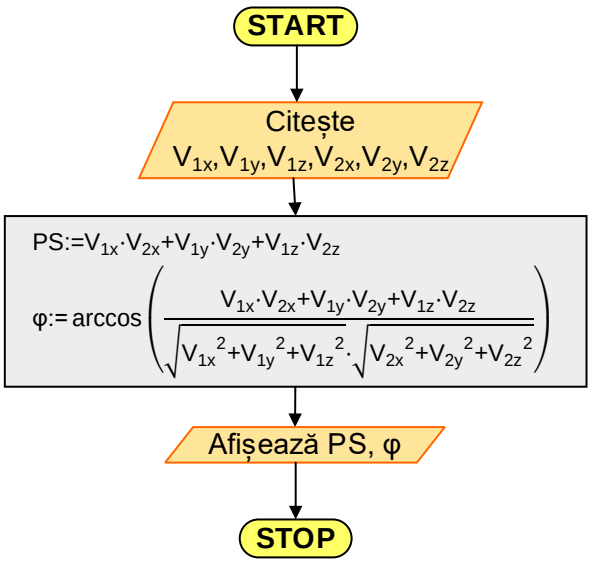
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** V1x,V1y,V1z,V2x,V2y,V2z PS ← V1x*V2x+V1y*V2y+V1z*V2z φ← arccos(PS/(sqrt(V1x*V1x+V1y*V1y+V1z*V1z)*sqrt(V2x*V2x+V2y*V2y+V2z*V2z))) **Scrie** PS, φ **Sfârşit**

Figura 3.6a. Reprezentarea algoritmului pentru calculul produsului scalar

Programul MATLAB	Execuția programului

`V1x = input('V1x=');`
`V1y = input('V1y=');`
`V1z = input('V1z=');`
`V2x = input('V2x=');`
`V2y = input('V2y=');`
`V2z = input('V2z=');`
`PS = V1x*V2x+V1y*V2y+V1z*V2z;`
`fi = PS/(sqrt(V1x*V1x+V1y*V1y+V1z*V1z)*sqrt(V2x*V2x+V2y*V2y+V2z*V2z));`
`s = sprintf ('Produsul scalar PS = %6.3f',PS);`
	`V1x = 1`
	`V1y = 2`
	`V1z = 3`
	`V2x = 5`
	`V2y = 2`
	`V2z = 4`
	`Produsul scalar PS = 21.000`
	`Unghiul fi = 33.211 [grade]`

```
disp(s)
s = sprintf ('Unghiul fi = %6.3f [grade]',
acos(fi)*180/pi); disp(s)
```

Figura 3.6b. Programul MATLAB și execuția acestuia pentru calculul produsului scalar

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.7. Rezolvarea ecuaţiei de gradul al doilea

Ecuaţia algebrică de gradul al doilea este o ecuaţie polinomială de gradul doi. Forma generală a ecuaţiei este:

$$ax^2 + bx + c = 0, \quad a \neq 0$$

unde x este variabila, iar a,b,c sunt coeficienţi.

Rădăcinile ecuaţiei algebrice de gradul doi se obţin cu ajutorul formulei:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

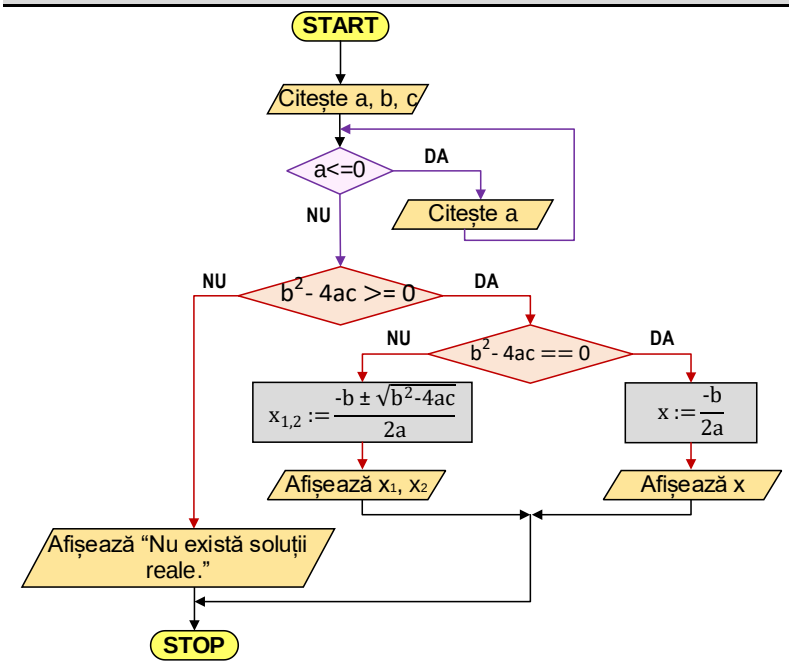
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** a,b,c **Cât timp** a<=0 **execută** **Citeşte** a **Sfârşit cât timp** **Dacă** b*b-4*a*c≥0 **atunci** **Dacă** b*b-4*a*c=0 **atunci** x←-b/(2*a) **Scrie** x **altfel** x1← (-b+sqrt(b*b-4*a*c))/(2*a) x2← (-b-sqrt(b*b-4*a*c))/(2*a) **Scrie** x1,x2 **Sfârşit dacă** **altfel** **Scrie** „Nu exista solutii reale!” **Sfârşit dacă** **Sfârşit**

Figura 3.7a. Reprezentarea algoritmului pentru calculul rădăcinilor ecuaţiei de gradul al doilea

Algoritmul de calcul al rădăcinilor presupune parcurgerea următorilor paşi:

- citirea coeficienţilor a, b şi c;
- verificarea condiției ca $b^2 - 4ac \geq 0$. Dacă condiţia este falsă înseamnă că nu există rădăcini reale ale ecuaţiei şi se va afişa un mesaj corespunzător. Dacă condiţia este adevărată, se verifică dacă $b^2 - 4ac = 0$. Dacă condiţia este adevărată, ecuaţia de gradul al doilea are o singură rădăcină reală, dată de relaţia:

$$x = \frac{-b}{2a}$$

După calculul rădăcinii reale se afişează valoarea acesteia.

Dacă condiţia este falsă atunci înseamnă că $b^2 - 4ac > 0$ atunci ecuația de gradul al doilea are două rădăcini reale. După calculul rădăcinilor se afişează valorile acestora.

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.7a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.7b.

Programul MATLAB	Execuția programului

`a=input('a=');`
`while a<=0`
`  a=input('a=');`
`end`
`b=input('b=');`
`c=input('c=');`
`if(b*b-4*a*c>=0)`
`  if(b*b-4*a*c==0)`
`  x=-b/(2*a);`
`  s=sprintf ('x = %6.3f',x);`
`  disp(s)`
`  else`
`  x1=(-b+sqrt(b*b-4*a*c))/(2*a);`
`  x2=(-b-sqrt(b*b-4*a*c))/(2*a);`
`  s=sprintf ('x1=%6.3f \t x2=%6.3f',x1,x2);`
`  disp(s)`
`  end`
`else`
`  s=sprintf('Nu exista solutii reale!');`
`  disp(s)`
`end`
	Rularea programului – cazul 1:
	a = **1**
	b = **-5**
	c = **6**
	x1=3.000
	x2=2.000
	Rularea programului – cazul 2:
	a = **1**
	b = **2**
	c = **1**
	x = -1.000
	Rularea programului – cazul 3:
	a = **0**
	a = **3**
	b = **2**
	c = **5**
	Nu există solutii reale!

Figura 3.7b. Programul MATLAB și execuția acestuia pentru calculul rădăcinilor ecuaţiei de gradul al doilea

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.8. Maximum a trei numere

Se consideră trei valori: **a**, **b**, **c** şi se cere să se determine valoarea maximă dintre cele trei.

Algoritmul este următorul:

- se compară prima dată valoarea variabilei **a** cu valoarea variabilei **b** punând condiţia **a > b**.
- dacă condiţia este adevărată atunci valoarea maximă obţinută până acum este **a**. În continuare, se compară valoarea variabilei **c** cu valoarea variabilei **a** punând condiţia **c > a**. Dacă condiţia este adevărată, maximul este variabila **c**, iar dacă condiţia este falsă atunci maximul este variabila **a**;
- dacă condiţia a > b este falsă, atunci valoarea maximă obţinută până acum este b. În continuare, se compară valoarea variabilei **c** cu valoarea variabilei **b** punând condiţia **c > b**. Dacă condiţia este adevărată, maximul este variabila **c**, iar dacă condiţia este falsă atunci maximul este variabila **b**;

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.8a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.8b.

Schema logică	Limbajul pseudocod

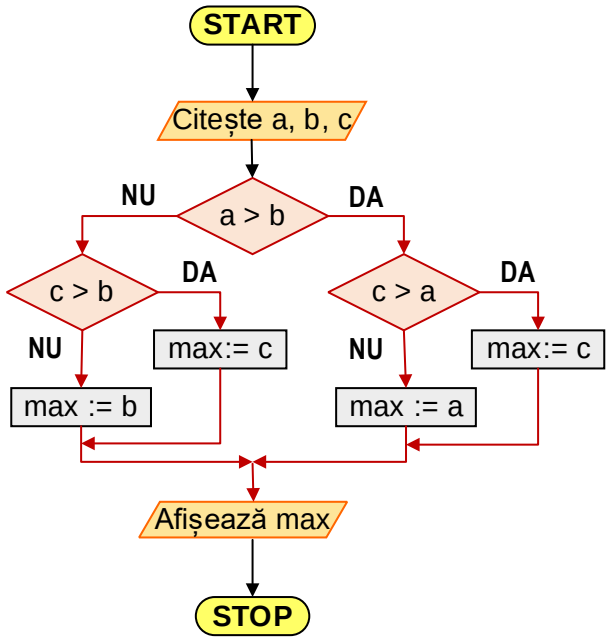
	**Început**
	  Citeşte a,b,c
	  Dacă a>b **atunci**
	  Dacă c>a **atunci**
	  max←c
	  altfel
	  max←a
	  Sfârşit dacă
	  altfel
	  Dacă c>b **atunci**
	  max←c
	  altfel
	  max←b
	  Sfârşit dacă
	  Sfârşit dacă
	Scrie max
	Sfârşit

Figura 3.8a. Reprezentarea algoritmului pentru determinarea maximului dintre trei valori

Programul MATLAB	Execuția programului

`a=input('Introdu a:');`
`b=input('Introdu b:');`
`c=input('Introdu c:');`
`if (a > b)`
`  if (c > a)`
`  s=sprintf ('max = %d',c);`
`  disp(s);`
`  else`
`  s=sprintf ('max = %d',a);`
`  disp(s);`
`  end`
`else`
`  if(c > b)`
`  s=sprintf ('max = %d',c);`
`  disp(s);`
`  else`
`  s=sprintf ('max = %d',b);`
`  disp(s);`
`  end`
`end`
	`Introdu a:1`
	`Introdu b:2`
	`Introdu c:3`
	`max = 3`
	
	`Introdu a:3`
	`Introdu b:2`
	`Introdu c:1`
	`max = 3`
	
	`Introdu a:1`
	`Introdu b:3`
	`Introdu c:2`
	`max = 3`

Figura 3.8b. Programul MATLAB și execuția acestuia pentru determinarea maximului dintre trei valori

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.9. Rezolvarea unui sistem de două ecuaţii cu două necunoscute

Se consideră un sistem de două ecuaţii cu două necunoscute:

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

unde: $a,\ b,\ c,\ d,\ e,\ f \in \mathbb{R}$.

Pentru rezolvarea sistemului se utilizează regula lui Cramer, care spune că dacă determinantul matricei principale obţinute pe baza coeficienţilor ecuaţiilor este diferit de zero, atunci soluţia sistemului de ecuaţii se determină cu ajutorul relaţiilor:

$$x = \frac{c \cdot e - b \cdot f}{a \cdot e - b \cdot d}, \quad y = \frac{a \cdot f - d \cdot c}{a \cdot e - b \cdot d};$$

Schema logică	Limbajul pseudocod

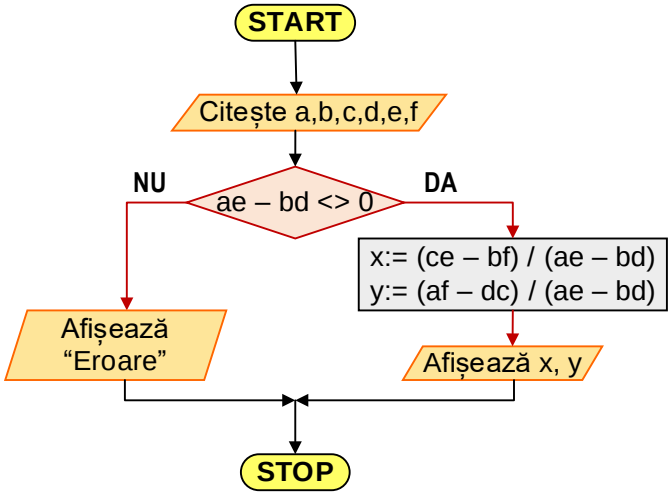
	**Început**
	  Citeşte a,b,c,d,e,f
	  Dacă a*e-b*d≠0 **atunci**
	  x← (c*e-b*f)/(a*e-b*d)
	  y← (a*f-d*c)/(a*e-b*d)
	  Scrie x,y
	  altfel
	  Scrie „NU are solutie!”
	  Sfârşit dacă
	Sfârşit

Figura 3.9a. Reprezentarea algoritmului pentru rezolvarea unui sistem de două ecuaţii cu două necunoscute

Programul MATLAB	Execuția programului

```
a=input('Introdu a:');
b=input('Introdu b:');
c=input('Introdu c:');
d=input('Introdu d:');
e=input('Introdu e:');
f=input('Introdu f:');
if(a*e-b*d~=0)
    x=(c*e-b*f)/(a*e-b*d);
    y=(a*f-d*c)/(a*e-b*d);
    s=sprintf ('x=%6.3f \t y=%6.3f',x,y);
    disp(s)
else
    s=sprintf('NU are solutie!!!');
    disp(s)
end
```

```
Rularea programului- cazul 1:
Introdu a:2
Introdu b:-3
Introdu c:5
Introdu d:1
Introdu e:1
Introdu f:10
x =  7.000        y =  3.000
Rularea programului- cazul 2:
Introdu a:1
Introdu b:1
Introdu c:1
Introdu d:1
Introdu e:1
Introdu f:1
NU are solutie!!!
```

Figura 3.9b. Programul MATLAB și execuția acestuia pentru rezolvarea unui sistem de două ecuaţii cu două necunoscute

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Algoritmul de rezolvare este următorul:

- se citesc coeficienţii ecuaţiilor şi termenii liberi, adică: a, b, c, d, e, f;
- se verifică dacă determinantul matricei principale este diferit de zero.
    1. dacă condiția impusă ($a \cdot e - b \cdot d \neq 0$) este adevărată se calculează soluţia cu relaţiile de mai sus şi se afişează;
    2. dacă condiţia impusă nu este adevărată, se afişează un mesaj corespunzător și se încheie algoritmul;

Reprezentarea algoritmului prin schema logică și limbaj pseudocod este prezentată în figura 3.9a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.9b.

3.10. Verificarea condiţiei de coliniaritate a trei puncte

Se consideră trei puncte $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, $A_3(x_3, y_3)$ situate în planul Oxy. Se doreşte să se verifice dacă cele trei puncte sunt coliniare. Condiţia de coliniaritate este:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

Algoritmul de verificare constă în parcurgerea următorilor paşi:

- se citesc coordonatele celor trei puncte A_1, A_2, A_3;
- se calculează valoarea determinantului conform relaţiei de mai sus;
- se verifică dacă determinantul este nul. Dacă determinantul este nul cele trei puncte sunt coliniare, iar dacă determinantul este diferit de zero atunci cele trei puncte nu sunt coliniare;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.10a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.10b.

Schema logică	Limbajul pseudocod

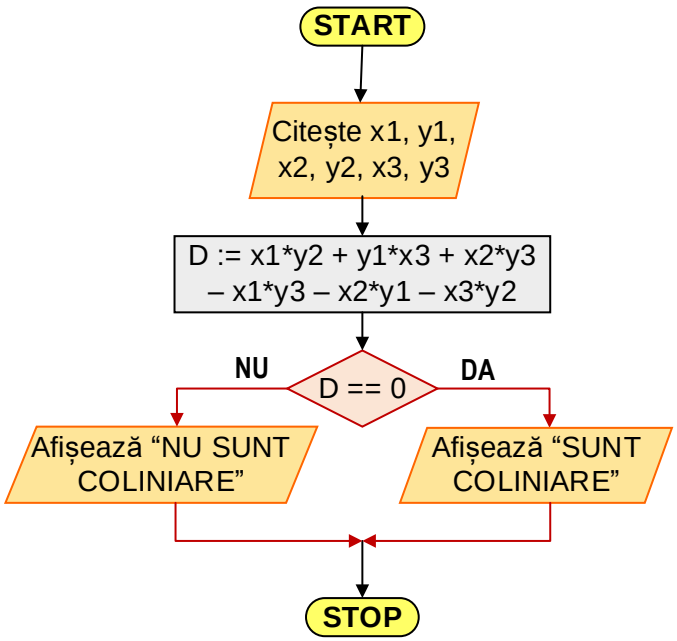
	**Început**
	  Citeşte x1,y1,x2,y2,x3,y3
	  D← x1*y2+y1*x3+x2*y3-x1*y3
	  -x2*y1-x3*y2
	  Dacă D=0 **atunci**
	  Scrie „SUNT COLINIARE”
	  altfel
	  Scrie „NU SUNT COLINIARE”
	  Sfârşit dacă
	Sfârşit

Figura 3.10a. Reprezentarea algoritmului pentru verificarea condiţiei ca trei puncte să fie coliniare

Programul MATLAB	Execuția programului

`x1=input('Introdu x1:');`
`y1=input('Introdu y1:');`
`x2=input('Introdu x2:');`
`y2=input('Introdu y2:');`
`x3=input('Introdu x3:');`
`y3=input('Introdu y3:');`
`D=x1*y2+y1*x3+x2*y3;`
`D=D-x1*y3-x2*y1-x3*y2;`
`if (D==0)`
`  disp('SUNT COLINIARE!!!');`
`else`
`  disp('NU SUNT COLINIARE!!!');`
`end`
	Rularea programului – cazul 1:
	Introdu x1:**1**
	Introdu y1:**1**
	Introdu x2:**2**
	Introdu y2:**2**
	Introdu x3:**3**
	Introdu y3:**3**
	SUNT COLINIARE!!!
	Rularea programului – cazul 2:
	Introdu x1:**0**
	Introdu y1:**0**
	Introdu x2:**2**
	Introdu y2:**0**
	Introdu x3:**0**
	Introdu y3:**3**
	NU SUNT COLINIARE!!!

Figura 3.10b. Programul MATLAB și execuția acestuia verificarea condiţiei ca trei puncte să fie coliniare

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.11. Determinarea coordonatelor punctului de intersecţie a două drepte

Se consideră două drepte de ecuaţii: $a_1 x + b_1 y + c_1 = 0$, $a_2 x + b_2 y + c_2 = 0$

Condiţia ca cele două drepte să se intersecteze este: $a_1 \cdot b_2 - a_2 \cdot b_1 \neq 0$

Coordonatele punctului de intersecţie sunt: $x_0 := \frac{b_1 \cdot c_2 - b_2 \cdot c_1}{a_1 \cdot b_2 - a_2 \cdot b_1}$; $y_0 := \frac{c_1 \cdot a_2 - c_2 \cdot a_1}{a_1 \cdot b_2 - a_2 \cdot b_1}$

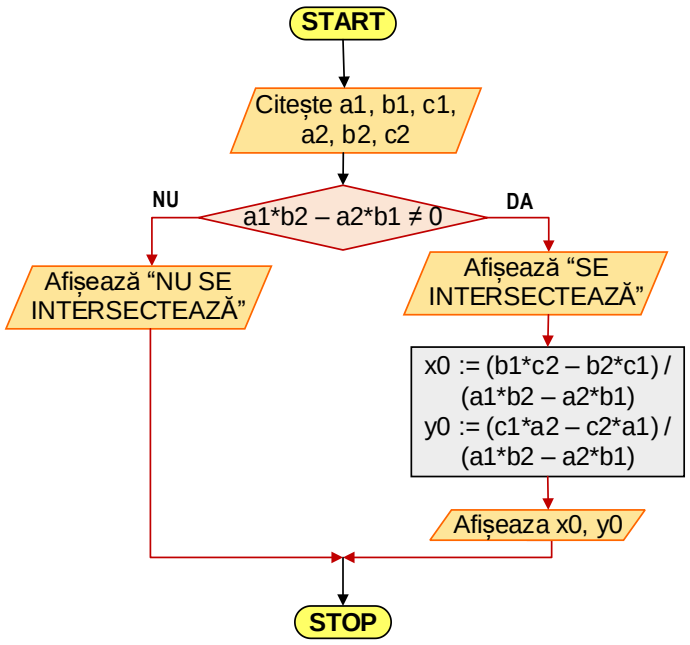
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** a1,b1,c1,a2,b2,c2 **Dacă** a1*b2-a2*b1≠0 **atunci** **Scrie** „SE INTERSECTEAZA” x0← (b1*c2-b2*c1)/(a1*b2-a2*b1) y0← (c1*a2-c2*a1)/(a1*b2-a2*b1) **Scrie** x0,y0 **altfel** **Scrie** „NU SE INTERSECTEAZA” **Sfârşit dacă** **Sfârşit**

Figura 3.11a. Reprezentarea algoritmului pentru calculul coordonatelor punctului de intersecție a două drepte

Programul MATLAB	Execuția programului

`a1=input('Introdu a1:');`
`b1=input('Introdu b1:');`
`c1=input('Introdu c1:');`
`a2=input('Introdu a2:');`
`b2=input('Introdu b2:');`
`c2=input('Introdu c2:');`
`if (a1*b2-a2*b1==0)`
`  disp('NU SE INTERSECTEAZA!!!');`
`else`
`  disp('SE INTERSECTEAZA!!!');`
`  x0=(b1*c2-b2*c1)/(a1*b2-a2*b1);`
`  y0=(c1*a2-c2*a1)/(a1*b2-a2*b1);`
`  s=sprintf('x0=%6.3f \t y0=%6.3f',x0,y0);`
`  disp(s);`
`end`
	Rularea programului – cazul 1:
	Introdu a1:**5**
	Introdu b1:**2**
	Introdu c1:**-4**
	Introdu a2:**1**
	Introdu b2:**-3**
	Introdu c2:**-11**
	SE INTERSECTEAZA!!!
	x0= 2.000
	y0=-3.000
	Rularea programului – cazul 2:
	Introdu a1:**1**
	Introdu b1:**2**
	Introdu c1:**3**
	Introdu a2:**2**
	Introdu b2:**4**
	Introdu c2:**6**
	NU SE INTERSECTEAZA!!!

Figura 3.11b. Programul MATLAB și execuția acestuia pentru calculul coordonatelor punctului de intersecție a două drepte

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Algoritmul de calcul al coordonatelor punctului de intersecţie presupune parcurgerea următorilor paşi:
- se citesc coeficienţii ecuaţiilor celor două drepte, adică: a_1, b_1, c_1, respectiv a_2. b_2. c_2;
- se verifică dacă se îndeplinește condiţia ca cele două drepte să se intersecteze:
  - dacă condiţia este adevărată se calculează cu relaţiile de mai sus coordonatele punctului de intersecţie;
  - dacă condiţia este falsă se afişează un mesaj corespunzător;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.11a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.11b.

3.12. Transformarea din coordonate carteziene în coordonate polare

Se consideră un punct P, pentru care se cunosc coordonatele x şi y ale punctului într-un sistem de coordonate cartezian Oxy. Se doreşte să se determine coordonatele polare (r şi θ) ale punctului.

Algoritmul de transformare din coordonate carteziene în coordonate polare constă în: citirea coordonatelor carteziene, calculul coordonatelor polare pe baza relaţiilor de mai jos, respectiv afişarea acestora.

Sistemul de coordonate polar este un sistem de coordonate bidimensional în care fiecărui punct i se asociază un unghi (φ) şi o distanţă (θ). Astfel, fiecare punct este determinat de două coordonate polare:

- coordonata radială (notată cu r) care reprezintă distanţa unui punct faţă de un punct central numit pol (echivalent cu originea sistemului de coordonate cartezian);
- coordonata unghiulară (denumită unghi polar sau azimut şi notată cu θ) care reprezintă unghiul măsurat în sens trigonometric de la direcţia de 0°, numită axa polară (echivalentă cu axa absciselor din coordonatele carteziene);

Relaţiile de calcul pentru coordonatele polare sunt:

$r = \sqrt{x^2 + y^2}$, respectiv:

$$\theta = \begin{cases} \arctan\left(\frac{y}{x}\right) & , \quad daca\ x > 0\ si\ y \geq 0 \\ \arctan\left(\frac{y}{x}\right) + 2\pi, & daca\ x > 0\ si\ y < 0 \\ \arctan\left(\frac{y}{x}\right) + \pi\ , & daca\ x < 0 \\ \frac{\pi}{2} & daca\ x = 0\ si\ y > 0 \\ \frac{3 \cdot \pi}{2} & daca\ x = 0\ si\ y < 0 \end{cases}$$

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.12a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.12b.

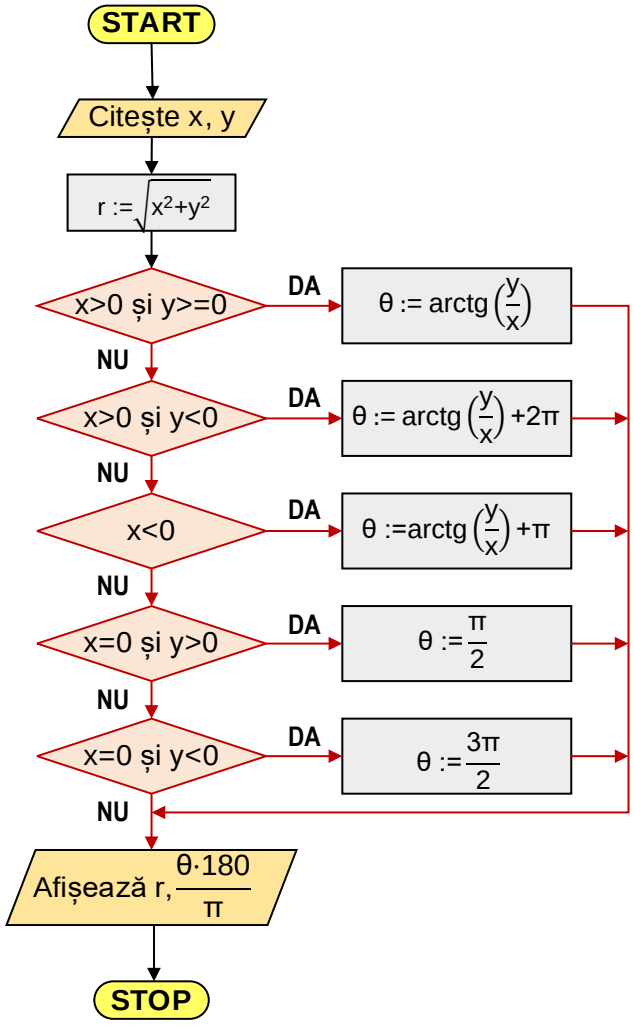
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** x,y r← sqrt(x*x+y*y) **Dacă** x>0 ŞI y≥0 **atunci** θ← arctg(y/x) **Sfârşit dacă** **Dacă** x>0 ŞI y<0 **atunci** θ← arctg(y/x)+2*π **Sfârşit dacă** **Dacă** x<0 **atunci** θ← arctg(y/x)+π **Sfârşit dacă** **Dacă** x=0 ŞI y>0 **atunci** θ← π/2 **Sfârşit dacă** **Dacă** x=0 ŞI y<0 **atunci** θ← 3*π/2 **Sfârşit dacă** **Scrie** r, θ*180/ π **Sfârşit**

Figura 3.12a. Reprezentarea algoritmului pentru transformarea din coordonate carteziene în coordonate polare

Programul MATLAB	Execuția programului

```
x=input('Introdu x [cm]:');
y=input('Introdu y [cm]:');
r=sqrt(x*x+y*y);
if (x>0 & y>=0)
    teta=atan(y/x);
end
if(x>0 & y< 0)
    teta=atan(y/x)+2*pi;
end
if(x<0)
    teta=atan(y/x)+pi;
end
if (x==0 & y>0)
    teta=pi/2;
end
if (x==0 & y<0)
    teta=3*pi/2;
end
s=sprintf('Raza polara r=%6.3f [cm]',r);
 disp(s);
s=sprintf('Unghiul polar=%6.3f[grade]',
teta*180/pi);
disp(s);
```

```
Introdu x [cm]:  18
Introdu y [cm]: -12
Raza polara r= 21.633 [cm]
Unghiul polar=326.310 [grade]
```

Figura 3.12b. Programul MATLAB și execuția acestuia pentru transformarea din coordonate carteziene în coordonate polare

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

3.13. Descompunerea în factori primi a unui număr natural

Descompunerea în factori primi a unui număr natural se bazează pe faptul că orice număr natural **n > 1** poate fi scris în mod unic sub forma:

$$n = p_1^{e1} \cdot p_2^{e2} \cdot \ldots \cdot p_k^{ek}$$

unde: $p_1 < p_2 < \ldots < p_n$ sunt numere prime (divizori ai numărului n), iar $ei > 0, \quad i = \overline{1,k}$

Algoritmul de descompunere a unui număr natura în factori primi este următorul:

- se iau pe rând divizorii lui **n** (notaţi cu **d**), începând cu **2**;
- atât timp cât **n > 1** se realizează etapele următoare:
    1. se iniţializează puterea divizorului curent (notată cu **p**) cu **0**;
    2. se verifică de câte ori **n** se divide la **d** prin împărţire directă, crescându-se puterea **p** cu **1**. În acelaşi timp se modifică valoarea curentă a lui **n**, prin împărţirea acestuia la **d** atât timp cât **d** divide pe noul **n**;
    3. dacă **p** este diferit de zero, se afişează divizorul lui **n** şi puterea acestuia;
- când **n** devine egal cu **1** procesul este încheiat;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 3.13a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 3.13b.

Schema logică	Limbajul pseudocod

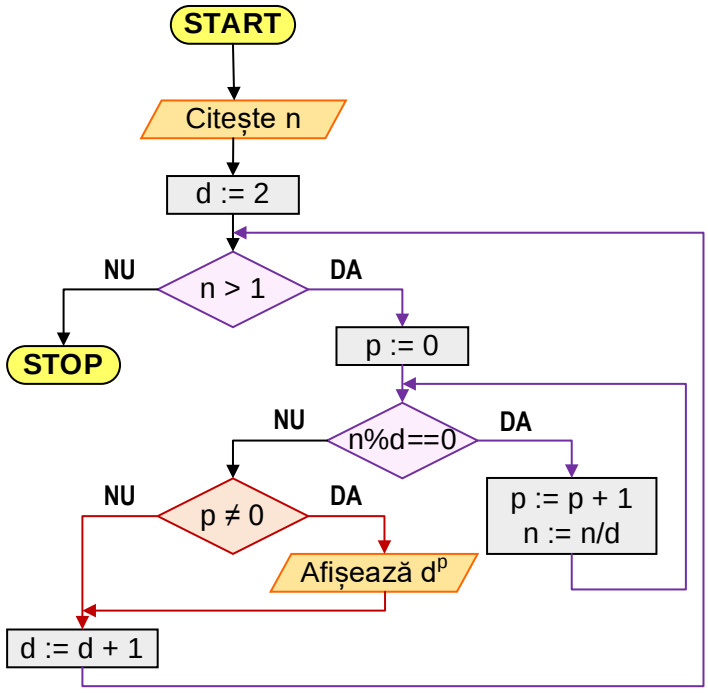
	**Început**
	  Citeşte n
	  d← 2
	  Cât timp n>1 **execută**
	  p← 0
	  Cât timp n MOD d=0 **execută**
	  p← p+1
	  n← n DIV d
	  Sfârşit cât timp
	  Dacă p≠0 **atunci**
	  Scrie d^p
	  Sfârşit dacă
	  d← d+1
	  Sfârşit cât timp
	Sfârşit

Figura 3.13a. Reprezentarea algoritmului pentru descompunerea unui număr în factori primi

Programul MATLAB	Execuția programului

`n=input('Introdu n:');`
`d=2;`
`while n>1`
`  p=0;`
`  while mod(n,d)==0`
`  p=p+1;`
`  n=n/d;`
`  end`
`  if(p~=0)`
`  s=sprintf('%d la puterea %d',d,p);`
`  disp(s);`
`  end`
`  d=d+1;`
`end`
	`Introdu n: 2520`
	`2 la puterea 3`
	`3 la puterea 2`
	`5 la puterea 1`
	`7 la puterea 1`

Figura 3.13b. Programul MATLAB și execuția acestuia pentru descompunerea unui număr în factori primi

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Capitolul 4. Calculul valorilor unor funcţii

4.1. Calculul valorii unui polinom

Un polinom de gradul n în nedeterminata **X** se scrie în formă canonică astfel:

$$P(X) = c_n \cdot X^n + c_{n-1} \cdot X^{n-1} + \ldots + c_1 \cdot X^1 + c_0$$

unde: c_0, c_1, c_2, ... , c_{n-1}, c_n se numesc coeficienţii polinomului.

Numărul $P(a) = c_n \cdot a^n + c_{n-1} \cdot a^{n-1} + \ldots + c_1 \cdot a^1 + c_0$ se numeşte valoare a polinomului **P(X)** pentru **X = a**.

Algoritmul pentru calculul valorii unui polinom presupune parcurgerea următorilor paşi:

- se citeşte gradul polinomului, adică variabila **n**;
- se citeşte valoarea variabilei **a**;
- se iniţializează valoarea polinomului cu **0**;
- utilizând un ciclu cu contor, se citesc coeficienţii polinomului şi se calculează valoarea polinomului adunând în fiecare etapă câte un termen, relaţia de calcul fiind:

$$P := P + c_i \cdot a^i, \quad i = \overline{0,n}$$

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.1a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.1b.

Schema logică	Limbajul pseudocod

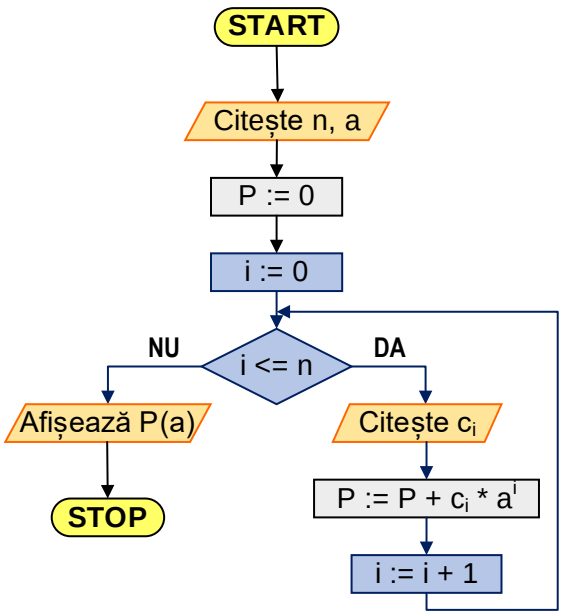
	**Început**
	  Citeşte n,a
	  P← 0
	  i← 0
	  Pentru i ← 1, n
	  Citeşte c_i
	  P ← P +c_i*a^i
	  i ← i + 1
	  Sfârşit pentru
	  Scrie P
	Sfârşit

Figura 4.1a. Reprezentarea algoritmului pentru calculul valorilor unui polinom

Programul MATLAB	Execuția programului

```
n=input('Gradul polinomului n= ');
a=input('Valoarea variabilei a=');
P=0;
for i=1:n+1
    s=sprintf(' c[%d] = ',i-1);
    c(i)=input(s);
    P = P + c(i)*a^i;
end
s=sprintf(' P(%6.3f) = %6.3f',a,P);
disp(s);
```

```
Gradul polinomului n = 3
Valoarea variabilei  a = 5
c[0] = -120
c[1] = 74
c[2] = -15
c[3] = 1
P( 5.000) = 0.000
```

Figura 4.1b. Programul MATLAB și execuția acestuia pentru calculul valorilor unui polinom

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

4.2. Calculul valorilor unei funcţii cu două ramuri

Se consideră funcţia: $y = f(x) = \begin{cases} x - 8 & daca \quad x \geq 1 \\ x^2 + 2x + 3 & daca \quad x < 1 \end{cases}$.

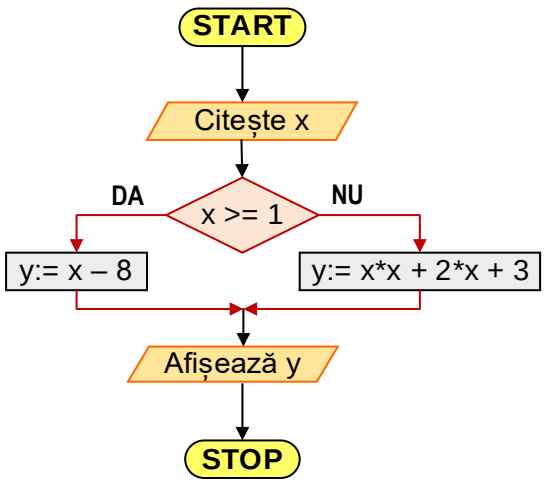
Se cere să se determine valoarea lui **y** pentru un **x** dat.

Algoritmul de calcul este următorul:

- se citeşte valoarea lui **x**;
- se compară valoarea lui **x** cu **1**, punând condiţia **x ≥ 1**. Dacă condiţia este adevărată, adică **x ≥ 1**, **y** se calculează cu relaţia: $y = x - 8$, iar dacă condiţia este falsă, adică **x < 1**, atunci **y** se calculează cu relaţia: $y = x^2 + 2x + 3$;
- se afişează valoarea lui **x** şi a lui **y**;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.2a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.2b.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte x
  Dacă x≥1 atunci
      y← x-8
  altfel
      y← x*x+2*x+3
  Sfârşit dacă
  Scrie x,y
Sfârşit
```

Figura 4.2a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu două ramuri

Programul MATLAB	Execuția programului

`x=input('Introdu x= ');`
`if(x>=1)`
`  y=x-8;`
`else`
`  y=x*x+2*x+3;`
`end`
`s=sprintf(' x = %f \t y = %f',x,y);`
`disp(s);`
	Exemplu numeric – cazul 1:
	Introdu x = **4**
	x = 4.000000 y = -4.000000
	Exemplu numeric – cazul 2:
	Introdu x = **0**
	x = 0.000000 y = 3.000000

Figura 4.2b. Programul MATLAB și execuția acestuia pentru calculul valorilor unei funcții cu două ramuri

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

4.3. Calculul valorilor unei funcţii cu trei ramuri – varianta 1

Se consideră funcţia: $y = f(x) = \begin{cases} x^2 - 3x + 4 & daca \quad x \geq 0 \\ \frac{x^2+2x+3}{x+2} & daca \quad x < 0 \end{cases}$

Se cere să se determine valoarea lui **y** pentru un **x** dat.

Algoritmul de calcul este următorul:

- se citeşte valoarea lui **x**;
- se compară valoarea lui **x** cu **0**, punând condiţia **x ≥ 0**. Dacă condiţia este adevărată, adică **x ≥ 0**, **y** se calculează cu relaţia: $y = x^2 - 3x + 4$ şi se afişează valorile lui **x** şi a lui **y**;
- dacă condiţia **x ≥ 0** este falsă, se observă că numitorul expresiei se anulează pentru **x = -2**, situaţie în care **y** nu se poate calcula. Astfel, pentru varianta în care **x < 0** trebuie să se impună o nouă condiţie, cea prin care se verifică dacă valoarea lui **x** este diferită de **-2**. Dacă această condiţie este adevărată atunci **y** se calculează cu relaţia: $y = \frac{x^2+2x+3}{x+2}$ şi se afişează valoarea lui **x** şi a lui **y**. Dacă a doua condiţie este falsă atunci nu se poate calcula **y** şi se afişează un mesaj corespunzător;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.3a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.3b.

Schema logică	Limbajul pseudocod

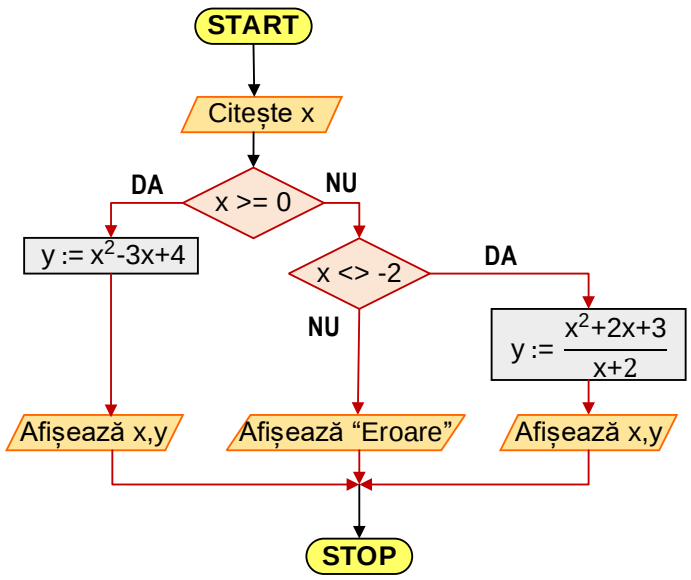
	**Început**
	  Citeşte x
	  Dacă x≥0 **atunci**
	  y← x*x-3*x+4
	  Scrie x,y
	  altfel
	  Dacă x≠-2 **atunci**
	  y← (x*x+2*x+3)/(x+2)
	  Scrie x,y
	  altfel
	  Scrie „Eroare”
	  Sfârşit dacă
	  Sfârşit dacă
	Sfârşit

Figura 4.3a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri – varianta 1

Programul MATLAB	Execuția programului

```
x=input('Introdu x= ');
if(x>=0)
    y=x*x-3*x+4;
    s=sprintf(' x = %f \t y = %f',x,y);
    disp(s);
elseif(x~=-2)
    y=(x*x+2*x+3)/(x+2);
    s=sprintf(' x = %f \t y = %f',x,y);
    disp(s);
else
    s=sprintf('Nu se poate calcula!');
    disp(s);
end
```

```
Rularea programului - cazul 1:
Introdu x = 1
x = 1.000000        y = 2.000000
Rularea programului - cazul 2:
Introdu x = -1
x = -1.000000       y = 2.000000
Rularea programului - cazul 3:
Introdu x = -2
Nu se poate calcula!
```

Figura 4.3b. Programul MATLAB și execuția acestuia pentru calculul valorilor unei funcţii cu trei ramuri – varianta 1

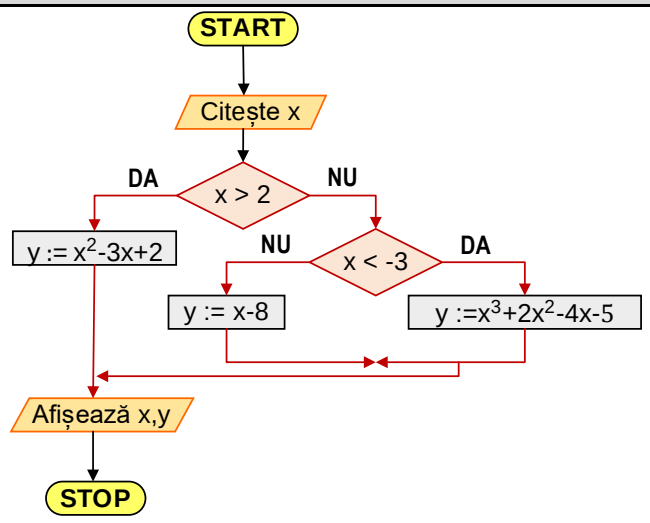
Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

4.4. Calculul valorilor unei funcţii cu trei ramuri – varianta 2

Se consideră funcția: $y = f(x) = \begin{cases} x^2 - 3x + 2 & x > 2 \\ x - 8 & -3 \le x \le 2 \\ x^3 + 2x^2 - 4x - 5 & x < -3 \end{cases}$

Se cere să se determine valoarea lui **y** pentru un **x** dat.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte x
  Dacă x>2 atunci
      y← x*x-3*x+2
  altfel
      Dacă x<-3 atunci
          y← x*x*x+2*x*x-4*x-5
      altfel
          y← x-8
      Sfârşit dacă
  Sfârşit dacă
  Scrie x,y
Sfârşit
```

Figura 4.4a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri – varianta 2

Algoritmul de calcul este următorul:

- se citeşte valoarea lui **x**;
- se compară valoarea lui **x** cu **2**, punând condiţia **x > 2**. Dacă condiţia este adevărată, **y** se calculează cu relaţia: $y = x^2 - 3x + 2$;

- dacă condiţia **x > 2** este falsă, se observă că pentru calculul valorii lui **y** avem în continuare două ramuri (două variante). Se impune astfel condiţia ca **x < -3**. Dacă această condiţie este adevărată y se calculează cu relaţia: $y = x^3 + 2x^2 - 4x - 5$, iar dacă condiția este falsă se calculează y cu relaţia: $y = x - 8$;
- se afişează valorile lui **x** şi a lui **y**;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.4a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.4b.

Programul MATLAB	Execuția programului

```
x=input('Introdu x= ');
if(x > 2)
    y=x*x-3*x+4;
elseif(x < -3)
    y = x*x*x+2*x*x-4*x-5;
else
    y=x-8;
end
s=sprintf('x = %6.3f \t y = %6.3f',x,y);
disp(s);
```

```
Exemplu numeric – cazul 1:
Introdu x = -5
x = -5.000, y = -40.000
Exemplu numeric – cazul 2:
Introdu x = 0
x = 0.000, y = -8.000
Exemplu numeric – cazul 3:
Introdu x = 4
x = 4.000, y = 6.000
```

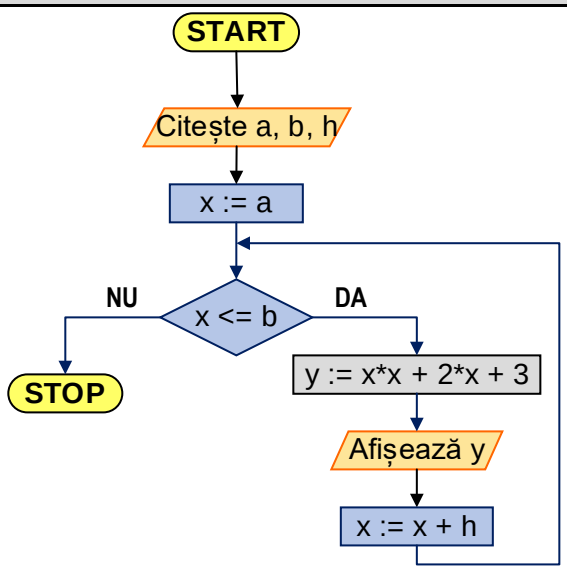
Figura 4.4b. Programul MATLAB și execuția acestuia pentru calculul valorilor unei funcţii cu trei ramuri – varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

4.5. Calculul valorilor unei funcții pe un interval

Se consideră funcția: $y = x^2 + 2x + 3$.Se cere să se calculeze toate valorile lui y pentru $x \in [a,b]$, parcurs cu pasul *h*.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte a,b,h
  Pentru x←a,b,h
    y ← x*x +2*x+3
    Scrie x,y
  Sfârşit pentru
Sfârşit
```

Figura 4.5a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii pe un interval

Algoritmul de calcul constă în:

- citirea limitelor intervalului în care variabila **x** ia valori (adică variabilele **a** şi **b**), respectiv a pasului cu care se parcurge intervalul (variabila **h**);
- pentru atribuirea către variabila **x** a valorilor din intervalul **[a,b],** parcurs cu pasul **h**, se utilizează un ciclu cu contor, astfel:
    1. se iniţializează valoarea variabilei **x** cu prima valoare din interval, adică cu **a**, deci **x := a**;
    2. se verifică dacă valoarea variabilei x este în intervalul [a,b], adică se verifică condiţia **x<=b**. Dacă condiţia este adevărată, se atribuie lui y valoarea $y = x^2 + 2x + 3$ se afișează **x** şi **y** şi se trece la pasul 3.
    3. Se modifică valoarea contorului cu pasul h, adică: **x := x + h** şi se revine la pasul 2;
    4. dacă condiţia de la pasul 2 nu este adevărată, se părăseşte ciclul cu contor;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.5a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.5b.

Programul MATLAB	Execuția programului

`a=input('Introdu a, a= ');`
`b=input('Introdu b, b= ');`
`h=input('Introdu h, h= ');`
`for x=a:h:b`
`  y=x*x+2*x+3;`
`  s=sprintf(' x = %6.3f \t y = %6.3f',x,y);`
`  disp(s);`
`end`
	`Introdu a, a = -5`
	`Introdu b, b = 5`
	`Introdu h, h = 1`
	`x = -5.000  y = 18.000`
	`x = -4.000  y = 11.000`
	`x = -3.000  y =  6.000`
	`x = -2.000  y =  3.000`
	`x = -1.000  y =  2.000`
	`x =  0.000  y =  3.000`
	`x =  1.000  y =  6.000`
	`x =  2.000  y = 11.000`
	`x =  3.000  y = 18.000`
	`x =  4.000  y = 27.000`
	`x =  5.000  y = 38.000`

Figura 4.5b. Programul MATLAB și execuția acestuia pentru calculul valorilor unei funcţii pe un interval

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

4.6. Calculul valorilor unei funcţii cu două ramuri pe un interval

Calculul valorii funcţiei: $y = f(x) = \begin{cases} x - 8 & daca \quad x \geq 1 \\ x^2 + 2x + 3 & daca \quad x < 1 \end{cases}$ pentru $x \in [a,b]$, parcurs cu pasul h.

Algoritmul de calcul constă în:

- citirea limitelor intervalului în care variabila **x** ia valori (adică variabilele **a** şi **b**), respectiv a pasului cu care se parcurge intervalul (variabila **h**);
- pentru atribuirea către variabila **x** a valorilor din intervalul **[a,b],** parcurs cu pasul **h**, se utilizează un ciclu cu contor, astfel:
    1. se iniţializează valoarea variabilei **x** cu prima valoare din interval, adică cu **a**, deci **x := a**;
    2. se verifică dacă valoarea variabilei **x** este în intervalul **[a,b]**, adică se verifică condiţia **x<=b**. Dacă condiţia este adevărată se trece la pasul următor, iar dacă condiţia este falsă se părăseşte ciclul, continuând cu secvenţa următoare ciclului;
    3. pentru calculul valorii lui **y** se verifică condiţia **x >= 1**. Dacă condiţia este adevărată **y** se calculează cu relația: $y = x - 8$, iar dacă condiția este falsă y se calculează cu relaţia: $y = x^2 + 2x + 3$;
    4. se afişează **x** şi **y** şi se trece la pasul următor.
    5. se modifică valoarea contorului cu pasul h, adică: **x := x + h**, după care se revine la pasul 2;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.6a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.6b.

Schema logică	Limbajul pseudocod

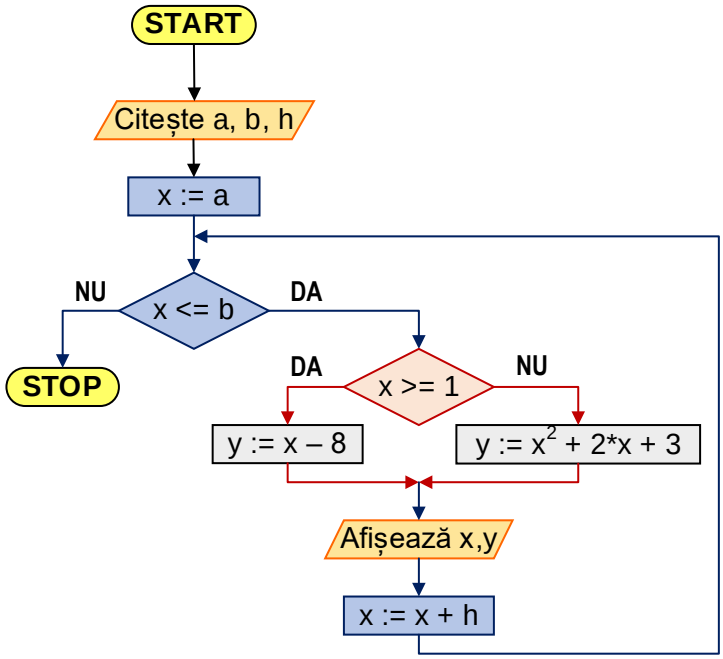
	**Început**
	  Citeşte a,b,h
	  Pentru x←a,b,h
	  Dacă x≥1 **atunci**
	  y← x-8
	  altfel
	  y ← x*x +2*x+3
	  Sfârşit dacă
	  Scrie x,y
	  Sfârşit pentru
	Sfârşit

Figura 4.6a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu două ramuri pe un interval

Programul MATLAB	Execuția programului

```
a=input('Introdu a, a= ');
b=input('Introdu b, b= ');
h=input('Introdu h, h= ');
for x=a:h:b
    if(x>=1)
        y = x-8;
    else
        y=x*x+2*x+3;
    end
    s=sprintf('x=%6.3f \t y=%6.3f',x,y);
    disp(s);
end
```

```
Introdu a, a = -3
Introdu b, b = 3
Introdu h, h = 1
x=-3.000            y= 6.000
x=-2.000            y= 3.000
x=-1.000            y= 2.000
x= 0.000            y= 3.000
x= 1.000            y=-7.000
x= 2.000            y=-6.000
x= 3.000            y=-5.000
```

Figura 4.6b. Programul MATLAB şi execuţia acestuia pentru calculul valorilor unei funcţii cu două ramuri pe un interval

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

4.7. Calculul valorilor unei funcţii cu trei ramuri pe un interval – varianta 1

Se consideră funcţia: $y = f(x) = \begin{cases} x^2 - 3x + 4 & daca \quad x \geq 0 \\ \frac{x^2+2x+3}{x+2} & daca \quad x < 0 \end{cases}$, $x \in [-a, a]$, parcurs cu pasul *p*.

Algoritmul de calcul constă în:

- se citeşte variabila **a,** respectiv pasul cu care se parcurge intervalul (variabila **p**);

- pentru atribuirea către variabila **x** a valorilor din intervalul **[-a,a],** parcurs cu pasul **p**, se utilizează un ciclu cu contor, astfel:

1. se iniţializează valoarea variabilei **x** cu prima valoare din interval, adică cu **-a**, deci **x := -a**;
2. se verifică dacă valoarea variabilei **x** este în intervalul **[-a,a]**, adică se verifică condiţia **x<=a**. Dacă condiţia este adevărată se trece la pasul următor, iar dacă condiţia este falsă se părăseşte ciclul, continuând cu secvenţa următoare ciclului;
3. pentru calculul valorii lui **y** se verifică condiţia condiţia **x ≥ 0**. Dacă condiţia este adevărată, adică **x ≥ 0**, **y** se calculează cu relaţia: $y = x^2 - 3x + 4$ şi se afişează valorile lui **x** şi a lui **y**; Dacă condiţia **x ≥ 0** este falsă, se observă că numitorul expresiei se anulează pentru **x = -2**, situaţie în care **y** nu se poate calcula. Astfel, pentru varianta în care **x < 0** trebuie să se impună o nouă condiţie, cea prin care se verifică dacă valoarea lui **x** este diferită de **-2**. Dacă această condiţie este adevărată atunci **y** se calculează cu relaţia: $y = \frac{x^2+2x+3}{x+2}$ şi se afişează valoarea lui **x** şi a lui **y**. Dacă a doua condiţie este falsă atunci nu se poate calcula **y** şi se afişează un mesaj corespunzător; se afişează **x** şi **y** şi se trece la pasul următor;
4. se modifică valoarea contorului cu pasul h, adică: **x := x + p**, după care se revine la pasul 2.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.7a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.7b.

Schema logică	Limbajul pseudocod

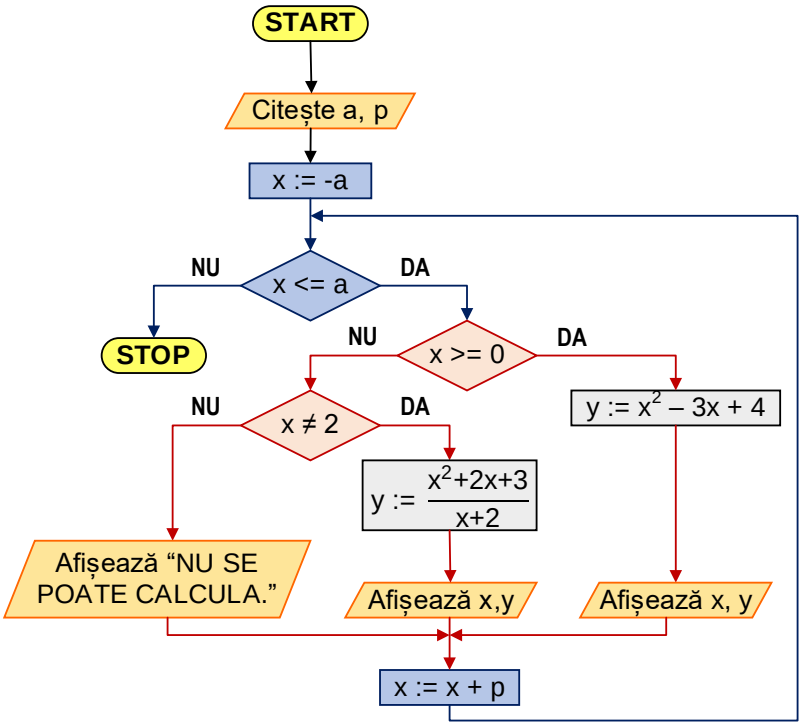
	**Început**
	  Citeşte a,p
	  Pentru x←-a,a,p
	  Dacă x≥0 **atunci**
	  y← x*x-3*x+4
	  Scrie x,y
	  altfel
	  Dacă x≠2 **atunci**
	  y← (x*x+2*x+3)/(x+2)
	  Scrie x,y
	  altfel
	  Scrie „Nu se poate calcula”
	  Sfârşit dacă
	  Sfârşit dacă
	  Sfârşit pentru
	Sfârşit

Figura 4.7a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri pe un interval – varianta 1

Programul MATLAB	Execuția programului

```
a=input('Introdu a, a= ');
p=input('Introdu p, p= ');
for x=-a:p:a
    if(x>=0)
        y=x*x-3*x+4;
        s=sprintf('x=%6.3f \t y=%6.3f',x,y);
        disp(s);
    elseif(x~=-2)
```

```
Introdu a, a = 3
Introdu p, p = 1
x=-3.000            y= 6.000
Nu se poate calcula!
x=-1.000            y= 2.000
x= 0.000            y= 4.000
x= 1.000            y= 2.000
x= 2.000            y= 2.000
```

```
        y=(x*x+2*x+3)/(x+2);
        s=sprintf('x=%6.3f \t y=%6.3f',x,y);
        disp(s);
    else
        disp(' Nu se poate calcula!');
    end
end
```

```
x= 3.000        y= 4.000
```

Figura 4.7b. Programul MATLAB și execuția acestuia pentru calculul valorilor unei funcţii cu trei ramuri pe un interval – varianta 1

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

4.8. Calculul valorilor unei funcţii cu trei ramuri pe un interval – varianta 2

Numerele lui Fibonacci sunt definite astfel: $F_n = \begin{cases} 0 & daca\ n = 0 \\ 1 & daca\ n = 1 \\ F_{n-1} + F_{n-2} & daca\ n > 1 \end{cases}$.

Primele 17 numere din şirul lui Fibonacci sunt: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987.

Prin împărţirea unui element (de la elementul 14 în sus) al şirului lui Fibonacci la precedentul său se obţine valoarea **1.61803** (233 : 144 = 1.61803**,** 377 : 233 = 1.61803, etc), denumită **numărul de aur**. Acesta se regăseşte în arhitectură, pictură, sculptură, estetică şi artă în general, la formarea unor proporţii armonioase, plăcute ochiului.

Schema logică	Limbajul pseudocod

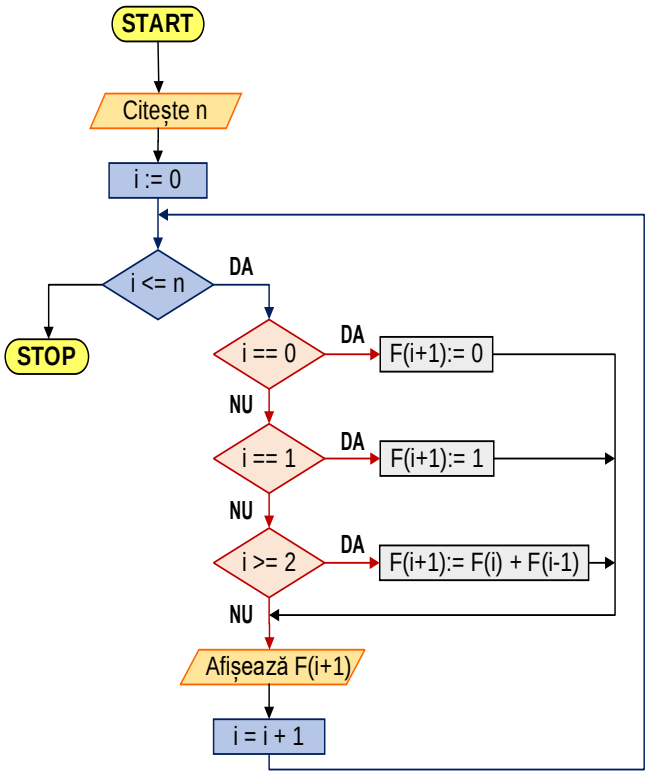
	**Început**
	  Citeşte n
	  Pentru i←0,n
	  Dacă i=0 **atunci**
	  F(i+1)← 0
	  Sfârşit dacă
	  Dacă x=1 **atunci**
	  F(i+1)← 1
	  Sfârşit dacă
	  Dacă x≥2 **atunci**
	  F(i+1)← F(i)+F(i-1)
	  Sfârşit dacă
	  Scrie F(i+1)
	  Sfârşit pentru
	Sfârşit

Figura 4.8a. Reprezentarea algoritmului pentru calculul valorilor unei funcţii cu trei ramuri pe un interval – varianta 2

Algoritmul de determinare a numerelor din şirul lui Fibonacci este următorul:

- se citeşte numărul de elemente, **n**;

- se parcurge intervalul **[0,n]**, cu pasul **1**, pentru generarea celor **n** elemente, utilizând un ciclu cu contor, astfel:
    1. se iniţializează contorul **i**, cu valoarea iniţială, adică **i := 0**;
    2. se verifică dacă **i ≤ n**, dacă condiţia este adevărată se trece la pasul următor, iar dacă condiţia este falsă se părăseşte ciclul cu contor, continuând cu secvenţa următoare ciclului;
    3. se verifică dacă **i** este egal cu **0**, caz în care **F(1) := 0,** (în MATLAB numerotarea indicilor tablourilor încep de la 1);
    4. se verifică dacă **i** este egal cu **1**, caz în care **F(2) := 1**;
    5. se verifică dacă **i** este mai mare sau egal cu **2**, caz în care **F(i+1) := F(i) + F(i-1)**;
    6. se modifică valoarea contorului **i** cu pasul de **1**, adică **i := i +1** şi se revine la pasul 2;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 4.8a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 4.8b.

Programul MATLAB	Execuția programului

`n=input('Introdu n= ');`
`for i=0:n`
`  if(i==0)`
`  F(i+1)=0;`
`  end`
`  if(i==1)`
`  F(i+1)=1;`
`  end`
`  if(i>=2)`
`  F(i+1)=F(i)+F(i-1);`
`  end`
`  s=sprintf('F[%2d]=%6d',i,F(i+1));`
`  disp(s);`
`end`
	`Introdu n = 8`
	`F[ 0]= 0`
	`F[ 1]= 1`
	`F[ 2]= 1`
	`F[ 3]= 2`
	`F[ 4]= 3`
	`F[ 5]= 5`
	`F[ 6]= 8`
	`F[ 7]= 13`
	`F[ 8]= 21`

Figura 4.8b. Programul MATLAB și execuția acestuia pentru calculul valorilor unei funcţii cu trei ramuri pe un interval– varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Capitolul 5. Operaţii cu tablouri unidimensionale

Tablourile reprezintă colecții de date de obicei de același tip, grupate sub un nume comun. Elementele lor sunt identificate prin indici care sunt numere naturale, indicele primului element al tabloului fiind 1. Tablourile unidimensionale sunt denumite și „vectori" sau șiruri unidimensionale.

În majoritatea limbajelor de programare, aplicarea unei operații (citire, afișare, adunare, înmulţire, etc) fiecărui element al unui tablou unidimensional se realizează de obicei utilizând cicluri cu contor. Limbajul de programare MATLAB, spre deosebire de alte limbaje de programare, are ca elemente de bază vectorii și matricele și are operațiile adaptate pentru acestea, fapt care face codul să fie ușor de citit și de executat.

În exemplele următoare sunt prezentate o serie de algoritmi utilizaţi în rezolvarea unor probleme ce implică tablouri unidimensionale sau probleme a căror rezolvare conduce la utilizarea tablourilor unidimensionale.

5.1. Introducerea / afişarea elementelor unui tablou unidimensional

În acest exemplu sunt prezentate operaţiile de introducere (citire), respectiv afişare (scriere) a elementelor unui tablou unidimensional cu „ **n** " elemente, numărul acestora fiind introdus de la tastatură.

Prima variantă este folosind funcții specifice limbajului de programare MATLAB. Citirea unui șir se face folosind funcția de intrare **input( )** și introducând efectiv elementele șirului prin enumerarea între paranteze pătrate, iar afișarea se poate face prin omiterea caracterului **;** sau folosind funcțiile **disp()** sau **display()**.

Programul MATLAB și execuția acestuia

```
a=input('Dati sirul a ')

Execuţia programului este următoarea:
Dati sirul a [1 2 7 -3 5 10]
a =
     1     2     7    -3     5    10
```

```
a=input('Dati sirul a ');
disp(a)

Execuţia programului este următoarea:
Dati sirul a [1 2 7 -3 5 10]
     1     2     7    -3     5    10
```

```
a=input('Dati sirul a ');
display(a)

Execuţia programului este următoarea:
Dati sirul a [1 2 7 -3 5 10]
a =
     1     2     7    -3     5    10
```

Figura 5.1a. Exemple de folosire a instrucțiunilor MATLAB pentru introducerea/afișarea elementelor unui tablou unidimensional

Se observă în exemplele din figura 5.1a. că nu este necesar ca întâi să se citească numărul de elemente ale șirului, adică dimensiunea acestuia.

A doua variantă se referă la algoritmul clasic, acesta fiind următorul:

- se citește **n** - numărul de elemente ale tabloului unidimensional;
- se utilizează un ciclu cu contor pentru citirea valorilor elementelor tabloului unidimensional, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **1**;

2. se evaluează condiţia „ **i <= n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunea care reprezintă corpul ciclului cu contor, în acest caz se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i** ;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- se utilizează un ciclu cu contor pentru afişarea valorilor elementelor tabloului unidimensional, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **1**;

2. se evaluează condiţia „ **i <= n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunea care reprezintă corpul ciclului cu contor, în acest caz se execută operaţia de afişare a valorii elementului curent al tabloului, adică **Afişează a_i** ;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;

Observaţie: Cu ajutorul primului ciclu cu contor se realizează citirea elementelor tabloului unidimensional, iar prin utilizarea celui de-al doilea ciclu cu contor se realizează afişarea elementelor tabloului unidimensional.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.1b, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.1c.

Schema logică	Limbajul pseudocod

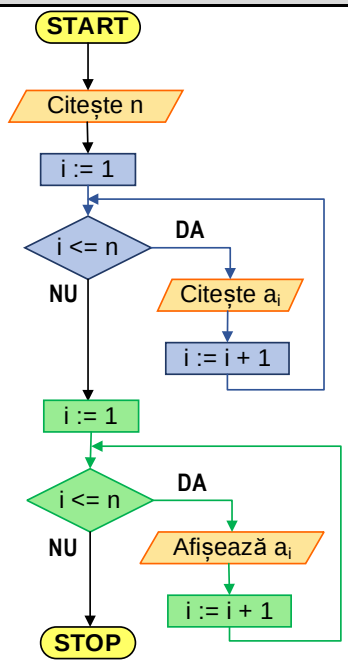
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  Pentru i←1,n
	  Scrie a_i
	  Sfârşit pentru
	Sfârşit

Figura 5.1b. Reprezentarea algoritmului pentru introducerea/afişarea elementelor unui tablou unidimensional

Programul MATLAB	Execuția programului

`n=input(' Introduceti n, n= ');`
`  for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`  end`
`  for i=1: n`
`  disp(a(i));`
`end`
	`Introduceti n, n = 5`
	`a[ 1] = 2`
	`a[ 2] = 4`
	`a[ 3] = 5`
	`a[ 4] = 3`
	`a[ 5] = 7`
	`2`
	`4`
	`5`
	`3`
	`7`

Figura 5.1c. Programul MATLAB și execuția acestuia pentru introducerea/afişarea elementelor unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.2. Calculul sumei elementelor unui tablou unidimensional

5.2.1. Calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu contor

Se consideră următoarea problemă: un strungar execută pe parcursul a **n** - zile, aceeași piesă, dar în cantități diferite de la o zi la alta. Se doreşte să se determine numărul total de piese pe care le-a realizat strungarul în cele **n** zile. Problema constă în calculul sumei elementelor unui șir de valori, adică în programare, calculul sumei elementelor unui tablou unidimensional.

Pentru calculul sumei, se inițializează suma cu valoarea **0** (zero) şi se repetă pentru fiecare element al tabloului unidimensional două operaţii: citirea valorii elementului curent al tabloului şi adăugarea acestuia la sumă utilizând o relaţie de forma:

$$\mathbf{S := S + a_i}$$

Relaţia de mai sus trebuie interpretată astfel: valoarea nouă a sumei este egală cu valoarea (sau primeşte valoarea) veche a sumei la care se adaugă valoarea elementului curent al tabloului unidimensional.

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma cu valoarea **0** (zero – element neutru pentru operaţia de adunare);
- se utilizează un ciclu cu contor în cadrul căruia se va realiza citirea valorilor elementelor tabloului unidimensional şi adăugarea acestei valori sumei. În cadrul ciclului cu contor se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **1**;

2. se evaluează condiţia „ **i <= n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor, adică:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i**;
- se adaugă valoarea elementului curent al tabloului la sumă, utilizând relația: **S := S + a_i**;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu contor (adică s-au citit cele **n** elemente ale tabloului şi s-au adunat la sumă) se afişează valoarea calculată a sumei, adică **Afişează S**;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.2a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.2b.

Schema logică	Limbajul pseudocod

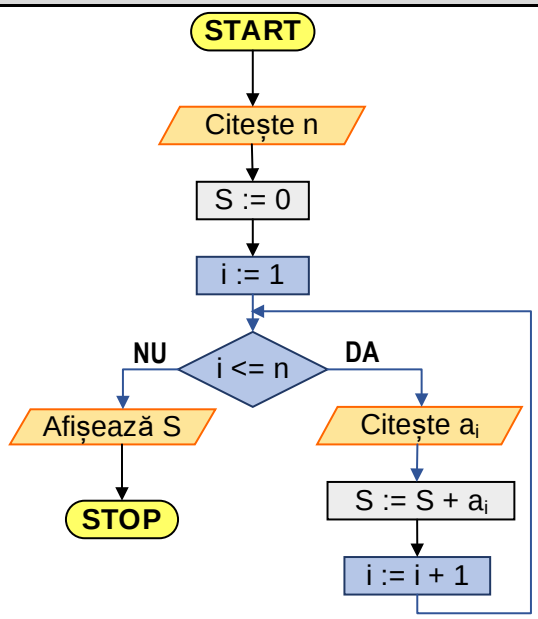
	**Început**
	  Citeşte n
	  S←0
	  Pentru i←1,n
	  Citeşte a_i
	  S←S+a_i
	  Sfârşit pentru
	  Scrie S
	Sfârşit

Figura 5.2a. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou unidimensional utilizând un ciclu cu contor

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`S=0;`
`for i=1:n`
`  s=sprintf(' Ziua %2d = ',i);`
`  a(i)=input(s);`
`  S=S+a(i);`
`end`
`mesaj=sprintf(' Suma este S = %d ',S);`
`disp(mesaj)`
	`Introduceti n, n = 5`
	` Ziua 1 = 23`
	` Ziua 2 = 21`
	` Ziua 3 = 24`
	` Ziua 4 = 22`
	` Ziua 5 = 23`
	`Suma este S = 113`

Figura 5.2b. Programul MATLAB și execuția acestuia pentru calculul sumei elementelor unui tablou unidimensional utilizând un ciclu cu contor

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.2.2. Calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test inițial

În general, prelucrarea elementelor tablourilor unidimensionale se realizează utilizând cicluri cu contor, însă se pot utiliza şi celelalte variante de cicluri. În acest exemplu este prezentat calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test iniţial. Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma cu 0 (zero – element neutru pentru operaţia de adunare);
- se atribuie variabilei i valoarea 1;
- se utilizează un ciclu cu test iniţial în corpul căruia se vor realiza următoarele operaţii: citirea valorilor elementelor tabloului unidimensional şi adăugarea acestora la sumă. Pentru aceasta trebuie parcurşi următorii paşi:

1. se evaluează condiţia „**i <= n** ” prin care se verifică dacă valoarea curentă a lui **i** este mai mică decât **n**. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 2). Dacă condiţia este **falsă**, se

continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

2. se execută instrucţiunile care reprezintă corpul ciclului cu contor, adică:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i**;
- se adaugă valoarea elementului curent al tabloului la sumă, utilizând relația: **S := S + a_i**;

3. se execută instrucţiunea prin care se modifică valoarea variabilei **i**, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **1**, adică la verificarea condiţiei **i <= n**;

- după ieşirea din ciclu cu test iniţial (adică s-au citit cele **n** elemente ale tabloului şi s-au adunat la sumă) se afişează valoarea calculată a sumei, adică **Afişează S**;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.2c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.2d.

Schema logică	Limbajul pseudocod

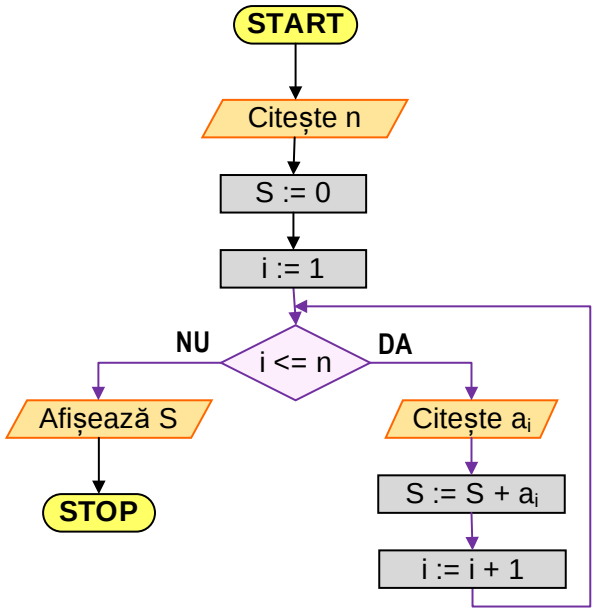
	**Început**
	  Citeşte n
	  S←0
	  i←1
	  Cât timp i≤n **execută**
	  Citeşte a_i
	  S←S+a_i
	  i←i+1
	  Sfârșit cât timp
	  Scrie S
	Sfârşit

Figura 5.2c. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test iniţial

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`S = 0; i = 1;`
`while i<=n`
`  s=sprintf('a[%2d] = ',i);`
`  a(i)=input(s);`
`  S = S + a(i);`
`  i = i + 1;`
`end`
`mesaj=sprintf(' Suma este S = %d ',S);`
`disp(mesaj)`
	`Introduceti n, n = 5`
	`a[1] = 2`
	`a[2] = 4`
	`a[3] = 5`
	`a[4] = 7`
	`a[5] = 3`
	`Suma este S = 21`

Figura 5.2d. Programul MATLAB şi execuţia acestuia pentru calculul sumei elementelor unui tablou unidimensional utilizând ciclu cu test iniţial

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Dacă nu se dorește neapărat folosirea unui ciclu cu contor sau a unui ciclu cu test inițial, se poate folosi o altă variantă de rezolvare, și anume folosind funcția **sum()** a limbajului de programare MATLAB. Reprezentarea algoritmului

prin schemă logică și limbaj pseudocod este prezentată în figura 5.2e, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.2f.

Schema logică	Limbajul pseudocod

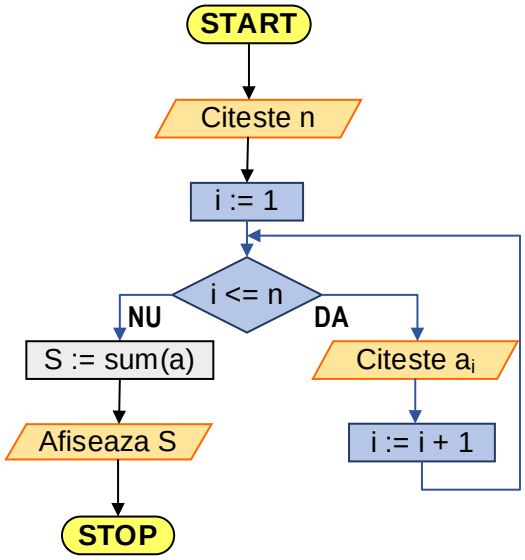
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  S←sum(a)
	  Scrie S
	Sfârşit

Figura 5.2e. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou unidimensional apelând funcția **sum()**

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`for i=1:n`
`  s=sprintf(' Ziua %2d = ',i);`
`  a(i)=input(s);`
`end`
`S=sum(a);`
`mesaj=sprintf(' Suma este S = %d ',S);`
`disp(mesaj)`
	`Introduceti n, n = 5`
	` Ziua 1 = 23`
	` Ziua 2 = 21`
	` Ziua 3 = 24`
	` Ziua 4 = 22`
	` Ziua 5 = 23`
	`Suma este S = 113`

Figura 5.2f. Folosirea unei funcții specifice limbajului MATLAB pentru suma elementelor unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.3. Calculul sumei elementelor strict pozitive ale unui tablou unidimensional

Se consideră următoarea problemă tehnică: din sistemul de preambalare a unui produs care se livrează la vrac rezultă cutii cu produse a căror greutate variază faţă de greutatea nominală. Se constată că unele cutii au abatere pozitivă, altele abatere negativă, iar unele cutii au exact greutatea nominală. Se măsoară greutatea fiecărei cutii şi din valoarea obţinută se scade greutatea nominală, obţinându-se un şir de valori care pot fi pozitive, negative sau nule (tabelul 5.3.1). Se doreşte să se determine cantitatea totală cu care cutiile cu abatere pozitivă depăşesc greutatea nominală.

Problema se reduce la calculul sumei elementelor strict pozitive din şirul de valori.

Tabelul 5.3.1. Şirul de valori

Element	1	2	3	4	5	6	7	8	9	10
Valoare	15	-12	0	8	14	-9	-15	0	10	-20

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma elementelor cu **0** (zero – element neutru pentru adunare);
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **1**;

2. se evaluează condiţia „ **i <= n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i**;
- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este strict pozitiv, impunând condiţia ca „ **a_i > 0** ". Dacă condiţia este adevărată înseamnă că elementul curent este strict pozitiv, astfel că pe ramura „ **DA** ”, se adaugă elementul curent la sumă. Ramura „ **NU** ” corespunde valorilor negative sau nule ale elementelor tabloului, astfel că pe aceasta nu se execută nimic.

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt strict pozitive, iar cele strict pozitive s-au adăugat la sumă) se afişează valoarea calculată a sumei, adică **Afişează S**;

Schema logică	Limbajul pseudocod

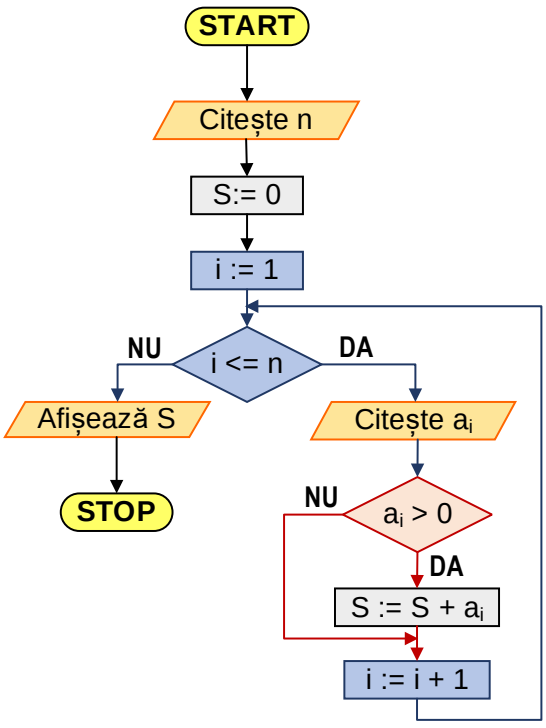
	**Început**
	  Citeşte n
	  S←0
	  Pentru i←1,n
	  Citeşte a_i
	  Dacă a_i>0 **atunci**
	  S←S+a_i
	  Sfârşit dacă
	  Sfârşit pentru
	  Scrie S
	Sfârşit

Figura 5.3a. Reprezentarea algoritmului pentru calculul sumei elementelor strict pozitive ale unui tablou unidimensional

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.3.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Suma primeşte iniţial valoarea **0** (zero).

La început (**i := 1**) se citeşte valoarea primului element al tabloului: **a_1 = 15**, şi se verifică dacă este strict pozitiv. Primul element fiind strict pozitiv, se adună la sumă, valoarea nouă a sumei fiind obţinută prin adunarea valorii anterioare a sumei (adică **0**) cu valoarea elementului curent (**a_1= 15**), astfel: **S :=** 0 + 15 = **15**;

În continuare, contorul **i** își creşte valoarea cu **1**, astfel că **i = 2**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (a_2 **= -12**) şi se verifică dacă este strict pozitiv. Deoarece valoarea acestuia nu este strict pozitivă, se continuă parcurgerea pe ramura **NU** a instrucţiunii de selecţie, în continuare mărindu-se valoarea contorului **i** cu **1** (**i = 3**).

Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n**, adică nu se mai respectă condiţia **i <= n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea sumei **S**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.3a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.3b.

Tabelul 5.3.2. Calculul sumei

Valori obținute pentru: **n = 10**				
i	**a_i**	**a_i > 0**	**S**	**Ecran**
			0	
1	15	DA	S= 0 + **15** = 15	
2	-12	NU		
3	0	NU		
4	8	DA	S= 15 + **8** = 23	
5	14	DA	S= 23 + **14** = 37	
6	-9	NU		
7	-15	NU		
8	0	NU		
9	10	DA	S= 37 + **10** = 47	
10	-20	NU		**47**

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`S=0;`
`for i=1:n`
`  s=sprintf('a[%2d] = ',i);`
`  a(i)=input(s);`
`  if a(i) > 0`
`  S = S + a(i);`
`  end`
`end`
`mesaj=sprintf('S = %d ',S);`
`disp(mesaj)`
	`Introduceti n, n = 10`
	`a[1] = 15`
	`a[2] = -12`
	`a[3] = 0`
	`a[4] = 8`
	`a[5] = 14`
	`a[6] = -9`
	`a[7] = -15`
	`a[8] = 0`
	`a[9] = 10`
	`a[10] = -20`
	`S = 47`

Figura 5.3b. Programul MATLAB și execuția acestuia pentru calculul sumei elementelor strict pozitive unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Schema logică	Limbajul pseudocod

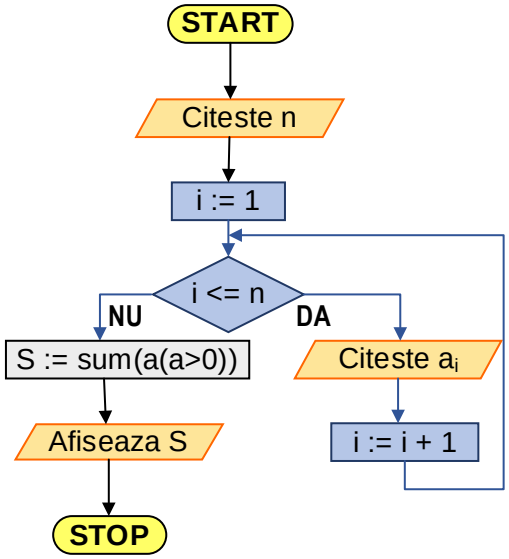
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  S←sum(a(a>0))
	  Scrie S
	Sfârşit

Figura 5.3c. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou unidimensional apelând funcția **sum()**

O altă variantă de rezolvare este folosind funcția **sum()** specifică limbajului de programare MATLAB. Pentru a identifica pozițiile pe care se află elementele pozitive se folosește condiția **a>0**,. Pur și simplu scriind această condiție MATLAB returnează un vector de valori 0 și 1, 0 pe pozițiile unde în șir se regăsesc valorile negative, iar 1 pe pozițiile unde în șir se regăsesc valori pozitive. Pentru a identifica valorile elementelor pozitive se folosește condiția **a(a>0)**, unde **a** este tabloul unidimensional.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.3c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.3d.

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`end`
`S=sum(a(a>0));`
`mesaj=sprintf('S = %d ',S);`
`disp(mesaj)`
	`Introduceti n, n = 10`
	` a[1] = 15`
	` a[2] = -12`
	` a[3] = 0`
	` a[4] = 8`
	` a[5] = 14`
	` a[6] = -9`
	` a[7] = -15`
	` a[8] = 0`
	` a[9] = 10`
	` a[10] = -20`
	`S = 47`

Figura 5.3d. Folosirea unei funcții specifice limbajului MATLAB pentru suma elementelor unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.4. Produsul elementelor strict pozitive ale unui tablou unidimensional

Se consideră şirul de valori din tabelul 5.4.1. Se doreşte calculul produsului elementelor strict pozitive din şir.

Tabelul 5.4.1. Şirul de valori

Element	1	2	3	4	5	6	7	8
Valoare	-5	2	0	1	-3	4	-2	3

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează produsul elementelor cu **1, P := 1** (unu – element neutru pentru înmulţire);
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** a_i;

- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este strict pozitiv, impunând condiţia ca „ a_i **> 0** ". Dacă condiţia este adevărată înseamnă că elementul curent este strict pozitiv, se continuă pe ramura „ **DA** ”, se înmulţeşte la produs valoarea elementului curent. Ramura „ **NU** ” corespunde valorilor negative sau nule ale elementelor tabloului, în consecintă pe această ramură nu se execută nimic;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;
- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt strict pozitive, iar cele strict pozitive s-au înmulţit la produs) se afişează valoarea calculată a produsului, adică **Afişează P**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.4.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Produsul primeşte iniţial valoarea **1** (unu).

La început (**i := 0**) se citeşte valoarea primului element al tabloului: a_0 **= -5**, şi se verifică dacă este strict pozitiv. Primul element nefiind strict pozitiv, nu se verifică condiţia a_i **> 0**, deci se trece la modificarea valorii contorului;

În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 1**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (a_1 **= 2**) şi se verifică dacă este strict pozitiv. Deoarece valoarea acestuia este strict pozitivă, se continuă parcurgerea pe ramura **DA** a instrucţiunii de decizie, astfel că se modifică valoarea produsului prin înmulţirea acestuia cu valoarea elementului curent astfel: **P := P *** a_i, apoi creşte valoarea contorului **i** cu **1** (**i = 2**).

Tabelul 5.4.2. Calculul produsului

Valori obţinute pentru: **n = 8**				
i	a_i	a_i **> 0**	**P**	**Ecran**
			1	
0	-5	NU		
1	2	DA	P:= 1 * **2** = 2	
2	0	NU		
3	1	DA	P:= 2 * **1** = 2	
4	-3	NU		
5	4	DA	P:= 2 * **4** = 8	
6	-2	NU		
7	3	DA	P:= 8 * **3** = 24	**24**

Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n-1**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea produsului **P**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.4a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.4b.

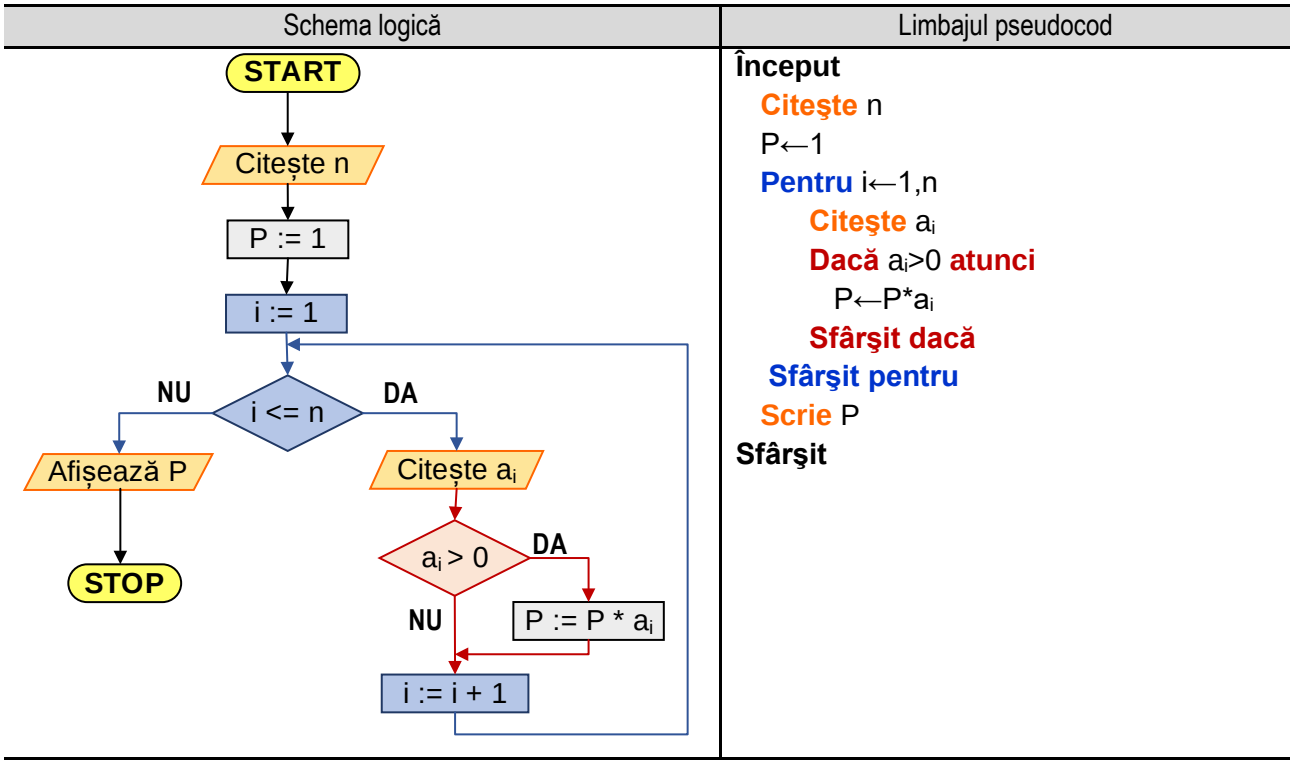


Figura 5.4a. Reprezentarea algoritmului pentru calculul produsului elementelor strict pozitive ale unui tablou unidimensional

Programul MATLAB	Execuția programului

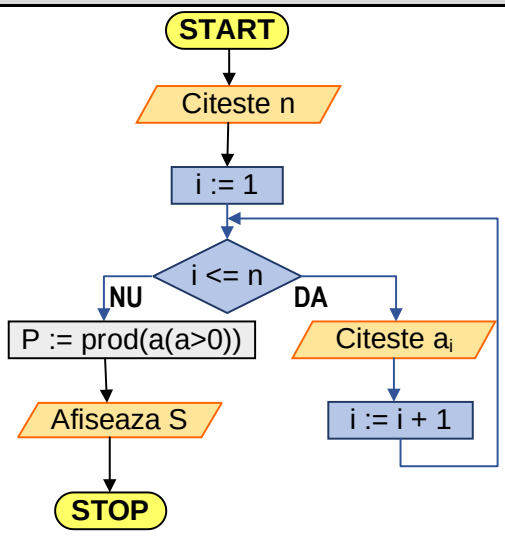
```
n=input('Introduceti n, n= ')
P=1;
for i=1:n
      s=sprintf(' a[%2d]= ',i);
      a(i)=input(s);
      if a[i] > 0
            P=P*a[i];
      end
end
mesaj=sprintf('P=%d ',P);
disp(mesaj)
```

```
Introduceti n, n = 8
a[ 0] = -5
a[ 1] =  2
a[ 2] =  0
a[ 3] =  1
a[ 4] = -3
a[ 5] =  4
a[ 6] = -2
a[ 7] =  3
P = 24
```

Figura 5.4b. Programul MATLAB şi execuţia acestuia pentru calculul produsului elementelor strict pozitive unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  Pentru i←1,n
    Citeşte a_i
  Sfârşit pentru
  P←prod(a(a>0))
  Scrie P
Sfârşit

Figura 5.4c. Reprezentarea algoritmului pentru calculul produsului elementelor pozitive ale unui tablou unidimensional apelând funcția **prod()**

Programul MATLAB	Execuția programului

```
n=input('Introduceti n, n= ');
for i=1:n
        s=sprintf(' a[%2d] = ',i);
        a(i)=input(s);
end
P=prod(a(a>0));
mesaj=sprintf('P = %d ',P);
disp(mesaj)
```

```
Introduceti n, n = 8
a[ 0] = -5
a[ 1] =  2
a[ 2] =  0
a[ 3] =  1
a[ 4] = -3
a[ 5] =  4
a[ 6] = -2
a[ 7] =  3
P = 24
```

Figura 5.4d. Folosirea unei funcții specifice limbajului MATLAB pentru produsul elementelor pozitive ale unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

La fel ca în exemplul cu determinarea sumei elementelor pozitive, se poate folosi o altă variantă de rezolvare și anume, utilizând funcția **prod()** specifică limbajului de programare MATLAB. Pentru a identifica pozițiile pe care se află elementele pozitive se folosește condiția **a>0**, iar pentru a identifica valorile elementelor pozitive se folosește condiția **a(a>0)**, unde **a** este tabloul unidimensional.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.4c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.4d.

5.5. Numărul de elemente nule ale unui tablou unidimensional

Se consideră şirul de valori din tabelul 5.5.1. Se doreşte calculul numărului de elemente nule din şir.

Tabelul 5.5.1. Şirul de valori

Element	1	2	3	4	5	6	7	8
Valoare	-5	2	0	1	0	4	-2	3

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează numărul elementelor nule (notat cu **NEN**) cu **0, NEN = 0** (zero – element neutru pentru adunare);
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** a_i;
- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este nul. Dacă condiţia este adevărată, se continuă pe ramura „ **DA** ”, se incrementează variabila **NEN**. Ramura „ **NU** ” corespunde valorilor negative sau pozitive ale elementelor tabloului, în consecință pe această ramură nu se execută nimic;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt nule, s-au numărat elementele nule) se afişează valoarea variabilei **NEN**, adică **Afişează NEN**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.5.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Variabila **NEN** primeşte iniţial valoarea **0** (zero).

La început (**i := 0**) se citeşte valoarea primului element al tabloului: a_0 **= -5**, şi se verifică dacă este nul. Primul element nefiind nul, nu se verifică condiţia, deci se trece la modificarea valorii contorului;

În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 1**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (a_1 **= 2**) şi se verifică dacă este nul. Deoarece valoarea acestuia nu este nulă, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 2.** Al treilea element este nul, astfel că se continuă parcurgerea pe ramura **DA** a instrucţiunii de decizie, se modifică valoarea variabilei **NEN**, incrementând-se, apoi creşte valoarea contorului **i** cu **1** (**i = 3**).

Tabelul 5.5.2. Calculul produsului

Valori obţinute pentru: **n = 8**				
i	a_i	a_i **nul**	**NEN**	**Ecran**
			0	
0	-5	NU		
1	2	NU		
2	0	DA	NEN := 0 + **1** = 1	
3	1	NU		
4	0	DA	NEN := 0 + **1** = 2	
5	4	NU		
6	-2	NU		
7	3	NU		**2**

Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n-1**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea variabilei **NEN**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.5a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.5b.

Schema logică	Limbajul pseudocod

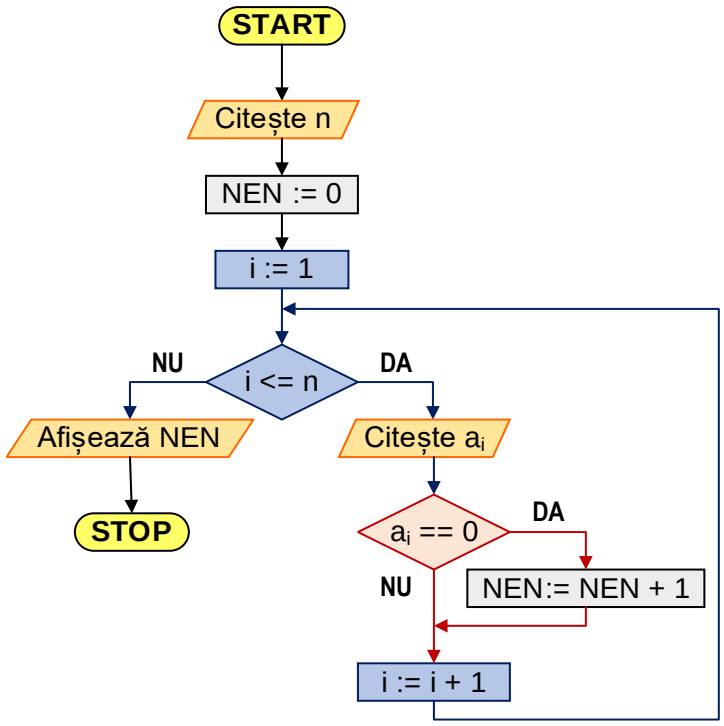
	**Început**
	  Citeşte n
	  NEN←0
	  Pentru i←1,n
	  Citeşte a_i
	  Dacă a_i=0 **atunci**
	  NEN←NEN+1
	  Sfârşit dacă
	  Sfârşit pentru
	  Scrie NEN
	Sfârşit

Figura 5.5a. Reprezentarea algoritmului pentru calculul numărului de elemente nule ale unui tablou unidimensional

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`NEN=0;`
`for i=1 : n`
`  s=sprintf(' a[%2d]= ',i);`
`  a(i)=input(s);`
`  if a(i) == 0`
`  NEN=NEN+1;`
`  end`
`end`
`mesaj=sprintf('NEN = %d ',NEN);`
`disp(mesaj)`
	`Introduceti n, n = 8`
	`a[ 1] = 2`
	`a[ 2] = 0`
	`a[ 3] = 1`
	`a[ 4] = -3`
	`a[ 5] = 0`
	`a[ 6] = -2`
	`a[ 7] = 3`
	`a[ 8] = -5`
	`NEN = 2`

Figura 5.5b. Programul MATLAB şi execuţia acestuia pentru calculul numărului de elemente nule ale unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

La fel ca în exemplele precedente, se poate folosi o altă variantă de rezolvare și anume, utilizând funcția **sum()** specifică limbajului de programare MATLAB. Pentru a identifica pozițiile pe care se află elementele nule se folosește condiția **a==0**, unde **a** este tabloul unidimensional. Folosind această instrucțiune se obține un șir de 0 și 1, 0 pentru pozițiile pe care nu sunt elemente nule, iar 1 pentru pozițiile pe care sunt elemente nule. Făcând suma șirului de valori 0 și 1 se pate determina numărul elementelor nule din șir.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.5c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.5d.

Schema logică	Limbajul pseudocod

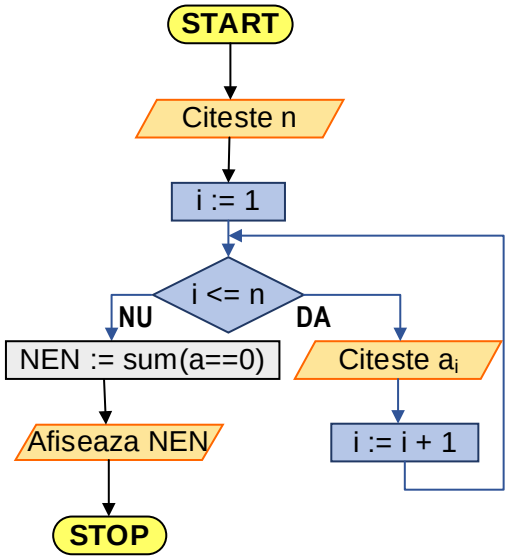
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  NEN←sum(a==0)
	  Scrie NEN
	Sfârşit

Figura 5.5c. Reprezentarea algoritmului pentru determinarea numărului elementelor nule ale unui tablou unidimensional apelând funcția **sum()**

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`end`
`NEN=sum(a==0);`
`mesaj=sprintf(' NEN = %d ',NEN);`
`disp(mesaj)`
	`Introduceti n, n = 8`
	`a[ 1] =  2`
	`a[ 2] =  0`
	`a[ 3] =  1`
	`a[ 4] = -3`
	`a[ 5] =  0`
	`a[ 6] = -2`
	`a[ 7] =  3`
	`a[ 8] = -5`
	`NEN =  2`

Figura 5.5d. Folosirea unei funcții specifice limbajului MATLAB pentru determinarea numărului elementelor nule ale unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.6. Media aritmetică a elementelor strict pozitive ale unei mulţimi

Se consideră şirul de valori din tabelul 5.6.1. Se doreşte determinarea mediei aritmetice a elementelor strict pozitive ale şirului. Pentru calculul mediei aritmetice trebuie să se calculeze suma elementelor strict pozitive precum şi numărul elementelor strict pozitive, media aritmetică fiind raportul dintre suma şi numărul elementelor strict pozitive.

Tabelul 5.6.1. Şirul de valori

Element	1	2	3	4	5	6	7	8	9	10
Valoare	15	-12	0	8	14	-9	-15	0	10	-20

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se iniţializează suma elementelor strict pozitive şi numărul de elemente strict pozitive cu **0** (zero – element neutru pentru adunare), adică **SP := 0**, respectiv **NP := 0**;
- se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente al tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „ **NU** ” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunile care reprezintă corpul ciclului cu contor:

- se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte** a_i;
- utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este strict pozitiv, impunând condiţia ca „ $a_i > 0$ ". Dacă condiţia este adevărată înseamnă că elementul curent este strict pozitiv, se continuă astfel pe ramura „**DA**”, se adaugă elementul curent la sumă şi se incrementează numărul de elemente strict pozitive. Ramura „**NU**” corespunde valorilor negative sau nule ale elementelor tabloului, astfel că pe aceasta nu se execută nimic;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat dacă sunt strict pozitive, iar cele strict pozitive s-au adăugat la sumă) se afişează calculează valoarea mediei aritmetice: **Ma := SP / NP**;
- se afişează valoarea calculată a mediei aritmetice, adică **Afişează Ma**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.6.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Suma şi numărul de elemente strict pozitve primesc iniţial valoarea **0** (zero), **SP := 0**, **NP := 0**.

La început (**i = 0**) se citeşte valoarea primului element al tabloului: a_0 **= 15**, şi se verifică dacă este strict pozitiv. Primul element fiind strict pozitiv, se adună la sumă, valoarea nouă a sumei fiind obţinută prin adunarea valorii anterioare a sumei (adică **0**) cu valoarea elementului curent (a_0 **= 15**), astfel: **SP :=** 0 + 15 = **15**. De asemenea, numărul de elemente strict pozitive **NP** se incrementează, adică **NP := NP + 1**;

În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 1**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (a_1 **= -12**) şi se verifică dacă este strict pozitiv. Deoarece valoarea acestuia nu este strict pozitivă, se continuă parcurgerea pe ramura **NU** a instrucţiunii de selecţie, în continuare mărindu-se valoarea contorului **i** cu **1** (**i = 2**).

Tabelul 5.6.2. Calculul mediei aritmetice

Valori obţinute pentru: **n = 10**					
i	a_i	$a_i > 0$	**SP**	**NP**	**Ecran**
			0	0	
0	15	DA	SP:= 0 + **15** = 15	1	
1	-12	NU			
2	0	NU			
3	8	DA	SP:= 15 + **8** = 23	2	
4	14	DA	SP:= 23 + **14** = 37	3	
5	-9	NU			
6	-15	NU			
7	0	NU			
8	10	DA	SP:= 37 + **10** = 47	4	
9	-20	NU			
			47	**4**	**11.750**

Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică calculul mediei aritmetice şi afişarea valorii acesteia.

Schema logică	Limbajul pseudocod

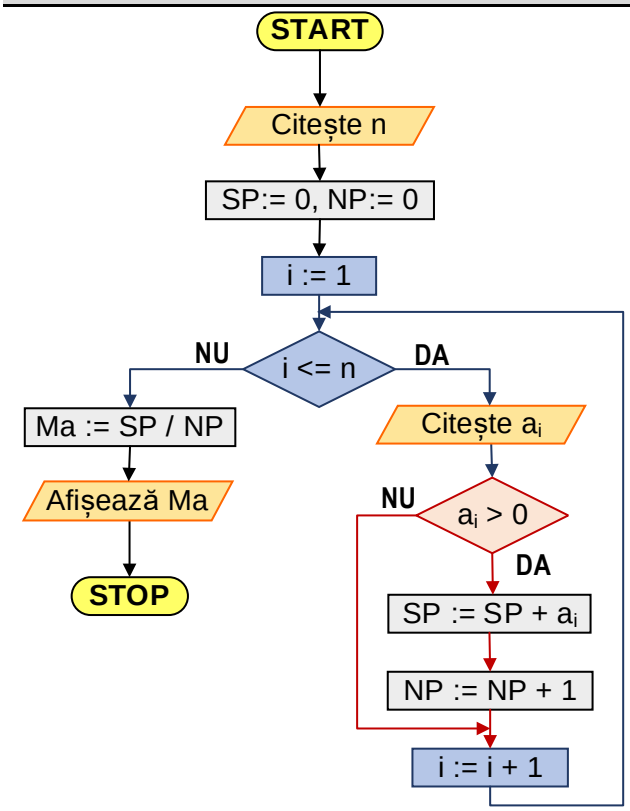
	**Început**
	  Citeşte n
	  SP←0
	  NP←0
	  Pentru i←1,n
	  Citeşte a_i
	  Dacă a_i>0 **atunci**
	  SP←SP+a_i
	  NP←NP+1
	  Sfârşit dacă
	  Sfârşit pentru
	  Ma←SP/NP
	  Scrie Ma
	Sfârşit

Figura 5.6a. Reprezentarea algoritmului pentru calculul mediei aritmetice a elementelor strict pozitive ale unui tablou unidimensional

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`SP=0;NP=0;`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`  if a(i) > 0`
`  SP = SP + a(i);`
`  NP=NP+1;`
`  end`
`end`
`Ma=SP/NP;`
`mesaj=sprintf(' MA = %f ',Ma);`
`disp(mesaj)`
	`Introduceti n, n = 10`
	`a[1] = 15`
	`a[2] = -12`
	`a[3] = 0`
	`a[4] = 8`
	`a[5] = 14`
	`a[6] = -9`
	`a[7] = -15`
	`a[8] = 0`
	`a[9] = 10`
	`a[10] = -20`
	`MA = 11.7500`

Figura 5.6b. Programul MATLAB şi execuţia acestuia pentru calculul mediei aritmetice a elementelor strict pozitive ale unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.6a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.6b.

La fel ca în exemplele precedente, se poate folosi o altă variantă de rezolvare și anume, utilizând funcția **mean()** specifică limbajului de programare MATLAB. Pentru a identifica pozițiile pe care se află elementele pozitive se folosește

condiția **a>0**, iar valorile elementelor de pe pozițiile identificate se determină cu condiția **a(a>0)**, unde **a** este tabloul unidimensional.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.6c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.6d.

Schema logică	Limbajul pseudocod

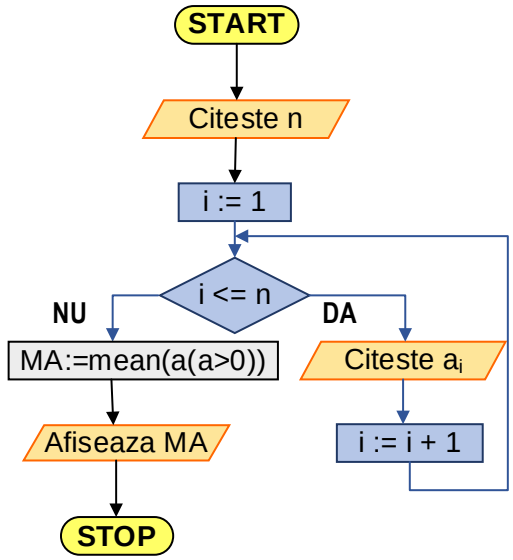
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  MA←mean(a(a>0))
	  Scrie MA
	Sfârşit

Figura 5.6c. Reprezentarea algoritmului pentru determinarea mediei aritmetice a elementelor pozitive ale unui tablou unidimensional apelând funcția mean**()**

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`end`
`MA=mean(a(a>0));`
`mesaj=sprintf(' MA = %f ',MA);`
`disp(mesaj)`
	`Introduceti n, n = 10`
	` a[1] = 15`
	` a[2] = -12`
	` a[3] = 0`
	` a[4] = 8`
	` a[5] = 14`
	` a[6] = -9`
	` a[7] = -15`
	` a[8] = 0`
	` a[9] = 10`
	` a[10] = -20`
	`MA = 11.7500`

Figura 5.6d. Folosirea unei funcții specifice limbajului MATLAB pentru determinarea mediei aritmetice a elementelor pozitive ale unui tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.7. Determinarea elementului maxim şi a poziţiei sale dintr-un tablou unidimensional

Se consideră şirul de valori din tabelul 5.7.1. Se doreşte determinarea elementului maxim şi a poziţiei sale din şir. În cazul în care există mai multe elemente maxime se determină poziția primului din șir.

Tabelul 5.7.1. Şirul de valori

Element	1	2	3	4	5	6	7	8
Valoare	-5	2	0	1	-3	4	-2	3

Algoritmul este următorul:

- se citeşte **n** - numărul de elemente ale tabloului unidimensional;
- se utilizează un ciclu cu contor pentru citirea valorilor elementelor tabloului unidimensional, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente a tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „**DA**”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă** se continuă pe ramura „**NU**” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. se execută instrucţiunea care reprezintă corpul ciclului cu contor, în acest caz se execută operaţia de citire a valorii elementului curent al tabloului, adică **Citeşte a_i** ;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i = i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- se iniţializează variabila **max** cu valoarea primului element din şir (**max = a_0**), iar variabila **pmax** se iniţializează cu **0**;
- pentru determinarea elementului maxim şi a poziţiei acestuia se utilizează un ciclu cu contor, în cadrul căruia se parcurg următorii paşi:

1. se iniţializează contorul „ **i** ” cu valoarea **0**;

2. se evaluează condiţia „ **i < n** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de elemente al tabloului. Dacă condiţia este **adevărată** se continuă pe ramura „**DA**”, se execută pasul următor (pasul 3). Dacă condiţia este **falsă**, se continuă pe ramura „**NU**” se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după ciclul cu contor;

3. utilizând o instrucţiune de decizie, se verifică dacă elementul curent al tabloului este mai mare decât **max**, impunând condiţia ca „ **a_i > max** ". Dacă condiţia este adevărată înseamnă că elementul curent este mai mare decât **max**, se continuă pe ramura „**DA**”, se atribuie variabilei **max** valoarea **a_i** a elementului curent, iar variabilei **pmax** i se atribuie valoarea indicelui **i** al elementului curent. Dacă elementul curent nu este mai mare decât variabila **max** (adică condiţia este falsă) se trece la pasul **4**;

4. se execută instrucţiunea prin care se modifică valoarea contorului cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **2**;

- după ieşirea din ciclu cu test iniţial (s-au citit cele **n** elemente ale tabloului, s-au verificat elementele şirului în raport cu variabila max) se afişează valorile variabilelor **max** şi **pmax**, adică **Afişează max, pmax**;

Exemplu numeric: Se utilizează valorile din şirul prezentat anterior. În tabelul 5.7.2 sunt ilustrate valorile obţinute la rularea programului, în fiecare etapă:

Variabila **max** primeşte iniţial valoarea **a_0**, respectiv variabila **pmax** primeşte valoarea **0,** adică **max = -5**, **pmax = 0**.

În prima etapă (**i = 1**), se citeşte valoarea celui de al doilea element al tabloului: **a_1 = 2**, şi se verifică dacă este mai mare decât **max**. Deoarece condiţia este adevărată, variabila **max** primeşte valoarea elementului curent al tabloului, adică **max = 2**, iar **pmax** primeşte valoarea indicelui elementului curent, adică **pmax = 1**.

În continuare, contorul **i** îşi creşte valoarea cu **1**, astfel că **i = 2**. Se citeşte valoarea elementului al doilea al tabloului unidimensional (**a_2 = 0**) şi se verifică dacă este mai mare decât

Tabelul 5.7.2. Determinarea elementului maxim şi a poziţiei acestuia în şir

Valori obţinute pentru: **n = 8**					
i	**a_i**	**a_i > 0**	**max**	**pmax**	**Ecran**
			-5	**0**	
1	2	**DA**	2	1	
2	0	NU			
3	1	NU			
4	-3	NU			
5	4	**DA**	4	5	
6	-2	NU			
7	3	NU			
					4, 5

max. Deoarece valoarea acestuia este mai mică, se continuă parcurgerea pe ramura „**NU**” a instrucţiunii de decizie, astfel că se continuă cu modificarea valorii contorului **i** cu **1** (**i = 3**).

Procedeul se repetă până când valoarea lui **i** devine mai mare decât **n-1**, adică nu se mai respectă condiţia **i < n**, astfel că se părăseşte ciclul cu contor şi se execută blocul următor, adică afişarea valorilor variabilelor **max**, respectiv **pmax**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.7a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.7b.

Schema logică	Limbajul pseudocod

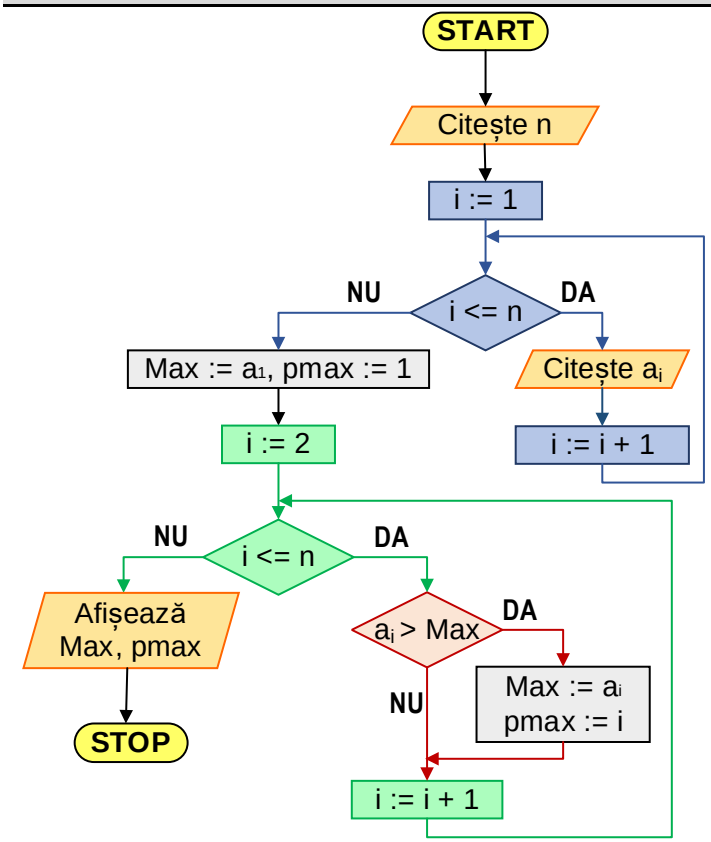
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  Max←a_1
	  pmax←1
	  Pentru i←2,n
	  Dacă a_i>Max **atunci**
	  Max← a_i
	  pmax←i
	  Sfârşit dacă
	  Sfârşit pentru
	  Scrie Max, pmax
	Sfârşit

Figura 5.7a. Reprezentarea algoritmului pentru determinarea elementului maxim şi a poziţiei sale într-un tablou unidimensional

La fel ca în exemplele precedente, se poate folosi o altă variantă de rezolvare și anume, utilizând funcția **max()** specifică limbajului de programare MATLAB. Această funcție are următoarele două forme mai des folosite:

```
M = max(a)
[M, poz] = max(a)
```

unde **a** este tabloul unidimensional. Prima formă returnează în **M** valoarea elementului maxim din **a**, iar a doua formă returnează pe lângă valoarea maximului și prima poziție pe care acesta apare în tablou și o salvează în variabila **poz**.

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`end`
	`Introduceti n, n = 8`
	`a[ 1] =  2`
	`a[ 2] =  0`
	`a[ 3] =  1`
	`a[ 4] = -3`

```
Max=a(1); pmax=1;
for i=2:n
    if a(i) > Max
        Max = a(i);
        pmax=i;
    end
end
mesaj=sprintf('Max: a[%d]=%d',pmax,Max);
disp(mesaj)
```

```
a[ 5] =  4
a[ 6] = -2
a[ 7] =  3
a[ 8] = -5
Max: a[5]=4
```

Figura 5.7b. Programul MATLAB şi execuţia acestuia pentru determinarea elementului maxim şi a poziţiei sale într-un tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Observații:

- *Dacă **a** este o matrice, atunci **max (a)** este un vector linie care conține valoarea maximă a fiecărei coloane.*
- *Dacă **a** este un tablou multidimensional, atunci **max (a)** operează de-a lungul primei dimensiuni diferite de 1, tratând elementele ca tablouri. Practic dimensiunea devine 1, în timp ce dimensiunile celelalte rămân aceleași.*
- *Dacă **a** este un tablou vid, atunci **max (a)** returnează tot un tablou vid.*

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.7c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.7d.

Schema logică	Limbajul pseudocod

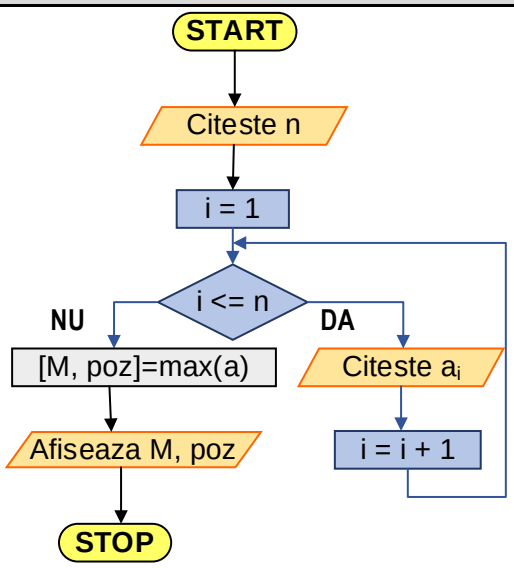
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  [M,poz]←max(a)
	  Scrie M, poz
	Sfârşit

Figura 5.7c. Reprezentarea algoritmului pentru determinarea elementului maxim şi a poziţiei sale într-un tablou unidimensional apelând funcția **max()**

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`end`
`[M, poz]=max(a);`
`mesaj=sprintf('Max: a[%d]=  %d',poz,M);`
	`Introduceti n, n = 8`
	`a[ 1] =  2`
	`a[ 2] =  0`
	`a[ 3] =  1`
	`a[ 4] = -3`
	`a[ 5] =  4`
	`a[ 6] = -2`
	`a[ 7] =  3`

`disp(mesaj)`	`a[ 8] = -5` `Max: a[5] = 4`

Figura 5.7d. Folosirea unei funcții specifice limbajului MATLAB pentru determinarea elementului maxim şi a poziţiei sale într-un tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.8. Ordonarea crescătoare / descrescătoare a elementelor unei mulţimi

Operaţia de ordonare a unor articole, în funcţie de diverse criterii, este foarte des întâlnită în practică, fiind dezvoltaţi o mulţime de algoritmi de sortare. În informatică, operaţia de sortare este o operaţie fundamentală, care constă în rearanjarea elementelor unui șir în ordine crescătoare sau descrescătoare. Operaţia de sortare poate fi realizată prin ocuparea aceleiaşi zone de memorie sau prin ocuparea unei alte zone de memorie. În continuare, sunt prezentate metode de sortare cu ocuparea aceleiaşi zone de memorie.

Problema sortării poate fi formulată în cazul general astfel: se consideră un șir de **n** valori a_0, a_1, ... , a_{n-1} care trebuie aranjat astfel pentru ordonare crescătoare,:

$$a_0 \le a_1 \le a_2 \le ... \le a_{n-1}$$, respectiv: $a_0 \ge a_1 \ge a_2 \ge ... \ge a_{n-1}$

pentru ordonare descrescătoare.

5.8.1. Sortarea prin selecţie directă

Algoritmul pentru ordonarea crescătoare este următorul:

- se compară primul element din mulţime cu toate elementele care urmează după el şi dacă se găseşte un element mai mic decât primul atunci se schimbă între ele cele două elemente;
- se compară al doilea element al mulţimii cu toate elementele care urmează după el şi dacă se găseşte un element mai mic decât acesta se schimbă între ele cele două elemente;
- se procedează în mod asemănător cu al treilea, al patrulea, etc. element al mulţimii, iar procesul continuă astfel până la penultimul element al mulţimii care va fi comparat cu ultimul.

Exemplul numeric:

Pentru exemplificare, se consideră o mulţime de 10 elemente şi se doreşte ordonarea acestora după valoare, în ordine crescătoare:

Indice	1	2	3	4	5	6	7	8	9	10
Valoare	15	-12	5	8	14	-9	-15	0	10	-20

Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.1:

Etapa 1: se compară primul element cu elementele care urmează după el şi dacă se găseşte un element mai mic decât primul atunci se schimbă între ele cele două elemente, astfel:

- Pasul 1a: se compară primul element (cu valoarea **15**) cu al doilea (cu valoarea **-12**) şi se constată că al doilea element este mai mic decât primul, deci **se schimbă între ele**;
- Pasul 1b: se compară primul element (cu valoarea **-12**) cu al treilea (cu valoarea **5**) şi se constată că primul element este mai mic decât al treilea, situaţie în care **se trece la pasul următor**;
- Pasul 1c: se compară primul element (cu valoarea **-12**) cu al patrulea (cu valoarea **8**) şi se constată că primul element este mai mic decât al patrulea, situaţie în care **se trece la pasul următor**;
- Pasul 1d: se compară primul element (cu valoarea **-12**) cu al cincilea (cu valoarea **14**) şi se constată că primul element este mai mic decât al cincilea, situaţie în care **se trece la pasul următor**;

- Pasul 1e: se compară primul element (cu valoarea **-12**) cu al şaselea (cu valoarea **-9**) şi se constată că primul element este mai mic decât al şaselea, situaţie în care **se trece la pasul următor**;
- Pasul 1f: se compară primul element (cu valoarea **-12**) cu al şaptelea (cu valoarea **-15**) şi se constată că primul element este mai mare decât al şaptelea, situaţie în care **se schimbă între ele**;
- Pasul 1g: se compară primul element (cu valoarea **-15**) cu al optulea (cu valoarea **0**) şi se constată că primul element este mai mic decât al optulea, situaţie în care **se trece la pasul următor**;
- Pasul 1h: se compară primul element (cu valoarea **-15**) cu al nouălea (cu valoarea **10**) şi se constată că primul element este mai mic decât al nouălea, situaţie în care **se trece la pasul următor**;
- Pasul 1i: se compară primul element (cu valoarea **-15**) cu al zecelea (cu valoarea **-20**) şi se constată că primul element este mai mare decât al zecelea, situaţie în care **se schimbă între ele**;

Etapa 2 - 9: se compară al doilea, al treilea, etc element cu elementele care urmează după el şi dacă se găseşte un element mai mic decât al doilea atunci se schimbă între ele cele două elemente.

Tabelul 5.8.1. Sortarea prin selecţie directă

Indice	1	2	3	4	5	6	7	8	9	10
Valoare	15	-12	5	8	14	-9	-15	0	10	-20
Etapa 1:	15	-12	5	8	14	-9	-15	0	10	-20
Pasul 1a:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1b:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1c:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1d:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1e:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1f:	-12	15	5	8	14	-9	-15	0	10	-20
Pasul 1g	-15	15	5	8	14	-9	-12	0	10	-20
Pasul 1h	-15	15	5	8	14	-9	-12	0	10	-20
Pasul 1i	-15	15	5	8	14	-9	-12	0	10	-20
Etapa 2:	-20	-15	5	8	14	-9	-12	0	10	15
Etapa 3:	-20	-15	-12	15	14	8	5	0	10	-9
...										
Final:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.8b.

Schema logică	Limbajul pseudocod

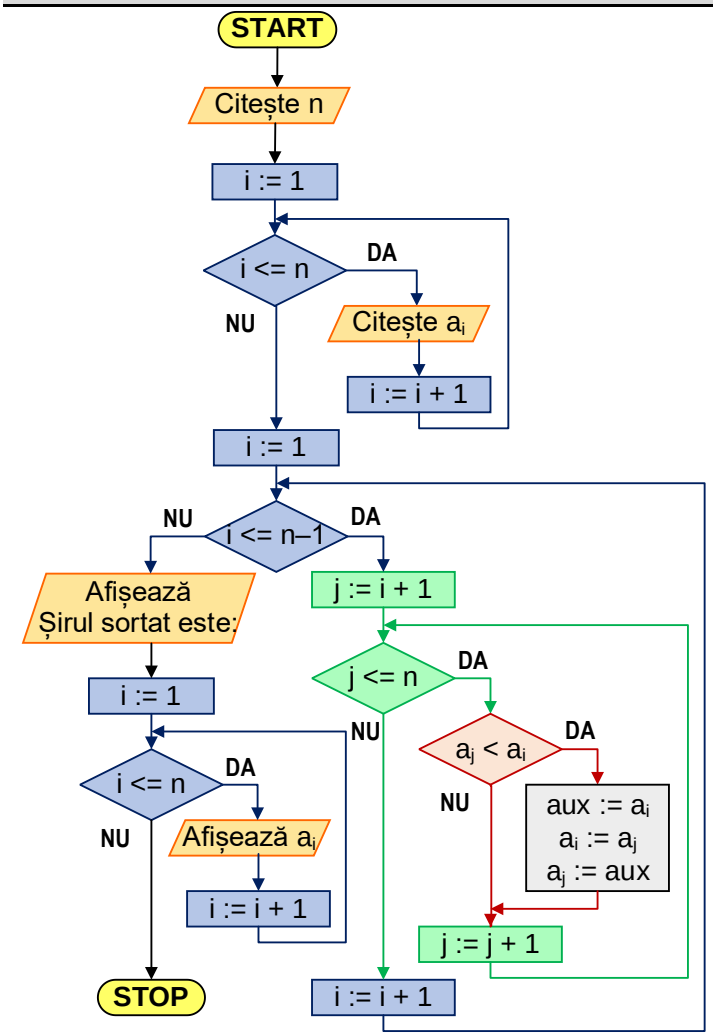
	**Început**
	  Citeşte n
	  Scrie „Introduceti elementele:”
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  Pentru i←1,n-1
	  Pentru j←i+1,n
	  Dacă $a_j < a_i$ **atunci**
	  aux← a_i
	  a_i ← a_j
	  a_j ←aux
	  Sfârşit dacă
	  Sfârşit pentru
	  Sfârşit pentru
	  Scrie „Sirul sortat este:”
	  Pentru i←1,n
	  Scrie a_i
	  Sfârşit pentru
	Sfârşit

Figura 5.8a. Reprezentarea algoritmului pentru sortarea (crescătoare) prin selecţie directă a elementelor unei mulţimi

Programul MATLAB	Execuția programului

```
n=input('Introduceti n, n= ');
for i=1:n
   s=sprintf(' a[%2d] = ',i);
   a(i)=input(s);
end
for i=1:n-1
   for j=i+1:n
       if a(j) < a(i)
           aux = a(i);
           a(i)= a(j);
           a(j)= aux;
       end
   end
end
disp('Sirul sortat este: ');
for i=1:n
```

```
Introduceti n, n= 5
 a[ 1] = 11
 a[ 2] = -2
 a[ 3] = 5
 a[ 4] = -1
 a[ 5] = 0
Sirul sortat este:
  -2
  -1
   0
   5
  11
```

```
    mesaj=sprintf('%4d',a(i));
disp(mesaj)
end
```

Figura 5.8b. Programul MATLAB și execuția acestuia pentru sortarea (crescătoare) prin selecţie directă a elementelor unei mulţimi

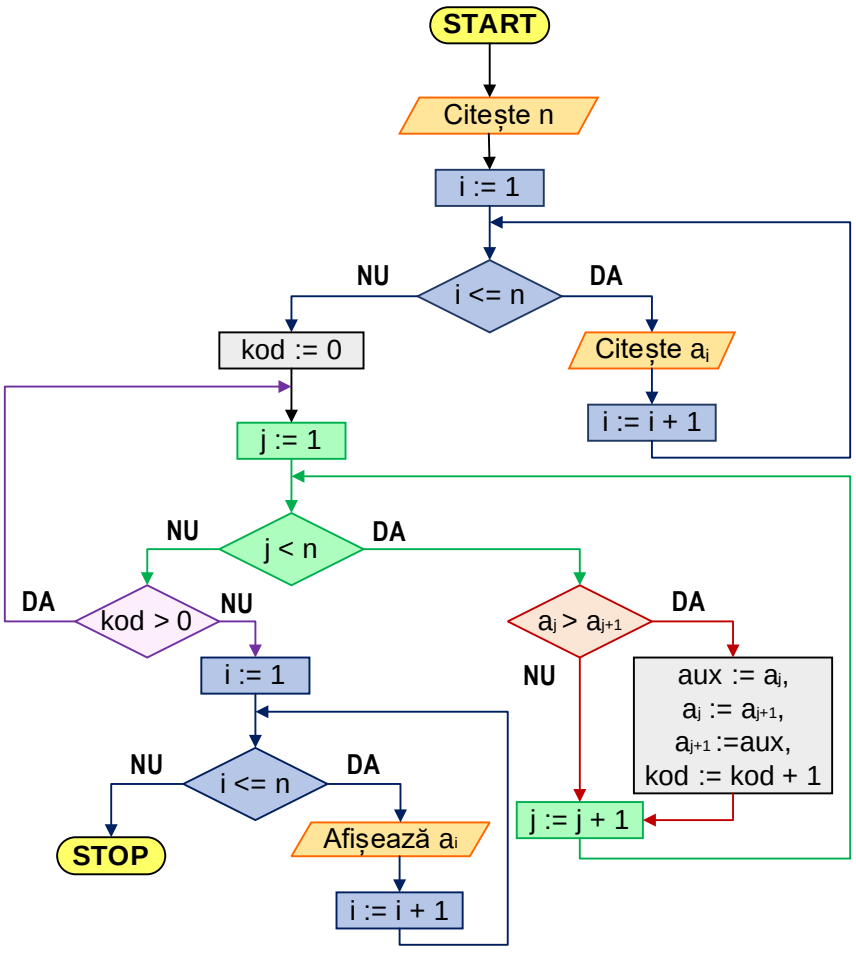
Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.8.2. Sortarea prin interschimbare – Bubble Sort (metoda bulelor)

În continuare este prezentat algoritmul pentru ordonarea descrescătoare prin metoda bulelor. Elementele unui șir sunt ordonate descrescător dacă între oricare două elemente alăturate ale șirului există relația:

$$a_i \geq a_{i+1}$$

Schema logică



Limbajul pseudocod

```
Început
  Citeşte n
  Pentru i←1,n
      Citeşte ai
  Sfârşit pentru
  kod← 1
  Cât timp kod>0
execută
      kod← 0
      Pentru j←1,n-1
        Dacă aj < aj+1atunci
           aux← aj
           aj ← aj+1
           aj+1 ←aux
           kod← kod+1
        Sfârşit dacă
      Sfârşit pentru
  Sfârşit cât timp
  Pentru i←1,n
      Scrie ai
  Sfârşit pentru
Sfârşit
```

Figura 5.8c. Reprezentarea algoritmului pentru sortarea (descrescătoare) prin interschimbare (metoda bulelor)

În cadrul metodei bulelor, se parcurge șir, de la primul element până la penultimul şi se verifică dacă două elemente alăturate sunt în relaţia de mai sus. Condiţia care se pune în acest caz este:

$$a_i < a_{i+1}$$

Dacă condiţia este adevărată, înseamnă că elementele nu sunt ordonate corespunzător, caz în care se realizează o interschimbare între ele a celor două elemente. Se utilizează o variabilă cu ajutorul căreia se marchează faptul că s-a realizat interschimbarea elementelor alăturate. La început, înaintea începerii parcurgerii mulţimii, această variabila primeşte valoarea 0.

Dacă condiţia este falsă, înseamnă că cele două elemente sunt ordonate corespunzător, situaţie în care se trece la verificarea următoarei perechi de elemente alăturate.

La final, după parcurgerea întregului șir, se verifică valoarea variabilei cu ajutorul căreia s-a realizat marcarea interschimbărilor. Dacă valoarea acesteia este diferită de zero, adică în timpul parcurgerii s-au realizat interschimbări, se reia parcurgerea șirului. Dacă valoarea acestei variabile este zero înseamnă că în timpul parcurgerii nu s-a realizat nicio interschimbare, deci şirul este ordonat.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.8d.

Programul MATLAB	Execuția programului

```
n=input('Introduceti n, n= ');
for i=1:n
    s=sprintf(' a[%2d] = ',i);
    a(i)=input(s);
end
kod=1;
while kod > 0
    kod=0;
    for j=1:n-1
        if a(j) < a(j+1)
            aux = a(j);
            a(j)=a(j+1);
            a(j+1)=aux;
            kod=kod+1;
        end
    end
end
disp('Sirul sortat este: ');
for i=1:n
    mesaj=sprintf('%4d',a(i));
    disp(mesaj)
end
```

```
Introduceti n, n= 5
 a[ 1] = 21
 a[ 2] = -5
 a[ 3] = 6
 a[ 4] = -1
 a[ 5] = 0
Sirul sortat este:
  21
  6
  0
  -1
  -5
```

Figura 5.8d. Programul MATLAB și execuția acestuia pentru sortarea (descrescătoare) prin interschimbare (metoda bulelor)

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Exemplul numeric: Modul de funcționare al algoritmului este prezentat în tabelul 5.8.2.

Etapa 1:

Pasul 1.1: se verifică elementele a_0 şi a_1, deoarece $a_0 > a_1$ se trece la următoarea pereche de elemente, valoarea variabilei kod nu se schimbă, adică kod = 0;

Pasul 1.2: se verifică elementele a_1 şi a_2, deoarece $a_1 < a_2$ se interschimbă între ele cele două elemente şi se creşte variabila kod cu 1, adică kod = 0 + 1 = 1;

Pasul 1.3: se verifică elementele a_2 şi a_3, deoarece $a_2 < a_3$ se interschimbă între ele cele două elemente şi se creşte variabila kod cu 1, adică kod = 1 + 1 = 2;

Pasul 1.4: se verifică elementele a_3 şi a_4, deoarece $a_3 < a_4$ se interschimbă între ele cele două elemente şi se creşte variabila kod cu 1, adică kod = 2 + 1 = 3;

Pasul 1.5: se verifică elementele a_4 şi a_5, deoarece $a_4 < a_5$ se interschimbă între ele cele două elemente şi se creşte variabila kod cu 1, adică kod = 3 + 1 = 4;

Pasul 1.6: se verifică elementele a_5 şi a_6, deoarece $a_5 > a_6$ se trece la următoarea pereche de elemente, valoarea variabilei kod nu se schimbă, adică kod = 4;

Pasul 1.7: se verifică elementele a_6 şi a_7, deoarece $a_6 < a_7$ se interschimbă între ele cele două elemente şi se creşte variabila kod cu 1, adică kod = 4 + 1 = 5;

Pasul 1.8: se verifică elementele a_7 şi a_8, deoarece $a_7 < a_8$ se interschimbă între ele cele două elemente şi se creşte variabila kod cu 1, adică kod = 5 + 1 = 6;

Pasul 1.9: se verifică elementele a_8 şi a_9, deoarece $a_8 > a_9$ se trece la următoarea pereche de elemente, valoarea variabilei kod nu se schimbă, adică kod = 6;

Deoarece după parcurgerea şirului de valori, variabila **kod** are o valoare strict pozitivă (**kod > 0**) se reia parcurgerea şirului de valori, variabila **kod** primind din nou valoarea **0**. Procedeul se repetă în mod asemănător până când după o parcurgere a şirului variabila kod nu-şi mai modifică valoarea, adică **kod = 0**. În acest moment, şirul este ordonat descrescător.

Tabelul 5.8.2. Sortarea prin interschimbare (metoda bulelor)

Indice	1	2	3	4	5	6	7	8	9	10	
Valoare	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**	**kod**
Etapa 1:	15	-12	5	8	14	-9	-15	0	10	-20	0
Pasul 1.1:	15	-12	5	8	14	-9	-15	0	10	-20	0
Pasul 1.2:	15	5	-12	8	14	-9	-15	0	10	-20	1
Pasul 1.3:	15	5	8	-12	14	-9	-15	0	10	-20	2
Pasul 1.4:	15	5	8	14	-12	-9	-15	0	10	-20	3
Pasul 1.5:	15	5	8	14	-9	-12	-15	0	10	-20	4
Pasul 1.6:	15	5	8	14	-9	-12	-15	0	10	-20	4
Pasul 1.7:	15	5	8	14	-9	-12	0	-15	10	-20	5
Pasul 1.8:	15	5	8	14	-9	-12	0	10	-15	-20	6
Pasul 1.9:	15	5	8	14	-9	-12	0	10	-15	-20	**6**
Etapa 2:	15	8	14	5	-9	0	10	-12	-15	-20	
Etapa 3:	15	14	8	5	0	10	-9	-12	-15	-20	
Etapa 4:	15	14	8	5	10	0	-9	-12	-15	-20	
Etapa 5:	15	14	8	10	5	0	-9	-12	-15	-20	
Etapa 6:	15	14	10	8	5	0	-9	-12	-15	-20	
Final:	**15**	**14**	**10**	**8**	**5**	**0**	**-9**	**-12**	**-15**	**-20**	

5.8.3. Sortarea prin metoda selecţiei naive (naive sort)

În acest caz, algoritmul de ordonare crescătoare poate fi descris astfel:

- în prima etapă, se caută în șirul neordonat cu n elemente, valoarea maximă şi poziţia acestei valori şi se realizează interschimbarea între valoarea maximă şi ultima valoare din șir (de indice n-1);
- în a doua etapă, se caută în șirul cu n-1 elemente (se omite ultimul element) valoarea maximă şi poziţia acesteia şi se realizează interschimbarea între valoarea maximă găsită şi valoarea de indice n-2;
- se repetă operaţiile până când se obţine un șir cu un singur element;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8e, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.8f.

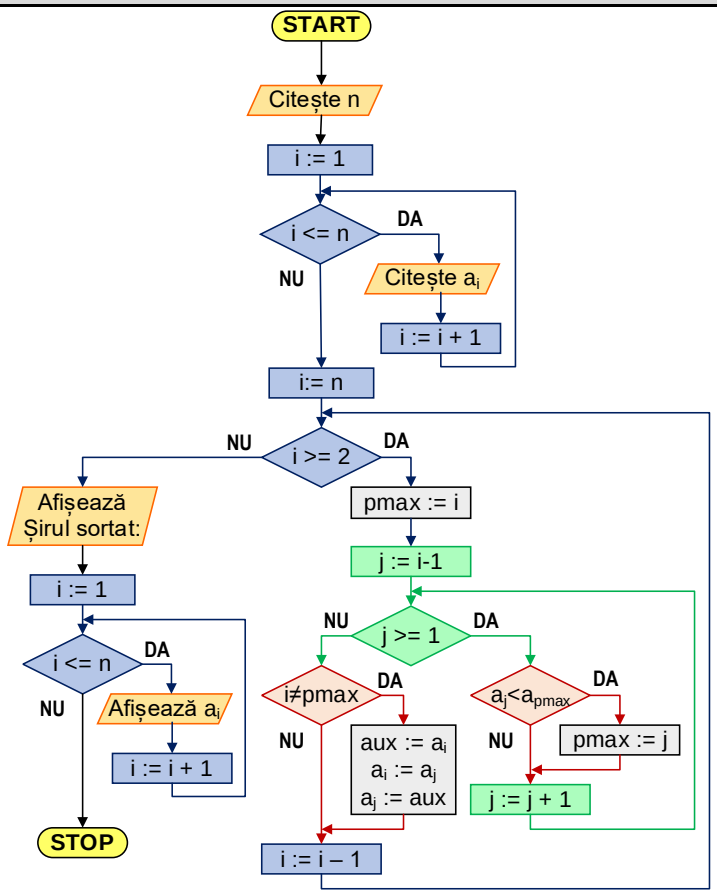
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n **Pentru** i←1,n **Citeşte** a_i **Sfârşit pentru** **Pentru** i←n,2,-1 pmax← i **Pentru** j←i-1,1,-1 **Dacă** $a_j < a_{pmax}$ **atunci** pmax← j **Sfârşit dacă** **Sfârşit pentru** **Dacă** i ≠ pmax **atunci** aux← a_i a_i← a_{pmax} a_{pmax} ← aux **Sfârşit dacă** **Sfârşit pentru** **Scrie** „Sirul sortat:” **Pentru** i←1,n **Scrie** a_i **Sfârşit pentru** **Sfârşit**

Figura 5.8e. Reprezentarea algoritmului pentru sortarea (crescătoare) prin metoda selecţiei naive (naive sort)

Programul MATLAB	Execuția programului

```
n=input('Introduceti n, n= ');
for i=1:n
  s=sprintf(' a[%2d] = ',i);
  a(i)=input(s);
end
for i=n:-1:2
  pmax=i;
   for j=i-1:-1:1
       if a(j) > a(pmax)
           pmax=j;
       end
   end
   if i~=pmax
           aux=a(i);
           a(i)=a(pmax);
           a(pmax)=aux;
       end
end
disp('Sirul sortat este: ');
for i=1:n
```

```
Introduceti n, n= 7
 a[ 1] = 21
 a[ 2] = -5
 a[ 3] = 6
 a[ 4] = -1
 a[ 5] = 0
 a[ 6] = 34
 a[ 7] = 100

Sirul sortat este:
  -5
  -1
   0
   6
  21
  34
100
```

```
    mesaj=sprintf('%4d',a(i));
disp(mesaj)
end
```

Figura 5.8f. Programul MATLAB și execuția acestuia pentru sortarea (crescătoare) prin metoda selecţiei naive (naive sort)

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Exemplul numeric: Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.3:

Tabelul 5.8.3. Sortarea prin metoda selecţiei naive

Indice	1	2	3	4	5	6	7	8	9	10
Valoare	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
Etapa 1a:	15	-12	5	8	14	-9	-15	0	10	-20
Etapa 1b:	-20	-12	5	8	14	-9	-15	0	10	**15**
Etapa 2a:	-20	-12	5	8	14	-9	-15	0	10	**15**
Etapa 2b:	-20	-12	5	8	10	-9	-15	0	**14**	**15**
Etapa 3a:	-20	-12	5	8	10	-9	-15	0	**14**	**15**
Etapa 3b:	-20	-12	5	8	0	-9	-15	**10**	**14**	**15**
Etapa 4a:	-20	-12	5	8	0	-9	-15	**10**	**14**	**15**
Etapa 4b:	-20	-12	5	-15	0	-9	**8**	**10**	**14**	**15**
Etapa 5a:	-20	-12	5	-15	0	-9	**8**	**10**	**14**	**15**
Etapa 5b:	-20	-12	-9	-15	0	**5**	**8**	**10**	**14**	**15**
Etapa 6a:	-20	-12	-9	-15	0	**5**	**8**	**10**	**14**	**15**
Etapa 6b:	-20	-12	-9	-15	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 7a:	-20	-12	-9	-15	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 7b:	-20	-12	-15	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 8a:	-20	-12	-15	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 8b:	-20	-15	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 9a:	-20	-15	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Etapa 9b:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**
Final:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**

Etapa 1a: se parcurge șirul neordonat (cu n elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 0** (valoarea **15**);
Etapa 1b: se schimbă între ele elementul de pe ultima poziţie (poziţia cu indicele **n-1**) şi elementul de pe poziţia **pmax**, adică primul (valoarea **15**) şi ultimul element (valoarea **-20**);
Etapa 2a: se parcurge mulţimea neordonată (cu n-1 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 4** (valoarea **14**);
Etapa 2b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n-2**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **14** şi **10**;
Etapa 3a: se parcurge șirul neordonat (cu n-2 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 4** (valoarea **10**);
Etapa 3b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n-3**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **10** şi **0**;
Etapa 4a: se parcurge șirul neordonat (cu n-3 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 3** (valoarea **8**);
Etapa 4b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n-4**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **8** şi **-15**;

Etapa 5a: se parcurge șirul neordonat (cu n-4 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 2** (valoarea **5**);
Etapa 5b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n-5**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **5** şi **-9**;
Etapa 6a: se parcurge șirul neordonat (cu n-5 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 4** (valoarea **0**);
Etapa 6b: deoarece valoarea maximă a elementului din subşir este chiar pe ultima poziţie, nu se schimbă nimic;
Etapa 7a: se parcurge șirul neordonat (cu n-6 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 2** (valoarea **-9**);
Etapa 7b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **n-7**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **-9** şi **-15**;
Etapa 8a: se parcurge șirul neordonat (cu 3 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 1** (valoarea **-12**);
Etapa 8b: se schimbă între ele elementul de pe ultima poziţie a subşirului (poziţia cu indicele **2**) şi elementul de pe poziţia **pmax**, adică elementele cu valorile **-12** şi **-15**;
Etapa 9a: se parcurge șirul neordonată (cu 2 elemente) şi se determină indicele elementului maxim, în acest caz **pmax = 1** (valoarea **-15**);
Etapa 9b: deoarece valoarea maximă a elementului din subşir este chiar pe ultima poziţie, nu se schimbă nimic;
După parcurgerea acestor etape, șirul este ordonat crescător.

5.8.4. Sortarea prin metoda inserţiei (Insertion Sort)

Pentru ordonarea crescătoare cu ajutorul acestei metode se consideră că primele **i** elemente ale șirului sunt ordonate, iar elementul de pe poziţia **i** va fi inserat în subşirul **[0, i - 1]**, astfel încât subşirul **[0, i]** să fie ordonat.

Inserarea se realizează astfel:

- se memorează într-o variabilă ajutătoare valoarea elementului de pe poziţia **i**;
- de la poziţia **i - 1** până la poziţia **0**, se deplasează cu o poziţie la dreapta toate elementele mai mari decât valoarea memorată în variabila ajutătoare;
- se inserează valoarea variabilei ajutătoare în locul ultimului element deplasat în etapa anterioară;

Reprezentarea algoritmului prin schemă logică şi limbaj pseudocod este prezentată în figura 5.8g, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.8h.

Schema logică	Limbajul pseudocod

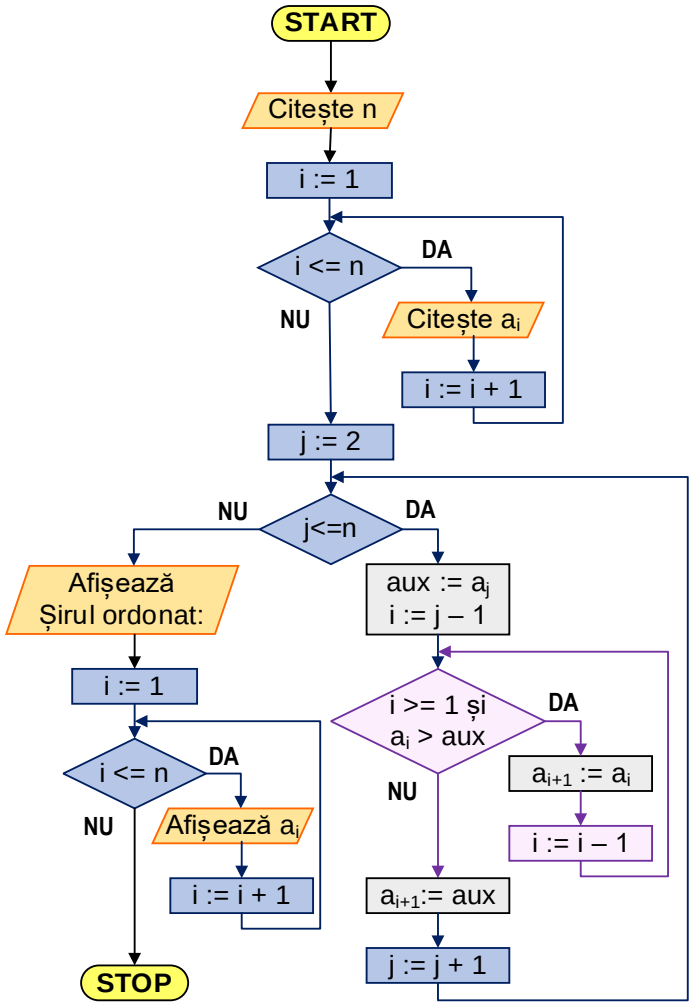
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  Pentru j←2,n
	  aux← a_j
	  i← j-1
	  Cât timp i≥1 ŞI a_i >aux **execută**
	  a_{i+1}← a_i
	  i ← i-1
	  a_{i+1} ←aux
	  Sfârşit cât timp
	  Sfârşit pentru
	  Scrie „Sirul ordonat:”
	  Pentru i←1,n
	  Scrie a_i
	  Sfârşit pentru
	Sfârşit

Figura 5.8g. Reprezentarea algoritmului pentru sortarea (crescătoare) prin metoda inserţiei (Insertion Sort)

Programul MATLAB	Execuția programului

```
n=input('Introduceti n, n= ');
for i=1:n
    s=sprintf(' a[%2d] = ',i);
    a(i)=input(s);
end
for j=2:n
    aux=a(j);    i=j-1;
    while i>=1 & a(i) >aux
      a(i+1)=a(i); i=i-1; a(i+1)=aux;
    end
end
disp('Sirul sortat este: ');
for i=1:n
    mesaj=sprintf('%4d',a(i));
disp(mesaj)
end
```

```
Introduceti n, n= 6
 a[ 1] = 11
 a[ 2] = -9
 a[ 3] = 8
 a[ 4] = -3
 a[ 5] = 0
 a[ 6] = 34
Sirul sortat este:
  -9
  -3
   0
   8
  11
  34
```

Figura 5.8h. Programul MATLAB și execuția acestuia pentru sortarea (crescătoare) prin metoda inserţiei (Insertion Sort)

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Exemplul numeric: Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.4:

Tabelul 5.8.4. Sortarea prin metoda inserţiei (Insertion Sort)

Indice	1	2	3	4	5	6	7	8	9	10	
Valoare	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**	
Etapa 1:	15	**-12**	5	8	14	-9	-15	0	10	-20	**-12**
	-12	15	5	8	14	-9	-15	0	10	-20	
Etapa 2:	-12	15	**5**	8	14	-9	-15	0	10	-20	**5**
	-12	5	15	8	14	-9	-15	0	10	-20	
Etapa 3:	-12	5	15	**8**	14	-9	-15	0	10	-20	**8**
	-12	5	8	15	14	-9	-15	0	10	-20	
Etapa 4:	-12	5	8	15	**14**	-9	-15	0	10	-20	**14**
	-12	5	8	14	15	-9	-15	0	10	-20	
Etapa 5:	-12	5	8	14	15	**-9**	-15	0	10	-20	**-9**
	-12	-9	5	8	14	15	-15	0	10	-20	
Etapa 6:	-12	-9	5	8	14	15	**-15**	0	10	-20	**-15**
	-15	-12	-9	5	8	14	15	0	10	-20	
Etapa 7:	-15	-12	-9	5	8	14	15	**0**	10	-20	**0**
	-15	-12	-9	0	5	8	14	15	10	-20	
Etapa 8:	-15	-12	-9	0	5	8	14	15	**10**	-20	**10**
	-15	-12	-9	0	5	8	10	14	15	-20	
Etapa 9:	-15	-12	-9	0	5	8	10	14	15	**-20**	**-20**
	-20	-15	-12	-9	0	5	8	10	14	15	
Final:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**	

Etapa 1: Variabila **j** are valoarea **1**, deci variabila **aux** primeşte valoarea **-12** (**aux = -12**). Se verifică în subşirul format din primul element dacă elementele acestuia sunt mai mari decât **-12**. Primul element al şirului având valoarea **15** este mai mare decât **-12**, deci se deplasează la dreapta cu o unitate, iar pe prima poziţie se aşează valoarea **-12**;
Etapa 2: Variabila **j** are valoarea **2**, deci variabila **aux** primeşte valoarea **5** (**aux = 5**). Se verifică în subşirul format din primele două elemente dacă elementele acestuia sunt mai mari decât **5**. Al doilea element al şirului având valoarea **15** este mai mare decât **5**, deci se deplasează la dreapta cu o unitate, iar pe poziţia acestuia se aşează valoarea **5**;
Etapa 3: Variabila **j** are valoarea **3**, deci variabila **aux** primeşte valoarea **8** (**aux = 8**). Se verifică în subşirul format din primele trei elemente dacă elementele acestuia sunt mai mari decât **8**. Al treilea element al şirului având valoarea **15** este mai mare decât **8**, deci se deplasează la dreapta cu o unitate, iar pe poziţia acestuia se aşează valoarea **8**;
Etapa 4: Variabila **j** are valoarea **4**, deci variabila **aux** primeşte valoarea **14** (**aux = 14**). Se verifică în subşirul format din primele patru elemente dacă elementele acestuia sunt mai mari decât **14**. Al patrulea element al şirului având valoarea **15** este mai mare decât **14**, deci se deplasează la dreapta cu o unitate, iar pe poziţia acestuia se aşează valoarea **14**;
Etapa 5: Variabila **j** are valoarea **5**, deci variabila **aux** primeşte valoarea **-9** (**aux = -9**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **-9**. Elementele cu valorile **5**, **8**, **14**, **15** având valori mai mari decât **-9**, se deplasează la dreapta cu o unitate, iar pe poziţia a doua se aşează valoarea **-9**;
Etapa 6: Variabila **j** are valoarea **6**, deci variabila **aux** primeşte valoarea **-15** (**aux = -15**). Se verifică în subşirul format din elementele din stânga dacă sunt mai mari decât **-15**. Toate elementele din stânga au având valori mai mari decât **-15**, deci se deplasează la dreapta cu o unitate, iar pe prima poziţie se aşează valoarea **-15**;
Etapa 7: Variabila **j** are valoarea **7**, deci variabila **aux** primeşte valoarea **0** (**aux = 0**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **0**. Elementele cu valorile **5**, **8**, **14**, **15** având valori mai mari decât **0**, se deplasează la dreapta cu o unitate, iar pe poziţia a patra se aşează valoarea **0**;
Etapa 8: Variabila **j** are valoarea **8**, deci variabila **aux** primeşte valoarea **10** (**aux = 10**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **10**. Elementele cu valorile **14**, **15** având valori mai mari decât **10**, se deplasează la dreapta cu o unitate, iar pe poziţia şapte se aşează valoarea **10**;

Etapa 9: Variabila **j** are valoarea **9**, deci variabila **aux** primeşte valoarea **-20** (**aux = -20**). Se verifică în subşirul format din elemente din stânga dacă sunt mai mari decât **-20**. Se observă că toate elementele din stânga au valori mai mari decât **-20**, astfel că se deplasează la dreapta cu o unitate, iar pe prima poziţie şapte se aşează valoarea **-20**.

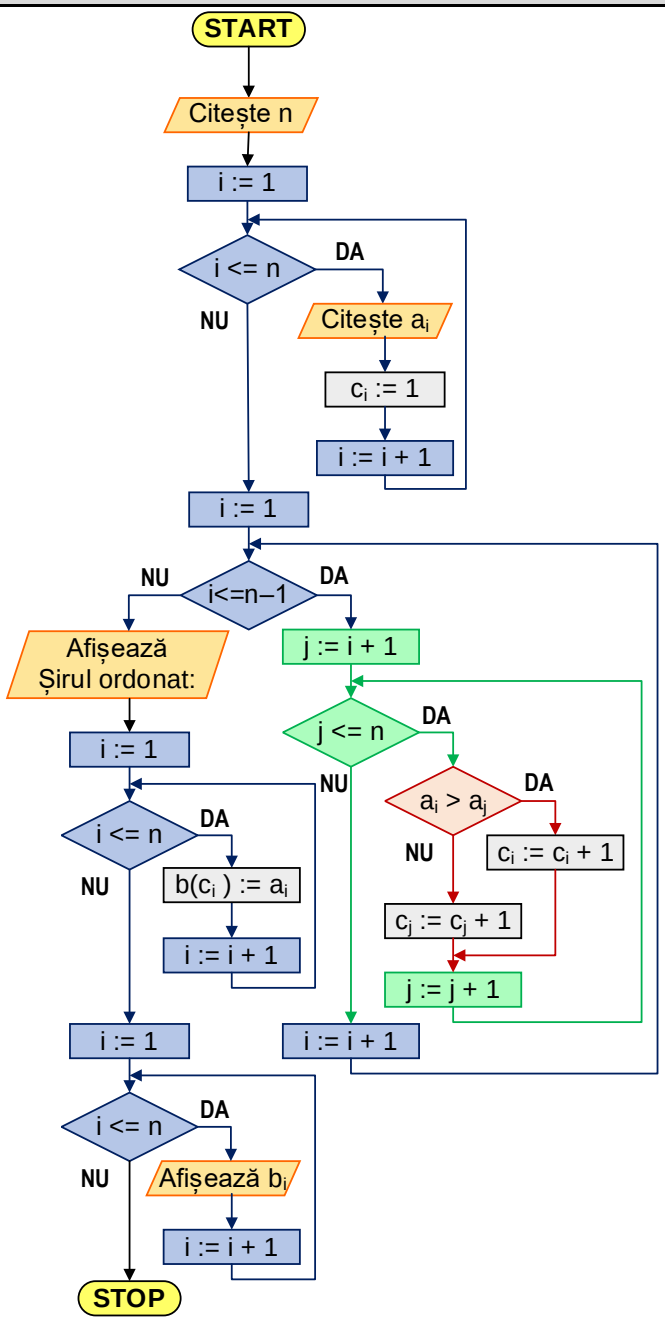
Se observă că şirul de valori este ordonat crescător.

5.8.5. Sortarea prin numărare (Counting Sort)

În cazul acestei metode se utilizează spaţiu suplimentar de memorie, utilizându-se următorii vectori:

- vectorul sursă (nesortat), notat cu a;
- vectorul destinaţie (sortat), notat cu b;
- vector numărător (care conţine contoarele), notat c.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte n
  Pentru i←1,n
      Citeşte ai
      ci← 1
  Sfârşit pentru
  Pentru i←1,n-1
      Pentru j←i+1,n
        Dacă ai > aj atunci
          ci← ci+1
        altfel
          cj← cj+1
        Sfârşit dacă
      Sfârşit pentru
  Sfârşit pentru
  Scrie „Sirul ordonat:”
  Pentru i←1,n
      bci← ai
  Sfârşit pentru
  Pentru i←1,n
      Scrie bi
  Sfârşit pentru
Sfârşit
```

Figura 5.8i. Reprezentarea algoritmului pentru sortarea (crescătoare) prin metoda inserţiei (Insertion Sort)

Algoritmul pentru ordonarea crescătoare utilizând această metodă constă în parcurgerea vectorului sursă şi pentru fiecare element al acestuia se numără câte elemente mai mici decât elementul curent sunt, valoarea astfel obţinută reprezintă poziţia pe care o va avea elementul curent al vectorului sursă în vectorul destinaţie, valoarea memorându-se în vectorul numărător c, pe poziţia corespunzătoare elementului curent al vectorului sursă;
Exemplul numeric: Modul de funcţionare al algoritmului este prezentat în tabelul 5.8.5:

Tabelul 5.8.5. Sortarea prin metoda inserţiei (Insertion Sort)

Indice	1	2	3	4	5	6	7	8	9	10
a [i]	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
c [i]	9	2	5	6	8	3	1	4	7	0
b [i]	-20	-15	-12	-9	0	5	8	10	14	15
Final:	**-20**	**-15**	**-12**	**-9**	**0**	**5**	**8**	**10**	**14**	**15**

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8i, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.8j.

Programul MATLAB	Execuția programului

```
n=input('Introduceti n, n= ');
for i=1:n
   s=sprintf(' a[%2d] = ',i);
   a(i)=input(s);
   c(i)=1;
end
for i=1:n-1
   for j=i+1:n
       if a(i) > a(j)
           c(i)=c(i)+1;
       else
           c(j)=c(j)+1;
       end
    end
end
disp('Sirul sortat este: ');
for i=1:n
    b(c(i))=a(i);
end
for i=1:n
    mesaj=sprintf('%4d',b(i));
disp(mesaj)
end
```

```
Introduceti n, n= 10
 a[ 1] = 15
 a[ 2] = -12
 a[ 3] = 5
 a[ 4] = 8
 a[ 5] = 14
 a[ 6] = -9
 a[ 7] = -15
 a[ 8] = 0
 a[ 9] = 10
 a[10] = -20
Sirul sortat este:
 -20
 -15
 -12
  -9
   0
   5
   8
  10
  14
  15
```

Figura 5.8j. Programul MATLAB și execuția acestuia pentru sortarea (crescătoare) prin metoda inserţiei (Insertion Sort)

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Dacă nu se dorește în mod special o anumită variantă de sortare, se poate folosi o altă metodă de rezolvare și anume, utilizând funcția **sort()** specifică limbajului de programare MATLAB. Această funcție are mai multe forme dar cel mai des folosită este:

```
b = sort(a, directie)
```

unde **a** este tabloul unidimensional. Practic se returnează în **b** tabloul **a** sortat conform **directie**, care poate fi 'ascend' pentru ordine crescătoare sau 'descend' pentru ordine descrescătoare. Al doilea parametru, **directie**, poate să și lipsească, situație în care se realizează implicit sortarea crescătoare.

Observații:

- *Dacă **a** este o matrice, atunci **sort (a)** sortează fiecare coloană.*
- *Dacă **a** este un tablou multidimensional, atunci **sort (a)** operează de-a lungul primei dimensiuni diferite de 1, tratând elementele ca tablouri.*

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.8k, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.8l.

Schema logică	Limbajul pseudocod

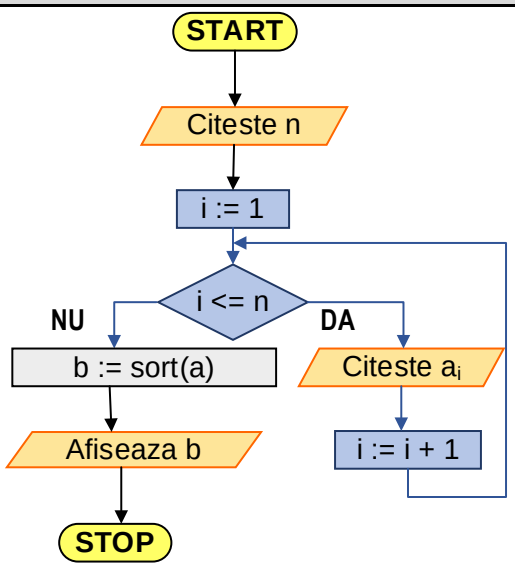
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Citeşte a_i
	  Sfârşit pentru
	  b← sort(a)
	  Scrie b
	Sfârşit

Figura 5.8k. Reprezentarea algoritmului pentru sortarea crescătoare a valorilor dintr-un tablou unidimensional apelând funcția **sort()**

Programul MATLAB	Execuția programului

`n=input('Introduceti n, n= ');`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`end`
`b=sort(a);`
`disp('Sirul sortat este: ');`
`disp(b)`
	`Introduceti n, n = 8`
	`a[ 1] =  2`
	`a[ 2] =  0`
	`a[ 3] =  1`
	`a[ 4] = -3`
	`a[ 5] =  4`
	`a[ 6] = -2`
	`a[ 7] =  3`
	`a[ 8] = -5`
	`Sirul sortat este:`
	`-5  -3  -2  0  1  2  3  4`

Figura 5.8l. Folosirea unei funcții specifice limbajului MATLAB pentru sortarea crescătoare a valorilor dintr-un tablou unidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.9. Căutarea unui element într-un șir neordonat (căutare secvenţială)

Algoritmul de căutare secvenţială este unul dintre cei mai simpli algoritmi de calcul studiaţi. Cu ajutorul acestuia se urmăreşte să se verifice dacă o valoarea (notată cu **v**) se află printre elementele unui vector, notat cu **a**.

Se parcurge vectorul, de la primul element, până la ultimul şi se compară valoarea v cu fiecare element al vectorului, utilizând operatorul de egalitate.

Dacă există egalitate între valoarea v şi valoarea unui element al vectorului, o variabilă ajutătoare numită **găsit** primeşte valoarea 1. Iniţial aceasta a primit valoarea 0.

În final, după parcurgerea vectorului, se verifică valoarea variabilei găsit, dacă aceasta este 1 se afişează pe ecran un mesaj prin care se specifică faptul că variabila **v** se află printre elementele vectorului, iar în cazul în care aceasta este 0 atunci se afişează pe ecran un mesaj prin care se specifică că variabila v nu se află printre elementele vectorului.

Exemplul numeric nr. 1: Se caută valoarea **5** în şirul de valori **15**, **-12**, **5**, **8**, **14**, **-9**, **-15**, **0**, **10**, **-20**.
Modul de funcţionare al algoritmului este prezentat în tabelul 5.9.1:

Tabelul 5.9.1. Căutarea unui element într-un vector (căutare secvenţială) – exemplul 1

Element	1	2	3	4	5	6	7	8	9	10
Indice i	1	2	3	4	5	6	7	8	9	10
a [i]	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
v == a[i]	**NU**	**NU**	**DA**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**
gasit:	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Observaţie: valoarea variabilei ajutătoare **gasit** rămâne **1** după ce s-a găsit corespondenţa între valoarea **v** şi unul dintre elementele vectorului.

Exemplul numeric nr. 2: Se caută valoarea **3** în şirul de valori **15**, **-12**, **5**, **8**, **14**, **-9**, **-15**, **0**, **10**, **-20**.
Modul de funcţionare al algoritmului este prezentat în tabelul 5.9.2:

Tabelul 5.9.2. Căutarea unui element într-un vector (căutare secvenţială) – exemplul 2

Element	1	2	3	4	5	6	7	8	9	10
Indice i	1	2	3	4	5	6	7	8	9	10
a [i]	**15**	**-12**	**5**	**8**	**14**	**-9**	**-15**	**0**	**10**	**-20**
v == a[i]	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**	**NU**
gasit:	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Observaţie: valoarea variabilei ajutătoare **gasit** rămâne **0** deoarece nu s-a găsit corespondenţă între valoarea **v** şi elementele vectorului.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.9a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.9b.

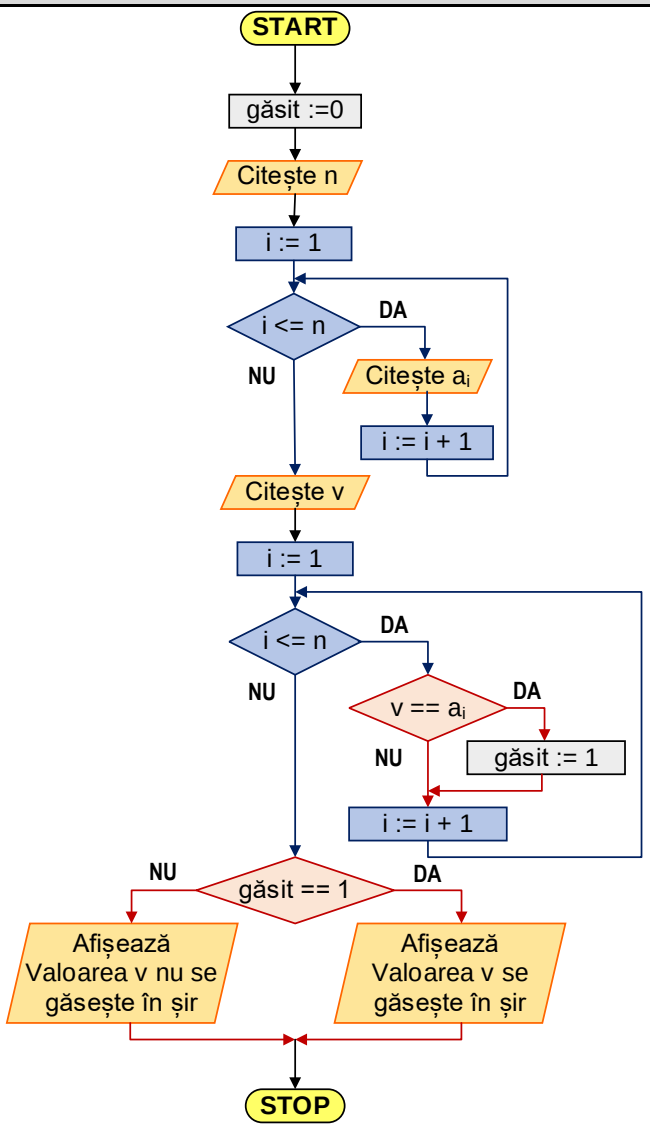
Schema logică	Limbajul pseudocod
	**Început** gasit← 0 **Citeşte** n **Pentru** i←1,n **Citeşte** a_i **Sfârşit pentru** **Citeşte** v **Pentru** i←1,n **Dacă** v==a_i **atunci** gasit← 1 **Sfârşit dacă** **Sfârşit pentru** **Dacă** gasit==1 **atunci** **Scrie** „Valoarea v se gaseste in sir” **altfel** **Scrie** „Valoarea v nu se gaseste in sir” **Sfârşit dacă** **Sfârşit**

Figura 5.9a. Reprezentarea algoritmului pentru căutarea unui element într-un vector (căutare secvenţială)

Programul MATLAB	Execuția programului

```
n=input('Introduceti n= ');
for i=1:n
   s=sprintf(' a[%2d] = ',i);
   a(i)=input(s);
end
v=input('Introduceti valoarea v= ');
gasit=0;
for i=1:n
   if v == a(i)
       gasit=1;
   end
end
if gasit == 1
    mesaj=sprintf('%4d e in sir!',v);
    disp(mesaj)
```

```
Exemplul rulare 1:
Introduceti n = 7
 a[1] = -12
 a[2] = 5
 a[3] = 8
 a[4] = 14
 a[5] = -9
 a[6] = -15
 a[7] = 0
Introduceti valoarea v = 5
  5 e in sir!

Exemplul rulare 2:
Introduceti n= 5
 a[ 1] = 11
 a[ 2] = 0
 a[ 3] = -2
```

```
else
    mesaj=sprintf('%4d nu e in sir!',v);
    disp(mesaj)
end
```

```
 a[ 4] = 9
 a[ 5] = 6
Introduceti valoarea v= 10
 10 nu e in sir!
```

Figura 5.9b. Programul MATLAB și execuția acestuia pentru căutarea unui element într-un vector (căutare secvenţială)

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.10. Căutarea unui element într-o listă ordonată (căutare binară)

Algoritmul de căutare binară este utilizat pentru găsirea unui element într-o listă ordonată de elemente (crescător sau descrescător). Algoritmul funcţionează pe baza tehnicii „divide et impera” (împarte şi cucereşte).

În cadrul algoritmului, valoarea căutată este comparată cu cea a elementului din mijlocul listei, existând trei situaţii posibile:

- dacă valoarea căutată este egală cu cea de la mijlocul listei, algoritmul se finalizează;
- dacă valoarea căutată este mai mare decât valoarea de la mijlocul listei, algoritmul se reia de la mijlocul listei până la sfârşit;
- dacă valoarea căutată este mai mică decât valoarea de la mijlocul listei, algoritmul se reia de la începutul listei până la mijlocul acesteia.

Se consideră un şir de numere ordonat crescător, notat cu **a** şi trei variabile **st**, **dr**, şi **mj** cu ajutorul cărora se memorează indicii extremităţilor, respectiv a mijlocului şirului considerat. Se utilizează o variabilă ajutătoare, cu numele **găsit**, cu valoarea iniţială 0. Când se găseşte egalitate între valoarea căutată şi un element din şir, variabila **gasit** primește valoarea **1**;

Algoritmul de căutare presupune parcurgerea următorilor paşi:

- se citeşte numărul de elemente din şir, notat cu **n**;
- se citesc în ordine crescătoare, elementele şirului;
- se citeşte valoarea căutată, notată cu **v**;
- se iniţializează variabilele **st**, **dr** şi **gasit** cu valorile **0**, **n-1**, respectiv **0**;
- cu ajutorul unui ciclu cu test iniţial de realizează restrângerea şirului în care se caută valoarea **v**, după metoda prezentată anterior, atât timp cât se îndeplinesc simultan condiţiile **st <= dr**, respectiv **gasit ~= 1**. În cadrul ciclului, se determină valoarea variabilei **mj** şi se compară valoarea căutată **v** cu valoarea elementului din şir a cărui indice este **mj**, astfel:

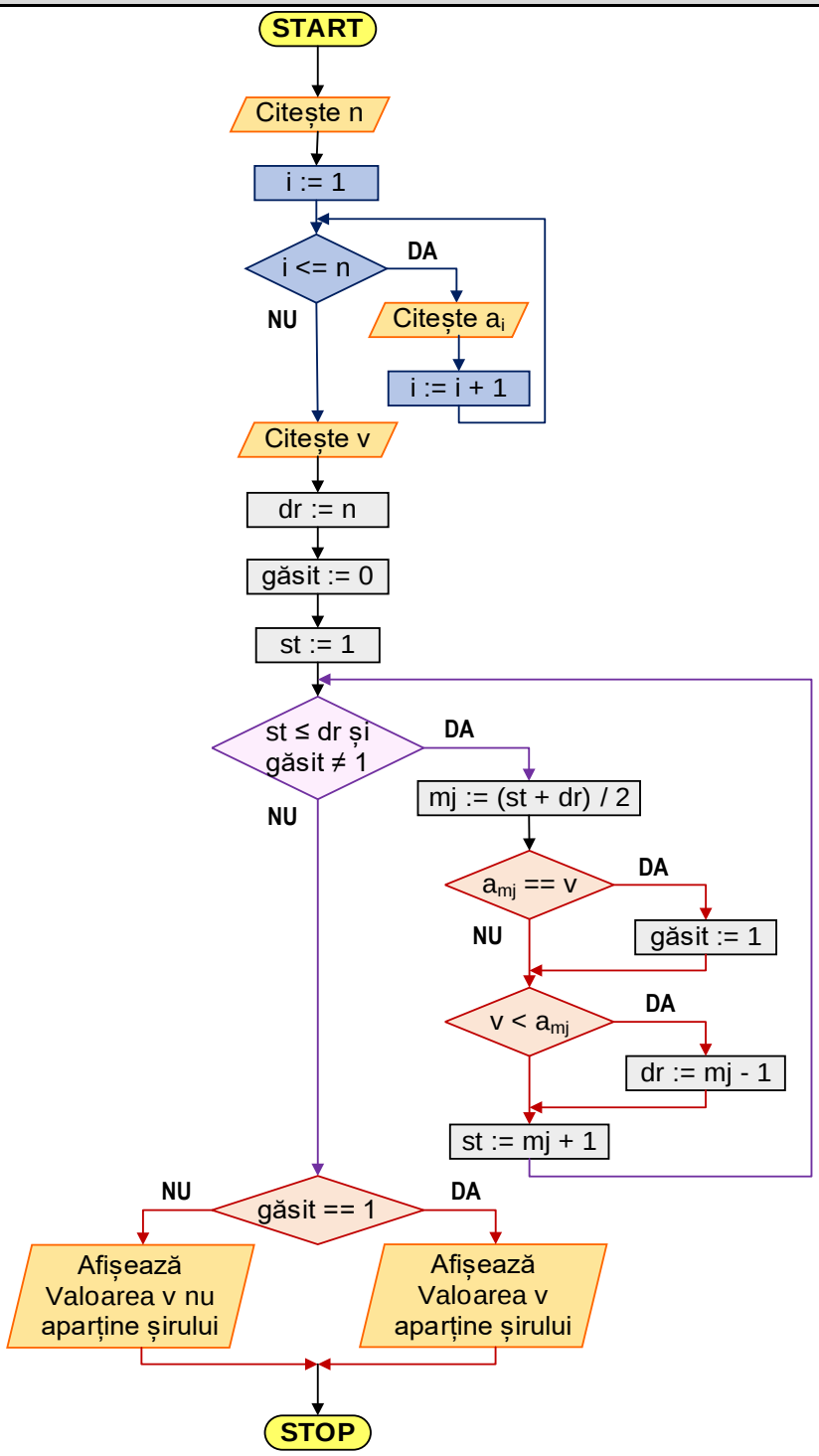
- dacă cele două valori sunt egale înseamnă că s-a găsit valoarea **v** în şir, variabila **gasit** primind valoarea 1, astfel că nu se mai îndeplineşte condiţia de execuţie a corpului ciclului şi se părăseşte ciclul, execuţia continuând cu instrucţiunea următoare ciclului;

- dacă cele două valori nu sunt egale, se verifică dacă valoarea variabilei **v** este mai mică decât cea a elementului cu indicele **mj**, existând două posibilităţi: **v < a[mj]** situaţie în care se restrânge şirul de elemente, variabila **dr** primind valoarea **mj-1**, respectiv dacă **v > a[mj]** se restrânge şirul de elemente prin atribuirea variabilei **st** a valorii **mj+1**;

Cu noul şir de valori se revine la căutarea valorii v.

- după părăsirea ciclului cu test iniţial se verifică valoarea variabilei **gasit**. Dacă aceasta este egală cu **1** înseamnă că valoarea căutată **v** se află în şir, în caz contrar valoarea nu se află în şir.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte n
   Pentru i←1,n
        Citeşte ai
   Sfârşit pentru
   Citeşte v
   gasit← 0,    st← 1,    dr← n
   Cât timp st≤dr ŞI gasit ≠ 1 execută
```

```
            mj← (st+dr) DIV 2
            Dacă v==a_mj atunci
               gasit← 1
            altfel
               Dacă v<a_mj atunci
                  dr← mj-1
               altfel
                  st← mj+1
               Sfârşit dacă
            Sfârşit dacă
         Sfârşit cât timp
   Dacă gasit==1 atunci
      Scrie „Valoarea v se gaseste in sir”
   altfel
      Scrie „Valoarea v nu se gaseste in sir”
   Sfârşit dacă
Sfârşit
```

Figura 5.10a. Reprezentarea algoritmului pentru căutarea unui element într-un vector (căutare binară)

Programul MATLAB	Execuția programului

```
n=input('Introduceti n= ');
for i=1:n
   s=sprintf(' a[%2d] = ',i);
   a(i)=input(s);
end
v=input('Introduceti valoarea v= ');
st = 1; dr = n; gasit = 0;
while st<=dr & gasit~=1
   mj=int32((st+dr)/2);
   if  a(mj) == v     gasit=1;
   elseif  v < a(mj)     dr=mj-1;
   else  st=mj+1;
   end
end
if gasit == 1
   mesaj=sprintf('%4d e in sir!',v);
   disp(mesaj)
else
   mesaj=sprintf('%4d nu e in sir!',v);
   disp(mesaj)
end
```

```
Exemplul rulare 1:
Introduceti n= 6
 a[ 1] = 1
 a[ 2] = 2
 a[ 3] = 3
 a[ 4] = 5
 a[ 5] = 8
 a[ 6] = 11
Introduceti valoarea v= 8
  8 e in sir!

Exemplul rulare 2:
Introduceti n= 6
 a[ 1] = 1
 a[ 2] = 2
 a[ 3] = 3
 a[ 4] = 5
 a[ 5] = 8
 a[ 6] = 11
Introduceti valoarea v= 20
  20 nu e in sir!
```

Figura 5.10b. Programul MATLAB și execuția acestuia pentru căutarea unui element într-un vector (căutare binară)

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.10a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.10b.

Exemplul numeric 1: Se caută valoarea **5** în şirul de valori: **-20, -15, -12, -9, 0, 5, 8, 10, 14, 15**.
Modul de funcţionare al algoritmului este prezentat în tabelul 5.10.1:

Tabelul 5.10.1. Căutarea unui element într-un vector (căutare secvenţială) – exemplul 1

	Element	1	2	3	4	5	6	7	8	9	10
	Indice i	1	2	3	4	5	6	7	8	9	10
Iniţial	a [i]	-20	-15	-12	-9	0	5	8	10	14	15
Etapa 1	st =	0		dr =		9		mj =		4	
	şir obţinut						5	8	10	14	15
Etapa 2	st =	5		dr =		9		mj =		7	
	şir obţinut						5	8			
Etapa 3	st =	5		dr =		6		mj =		5	
	şir obţinut						5				
Valoarea **5** apartine sirului!											

5.11. Inserarea unui element într-un șir

5.11.1. Inserarea unui element într-un șir, pe o poziţie specificată

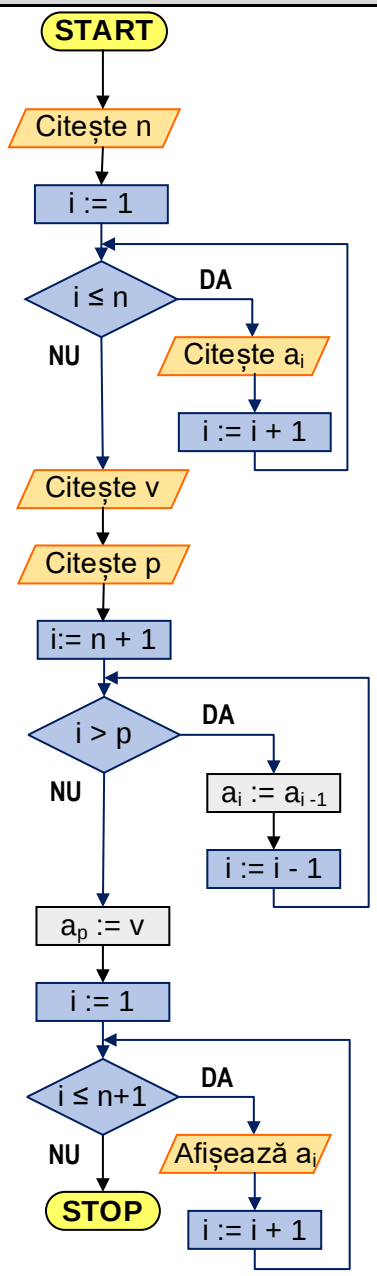
Se consideră un șir notat cu **a**, cu **n** elemente şi se doreşte inserarea unui element în șir, pe o anumită poziţie – notată cu **p**, șirul astfel obţinut având **n+1** elemente. Șirul nou se va forma astfel:
- până la poziţia p-1, elementele șiruluii rămân la fel, ca la început;
- pe poziţia p se inserează valoarea v;
- de la poziţia **p+1** la **n+1** se inserează elementele șirului iniţial de la poziţia **p** la **n**;

Astfel, pentru crearea noului șir este necesar şi suficient să se parcurgă șirul de la ultimul element al său (de indice **n+1**) până la poziţia **p+1**, atribuindu-se elementului de pe poziţia **i**, valoarea elementului de pe poziţia **i-1** din șir, adică se realizează o deplasare la dreapta a elementelor șirului de la poziţia **p** şi până la sfârşit, lăsând astfel liberă poziţia **p**, pe care se inserează valoarea **v**.

Algoritmul de inserare constă în parcurgerea următorilor paşi:
- se citeşte numărul de elemente din șir, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale șirului;
- se citeşte valoarea **v** care se doreşte a fi inserată;
- se citeşte valoarea **p** a poziţiei în care se doreşte inserarea variabilei **v**;
- cu ajutorul unui ciclu cu contor se realizează deplasarea la dreapta cu o poziţie a elementelor mulţimii situate de la poziţia **p** şi până la sfârşit, operaţie care presupune parcurgerea noului șir în ordine inversă de la poziţia **n+1** până la poziţia **p+1**;
- pe poziţia **p** se inserează valoarea **v**, restul șirului rămâne la fel;
- în final se afişează șirul astfel obţinut;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.11a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.11b.

Schema logică	Limbajul pseudocod



Început
  Citeşte n
  Pentru i←1,n
    Citeşte a_i
  Sfârşit pentru
  Citeşte v
  Citeşte p
  Pentru i←n+1,p+1,-1
    a_i← a_{i-1}
  Sfârşit pentru
  a_p← v
  Scrie a
Sfârşit

Figura 5.11a. Reprezentarea algoritmului pentru inserarea unui element într-o mulţime pe poziţia p

Exemplul numeric: Se inserează valoarea **0** pe poziţia **4** în şirul de valori: **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**.
Modul de funcţionare al algoritmului este prezentat în tabelul 5.11.1:

Tabelul 5.11.1. Inserarea unui element într-o mulţime pe poziţia p

	Element	1	2	3	4	5	6	7	8	9	
Iniţial	**Indice i**	1	2	3	4	5	6	7	8	9	
	a [i]	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Intermediar	**Indice i**	1	2	3	4	5	6	7	8	9	**10**
	a [i]	**1**	**2**	**3**		**4**	**5**	**6**	**7**	**8**	**9**
Final	**Indice i**	1	2	3	4	5	6	7	8	9	**10**
	a[i]	**1**	**2**	**3**	**0**	**4**	**5**	**6**	**7**	**8**	**9**

Programul MATLAB	Execuția programului

`n=input('Introdu n= ');`
`for i=1:n`
`  s=sprintf(' a[%2d] = ',i);`
`  a(i)=input(s);`
`end`
`v=input('Introdu valoarea v= ');`
`p=input('Introdu pozitia p= ');`
`for i = n+1:-1:p+1`
`  a(i) = a(i-1);`
`end`
`a(p) = v;`
`disp('Sirul nou este: '); disp(a)`
	`Introdu n = 9`
	` a[1] = 1`
	` a[2] = 2`
	` a[3] = 3`
	` a[4] = 4`
	` a[5] = 5`
	` a[6] = 6`
	` a[7] = 7`
	` a[8] = 8`
	` a[9] = 9`
	` Introdu valoarea v = 0`
	` Introdu pozitia p = 4`
	`Sirul nou este:`
	` 1 2 3 0 4 5 6 7 8 9`

Figura 5.11b. Programul MATLAB și execuția acestuia pentru inserarea unui element într-o mulţime pe poziţia p

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

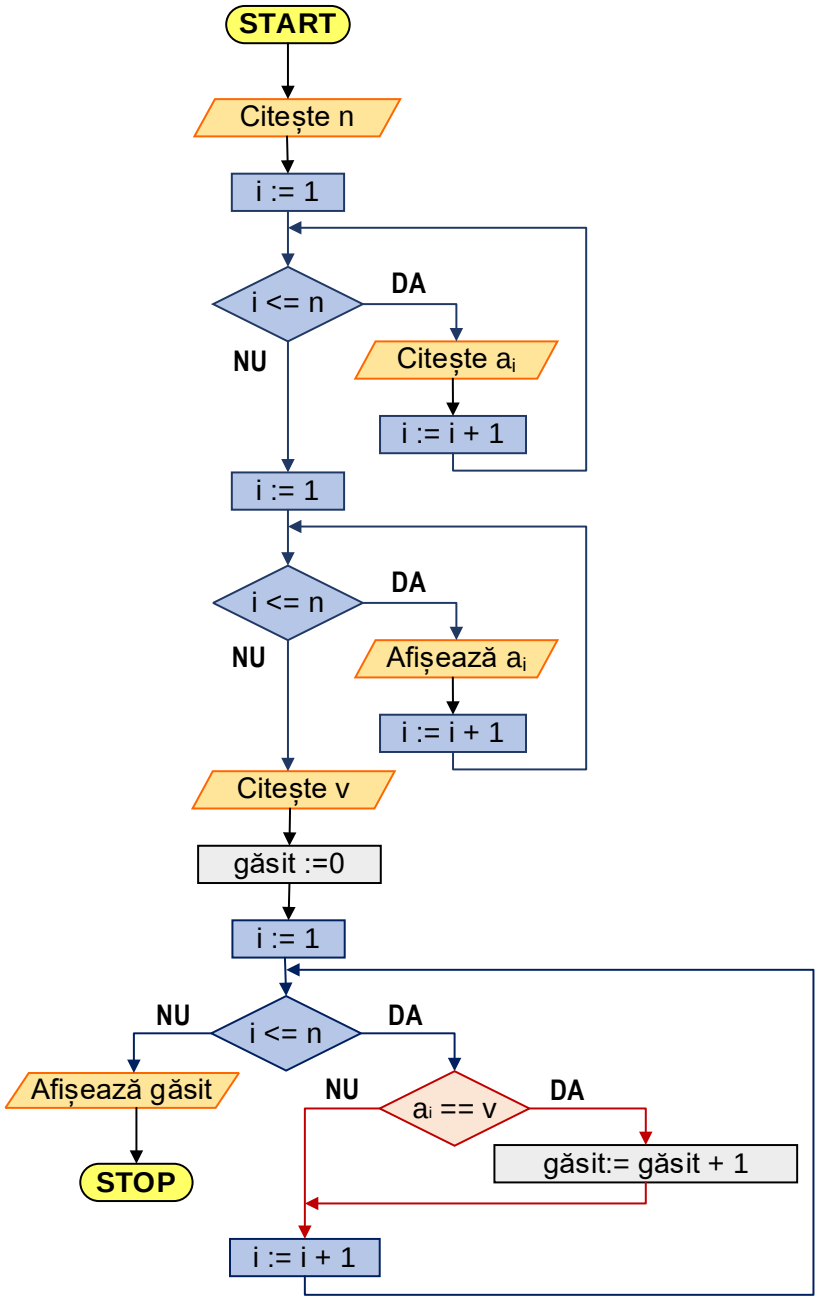
5.12. Determinarea numărului de apariţii a unei valori într-o mulţime

Se consideră o mulţime cu **n** elemente şi se doreşte determinarea numărului de apariţii a unei valori **v** în cadrul mulţimii. Algoritmul constă în parcurgerea următorilor paşi:

- se citeşte numărul de elemente din mulţime, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale mulţimii;
- se afişează elementele mulţimii citite anterior;
- se citeşte valoarea **v** căutată;
- se iniţializează o variabilă ajutătoare, cu numele **gasit**, cu valoarea 0 (zero). Cu ajutorul acesteia se va calcula numărul de apariţii a variabilei **v** în mulţime;
- cu ajutorul unui ciclu cu contor se realizează parcurgerea mulţimii element cu element, pornind de la primul element spre ultimul. Cu ajutorul unei instrucţiuni de decizie se verifică pentru fiecare element în parte dacă este egal cu valoarea **v**. Dacă se constată egalitatea se incrementează valoarea variabilei **gasit**. Dacă nu se constată egalitatea se trece la următorul element din mulţime;
- în final se afişează numărul de apariţii a variabilei **v** în mulţimea considerată prin afişarea valorii variabilei **gasit**;

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 5.12a, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 5.12b.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte n
  Pentru i←1,n
      Citeşte ai
  Sfârşit pentru
  Pentru i←1,n
      Scrie ai
  Sfârşit pentru
  Citeşte v
  gasit← 0
  Pentru i←1,n
      Dacă v==ai atunci
          gasit← gasit+1
      Sfârşit dacă
  Sfârşit pentru
  Scrie gasit
Sfârşit
```

Figura 5.12a. Reprezentarea algoritmului pentru determinarea numărului de apariţii a unei valori într-o mulţime

Programul MATLAB	Execuția programului

```
n=input('Introduceti n= ');
for i=1:n
    s=sprintf('Introduceti a[%2d] = ',i);
    a(i)=input(s);
end
s='';
for i=1:n
    s=[s,num2str(a(i)),' '];
end
disp(s)
```

```
Rularea programului -
cazul 1:
 Introduceti n, n = 6
 Introduceti a[1] = 1
 Introduceti a[2] = 2
 Introduceti a[3] = 5
 Introduceti a[4] = 2
 Introduceti a[5] = 3
 Introduceti a[6] = 4
  1   2   5   2   3   4
 Introduceti v, v = 2
```

```
v=input('Introduceti v= ');
gasit = 0;
for i=1:n
    if a(i) == v
       gasit=gasit+1;
    end
end
s=sprintf('S-au gasit %d elemente! ',gasit);
disp(s)
```

```
 S-au gasit 2 elemente!

Rularea programului –
cazul 2:
 Introduceti n, n = 6
 Introduceti a[1] = 1
 Introduceti a[2] = 2
 Introduceti a[3] = 3
 Introduceti a[4] = 4
 Introduceti a[5] = 5
 Introduceti a[6] = 6
  1   2   3   4   5   6
 Introduceti v, v = 8
 S-au gasit 0 elemente!
```

Figura 5.12b. Programul MATLAB și execuția acestuia pentru determinarea numărului de apariţii a unei valori într-o mulţime

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.13. Eliminarea unui element dintr-o mulţime

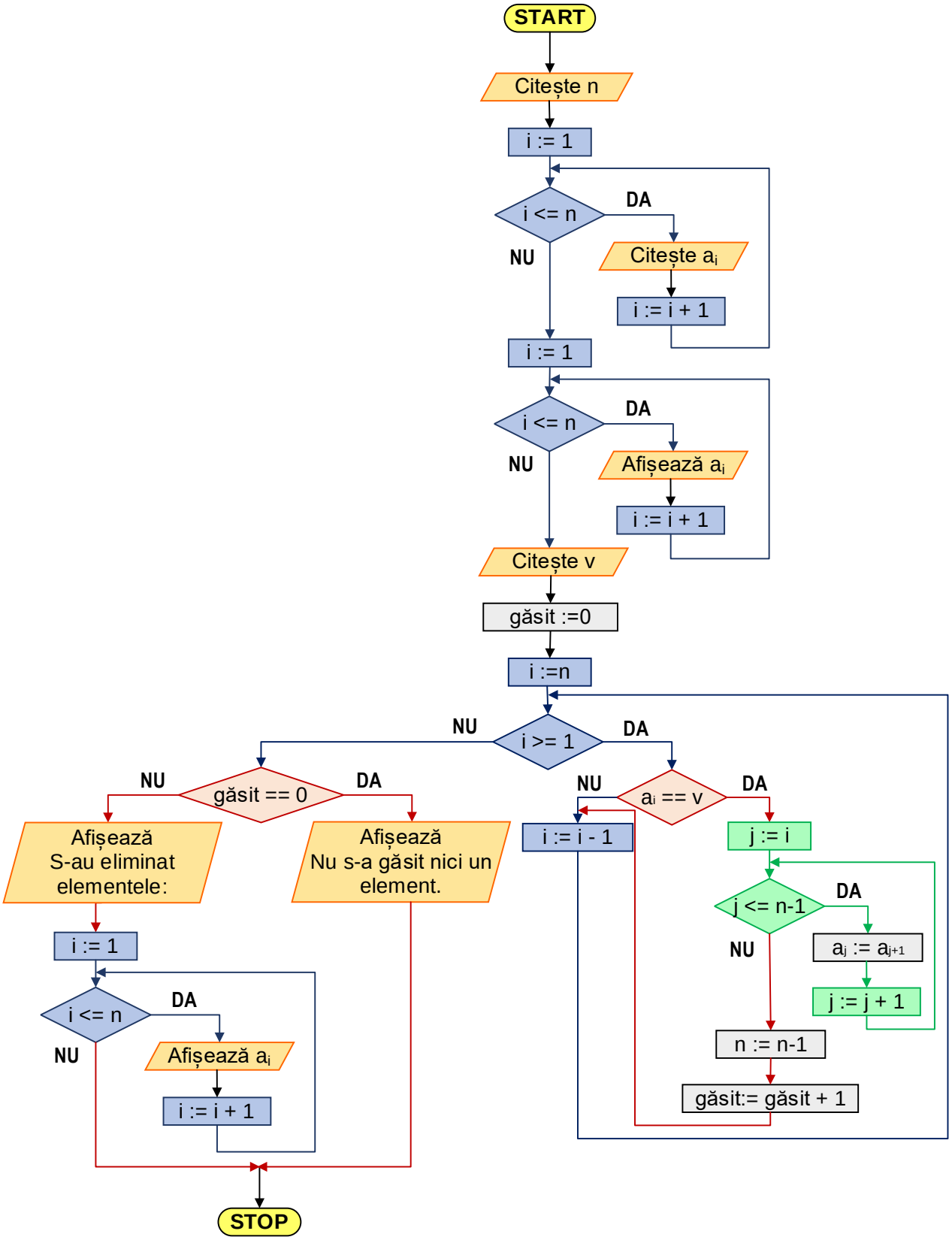
Se consideră o mulţime (ordonată sau nu) cu **n** elemente şi se doreşte eliminarea unui element din mulţime. Se va obţine o mulţime cu mai puţine elemente, numărul acestora fiind mai mic cu numărul elementelor găsite şi eliminate.

Algoritmul de eliminare constă în parcurgerea următorilor paşi:

- se citeşte numărul de elemente din mulţime, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale mulţimii;
- se afişează elementele mulţimii citite anterior;
- se citeşte valoarea **v** care se doreşte a fi eliminată;
- se iniţializează o variabilă ajutătoare, cu numele **gasit**, cu valoarea 0 (zero). Cu ajutorul acesteia se va calcula numărul de elemente eliminate;
- cu ajutorul unui ciclu cu contor se realizează parcurgerea mulţimii element cu element, pornind de la ultimul element spre primul. Pentru fiecare element al mulţimii se verifică, cu ajutorul unei decizii cu o ramură, dacă valoarea acestuia este egală cu valoarea variabilei ajutătoare **v**. În cazul în care valoarea elementului curent al mulţimii este egală cu valoarea variabilei **v**, se realizează următoarele operaţii:
  - deoarece s-a găsit un element care trebuie eliminat, eliminarea acestuia se realizează prin mutarea cu o poziţie spre primul element al elementelor care au fost verificate până acum, adică a elementelor de la poziţia **i + 1** până la ultimul, astfel că elementul de pe poziţia **i + 1** va „sări” pe poziţia **i**, cel de pe poziţia **i + 2** va „sări” pe poziţia **i + 1**, ş.a.m.d;
  - deoarece numărul de elemente a scăzut cu **1**, se decrementează variabila **n**;
  - deoarece s-a găsit un element care trebuie eliminat, se incrementează variabila **gasit**;
- în final cu ajutorul unei decizii se verifică valoarea variabilei **gasit**. Dacă aceasta este **0**, înseamnă că şirul nu conţine elemente egale cu valoarea căutată şi se afişează un mesaj corespunzător. Dacă valoarea variabilei **gasit** este diferită de zero (pozitivă) atunci se afişează numărul de elemente eliminate şi utilizând un ciclu cu contor se afişează elementele mulţimii;

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 5.13a, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 5.13b.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte n
  Pentru i←1,n
      Citeşte ai
  Sfârşit pentru
  Pentru i←1,n
      Scrie ai
  Sfârşit pentru
  Citeşte v
  gasit← 0
  Pentru i←n,1,-1
      Dacă v==ai atunci
          Pentru j←i,n-1
            aj← aj+1
          Sfârşit pentru
          n←n-1
          gasit← gasit+1
      Sfârşit dacă
  Sfârşit pentru
  Dacă gasit==0 atunci
     Scrie „Nu s-au gasit elemente”
    altfel
      Scrie gasit
    Sfârşit dacă
Sfârşit
```

Figura 5.13a. Reprezentarea algoritmului pentru eliminarea unui element dintr-o mulţime

Programul MATLAB şi execuţia acestuia

```
n=input('Introduceti n= ');
for i=1:n
   s=sprintf('Introduceti a[%2d] = ',i);
   a(i)=input(s);
end
s=' ';
for i=1:n
   s=[s,num2str(a(i)),' '];
end
disp(s)
v=input('Introduceti v= ');
gasit = 0;
for i=n:-1:1
  if a(i) == v
    for j = i :n-1
       a(j) = a(j+1);
    end
    n=n-1; gasit=gasit+1;
  end
```

```
Rularea programului -
Cazul 1:
 Introduceti n, n = 8
 Introduceti a[1] = 2
 Introduceti a[2] = 3
 Introduceti a[3] = 1
 Introduceti a[4] = 5
 Introduceti a[5] = 4
 Introduceti a[6] = 6
 Introduceti a[7] = 5
 Introduceti a[8] = 9
  2  3  1  5  4  6  5  9
 Introduceti v, v = 5

 S-au eliminat 2 elemente!
  2  3  1  4  6  9

Rularea programului -
Cazul 2:
```

```
end
if gasit == 0
 disp(' Nu s-a gasit niciun element!');
else
    s=sprintf('S-au eliminat %d elemente!
',gasit); disp(s); s='';
    for i=1:n
      s=[s,num2str(a(i)),' '];
    end
    disp(s)
end
```

```
 Introduceti n, n = 4
 Introduceti a[1] = 2
 Introduceti a[2] = 3
 Introduceti a[3] = 1
 Introduceti a[4] = 4
  2  3  1  4
 Introduceti v, v = 5

 Nu s-a gasit niciun
element!
```

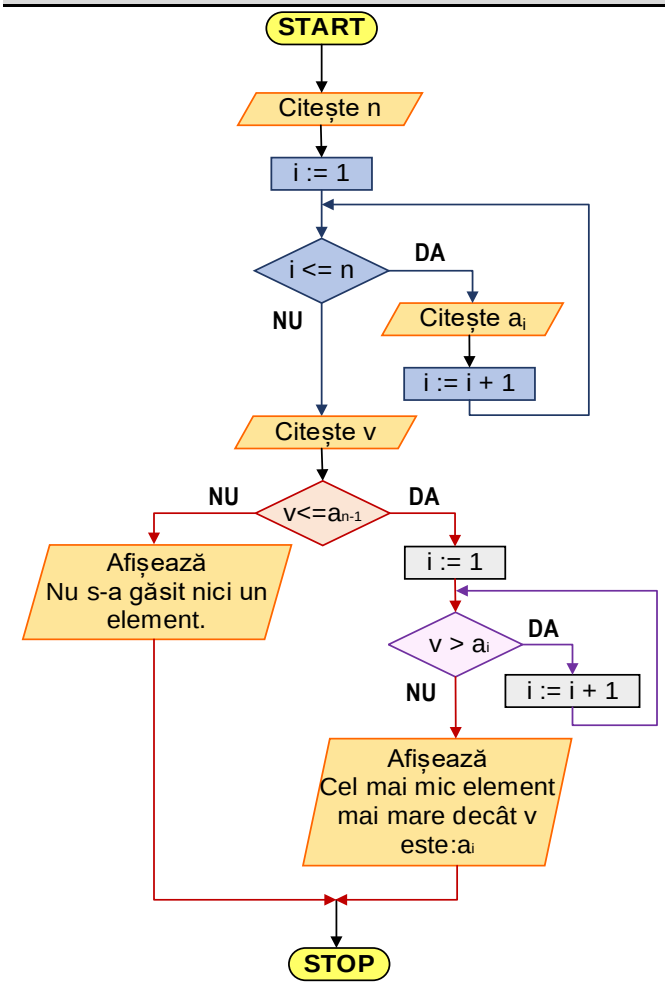
Figura 5.13b. Programul MATLAB şi execuţia acestuia pentru eliminarea unui element dintr-o mulţime

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

5.14. Determinarea celui mai mic număr mai mare decât o valoare impusă, dintr-o mulţime ordonată crescător

Se consideră o mulţime cu **n** elemente şi se doreşte determinarea celui mai mic număr din mulţime, mai mare decât o valoare impusă **v**.

Schema logică



Limbajul pseudocod

Început
  Citeşte n
  Pentru i←1,n
    Citeşte a_i
  Sfârşit pentru
  Citeşte v
  Dacă v≤a_{n-1} **atunci**
    i←1
    Cât timp v>a_i **execută**
      i← i+1
    Sfârşit cât timp
    Scrie a_i
  altfel
    Scrie „Nu s-a gasit”
  Sfârşit dacă
Sfârşit

Figura 5.14a. Reprezentarea algoritmului pentru determinarea celui mai mic număr mai mare decât o valoare impusă, dintr-o mulţime ordonată crescător

Algoritmul constă în parcurgerea următorilor paşi:
- se citeşte numărul de elemente din mulţime, adică valoarea variabilei **n**;
- se citesc valorile celor **n** elemente ale mulţimii;
- se afişează elementele mulţimii citite anterior;
- se citeşte valoarea variabilei **v**;
- cu ajutorul unei instrucţiuni de decizie se verifică dacă variabila **v** este mai mică decât ultimul element al mulţimii **a[n-1]**, existând două posibilităţi:

- dacă variabila **v** este mai mică înseamnă că există un element al mulţimii mai mare sau egal decât **v**. În acest caz, se iniţializează contorul **i** cu valoarea **0**, iar cu ajutorul unei instrucţiuni de ciclare cu test iniţial se verifică pe rând dacă valoarea **v** este mai mare decât valoarea elementului curent al mulţimii. Atunci când valoarea variabilei v este mai mică decât elementul curent al mulţimii, se părăseşte ciclul cu test iniţial şi se afişează valoarea variabilei **v** precum şi cea a elementului curent al mulţimii **a[i]**;

- dacă variabila **v** nu este mai mică decât ultimul element al mulţimii, căutarea nu se mai efectuează şi se afişează un mesaj corespunzător.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.14a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.14b.

Programul MATLAB	Execuția programului

```
n=input('Introduceti n= ');
for i=1:n
   s=sprintf('Introduceti a[%2d] = ',i);
   a(i)=input(s);
end
v=input('Introduceti v= ');
if v <= a(n-1)
   i = 1;
   while(v > a(i))
          i=i+1;
   end
   s=sprintf(' Cel mai mic element mai
mare decat %6.3f este %6.3f',v,a(i));
   disp(s);
else
   disp('Nu s-a gasit niciun element!');
end
```

```
Rularea programului – cazul 1:
 Introduceti n, n = 6
 Introduceti a[1] = 1
 Introduceti a[2] = 2
 Introduceti a[3] = 3
 Introduceti a[4] = 4
 Introduceti a[5] = 5
 Introduceti a[6] = 6
 Introduceti v, v = 3.14159
 Cel mai mic element mai mare
decat  3.142 este  4.000

Rularea programului – cazul 2:
 Introduceti n, n = 6
 Introduceti a[1] = 1
 Introduceti a[2] = 2
 Introduceti a[3] = 3
 Introduceti a[4] = 4
 Introduceti a[5] = 5
 Introduceti a[6] = 6
 Introduceti v, v = 7
 Nu s-a gasit niciun element!
```

Figura 5.14b. Programul MATLAB și execuția acestuia pentru determinarea celui mai mic număr mai mare decât o valoare impusă, dintr-o mulţime ordonată crescător

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.15. Conversia unui număr din baza 10 într-o bază de la 2 la 9

Într-un sistem de numeraţie poziţional un număr oarecare **N** (cu parte întreagă şi parte fracţionară, separate prin virgulă) se poate scrie sub una din următoarele forme:

$$N = a_{n-1}a_{n-2}...a_1a_0, a_{-1}a_{-2}...a_{(m-1)}a_{-m}$$

$$N = a_{n-1}q^{n-1} + a_{n-2}q^{n-2} + ... + a_1q^1 + a_0q^0 + a_{-1}q^{-1} + a_{-2}q^{-2} + ... + a_{-m}q^{-m}$$

$$N = \sum_{i=-m}^{n-1} a_i q^i$$

unde: ***q*** reprezintă baza sistemului de numeraţie, a_i reprezintă cifrele utilizate în sistemul de numeraţie, ***n*** reprezintă numărul de cifre întregi ale numărului, ***m*** fiind numărul de cifre fracţionare ale numărului.

În relaţiile de mai sus, cifrele a_i reprezintă coeficienţii cu care se înmulţesc puterile q^i ale bazei ***q***. Cifra a_{n-1} este cifra cea mai semnificativă în timp ce cifra a_{-m} este cifra cea mai puţin semnificativă.

Pentru înţelegerea modului de conversie a părţii întregi a unui număr dintr-o bază de numeraţie în alta se pleacă de la faptul cunoscut că dacă ***N*** şi ***q*** sunt numere întregi, există întotdeauna un singur întreg ***r*** – numit rest (pozitiv) mai mic decât ***q*** şi un singur întreg ***C***, astfel încât:

$$\frac{N}{q} = C + \frac{r}{q},\ 0 \le r < q$$

Pe baza regulii prezentate anterior pentru determinarea algoritmului de conversie a unui număr întreg ***N*** din baza ***p*** în baza ***q*** se porneşte de la expresia numărului ***N*** scris în baza ***q*** :

$$N_p = a_{n-1} \cdot q^{n-1} + ... + a_2 \cdot q^2 + a_1 \cdot q^1 + a_0 \cdot q^0$$

Prin împărţire cu baza ***q*** se obţine :

$$\frac{N_p}{q} = \underbrace{a_{n-1} \cdot q^{n-2} + ... + a_2 \cdot q^1 + a_1 \cdot q^0}_{Cat} + \underbrace{\frac{a_0}{q}}_{\text{Re } st} = C_0 + \frac{r_0}{q}$$

astfel : $a_0 = r_0$, respectiv $C_0 = a_{n-1} \cdot q^{n-2} + ... + a_2 \cdot q^1 + a_1 \cdot q^0$.

Se împarte câtul obţinut la ***q***, obţinându-se :

$$\frac{C_0}{q} = \underbrace{a_{n-1} \cdot q^{n-3} + ... + a_2 \cdot q^0}_{Cat} + \underbrace{\frac{a_1}{q}}_{\text{Re } st} = C_1 + \frac{r_1}{q}$$

adică : $a_1 = r_1$, respectiv $C_1 = a_{n-1} \cdot q^{n-3} + ... + a_2 \cdot q^0$.

Operaţia se continuă până când se obţine un cât egal cu zero.

Algoritmul de conversie a unui număr întreg dintr-o bază în alta poate fi enunţat astfel :

Pentru conversia părţii întregi se realizează împărţirea succesivă a numărului scris în baza p la q (baza în care se doreşte conversia). Astfel, se împarte numărul la bază obţinându-se un cât şi un rest. Se împarte câtul din nou la bază obţinându-se un nou cât şi un nou rest, ş.a.m.d. până când câtul devine 0. Resturile obţinute aşezate în ordine inversă reprezintă cifrele numărului în baza cerută.

Pe baza aspectelor teoretice prezentate anterior, se consideră un număr natural **N** scris în baza **10**. În continuare sunt prezentate algoritmul, schema logică si programul pentru conversia numărului natural din baza **10** în baza **2** - **9**.

- se citeşte numărul natural **N** în baza **10**;
- într-un ciclu cu test final se citeşte baza **b** în care se doreşte conversia (de la 2 la 9), repetându-se operaţia de citire a valorii variabilei **b** atât timp cât se introduce un număr mai mic ca 2 sau mai mare ca 9;
- se iniţializează variabila **Nb** cu **0**, respectiv **fact** cu **1**, adică **Nb := 0, fact := 1**; Variabila **Nb** reprezintă numărul natural **N** scris în baza **b**, iar variabila **fact** se utilizează pentru a plasa fiecare cifră a numărului **Nb** în poziţia corespunzătoare;
- cu ajutorul unui ciclu cu test final, se execută următoarele operaţii, atât timp cât **N > 0**:

- se determină restul împărţirii numărului **N** la baza **b** (notat **rest**), valoarea astfel obţinută reprezintă ultima cifră din numărul **N** scris în baza **b**;

- se modifică valoarea numărului **N** scris în baza **10** prin atribuirea câtului obţinut prin împărţirea numărului **N** la baza **b**;

- se modifică valoarea numărului **N** scris în baza **b**, notat **Nb**, cu ajutorul relaţiei **Nb := Nb + fact * rest**;

- se modifică valoarea variabilei fact cu ajutorul relaţiei **fact := fact * 10**. Valoarea variabilei **rest**, obţinută în fiecare etapă se înmulţeşte cu valoarea curentă a variabilei **fact** astfel se poziţionează cifra numărului în baza **b** pe poziţia corespunzătoare în cadrul numărului scris în baza **b**;

- când valoarea variabilei **N** nu mai este strict pozitivă (devine zero), se părăseşte ciclul şi se continuă cu următoarea secvenţă din schema logică, adică se afişează valoarea numărului natural **N** în baza **b**, adică **Afişează Nb**;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.15a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.15b.

Schema logică	Limbajul pseudocod

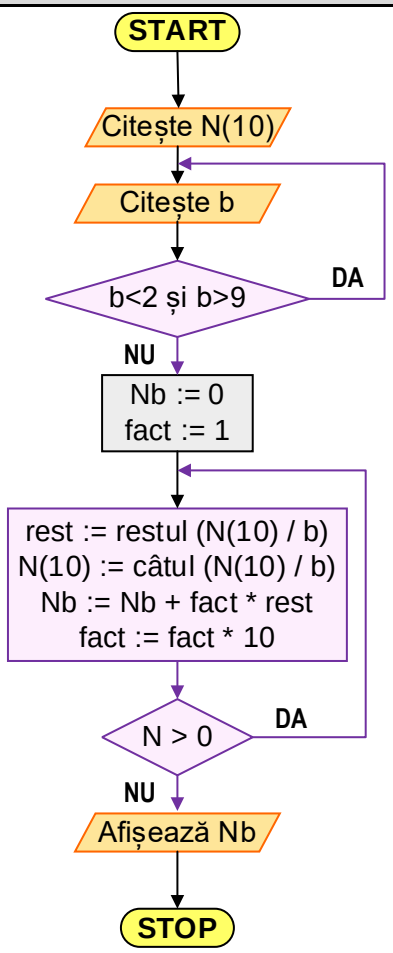
	**Început**
	  Citeşte N
	  NC←N
	  Citeşte b
	  Cât timp b<2 SAU b>9 **execută**
	  Citeşte b
	  Sfârşit cât timp
	  NB←0
	  fact← 1
	  Cât timp N>0 **execută**
	  rest← N MOD b
	  N← N DIV b
	  Nb← Nb+fract*rest
	  fact← fact*10
	  Sfârşit cât timp
	  Scrie Nb
	Sfârşit

Figura 5.15a. Reprezentarea algoritmului pentru calculul conversia unui număr natural din baza 10 în baza 2 - 9

Programul MATLAB	Execuția programului

`s=input('Introduceti numarul N10=','s');`
`N = sscanf(s,'%d');`
`Nc=N;`
`b=input('Introduceti baza b=');`
`while b<2 | b>9`
`  b=input('Introduceti baza b=');`
`end`
`Nb = 0; fact = 1;`
`while N>0`
`  rest=mod(N,b);`
`  N=floor(N/b);`
`  Nb=Nb+fact*rest;`
`  fact=fact*10;`
`end`
`s=sprintf(' N(10)=%d, N(%d)=%d',Nc,b,Nb);`
`disp(s)`
	Execuţia programului – cazul 1:
	
	`Introduceti numarul N10 = 123`
	`Introducet baza, b = 2`
	`N(10) = 123, N(2) = 1111011`
	
	Execuţia programului – cazul 2:
	
	`Introduceti numarul N10 = 123`
	`Introducet baza, b = 8`
	`N(10) = 123, N(8) = 173`

Figura 5.15b. Programul MATLAB şi execuţia acestuia pentru conversia unui număr natural din baza 10 în baza 2 - 9

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Exemplu numeric (tabelul 5.15):

Se consideră **N = 123** şi **b = 2** şi se aplică algoritmul prezentat anterior. Se iniţializează **Nb := 0** şi **fact := 1**.

În prima etapă, se împarte numărul **123** la baza **2**, obţinându-se câtul **61** şi restul **1**. Restul obţinut este ultima cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb := Nb + fact * rest**, adică **Nb := 0 + 1 * 1 = 1**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact := fact * 10 = 1 * 10 = 10**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **123** la baza **2**, adică **61**;

Tabelul 5.15. Conversia unui număr natural din baza 10 în baza 2 - 9

Valori obţinute pentru: **N = 123, b = 2**						
Etapa	**N**	**[N/b]**	**rest**	**Nb = Nb + fact * rest**	**fact**	**Ecran**
0	123			**0**	**1**	
1	123	61	**1**	Nb := 0 + 1 * 1 = 1	10	
2	61	30	**1**	Nb := 1 + 10 * 1 = 11	100	
3	30	15	**0**	Nb := 11 + 100 * 0 = 11	1000	
4	15	7	**1**	Nb := 11 + 1000 * 1 = 1011	10000	
5	7	3	**1**	Nb := 1011 + 10000 * 1 = 11011	100000	
6	3	1	**1**	Nb := 11011 + 100000 * 1 = 111011	1000000	
7	1	0	**1**	Nb := 111011 + 1000000 * 1 = 1111011	10000000	
						1111011

În următoarea etapă, se împarte numărul **61** la baza **2**, obţinându-se câtul **30** şi restul **1**. Restul obţinut este penultima cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb := Nb + fact * rest**, adică **Nb := 1 + 10 * 1 = 11**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact := fact * 10 = 10 * 10 = 100**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **61** la baza **2**, adică **30**;

În următoarea etapă, se împarte numărul **30** la baza **2**, obţinându-se câtul **15** şi restul **0**. Restul obţinut este antepenultima cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb := Nb + fact * rest**, adică **Nb := 11 + 10 * 0 = 11**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact := fact * 10 = 100 * 10 = 1000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **30** la baza **2**, adică **15**;

În următoarea etapă, se împarte numărul **15** la baza **2**, obţinându-se câtul **7** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb := Nb + fact * rest**, adică **Nb := 11 + 1000 * 1 = 1011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact := fact * 10 = 1000 * 10 = 10000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **15** la baza **2**, adică **7**;

În următoarea etapă, se împarte numărul **7** la baza **2**, obţinându-se câtul **3** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb := Nb + fact * rest**, adică **Nb := 1011 + 10000 * 1 = 11011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact := fact * 10 = 10000 * 10 = 100000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **7** la baza **2**, adică **3**;

În următoarea etapă, se împarte numărul **3** la baza **2**, obţinându-se câtul **1** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb := Nb + fact * rest**, adică **Nb := 11011 + 100000 * 1 = 111011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact := fact * 10 = 100000 * 10 = 1000000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **3** la baza **2**, adică **1**;

În următoarea etapă, se împarte numărul **1** la baza **2**, obţinându-se câtul **0** şi restul **1**. Restul obţinut este următoarea cifră a numărului **123** în baza **2**. Numărul scris în baza **2** (notat **Nb**) se calculează cu relaţia **Nb := Nb + fact * rest**, adică **Nb := 111011 + 1000000 * 1 = 1111011**, iar variabila **fact** îşi modifică valoarea pe baza relaţiei: **fact := fact * 10 = 1000000 * 10 = 10000000**. Se continuă algoritmul cu modificarea valorii variabilei **N**, atribuindu-se valoarea câtului obţinut prin împărţirea numărului **1** la baza **2**, adică **0**;

Deoarece **N = 0**, condiţia **N > 0** nu se mai îndeplineşte, se continuă pe ramura **"NU"** a condiţiei **N > 0** şi se execută secvenţa următoare ciclului cu test iniţial, adică **Afişeaza Nb**.

5.16. Reversul unui număr

Se consideră un număr natural **N**. În continuare sunt prezentate algoritmul, schema logică si programul pentru determinarea reversului unui număr. Algoritmul este următorul:

- se citeşte numărul natural **N**;
- se iniţializează numărul revers cu zero, adică **rev := 0**;
- cu ajutorul unui ciclu cu test iniţial, atât timp cât **N > 0**, se realizează următoarele operaţii:
  - se determină ultima cifră (notată **uc**) a numărului **N** prin împărţirea cu rest a acestuia la **10**, restul obţinut fiind ultima cifră a numărului;
  - se formează reversul numărului N cu ajutorul relaţiei: **rev := rev * 10 + uc**;
  - se modifică valoarea numărului natural **N**, adică se elimină ultimă cifră, obţinându-se un număr mai scurt cu o cifră. Acest lucru se realizează prin atribuirea către variabila **N** a valorii câtului împărţirii numărului natural **N** la **10**;
  - se revine la evaluarea valorii numărului natural **N** nou obţinut prin revenirea la condiţia **N > 0**;
- când valoarea variabilei **N** nu mai este strict pozitivă (devine zero), se părăseşte ciclul şi se continuă cu următoarea secvenţă din schema logică, adică se afişează valoarea numărului revers, adică **Afişează rev**;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.16a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.16b.

Schema logică	Limbajul pseudocod

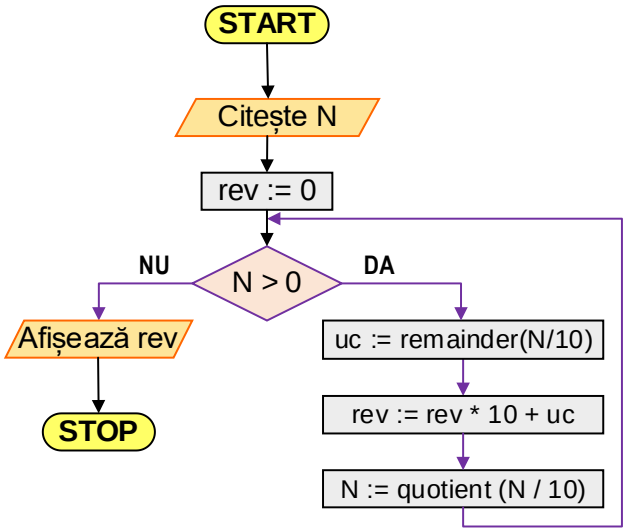
	**Început**
	  Citeşte N
	  rev←0
	  Cât timp N>0 **execută**
	  uc← N MOD 10
	  rev← rev*10+uc
	  N← N DIV 10
	  Sfârşit cât timp
	  Scrie rev
	Sfârşit

Figura 5.16a. Reprezentarea algoritmului pentru formarea numărului invers

Programul MATLAB	Execuția programului

`s=input('Introduceti numarul N=','s');`
`N = sscanf(s,'%d');`
`rev=0;`
`while N>0`
`  uc=mod(N,10);`
`  rev=rev*10+uc;`
`  N=floor(N/10);`
`end`
`s=sprintf(' Inversul este %d',rev);`
`disp(s)`
	`Introduceti numarul N = 12345`
	`Inversul este 54321`

Figura 5.16b. Programul MATLAB şi execuţia acestuia pentru formarea numărului invers

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Exemplu numeric (tabelul 5.16):

Se consideră **N = 12345** şi se aplică algoritmul prezentat anterior.

Se iniţializează **rev = 0**.

În prima etapă, se obţine ultima cifră a numărului **12345**, prin împărţirea cu rest a numărului **12345** la **10**. Rezultatul obţinut este **5** şi se modifică valoarea variabilei **rev** conform relaţiei: **rev:=rev*10+uc**, deci **rev := 0 * 10 + 5 = 5**. În continuare, se modifică valoarea numărului **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1234**, deci **N = 1234**;

Tabelul 5.16. Determinarea reversului unui număr natural

Valori obţinute pentru: **N = 12345**					
Etapa	**N**	**uc**	**rev**	**[N/10]**	**Ecran**
0	12345		0		
1	12345	5	rev := 0 * 10 + **5** = 5	1234	
2	1234	4	rev := 5 * 10 + **4** = 54	123	
3	123	3	rev := 54 * 10 + **3** = 543	12	
4	12	2	rev := 543 * 10 + **2** = 5432	1	
5	1	1	rev := 5432 * 10 + **1** = 54321	0	
					54321

În a doua etapă, se obţine ultima cifră a numărului natural **N** (**N = 1234**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **4**. Se modifică valoarea variabilei **rev** conform relaţiei:

rev:=rev*10+uc, deci **rev := 5 * 10 + 4 = 54**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **123**, deci **N = 123**;

În a treia etapă, se obţine ultima cifră a numărului natural **N** (**N = 123**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **3**. Se modifică valoarea variabilei **rev** conform relaţiei: **rev:=rev*10+uc**, deci **rev := 54 * 10 + 3 = 543**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **12**, deci **N = 12**;

În a patra etapă, se obţine ultima cifră a numărului natural **N** (**N = 12**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **2**. Se modifică valoarea variabilei **rev** conform relaţiei: **rev:=rev*10+uc**, deci **rev := 543 * 10 + 2 = 5432**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1**, deci **N = 1**;

În a cincea etapă, se obţine ultima cifră a numărului natural **N** (**N = 1**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **1**. Se modifică valoarea variabilei **rev** conform relaţiei: **rev:=rev*10+uc**, deci **rev := 5432 * 10 + 1 = 54321**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **0**, deci **N = 0**;

Deoarece condiţia **N > 0** nu se mai îndeplineşte, se continuă pe ramura **"NU"** a condiţiei **N > 0** şi se execută secvenţa următoare ciclului cu test iniţial, adică **Afişare rev**.

5.17. Calculul sumei cifrelor unui număr natural

Se consideră un număr natural **N**. În continuare sunt prezentate algoritmul, schema logică si programul pentru calculul sumei cifrelor numărului.

Algoritmul este următorul:

- se citeşte numărul natural **N**;
- se iniţializează suma cifrelor **S** cu zero, adică **S := 0**;
- cu ajutorul unui ciclu cu test iniţial, atât timp cât **N > 0**, se realizează următoarele operaţii:
  - se determină ultima cifră (notată **uc**) a numărului **N** prin împărţirea cu rest a acestuia la **10**, restul obţinut fiind ultima cifră a numărului;
  - se adaugă valoarea ultimei cifre la sumă: **S := S + uc**;
  - se modifică valoarea numărului natural **N**, adică se elimină ultimă cifră, obţinându-se un număr mai scurt cu o cifră. Acest lucru se realizează prin atribuirea către variabila **N** a valorii câtului împărţirii numărului natural **N** la **10**;
  - se revine la evaluarea valorii numărului natural **N** nou obţinut prin revenirea la condiţia **N > 0**;
- când valoarea variabilei **N** nu mai este strict pozitivă (devine zero), se părăseşte ciclul şi se continuă cu următoarea secvenţă din schema logică, adică se afişează valoarea calculată a sumei, adică **Afişează S**;

Exemplu numeric (tabelul 5.17):

Se consideră **N = 12345** şi se aplică algoritmul prezentat anterior. Se iniţializează **S = 0**.

În prima etapă, se obţine ultima cifră a numărului **12345**, prin împărţirea cu rest a numărului **12345** la **10**. Rezultatul obţinut este **5** şi se adună la valoarea sumei **S** a cărui valoare iniţială este **0**, deci **S := 0 + 5 = 5**. În continuare, se modifică valoarea numărului **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1234**, deci **N = 1234**;

Tabelul 5.17. Calculul sumei cifrelor unui număr natural

Valori obţinute pentru: **N = 12345**					
Etapa	**N**	**uc**	**S**	**[N / 10]**	**Ecran**
0	12345		0		
1	12345	5	S = 0 + **5** = 5	1234	
2	1234	4	S = 5 + **4** = 9	123	
3	123	3	S = 9 + **3** = 12	12	
4	12	2	S = 12 + **2** = 14	1	
5	1	1	S = 14 + **1** = 15	0	
					15

În a doua etapă, se obţine ultima cifră a numărului natural **N** (**N = 1234**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **4**. Se adaugă această valoare la suma **S**, deci **S := 5 + 4**, adică **S = 9**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **123**, deci **N = 123**;

În a treia etapă, se obţine ultima cifră a numărului natural **N** (**N = 123**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **3**. Se adaugă această valoare la suma **S**, deci **S := 9 + 3**, adică **S = 12**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **12**, deci **N = 12**;

În a patra etapă, se obţine ultima cifră a numărului natural **N** (**N = 12**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **2**. Se adaugă această valoare la suma **S**, deci **S := 12 + 2**, adică **S = 14**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **1**, deci **N = 1**;

În a cincea etapă, se obţine ultima cifră a numărului natural **N** (**N = 1**) prin împărţirea acestuia la **10** cu rest. Valoarea obţinută este ultima cifră a numărului, adică **1**. Se adaugă această valoare la suma **S**, deci **S := 14 + 1**, adică **S = 15**. Se modifică valoarea lui **N** prin atribuirea rezultatului împărţirii întregi (a câtului) a lui **N** la **10**, adică **0**, deci **N = 0**;

Deoarece condiţia **N > 0** nu se mai îndeplineşte, se continuă pe ramura **"NU"** a condiţiei **N > 0** şi se execută secvenţa următoare ciclului cu test iniţial, adică **Afişare S**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.17a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.17b.

Schema logică	Limbajul pseudocod

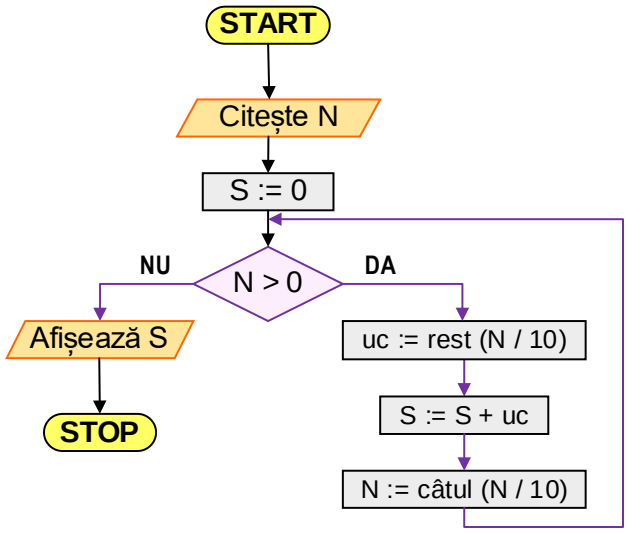
	**Început**
	  Citeşte N
	  S←0
	  Cât timp N>0 **execută**
	  uc← N MOD 10
	  S← S+uc
	  N← N DIV 10
	  Sfârşit cât timp
	  Scrie S
	Sfârşit

Figura 5.17a. Reprezentarea algoritmului pentru calculul sumei cifrelor unui număr natural

Programul MATLAB	Execuția programului

`s=input('Introduceti numarul N=','s');`
`N = sscanf(s,'%d');`
`S=0;`
`while N>0`
`  uc=mod(N,10);`
`  S=S+uc;`
`  N=floor(N/10);`
`end`
`s=sprintf(' Suma cifrelor este %d',S);`
`disp(s)`
	`Introduceti numarul N = 12345`
	`Suma cifrelor este 15`

Figura 5.17b. Programul MATLAB şi execuţia acestuia pentru calculul sumei cifrelor unui număr natural

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.18. Verificarea CNP-ului (Codul Numeric Personal)

Codul numeric personal (CNP) este un cod numeric de 13 cifre, unic, atribuit fiecărei persoane născute în România. Codul numeric personal se atribuie fiecărei persoane la naştere şi este specificat în certificatul de naştere, actele de identitate (buletin sau carte de identitate) cât şi în permisul de conducere. Codul numeric personal a fost introdus ca element obligatoriu printr-un decret prezidenţial în 2 martie 1978. Codul numeric personal conţine 13 cifre, astfel:

S	A	A	L	L	Z	Z	J	J	N	N	N	C

S : primă cifră defineşte sexul (sex bărbătesc / sex femeiesc), astfel:

1 / 2 : născuți între 1 ianuarie 1900 și 31 decembrie 1999;

3 / 4 - născuți între 1 ianuarie 1800 și 31 decembrie 1899;

5 / 6 - născuți între 1 ianuarie 2000 și 31 decembrie 2099;

7 / 8 - pentru persoanele străine rezidente în România;

AA : număr format din două cifre care reprezintă ultimele două cifre din anul naşterii persoanei;

LL : număr format din două cifre care reprezintă luna naşterii persoanei;

ZZ : reprezintă ziua naşterii persoanei. Pentru zilele de la 1 la 9 se adaugă un 0 înaintea zilei;

JJ : reprezintă un număr format din două cifre care reprezintă codul judeţului sau sectorului (pentru municipiul Bucureşti), astfel: 01–Alba, 02–Arad, 03–Argeş, 04–Bacău, 05–Bihor, 06–Bistriţa-Năsăud, 07–Botoşani, 08–Braşov, 09–Brăila, 10–Buzău, 11–Caraş-Severin, 12–Cluj, 13–Constanţa, 14–Covasna, 15–Dâmboviţa, 16–Dolj, 17–Galaţi, 18–Gorj, 19–Harghita, 20–Hunedoara, 21–Ialomiţa, 22–Iaşi, 23–Ilfov, 24–Maramureş, 25–Mehedinţi, 26–Mureş, 27–Neamţ, 28–Olt, 29–Prahova, 30–Satu Mare, 31–Sălaj, 32-Sibiu, 33–Suceava, 34–Teleorman, 35–Timiş, 36–Tulcea, 37–Vaslui, 38–Vâlcea, 39–Vrancea, 40–Bucureşti, 41:46 – Bucureşti – Sectorul 1:6, 51–Călăraşi, 52–Giurgiu.

NNN : număr format din trei cifre (din intervalul 001 – 999), care se împart în judeţ birourilor de Evidenţă a Persoanei, astfel încât un anumit număr din interval să fie alocat unei singure persoane, într-o anumită zi.

C : cifră de control, aflată în relaţie cu celelalte cifre ale CNP-ului. Valoarea cifrei de control se calculează astfel: fiecare cifră din primele 12 cifre ale CNP-ului se înmulţeşte cu fiecare cifră de pe aceeaşi poziţie din numărul '279146358279', rezultatele se însumează, valoarea obţinută se împarte cu rest la 11. Dacă restul este 10 acesta se consideră ca fiind 1. Dacă cifra de control este egală cu restul obţinut anterior atunci CNP este valid, în caz contrar CNP este invalid.

Algoritmul este următorul:

1. Cu ajutorul unui ciclu cu contor se citesc cele **13** cifre ale **CNP**;

2. Se iniţializează suma **S** cu valoarea **0**;

3. Cu ajutorul unui ciclu cu contor, în care contorul ia valori de la **0** la **11** (adică se utilizează primele **12** cifre ale **CNP**), se calculează suma, cu ajutorul relaţiei **S := S + cnp[i]*sirc[i]**;

4. Se împarte cu rest suma obţinută anterior la **11**. Dacă restul obţinut este **10**, atunci variabila control primeşte valoarea 1, altfel variabila control are valoarea restului împărţirii sumei la **11**;

5. Dacă valoarea variabilei control este egală cu ultima cifră a **CNP**-ului atunci acesta este valid, în caz contrar **CNP**-ul este invalid.

Exemplul numeric:

1. Se citeşte un CNP cu valoarea **1570330121114**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12
CNP	**1**	**5**	**7**	**0**	**3**	**3**	**0**	**1**	**2**	**1**	**1**	**1**	**4**
Şir de verificare	2	7	9	1	4	6	3	5	8	2	7	9	
Produse parţiale	2	35	63	0	12	18	0	5	16	2	7	9	
Suma produselor parţiale (S) =			**169**		Restul împărţirii sumei la 11 (S%11) =								**4**

În acest caz cifra de control al CNP-ului (ultima cifră) este egală cu restul obţinut prin împărţirea sumei la 11, deci CNP-ul este valid.

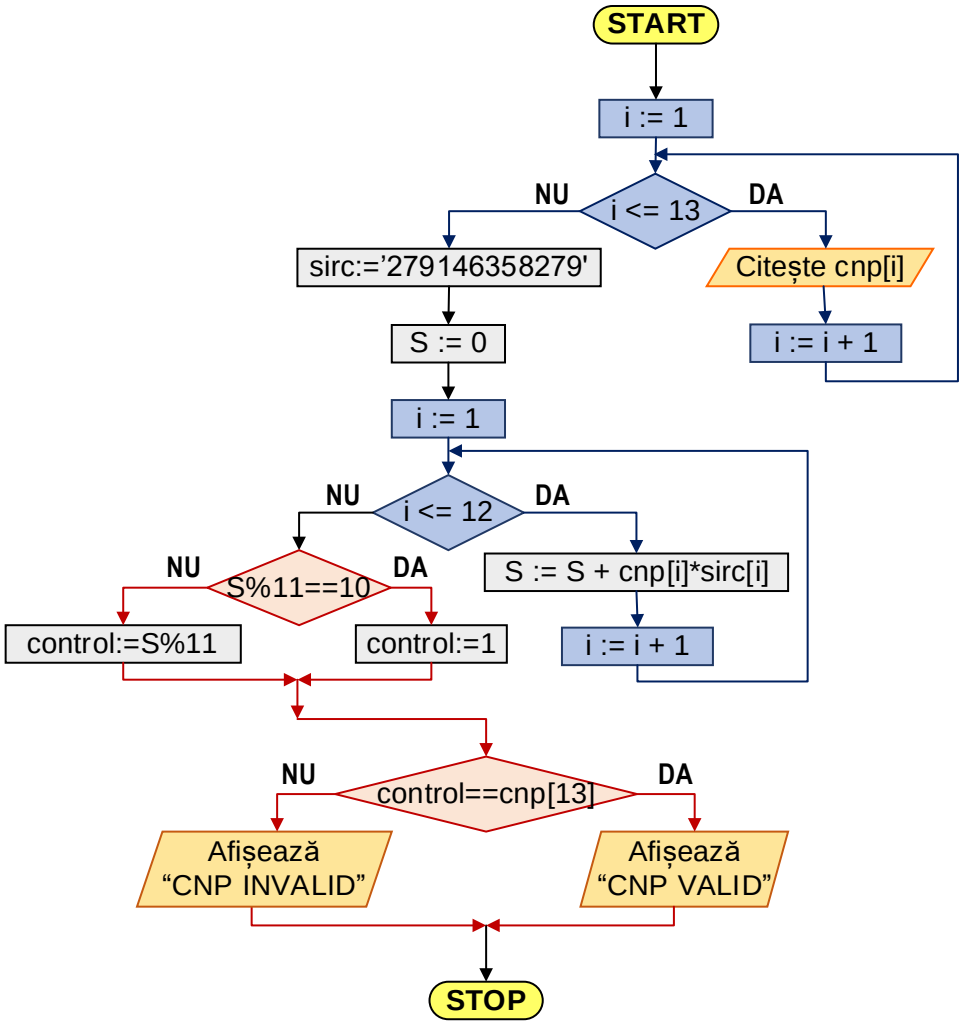
2. Se citeşte un CNP cu valoarea **1570330121111**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la **11**.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12
CNP	**1**	**5**	**7**	**0**	**3**	**3**	**0**	**1**	**2**	**1**	**1**	**1**	**1**
Şir de verificare	2	7	9	1	4	6	3	5	8	2	7	9	
Produse parţiale	2	35	63	0	12	18	0	5	16	2	7	9	
Suma produselor parţiale (S) =			**169**		Restul împărţirii sumei la 11 (S%11) =								**4**

În acest caz cifra de control al CNP-ului nu este egală cu restul obţinut prin împărţirea sumei la 11, deci CNP nu este valid.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.18a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.18b.

Schema logică



Pseudocod

```
Început
  Pentru i←1,13
      Citeşte cnpi
  Sfârşit pentru
  sirc← [ 2,7,9,1,4,6,3,5,8,2,7,9];
  S←0
  Pentru i←1,12
      S←S+cnpi *sirci
  Sfârşit pentru
  Dacă S MOD 11=10 atunci
       control←1
   altfel
       control←S MOD 11
  Sfârşit dacă
  Dacă control=cnp13 atunci
       Scrie „CNP VALID”
   altfel
       Scrie „CNP INVALID”
  Sfârşit dacă
Sfârşit
```

Figura 5.18a. Reprezentarea algoritmului pentru verificarea CNP-ului

Programul MATLAB	Execuția programului

`sirc=[ 2,7,9,1,4,6,3,5,8,2,7,9];`
`cnp=input('Introduceti CNP-ul: ','s');`
`S=0;`
`for i=1:12`
`  cifra=str2num(cnp(i));`
`  S=S+cifra*sirc(i);`
`end`
`if mod(S,11)==10`
`  control=1;`
`else`
`  control=mod(S,11);`
`end`
`if control ==str2num(cnp(13))`
`  disp('CNP valid!');`
`else`
`  disp('CNP invalid!');`
`end`
	`Introduceti CNP-ul:`
	`1570330121114`
	`CNP valid!`
	
	`Introduceti CNP-ul:`
	`1570330121111`
	`CNP invalid!`

Figura 5.18b. Programul MATLAB şi execuţia acestuia pentru verificarea CNP-ului

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

5.19. Verificarea codului de pe card

Majoritatea cardurilor bancare sunt identificate printr-un număr de 16 cifre. În ordine, de la stânga la dreapta acestea reprezintă:

S : reprezintă tipul cardului (4 – VISA, 5 – MasterCard) sau Major Industry Identifier (MII);
BBBBB : reprezintă codul băncii emitente;
NN ... N : reprezintă numărul contului;
C : reprezintă cifra de control.

Verificarea codului de pe card se realizează conform algoritmului lui Luhn:

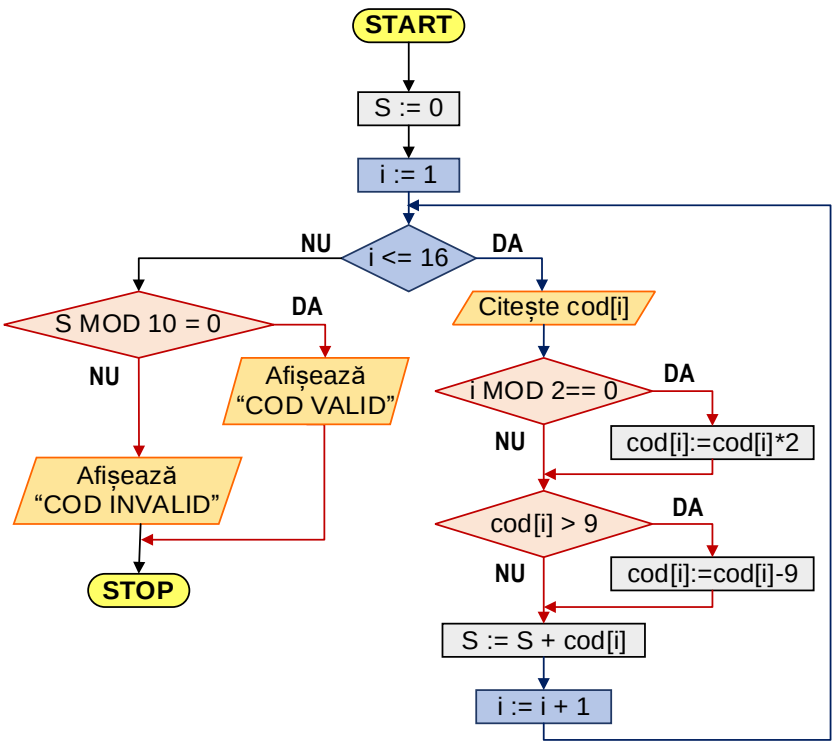
Pas 1: Se înmulţeşte fiecare cifră din codul de card cu ponderea sa. Dacă un card are un număr par de cifre, prima cifră are ponderea 2, dacă nu, cifra are ponderea 1. În continuare, ponderile cifrelor alternează 1,2,1,2;
Pas 2: Dacă o cifră are o valoare ponderată mai mare decât 9, se scade 9 din valoarea ei;
Pas 3: Se adună toate valorile ponderate pentru primele 15 cifre şi se calculează restul împărţirii la 10 (MODULO 10);
Pas 4: Un cod de card este valid dacă rezultatul operatiei MODULO 10 este egal cu valoarea cifrei de control **C**.

Algoritmul este următorul:

1. Cu ajutorul unui ciclu cu contor se citesc cele 16 cifre ale codului;
2. Se iniţializează suma S cu valoarea 0;
3. Cu ajutorul unui ciclu cu contor, în care contorul ia valori de la 0 la 15, se extrage fiecare cifră din cod, se înmulţeşte cu ponderea, iar dacă valoarea obţinută este mai mare decât 9, se scade 9 din valoarea obţinută. Cu noua valoare se calculează suma, cu ajutorul relaţiei **S := S + cod[i]**.
4. Se împarte cu rest suma obţinută anterior la 10;
5. Dacă restul împărţirii este egal cu zero atunci codul este valid, în caz contrar codul este invalid.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.19a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.19b.

Schema logică



Pseudocod

```
Început
  S←0
  Pentru i←1,16
      Citeşte cod_i
      Dacă i+1 MOD 2=0 atunci
        cod_i ← cod_i*2
      Sfârşit dacă
      Dacă cod_i>9 atunci
        cod_i ← cod_i-9
      Sfârşit dacă
      S← S+cod_i
  Sfârşit pentru
  Dacă S MOD 10=0 atunci
       Scrie „COD VALID”
   altfel
       Scrie „COD INVALID”
  Sfârşit dacă
Sfârşit
```

Figura 5.19a. Reprezentarea algoritmului pentru verificarea codului de pe card

Programul MATLAB	Execuția programului

`disp('Introduceti codul: ');`
`S=0;`
`for i=1:16`
`  cod(i)=input(' ');`
`  if mod(i+1,2)==0`
`  cod(i)=2*cod(i);`
`  end`
`  if cod(i)>9`
`  cod(i)=cod(i)-9;`
`  end`
`  S=S+cod(i);`
`end`
`if mod(S,10)==0`
`  disp('Cod valid!');`
`else`
`  disp('Cod invalid!');`
`end`
	`Introduceti codul:` **`5123456789123457`**
	`Cod valid!`
	
	`Introduceti codul:` **`4123456789123455`**
	`Cod invalid!`

Figura 5.19b. Programul MATLAB şi execuţia acestuia pentru verificarea codului de pe card

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Exemplul numeric:

1. Se citeşte un cod cu valoarea **5123456789123457**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cod card	**5**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1**	**2**	**3**	**4**	**5**	**7**
Ponderea	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1
Produs parţial	10	1	4	3	8	5	12	7	16	9	2	2	6	4	10	7
Produs parţial recalculat	1	1	4	3	8	5	3	7	7	9	2	2	6	4	1	7
Suma produselor parţiale (S)	**70**			Restul împărţirii sumei la 10 (S%10) =												**0**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 10, deci codul este valid.

2. Se citeşte un cod cu valoarea **4123456789123455**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Cod card	**4**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**1**	**2**	**3**	**4**	**5**	**5**
Ponderea	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1
Produs parţial	8	1	4	3	8	5	12	7	16	9	2	2	6	4	10	5
Produs parţial recalculat	8	1	4	3	8	5	3	7	7	9	2	2	6	4	1	5
Suma produselor parţiale (S)	**75**			Restul împărţirii sumei la 10 (S%10) =												**5**

În acest caz cifra de control al codului nu este egală cu restul obţinut prin împărţirea sumei la 10, deci codul este invalid.

5.20. Verificarea codului ISBN

Codul **ISBN** (**I**nternational **S**tandard **B**ook **N**umber) este un cod internaţional de identificare a cărţilor, definit prin ISO 2108. Este format din 13 cifre grupate în 5 segmente de lungimi variabile, separate de cratimă, cu următoarele specificaţii:

- prefixul 978 : reprezintă producţia editorială de carte la nivel internaţional;
- codul de ţară : indică grupul naţional, lingvistic sau geografic şi desemnează limba editorului şi nu limba în care este publicată cartea. Pentru România se utilizează 973 sau 606;
- codul editurii : identifică editorul documentului, lungimea acestuia variază în funcţie de numărul lucrărilor publicate de editor;
- numărul de identificare a documentului : se utilizează pentru numerotarea documentului printre publicaţiile editorului;
- cifra de control : reprezintă ultima cifră a codului ISBN cu ajutorul căreia se verifică validitatea codului ISBN;

Există două categorii de coduri ISBN, ISBN-10 mai vechi care nu conţine prefixul 978 şi codul ISBN-13.

A. Algoritmul de validare al unui cod **ISBN-10**:

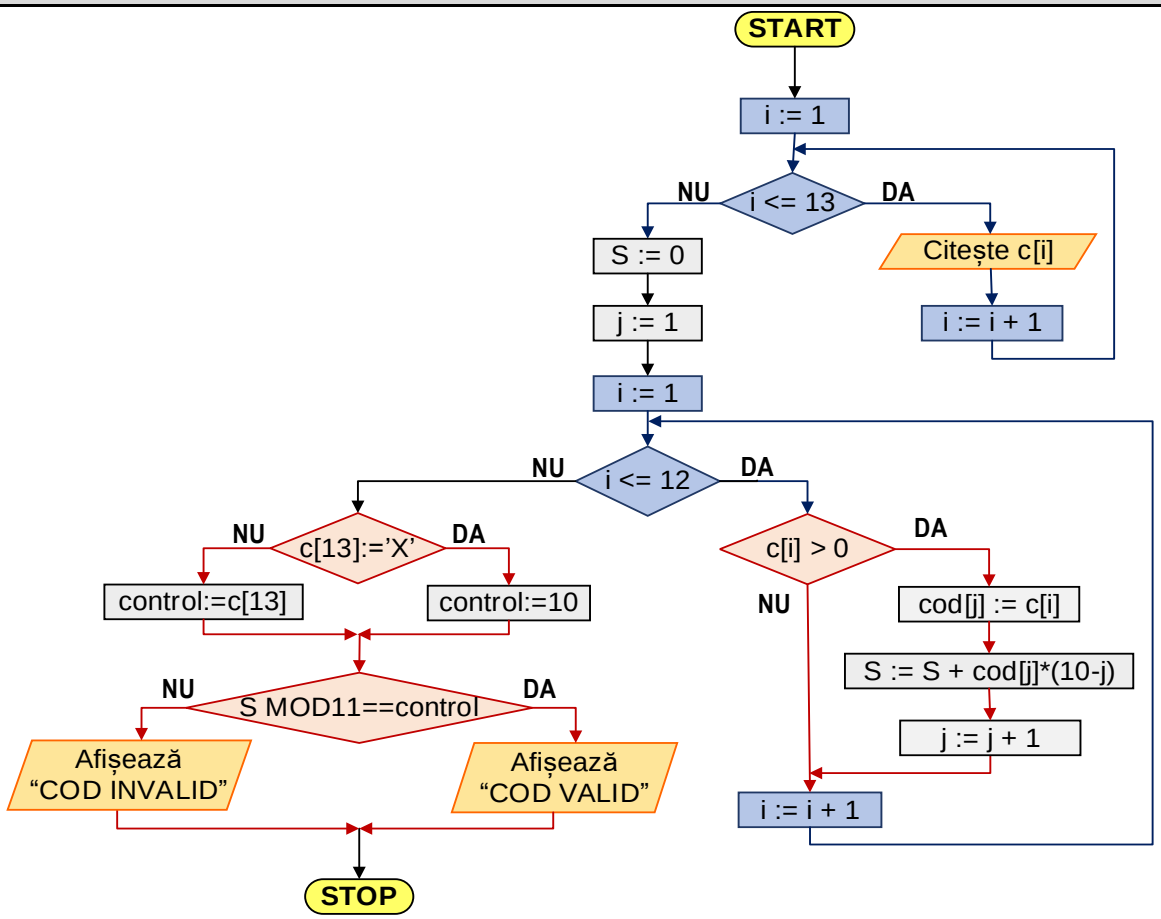
A.1. Se citeşte codul. Se elimină spaţiile şi cratimele. Ultimul caracter se ignoră;

A.2. Se înmulţeşte fiecare cifră cu ponderea asociată ei, ponderea este dată sub forma 11 – poziţia cifrei. Se adună valorile obţinute;

A.3. Se determină valoarea variabilei **control**, astfel: dacă caracterul de control este ‚X', atunci valoarea variabilei **control** este 10, altfel valoarea variabilei **control** este egală cu valoarea cifrei de control;

A.4. Se verifică validitatea codului comparând restul împărţirii sumei obţinute la 11 (modulo 11) cu cifra de control. Dacă cele două valori sunt egale, codul este valid, iar dacă nu sunt egale, codul este invalid.

Schema logică



Pseudocod

```
Început
  Pentru i←1,13
      Citeşte ci
  Sfârşit pentru
  S←0,   j←1
  Pentru i←1,12
    Dacă ci >0 atunci
        codj ← ci,  S← S+codj *(11-j),   j← j+1
    Sfârşit dacă
  Sfârşit pentru
  Dacă c13='X' atunci
      control←10
   altfel
       control←c(13)
  Sfârşit dacă
  Dacă S MOD 11=control atunci
       Scrie „COD VALID”
   altfel
       Scrie „COD INVALID”
  Sfârşit dacă
Sfârşit
```

Figura 5.20a. Reprezentarea algoritmului pentru verificarea codului **ISBN-10**

Programul MATLAB	Execuția programului

```
c=input('Introduceti codul: ','s');
S=0; j=1;
for i=1:12
    if c(i)-'0'>=0
       cod(j)=str2num(c(i));
       S=S+cod(j)*(11-j); j=j+1;
    end
end
if c(13)=='X'     control=10;
else     control=c(13)-'0';
end
if mod(S,11)==control
   disp('Cod valid!');
else
   disp('Cod invalid!');
end
```

```
Introduceti codul: 973-356-432-1
COD VALID!!!

Introduceti codul: 973-456-432-5
COD INVALID!!!
```

Figura 5.20b. Programul MATLAB şi execuţia acestuia pentru verificarea codului **ISBN-10**

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.20a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.20b.

Exemplul numeric:

1. Se citeşte un cod cu valoarea **973-356-432-1**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**3**	**5**	**6**	**4**	**3**	**2**	**1**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	21	30	30	16	9	4	
Suma produselor parţiale S =	**287**		Restul împărţirii sumei S%11 =							**1**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **valid**.

2. Se citeşte un cod cu valoarea **973-456-432-5**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**4**	**5**	**6**	**4**	**3**	**2**	**5**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	28	30	30	16	9	4	
Suma produselor parţiale S =	**294**		Restul împărţirii sumei S%11 =							**8**

În acest caz cifra de control al codului nu este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **invalid**.

3. Se citeşte un cod cu valoarea **973-456-433-X**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**4**	**5**	**6**	**4**	**3**	**3**	**X**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	28	30	30	16	9	6	
Suma produselor parţiale S =	**296**		Restul împărţirii sumei S%11 =							**10**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **valid**.

4. Se citeşte un cod cu valoarea **973-456-439-0**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 11.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10
Cod ISBN-10	**9**	**7**	**3**	**4**	**5**	**6**	**4**	**3**	**9**	**0**
Ponderea	10	9	8	7	6	5	4	3	2	
Produs parţial	90	63	24	28	30	30	16	9	18	
Suma produselor parţiale S =	**308**		Restul împărţirii sumei S%11 =							**0**

În acest caz cifra de control al codului este egală cu restul obţinut prin împărţirea sumei la 11, deci codul este **valid**.

B. Algoritmul de validare al unui cod **ISBN-13**:

B.1. Se citeşte codul. Se elimină spaţiile şi cratimele. Ultimul caracter se ignoră;

B.2. Se înmulţeşte fiecare cifră cu ponderea asociată ei, ponderea se atribuie pentru fiecare cifră, începând cu prima cifră, sub forma 1,3,1,3.... şi se adună valorile obţinute;

B.3. Se împarte suma obţinută la 10 şi se extrage restul împărţirii (modulo 10);

B.4. Dacă restul este 0 atunci variabila **control** trebuie să fie 0. Dacă restul este diferit de 0, atunci variabila **control** se obţine scăzând din 10 restul obţinut;

B.4. Se verifică validitatea codului comparând variabila **control** cu cifra de control, astfel: pentru un ISBN-13 valid variabila **control** rezultată va trebui să fie egală cu ultima cifră a codului (cifra 13).

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 5.20c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 5.20d.

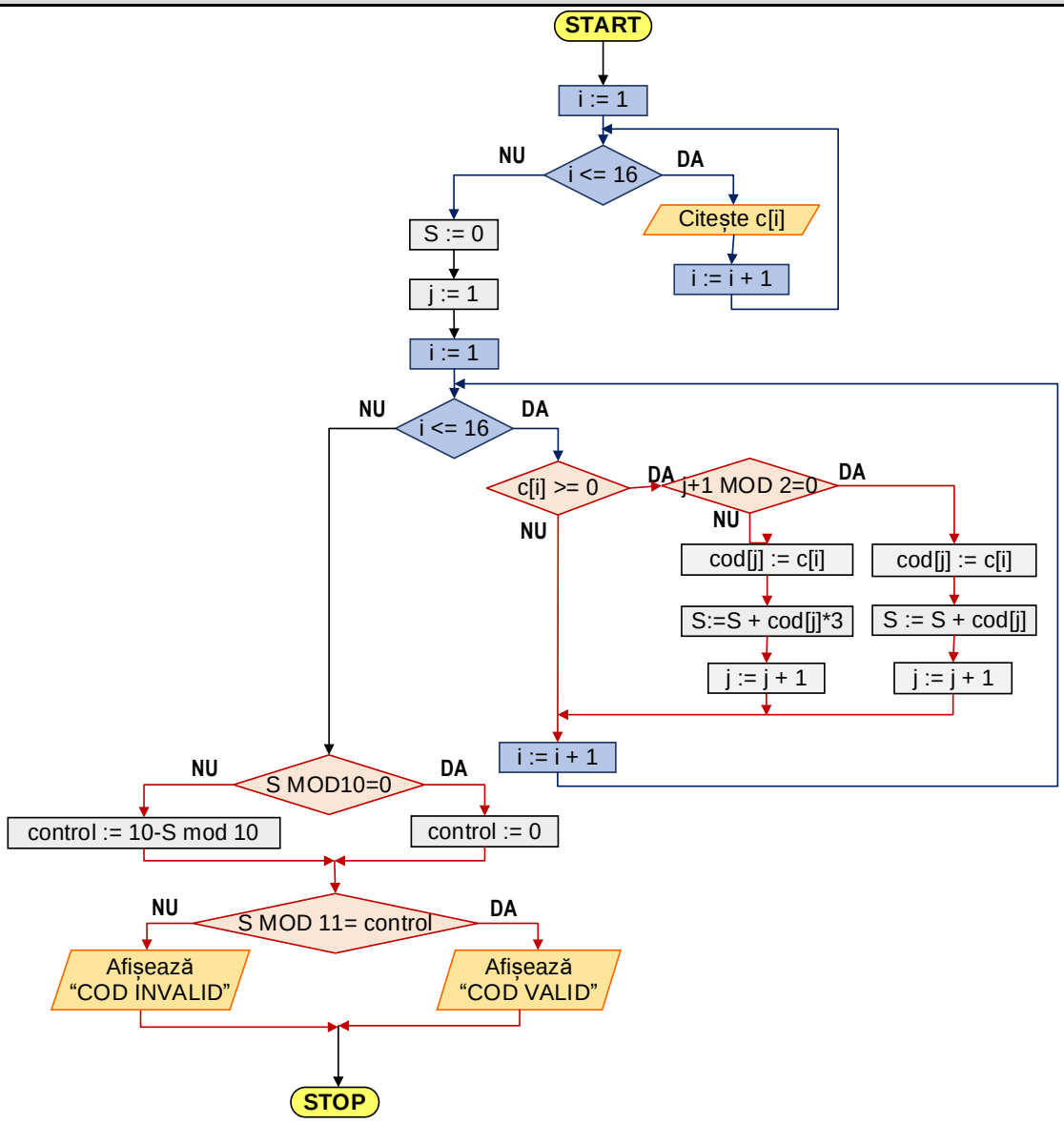
Pseudocod

```
Început
  Pentru i←1,16
      Citeşte ci
  Sfârşit pentru
  S←0,   j←1
 Pentru i←1,16
   Dacă ci≥0 atunci
    Dacă j+1 MOD 2=0 atunci
       codj← ci;  S←S+codj;   j←j+1;
    altfel
       codj←ci;  S←S+codj*3;  j←j+1;
    Sfârşit dacă
  Sfârşit dacă
 Sfârşit pentru
 Dacă S MOD 10=0 atunci    control←0
  altfel     control←10-S MOD 10
 Sfârşit dacă
  Dacă S MOD 11=control atunci
```

Scrie „COD VALID”
altfel
Scrie „COD INVALID”
Sfârşit dacă
Sfârşit

Figura 5.20c. Schema logică pentru verificarea codului **ISBN-13**

Schema logică



Programul MATLAB şi execuţia acestuia

```
c=input('Introduceti codul: ','s');
S=0; j=1;
for i=1:16
    if c(i)-'0'>=0
        if mod(j+1,2)==0
            cod(j)=c(i)-'0';
            S=S+cod(j);  j=j+1;
        else
```

```
Execuţia programului:
Introduceti codul: 978-606-737-208-3.
COD VALID!!!

Introduceti codul: 978-973-755-835-0
          COD VALID!!!
```

```
            cod(j)=c(i)-'0';
            S=S+cod(j)*3;    j=j+1;
        end
    end
end
if mod(S,10)==0
        control=0;
else
        control=10-mod(S,10);
end
if control==(c(17)-'0')
        disp('Cod valid!');
else    disp('Cod invalid!');
end
```

Figura 5.20d. Programul MATLAB şi execuţia acestuia pentru verificarea codului **ISBN-13**

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Exemplul numeric:

1. Se citeşte un cod cu valoarea **978-606-737-208-3**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 10.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10	11	12	13
Cod ISBN-13	**9**	**7**	**8**	**6**	**0**	**6**	**7**	**3**	**7**	**2**	**0**	**8**	**3**
Ponderea	1	3	1	3	1	3	1	3	1	3	1	3	
Produs parţial	9	21	8	18	0	18	7	9	7	6	0	24	
Suma produselor parţiale S =	**127**					Restul împărţirii sumei S%10 =							**7**

În acest caz cifra de control al codului este egală cu diferenţa dintre 10 şi restul obţinut, deci codul este **valid**.

2. Se citeşte un cod cu valoarea **978-973-755-835-0**. În tabelul următor se ilustrează modul de calcul al sumei şi se determină restul împărţirii sumei la 10.

Poziţia din şir (indice)	1	2	3	4	5	6	7	8	9	10	11	12	13
Cod ISBN-13	**9**	**7**	**8**	**9**	**7**	**3**	**7**	**5**	**5**	**8**	**3**	**5**	**0**
Ponderea	1	3	1	3	1	3	1	3	1	3	1	3	
Produs parţial	9	21	8	27	7	9	7	15	5	24	3	15	
Suma produselor parţiale S =	**150**					Restul împărţirii sumei S%10 =							**0**

În acest caz restul obţinut este 0 iar cifra de control al codului este tot 0, deci codul este **valid**.

Capitolul 6. Operaţii cu tablouri bidimensionale - matrice

Matricea reprezintă o colecţie omogenă şi bidimensională de elemente, care pot fi accesate prin intermediul a doi indici, numerotaţi în MATLAB începând de la 1.

Reprezentarea matricelor se realizează printr-un tabel dreptunghiular de valori:

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

Elementul general al matricei se notează cu a_{ij}, indicele „ **i** ” arată linia pe care se află elementul, iar al doilea indice, „ **j** ”, arată coloana pe care se află elementul.

6.1. Introducerea / afişarea elementelor unui tablou bidimensional

În acest exemplu sunt prezentate operaţiile de introducere (citire), respectiv afişare (scriere) a elementelor unui tablou bidimensional cu „ **m** x **n** ” elemente, numărul de linii şi numărul de coloane fiind introduse de la tastatură.

Parcurgerea matricelor se realizează linie cu linie, astfel că pentru parcurgerea liniilor se utilizează un ciclu cu contor, contorul fiind notat cu „ **i** ”, acesta luând valori de la **1** la **m**, iar pentru parcurgerea fiecărei linii se utilizează un al doilea ciclu cu contor (în corpul primului ciclu cu contor), contorul fiind notat cu „ **j** ”, acesta luând valori de la **1** la **n**.

Algoritmul de parcurgere a unei matrice este următorul:

1. se citeşte **m** - numărul de linii ale tabloului bidimensional, respectiv **n** - numărul de coloane ale tabloului bidimensional;

2. se iniţializează contorul „ **i** ” cu valoarea iniţială **1**;

3. se evaluează condiţia „ **i <= m** ” prin care se verifică dacă valoarea curentă a contorului este mai mică decât valoarea numărului de linii. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ” (pasul 3.1), iar dacă condiţia este **falsă** se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează după acest ciclu cu contor;

Pentru fiecare valoare a lui „ **i** ” (adică pentru fiecare linie) se realizează parcurgerea liniei astfel:

3.1. se iniţializează contorul „ **j** ” cu valoarea iniţială **1**;

3.2. se evaluează condiţia „ **j <= n** ”. Dacă condiţia este **adevărată** se continuă pe ramura „ **DA** ” (pasul 3.3), iar dacă condiţia este **falsă** se continuă pe ramura „ **NU** ”, se părăseşte corpul ciclului şi se continuă execuţia cu secvenţa care urmează, adică pasul **4**;

3.3. se execută corpul ciclului condus de variabila „ **j** ”, în care se realizează operaţia de citire sau scriere a elementelor matricei;

3.4. se execută instrucţiunea prin care se modifică valoarea contorului „ **j** ” cu pasul, adică **j := j + 1**. După executarea acestei operaţii se revine la pasul **3**;

4. se execută instrucţiunea prin care se modifică valoarea contorului „ **i** ” cu pasul, adică **i := i + 1**. După executarea acestei operaţii se revine la pasul **3**;

În exemplul următor este prezentată modalitatea de citire a elementelor unei matrice, respectiv modalitatea de afişare a elementelor unei matrice.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.1a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.1b.

Schema logică	Limbajul pseudocod

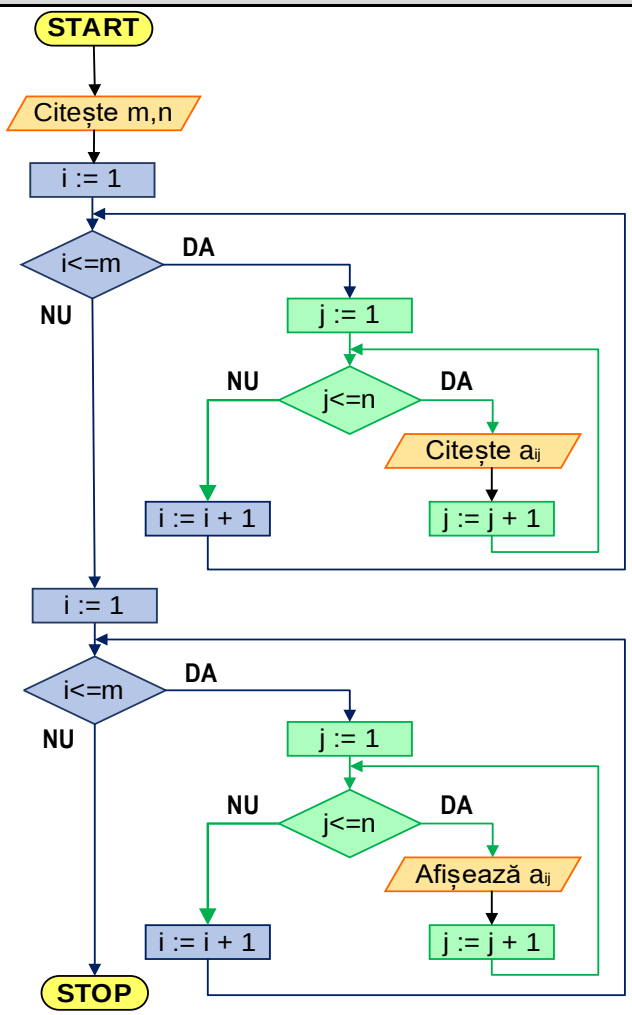
	**Început**
	  Citeşte m,n
	  Pentru i←1,m
	  Pentru j←1,n
	  Citeşte a_{ij}
	  Sfârşit pentru
	  Sfârşit pentru
	  Pentru i←1,m
	  Pentru j←1,n
	  Scrie a_{ij}
	  Sfârşit pentru
	  Sfârşit pentru
	Sfârşit

Figura 6.1a. Reprezentarea algoritmului pentru introducerea/afişarea elementelor unui tablou bidimensional

Programul MATLAB	Execuția programului

```
m=input(' Introdu m= ');
n=input(' Introdu n= ');
for i=1:m
    for j=1:n
       s=sprintf(' a[%d,%d] = ',i,j);
       a(i,j)=input(s);
    end
end
for i=1:m
    s='';
    for j=1:n
        s=[s, num2str(a(i,j)), ' '];
    end
    disp(s)
end
```

```
Introdu m = 2
Introdu n = 3
 a[1,1] = 2
 a[1,2] = 4
 a[1,3] = 5
 a[2,1] = 3
 a[2,2] = 7
 a[2,3] = 1

   2    4    5
   3    7    1
```

Figura 6.1b. Programul MATLAB şi execuţia acestuia pentru introducerea/afişarea elementelor unui tablou bidimensional

Observaţie: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

O altă variantă este folosind funcții specifice limbajului de programare MATLAB. Citirea unei matrice se face folosind funcția de intrare **input( )** și introducând efectiv elementele matricei prin enumerarea între paranteze pătrate, iar afișarea se poate face prin omiterea caracterului **;** sau folosind funcțiile **disp()** sau **display()**. Separarea elementelor în cadrul unei linii se poate face folosind spații sau simbolul virgula **(,),** iar separarea liniilor se face folosind simbolul punct și virgulă **(;).**

Programul MATLAB și execuția acestuia

```
A=input('Dati matricea A ')

Execuţia programului este următoarea:
Dati matricea A [1 2 7; -3, 5,10]
A =
     1     2     7
    -3     5    10
```

```
A=input('Dati matricea A ');
disp(A)

Execuţia programului este următoarea:
Dati matricea A [1 2; 7 -3; 5 10]
     1     2
     7    -3
     5    10
```

```
A=input('Dati matricea A ');
display(A)

Execuţia programului este următoarea:
Dati matricea A [1, 2; 7, -3; 5, 10]
A =
     1     2
     7    -3
     5    10
```

Figura 6.1c. Exemple de folosire a instrucțiunilor MATLAB pentru introducerea/afișarea elementelor unui tablou bidimensional

Se observă în exemplele din figura 6.1c că nu este necesar ca întâi să se citească numărul de linii și numărul de coloane ale matricei.

6.2. Calculul sumei elementelor unui tablou bidimensional

Pentru calculul sumei elementelor unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se inţializează suma **S** cu valoarea zero şi se parcurge matricea, aşa cum s-a arătat în exemplul anterior. În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei şi adăugarea valorii acesteia la suma S.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.2a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.2b.

Schema logică	Limbajul pseudocod

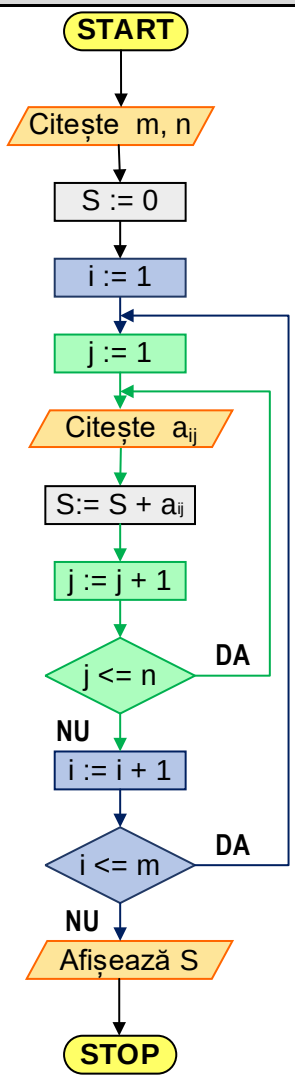
	**Început**
	  Citeşte m,n
	  S←0
	  Pentru i←1,m
	  Pentru j←1,n
	  Citeşte a_{ij}
	  S←S+ a_{ij}
	  Sfârşit pentru
	  Sfârşit pentru
	  Scrie S
	Sfârşit

Figura 6.2a. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou bidimensional – varianta 1

Programul MATLAB	Execuția programului

`m=input(' Introdu m= ');`
`n=input(' Introdu n= ');`
`S=0;`
`for i=1:m`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  S=S+a(i,j);`
`  end`
`end`
`s=sprintf(' S=%d',S);`
`disp(s)`
	`Introdu m = 2`
	`Introdu n = 3`
	`a[1,1] = 2`
	`a[1,2] = 4`
	`a[1,3] = 5`
	`a[2,1] = 3`
	`a[2,2] = 7`
	`a[2,3] = 1`
	
	`S = 22`

Figura 6.2b. Programul MATLAB şi execuţia acestuia pentru calculul sumei elementelor unui tablou bidimensional – varianta 1

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

O altă variantă de rezolvare este folosind funcția **sum()** specifică limbajului de programare MATLAB. Pentru determinarea sumei elementelor pe coloane se folosește condiția **sum(a)**, iar pentru determinarea sumei tuturor elementelor matricei se folosește instrucțiunea **sum(sum(a))**. Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.2c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.2d.

Schema logică	Limbajul pseudocod

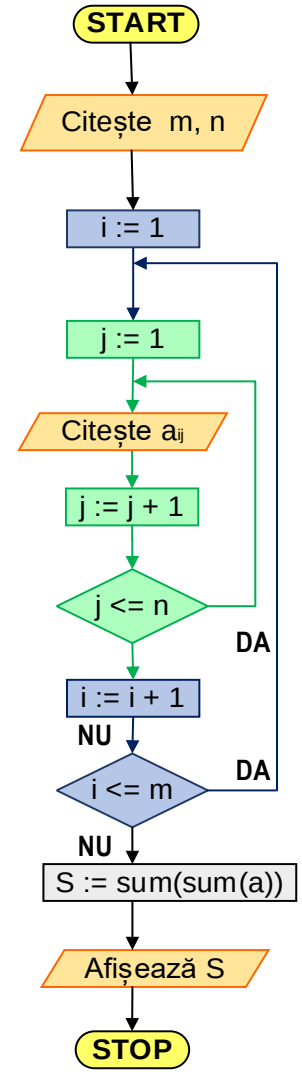
	**Început**
	  Citeşte m,n
	  Pentru i←1,m
	  Pentru j←1,n
	  Citeşte a_{ij}
	  Sfârşit pentru
	  Sfârşit pentru
	  S←sum(sum(a))
	  Scrie S
	Sfârşit

Figura 6.2c. Reprezentarea algoritmului pentru calculul sumei elementelor unui tablou bidimensional - varianta 2

Programul MATLAB	Execuția programului

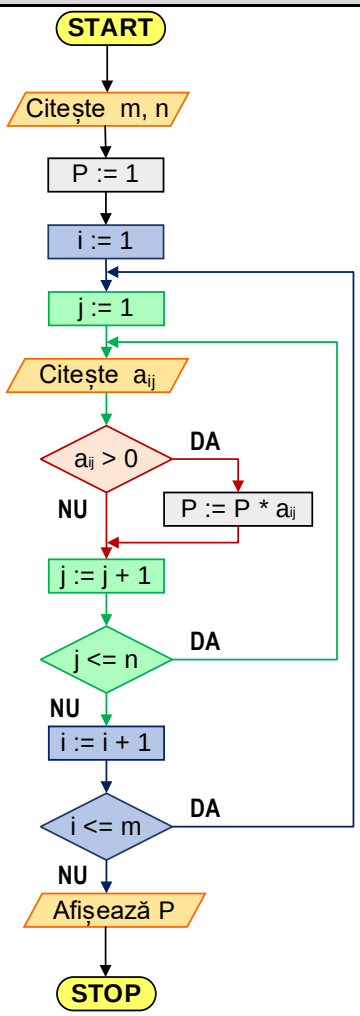
`m=input(' Introdu m= ');`
`n=input(' Introdu n= ');`
`for i=1:m`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  end`
`end`
`S=sum(sum(a));`
`s=sprintf(' S=%d',S); disp(s)`
	`Introdu m = 2`
	`Introdu n = 3`
	`a[1,1] = 2`
	`a[1,2] = 4`
	`a[1,3] = 5`
	`a[2,1] = 3`
	`a[2,2] = 7`
	`a[2,3] = 1`
	`S = 22`

Figura 6.2d. Programul MATLAB şi execuţia acestuia pentru calculul sumei elementelor unui tablou bidimensional - varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

6.3. Calculul produsului elementelor strict pozitive ale unui tablou bidimensional

Schema logică	Limbajul pseudocod




```
Început
  Citeşte m,n
  P←1
  Pentru i←1,m
    Pentru j←1,n
      Citeşte aij
      Dacă aij >0 atunci
          P←P * aij
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Scrie P
Sfârşit
```

Figura 6.3a. Reprezentarea algoritmului pentru calculul produsului elementelor strict pozitive ale unui tablou bidimensional – varianta 1

Programul MATLAB	Execuția programului

```
m=input(' Introdu m= ');
n=input(' Introdu n= ');
P=1;
for i=1:m
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
        if a(i,j)>0
            P=P*a(i,j);
        end
    end
end
s=sprintf('P=%d',P); disp(s)
```

```
Introdu m = 3
Introdu n = 3
 a[1,1] = 2
 a[1,2] = -4
 a[1,3] = 5
 a[2,1] = 0
 a[2,2] = 3
 a[2,3] = 1
 a[3,1] = 4
 a[3,2] = -2
 a[3,3] = -1
P = 120
```

Figura 6.3b. Programul MATLAB şi execuţia acestuia pentru calculul produsului elementelor strict pozitive ale unui tablou bidimensional – varianta 1

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Pentru calculul produsului elementelor strict pozitive ale unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se iniţializează produsul **P** cu valoarea **1** şi se parcurge matricea. În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei.

Schema logică	Limbajul pseudocod

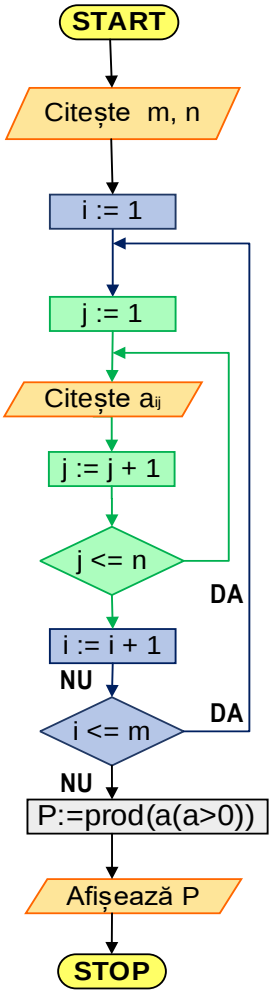
	**Început**
	  Citeşte m,n
	  Pentru i←1,m
	  Pentru j←1,n
	  Citeşte a_{ij}
	  Sfârşit pentru
	  Sfârşit pentru
	  P←prod(a(a>0))
	  Scrie P
	Sfârşit

Figura 6.3c. Reprezentarea algoritmului pentru calculul produsului elementelor strict pozitive ale unui tablou bidimensional – varianta 2

Programul MATLAB	Execuția programului

`m=input(' Introdu m= ');`
`n=input(' Introdu n= ');`
`for i=1:m`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  end`
`end`
`P=prod(a(a>0));`
`s=sprintf(' P=%d',P);  disp(s)`
	`Introdu m = 3`
	`Introdu n = 3`
	`a[1,1] = 2`
	`a[1,2] = -4`
	`a[1,3] = 5`
	`a[2,1] = 0`
	`a[2,2] = 3`
	`a[2,3] = 1`
	`a[3,1] = 4`
	`a[3,2] = -2`
	`a[3,3] = -1`
	`P = 120`

Figura 6.3d. Programul MATLAB şi execuţia acestuia pentru calculul sumei elementelor unui tablou bidimensional - varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Dacă valoarea acestuia este strict pozitivă, adică condiţia $a_{ij} > 0$ este adevărată, se înmulţeşte produsul **P** cu valoarea elementului curent. Dacă valoarea elementului curent nu este strict pozitivă, se continuă cu incrementarea valorii variabilei **j**. Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.3a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.3b.

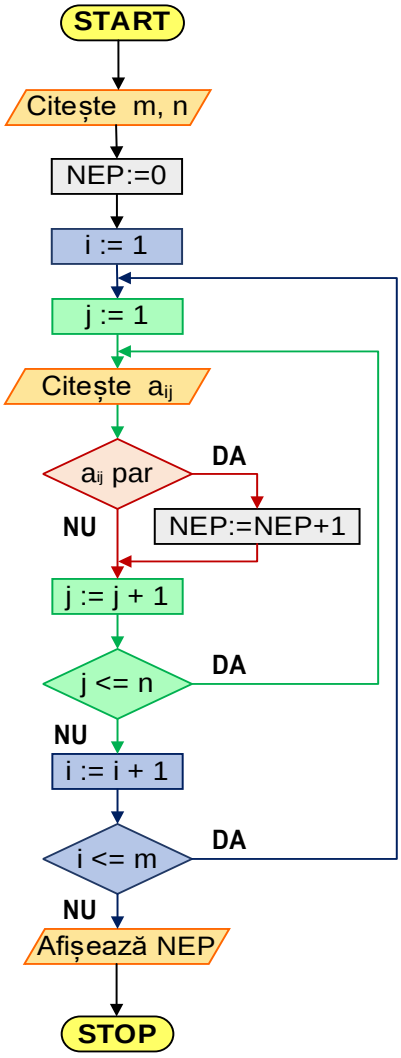
O altă variantă de rezolvare este folosind funcția **prod()** specifică limbajului de programare MATLAB. Pentru determinarea elementelor pozitive se folosește condiția **a(a>0)**, iar pentru determinarea produsului acestora se folosește instrucțiunea **prod(a(a>0))**. Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.3c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.3d.

6.4. Calculul numărului de elemente pare ale unui tablou bidimensional

Pentru calculul numărului de elemente pare ale unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se iniţializează numărul de elemente pare, notat **NEP** cu valoarea **0** şi se parcurge matricea. În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei.

Dacă valoarea acestuia este pară, adică condiţia a_{ij} **par** este adevărată, se incrementează variabila **NEP**. Dacă valoarea elementului curent nu este pară, se continuă cu incrementarea valorii variabilei **j**.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte m,n
  NEP←0
  Pentru i←1,m
    Pentru j←1,n
      Citeşte aij
      Dacă aij par atunci
          NEP←NEP +1
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Scrie NEP
Sfârşit
```

Figura 6.4a. Reprezentarea algoritmului pentru calculul numărului de elemente pare ale unui tablou bidimensional-varianta 1

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.4a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.4b.

O altă variantă de rezolvare este folosind funcția **sum()** specifică limbajului de programare MATLAB. Pentru determinarea poziției elementelor pare se folosește condiția **mod(a,2)==0**, iar pentru determinarea numărului acestora se folosește instrucțiunea **sum(sum(mod(a,2)==0))**. Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.4c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.4d.

Programul MATLAB	Execuţia programului

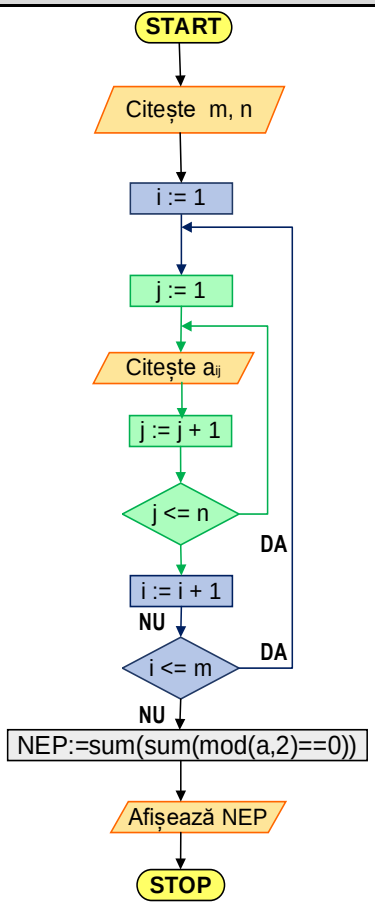
```
m=input(' Introdu m= ');
n=input(' Introdu n= ');
NEP=0;
for i=1:m
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
        if mod(a(i,j),2)==0
            NEP=NEP+1;
        end
    end
end
s=sprintf('NEP=%d',NEP); disp(s)
```

```
Introdu m = 3
Introdu n = 3
 a[1,1] = 2
 a[1,2] = -4
 a[1,3] = 5
 a[2,1] = 0
 a[2,2] = 3
 a[2,3] = 1
 a[3,1] = 4
 a[3,2] = -2
 a[3,3] = -1
    NEP = 5
```

Figura 6.4b. Programul MATLAB şi execuţia acestuia pentru calculul numărului de elemente pare ale unui tablou bidimensional-varianta 1

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Schema logică	Limbajul pseudocod



Început
  Citeşte m,n
  Pentru i←1,m
    Pentru j←1,n
      Citeşte a_{ij}
    Sfârşit pentru
  Sfârşit pentru
  NEP←sum(sum(mod(a,2)=0))
  Scrie NEP
Sfârşit

Figura 6.4c. Reprezentarea algoritmului pentru calculul numărului de elemente pare ale unui tablou bidimensional -varianta 2

Programul MATLAB	Execuţia programului

`m=input(' Introdu m= ');`
`n=input(' Introdu n= ');`
`for i=1:m`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  end`
`end`
`NEP=sum(sum(mod(a,2)==0));`
`s=sprintf('NEP=%d',NEP);`
`disp(s)`
	`Introdu m = 3`
	`Introdu n = 3`
	`a[1,1] = 2`
	`a[1,2] = -4`
	`a[1,3] = 5`
	`a[2,1] = 0`
	`a[2,2] = 3`
	`a[2,3] = 1`
	`a[3,1] = 4`
	`a[3,2] = -2`
	`a[3,3] = -1`
	`NEP=5`

Figura 6.4d. Programul MATLAB şi execuţia acestuia pentru calculul sumei elementelor unui tablou bidimensional – varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

6.5. Calculul mediei aritmetice a elementelor divizibile cu 5 ale unui tablou bidimensional

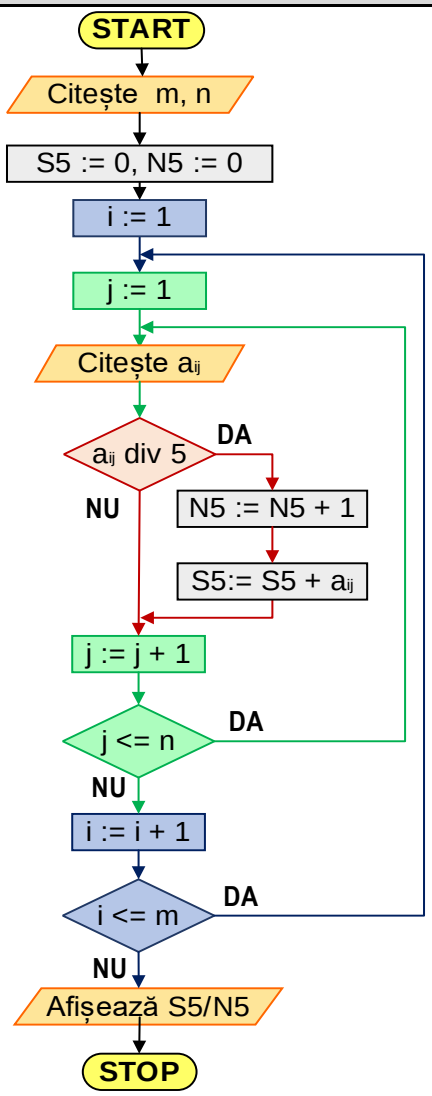
Pentru calculul numărului de elemente divizibile cu **5** ale unui tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se iniţializează numărul de elemente divizibile cu **5**, notat **N5** cu valoarea **0**, respectiv suma elementelor divizibile cu **5**, notată cu **S5** cu valoarea **0**.

În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei. Dacă valoarea acestuia este divizibilă cu **5**, adică condiţia a_{ij} **div 5** este adevărată, se incrementează variabila **N5**, iar la suma **S5** se adaugă valoarea a_{ij}. Dacă valoarea elementului curent nu este divizibilă cu **5**, se continuă cu incrementarea valorii variabilei **j**.

La final în variabila **N5** va fi stocat numărul de elemente divizibile cu 5 din tabloul unidimensional, iar în variabila **S5** va fi stocată suma acestora. Astfel, prin împărțirea variabilei **S5** la **N5** se obține media aritmetică a elementelor divizibile cu 5 dintr-un tablou unidimensional dat.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.5a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.5b.

Schema logică	Limbajul pseudocod




```
Început
  Citeşte m,n
  S5←0
  N5←0
  Pentru i←1,m
    Pentru j←1,n
      Citeşte aij
      Dacă aij DIV 5=0 atunci
          N5←N5 +1
          S5←S5 +aij
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Scrie S5/N5
Sfârşit
```

Figura 6.4a. Reprezentarea algoritmului pentru calculul mediei aritmetice a elementelor divizibile cu 5 – varianta 1

Programul MATLAB	Execuţia programului

```
m=input(' Introdu m= ');
n=input(' Introdu n= ');
S5=0; N5=0;
for i=1:m
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
        if mod(a(i,j),5)==0
            N5=N5+1;
            S5=S5+a(i,j);
        end
    end
end
s=sprintf('Ma=%d',S5/N5); disp(s)
```

```
Introdu m = 3
Introdu n = 3
 a[1,1] = 15
 a[1,2] = 6
 a[1,3] = -3
 a[2,1] = 0
 a[2,2] = 5
 a[2,3] = 10
 a[3,1] = 4
 a[3,2] = 2
 a[3,3] = -5
Ma = 5.000000
```

Figura 6.5b. Programul MATLAB şi execuţia acestuia pentru calculul mediei aritmetice a elementelor divizibile cu 5 -varianta 1

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Schema logică	Limbajul pseudocod

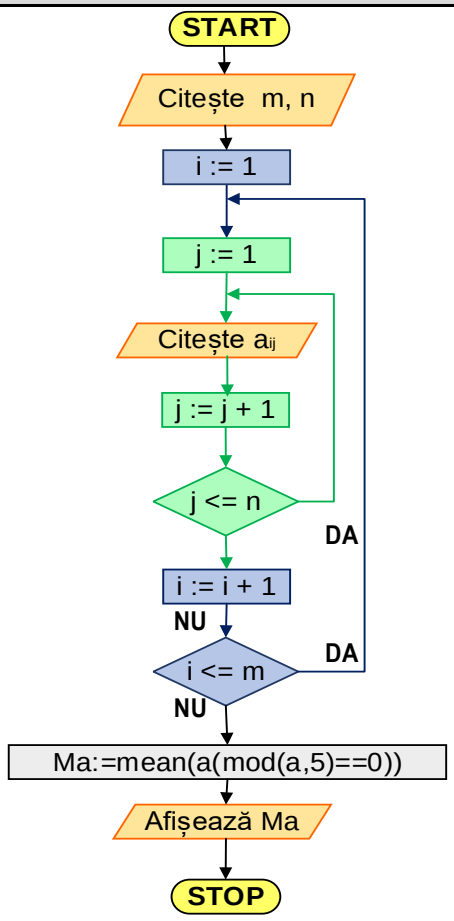
	**Început**
	  Citeşte m,n
	  Pentru i←1,m
	  Pentru j←1,n
	  Citeşte a_{ij}
	  Sfârşit pentru
	  Sfârşit pentru
	  Ma←mean(a(mod(a,5)=0))
	  Scrie Ma
	Sfârşit

Figura 6.4c. Reprezentarea algoritmului pentru calculul mediei aritmetice a elementelor divizibile cu 5 pare ale unui tablou bidimensional – varianta 2

O altă variantă de rezolvare este folosind funcția **mean()** specifică limbajului de programare MATLAB. Pentru determinarea elementelor pare se folosește condiția **a(mod(a,5)==0)**, iar pentru determinarea sumei acestora se folosește instrucțiunea **mean(a(mod(a,2)==0))**. Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.5c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.5d.

Programul MATLAB	Execuţia programului

`m=input(' Introdu m= ');`
`n=input(' Introdu n= ');`
`for i=1:m`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  end`
`end`
`Ma=mean(a(mod(a,5)==0));`
`s=sprintf('Ma=%d',Ma); disp(s)`
	`Introdu m = 3`
	`Introdu n = 3`
	`a[1,1] = 15`
	`a[1,2] = 6`
	`a[1,3] = -3`
	`a[2,1] = 0`
	`a[2,2] = 5`
	`a[2,3] = 10`
	`a[3,1] = 4`
	`a[3,2] = 2`
	`a[3,3] = -5`
	`Ma = 5.000000`

Figura 6.5d. Programul MATLAB şi execuţia acestuia pentru calculul mediei aritmetice a elementelor divizibile cu 5 pare ale unui tablou bidimensional – varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

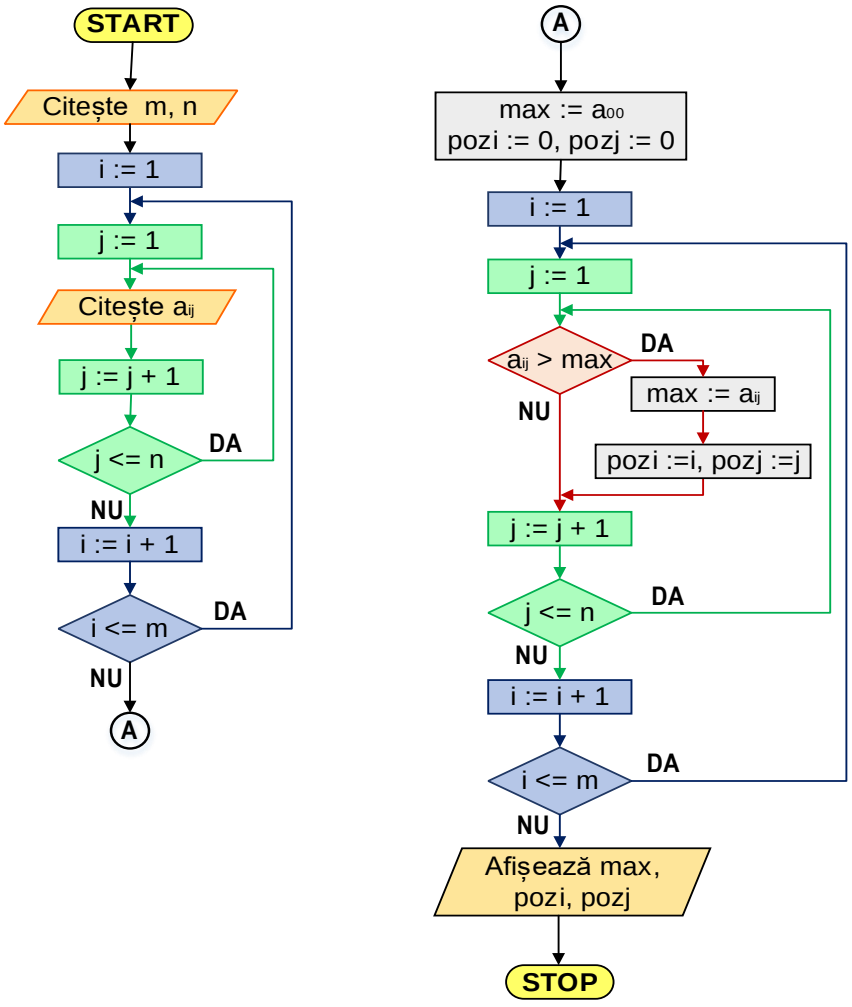
6.6. Determinarea elementului maxim şi a poziţiei acestuia dintr-un tablou bidimensional

Pentru determinarea elementului maxim dintr-un tablou bidimensional se citesc valorile variabilelor **m** şi **n** (numărul de linii şi de coloane), se iniţializează variabila **max** cu valoarea elementului de pe prima linie și coloană $\mathbf{a_{11}}$, iar variabilele **pozi**, respectiv **pozj** cu **1**.

În corpul ciclului cu contor interior, cel cu ajutorul căruia se parcurge linia, se realizează două operaţii: citirea elementului curent al matricei, respectiv verificarea valorii elementului curent al matricei. Dacă valoarea acestuia este mai mare decât valoarea variabilei **max**, se atribuie lui **max** valoarea elementului curent, iar variabilelor **pozi** şi **pozj** li se atribuie valorile curente ale lui **i** şi **j**, astfel actualizându-se la întâlnirea unui element mai mare decât maximul de până atunci, atât valoarea maximului cât și poziția pe care se regăsește acesta. În caz contrar, adică, dacă valoarea elementului curent nu este mai mare decât valoarea reținută în variabila **max** se continuă căutarea în tabloul bidimensional prin incrementarea valorii variabilei **j**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.6a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.6b.

Schema logică



Limbajul pseudocod

```
Început
   Citeşte m,n
   Pentru i←1,m
      Pentru j←1,n
        Citeşte aij
      Sfârşit pentru
   Sfârşit pentru
   max←a11;   pozi←1;   pozj←1
   Pentru i←1,m
      Pentru j←1,n
        Dacă aij >max atunci
            max←aij;   pozi←i ;  pozj←j
        Sfârşit dacă
      Sfârşit pentru
   Sfârşit pentru
    Scrie max, pozi, pozj
Sfârşit
```

Figura 6.6a. Reprezentarea algoritmului pentru determinarea elementului maxim şi a poziţiei acestuia dintr-un tablou bidimensional

Programul MATLAB	Execuția programului

```
m=input(' Introdu m= ');
n=input(' Introdu n= ');
for i=1:m
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
    end
end
M=a(1,1); pozi=1; pozj=1;
for i=1:m
    for j=1:n
        if a(i,j)>M
            M=a(i,j);
            pozi=i;
            pozj=j;
        end
    end
end
s=sprintf('Max=%d',M);
disp(s)
s=sprintf('pozi=%d, pozj=%d',pozi,pozj);
disp(s)
```

```
Introdu m= 3
 Introdu n= 3
 a[1,1] = 5
 a[1,2] = 6
 a[1,3] = -3
 a[2,1] = 0
 a[2,2] = 5
 a[2,3] = 10
 a[3,1] = 4
 a[3,2] = 2
 a[3,3] = -5
Max=10
pozi=2, pozj=3
```

Figura 6.6b. Programul MATLAB şi execuţia acestuia pentru determinarea elementului maxim şi a poziţiei acestuia dintr-un tablou bidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

6.7. Înmulţirea a două tablouri bidimensionale

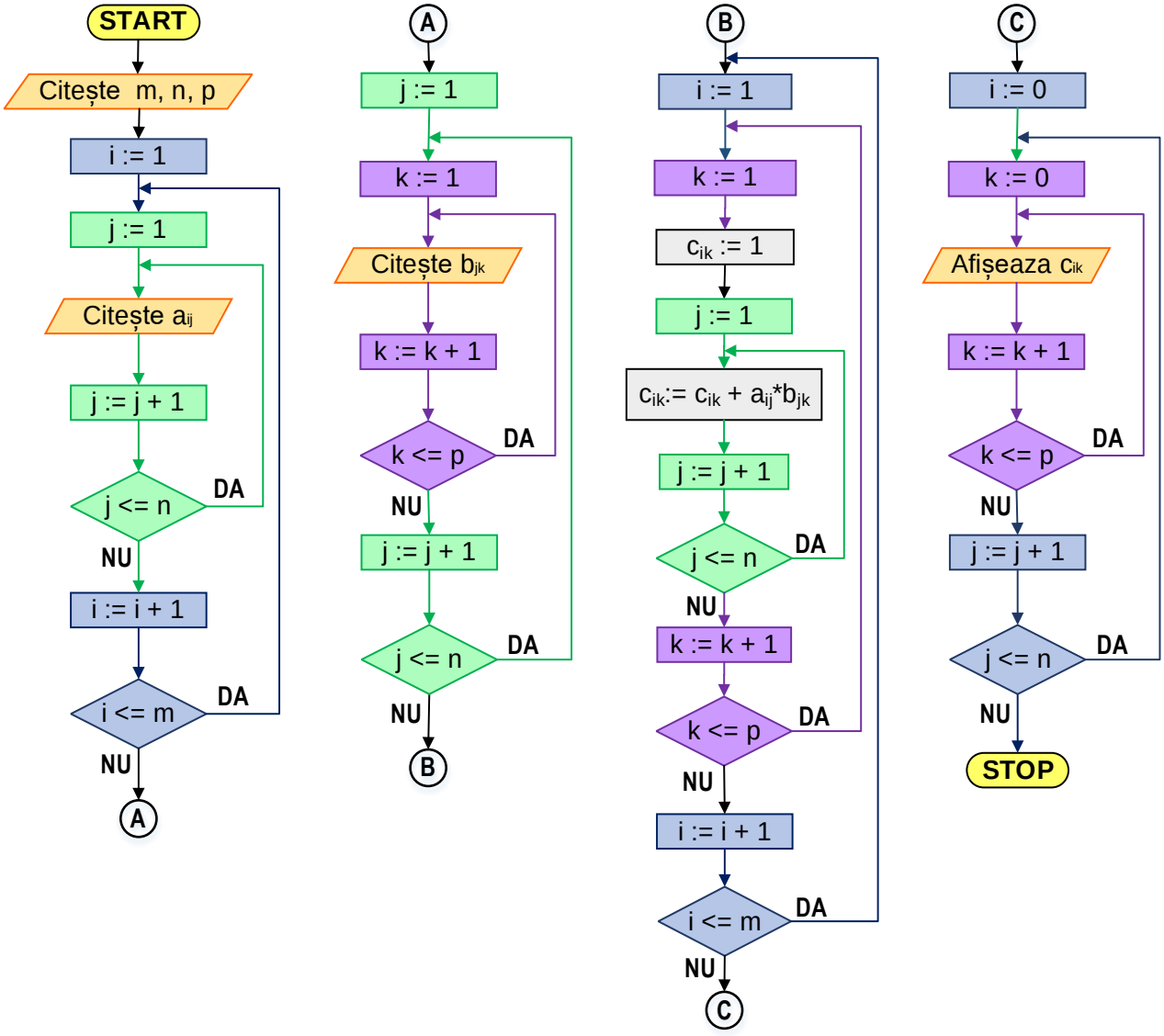
Se consideră două matrice $\mathbf{A}_{mxn}$ şi $\mathbf{B}_{nxp}$ şi se doreşte să se determine matricea $\mathbf{C}_{mxp} = \mathbf{A}_{mxn} \times \mathbf{B}_{nxp}$. Înmulţirea este posibilă doar dacă numărul de coloane de la prima matrice este egal cu numărul de linii de la a doua matrice. Relaţia de calcul a elementelor matricei **C** este:

$$c_{ik} = \sum_{j=0}^{n-1} a_{ij} \cdot b_{jk}$$

În cadrul programului se citesc cele două matrice $\mathbf{A}_{mxn}$ şi $\mathbf{B}_{nxp}$ şi se calculează elementele matricei $\mathbf{C}_{mxp} = \mathbf{A}_{mxn} \times \mathbf{B}_{nxp}$ conform relaţiei de mai sus. În final, se afişează elementele matricei **C**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.7a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.7b.

Schema logică



Pseudocod

```
Început
  Citeşte m,n,p
  Pentru i←1,m
    Pentru j←1,n
      Citeşte aij
    Sfârşit pentru
  Sfârşit pentru
  Pentru j←1,n
    Pentru k←1,p
      Citeşte bjk
    Sfârşit pentru
  Sfârşit pentru
  Pentru i←1,m
    Pentru k←1,p
      cik←0
      Pentru j←1,n
        cik←cik+aij*bjk
      Sfârşit pentru
    Sfârşit pentru
  Sfârşit pentru
  Pentru i←1,m
    Pentru k←1,p
      Scrie cik
    Sfârşit pentru
  Sfârşit pentru
Sfârşit
```

Figura 6.7a. Reprezentarea algoritmului pentru înmulțirea a două tablouri bidimensionale – varianta 1

Programul MATLAB	Execuția programului

`m=input(' Introdu m= ');`
`n=input(' Introdu n= ');`
`p=input(' Introdu p= ');`
`for i=1:m`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  end`
`end`
`for j=1:n`
`  for k=1:p`
`  s=sprintf(' b[%d,%d] = ',j,k);`
`  b(j,k)=input(s);`
`  end`
`end`
`for i=1:m`
`  for k=1:p`
`  c(i,k)=0;`
	`Introdu m= 3`
	` Introdu n= 2`
	` Introdu p= 3`
	` a[1,1] = 1`
	` a[1,2] = 0`
	` a[2,1] = 2`
	` a[2,2] = -1`
	` a[3,1] = 3`
	` a[3,2] = 1`
	` b[1,1] = 3`
	` b[1,2] = 1`
	` b[1,3] = 2`
	` b[2,1] = 1`
	` b[2,2] = 1`
	` b[2,3] = 0`
	`Matricea C:`
	`3 1 2`
	`5 1 4`

```
        for j=1:n
            c(i,k)=c(i,k)+a(i,j)*b(j,k);
        end
    end
end
disp('Matricea C:');
for i=1:m
    s='';
    for k=1:p
        s=[s, num2str(c(i,k)), ' '];
    end
    disp(s)
end
```

```
10  4  6
```

Figura 6.7b. Programul MATLAB şi execuţia acestuia pentru înmulțirea a două tablouri bidimensionale – varianta 1

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

O altă variantă de rezolvare este folosind operatorul * de înmulțire. În limbajul de programare MATLAB, acesta poate fi folosit și pentru matrice, nu doar pentru numere, atât timp cât matricele îndeplinesc condiția matematică legată de numărul de linii și coloane necesară pentru a se putea realiza înmulțirea. Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.7c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.7d.

Schema logică	Limbajul pseudocod

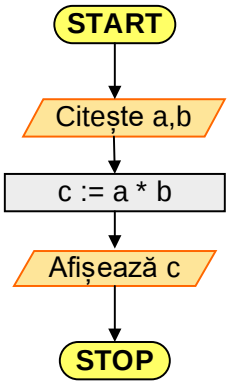
	**Început**
	  Citeşte a,b
	  c←a*b
	  Scrie c
	Sfârşit

Figura 6.7c. Reprezentarea algoritmului pentru înmulțirea a două tablouri bidimensionale – varianta 2

Programul MATLAB	Execuţia programului

`a=input(' Introdu matricea a= ');`
`b=input(' Introdu matricea b= ');`
`c=a*b;`
`disp('Matricea C:');`
`disp(c)`
	`Introdu matricea a= [1 2 3; 3 4 5]`
	`Introdu matricea b= [1 2;3 4; 5 6]`
	`Matricea C:`
	`22  28`
	`40  52`

Figura 6.7d. Programul MATLAB şi execuţia acestuia pentru calculul sumei elementelor unui tablou bidimensional – varianta 2

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

6.8. Eliminarea unei linii dintr-un tablou bidimensional

Se consideră o matrice $\mathbf{A}_{mxn}$ şi se doreşte eliminarea liniei cu indicele „**k**”, cu formarea unei noi matrice $\mathbf{B}_{m-1xn}$, care conţine **m-1** linii. În cadrul programului se realizează citirea matricei $\mathbf{A}_{mxn}$, după care se realizează citirea valorii **k** a indicelui liniei care se elimină.

Se parcurge matricea $\mathbf{A}_{mxn}$ şi li se atribuie elementelor matricei **B** valorile elementele matricei **A**, până la linia cu indicele **k**, adică atât timp cât este valabilă condiţia **i < k**. În continuare, de la linia **k** şi până la ultima linie a matricei **B**, elementele de pe liniile **i** ale matricei **B** vor primi valorile elementelor de pe liniile **i+1** ale matricei **A**. Astfel se creează noua matrice **B** cu **n** coloane dar cu **m-1** linii. Mai exact matricea **B** conține doar liniile de la **1** la **k**, respectiv de la **k+1** la **m** din matricea **A**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.8a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.8b.

Schema logică

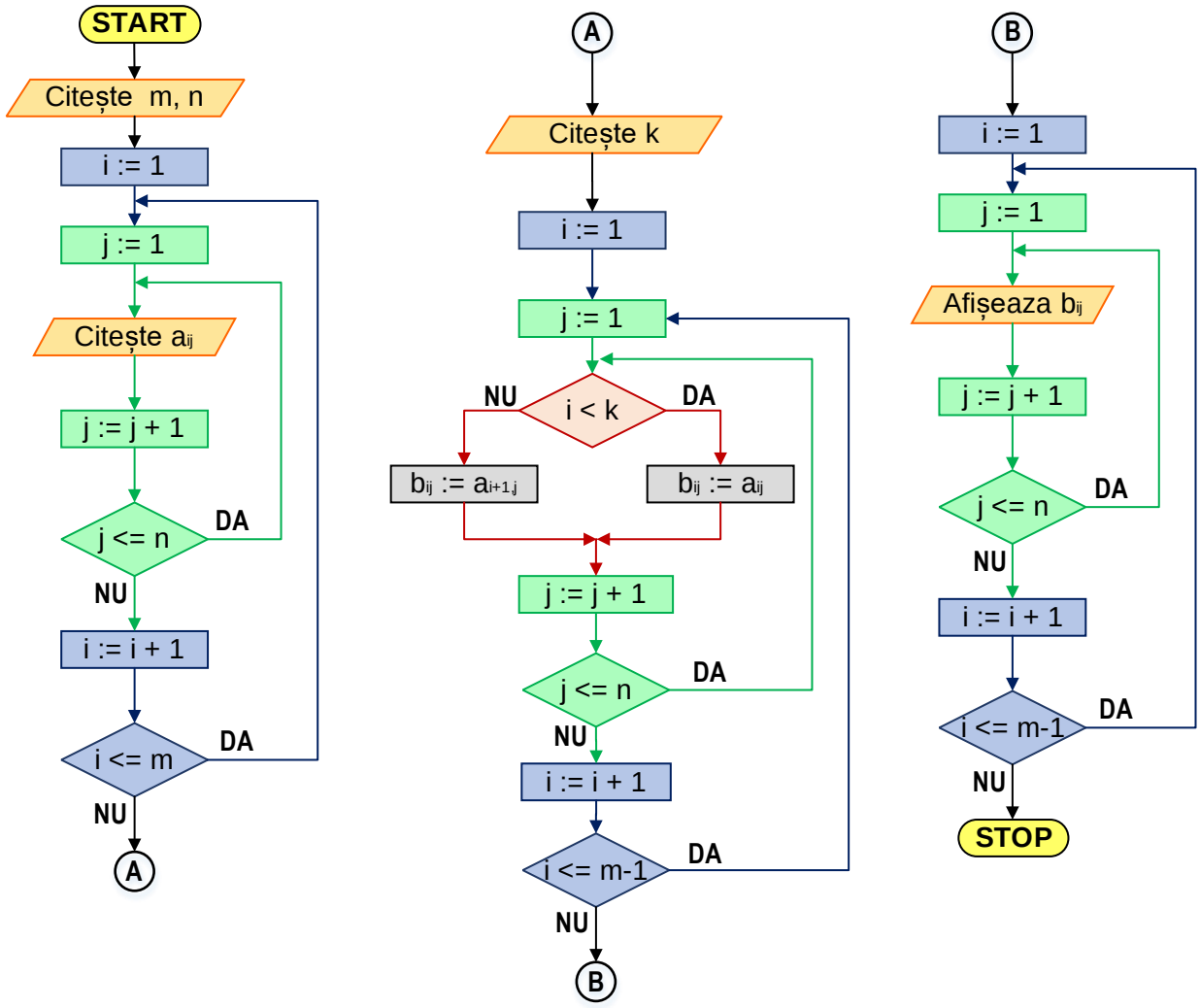


Figura 6.8a. Reprezentarea algoritmului pentru eliminarea liniei „k” dintr-un tablou bidimensional

Programul MATLAB	Execuția programului

```
m=input(' Introdu m= ');
n=input(' Introdu n= ');
for i=1:m
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
    end
end
k=input(' Introdu k= ');
for i=1:m-1
    for j=1:n
        if i<k
            b(i,j)=a(i,j);
        else
            b(i,j)=a(i+1,j);
        end
    end
end
disp('Matricea B:');
for i=1:m-1
    s='';
    for j=1:n
        s=[s, num2str(b(i,j)), ' '];
    end
    disp(s)
end
```

```
Introdu m = 4
Introdu n = 3
 a[1,1] = 1
 a[1,2] = 0
 a[1,3] = 2
 a[2,1] = -1
 a[2,2] = 3
 a[2,3] = 1
 a[3,1] = 3
 a[3,2] = 1
 a[3,3] = 2
 a[4,1] = -5
 a[4,2] = 4
 a[4,3] = -2
Introdu k = 2
Matricea B:
     1     0     2
     3     1     2
    -5     4    -2
```

Figura 6.8b. Programul MATLAB şi execuţia acestuia pentru eliminarea liniei „k" dintr-un tablou bidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

6.9. Eliminarea unei coloane dintr-un tablou bidimensional

Se consideră o matrice $\mathbf{A}_{mxn}$ şi se doreşte eliminarea coloanei cu indicele „**k**", cu formarea unei matrice $\mathbf{B}_{mxn-1}$, care conţine **n-1** coloane. În cadrul programului se procedează ca la problema anterioară, doar că se elimină o coloană în loc de o linie.

Întâi se realizează citirea matricei $\mathbf{A}_{mxn}$, după care se realizează citirea valorii **k** a indicelui coloanei care se elimină. Apoi se parcurge matricea $\mathbf{A}_{mxn}$ şi li se atribuie elementelor matricei **B** valorile elementele matricei **A**, până la coloana cu indicele **k**, adică atât timp cât este valabilă condiţia **j < k**. În continuare, de la coloana **k** şi până la ultima coloană a matricei **B**, elementele de pe coloanele **i** ale matricei **B** vor primi valorile elementelor de pe coloanele **j+1** ale matricei **A**. Astfel se creează noua matrice **B** cu **m** linii, dar cu **n-1** coloane. Mai exact matricea **B** conține doar coloanele de la **1** la **k**, respectiv de la **k+1** la **n** din matricea **A**.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.9a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.9b.

Schema logică

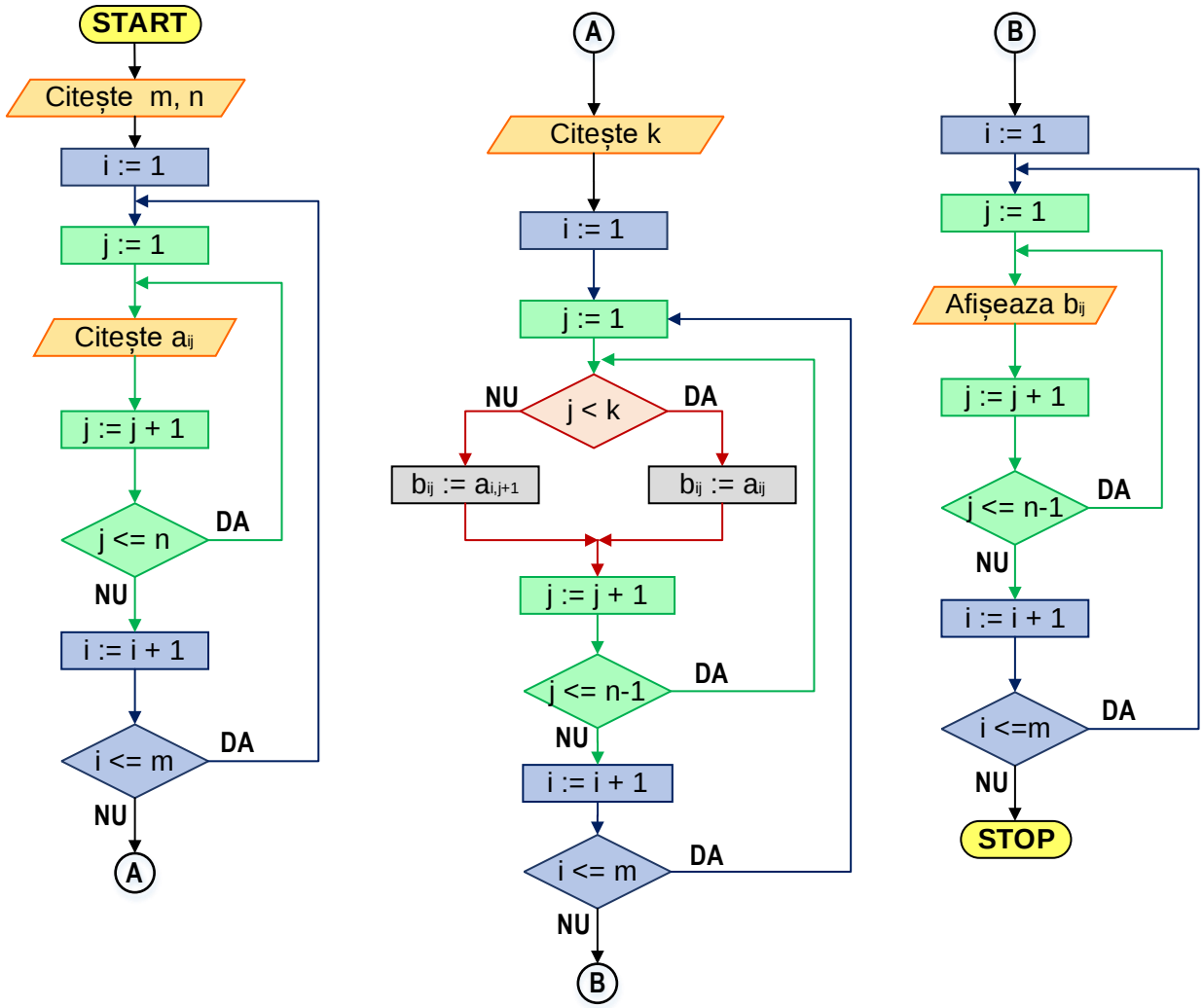


Figura 6.9a. Reprezentarea algoritmului pentru eliminarea coloanei „k” dintr-un tablou bidimensional

Programul MATLAB	Execuția programului

```
m=input(' Introdu m= ');
n=input(' Introdu n= ');
for i=1:m
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
    end
end
k=input(' Introdu k= ');
for i=1:m
    for j=1:n-1
        if j<k
            b(i,j)=a(i,j);
        else
            b(i,j)=a(i,j+1);
        end
```

```
Introdu m = 4
Introdu n = 3
 a[1,1] = 1
 a[1,2] = 0
 a[1,3] = 2
 a[2,1] = -1
 a[2,2] = 3
 a[2,3] = 1
 a[3,1] = 3
 a[3,2] = 1
 a[3,3] = 2
 a[4,1] = -5
 a[4,2] = 4
 a[4,3] = -2

Introdu k = 2
```

```
    end
end
disp('Matricea B:');
for i=1:m
    s='';
    for j=1:n-1
        s=[s, num2str(b(i,j)), ' '];
    end
    disp(s)
end
```

```
Matricea B:
     1     2
    -1     1
     3     2
    -5    -2
```

Figura 6.9b. Programul MATLAB şi execuţia acestuia pentru eliminarea coloanei „k" dintr-un tablou bidimensional

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

.

6.10. Suma elementelor situate deasupra diagonalei principale (matrice pătratice)

Un tablou bidimensional se numeşte tablou pătratic sau matrice pătratică dacă numărul de linii este egal cu numărul de coloane. În acest caz se utilizează pentru specificarea numărului de linii şi de coloane o singură variabilă, de obicei n.

În cazul matricelor pătratice există două elemente definitorii: **diagonala principală** (figura 6.10a), respectiv **diagonala secundară** (figura 6.10b), care permit împărțirea matricei în diferite zone. Aceste zone pot fi specificate cu ajutorul unor relații pe care indicii pentru linie, respectiv coloană trebuie să le îndeplinească.

	1	2	3	...	n-1	n
1	a_{11}	a_{12}	a_{13}	...	a_{1n-1}	a_{1n}
2	a_{21}	a_{22}	a_{23}	...	a_{2n-1}	a_{2n}
3	a_{31}	a_{32}	a_{33}	...	a_{3n-1}	a_{3n}
...	...	...	...	...	...	...
n-1	a_{n-11}	a_{n-12}	a_{n-13}	...	a_{n-1n-1}	a_{n-1n}
n	a_{n1}	a_{n2}	a_{n3}	...	a_{nn-1}	a_{nn}

Figura 6.10a. Diagonala principală

	1	2	3	...	n-1	n
1	a_{11}	a_{12}	a_{13}	...	a_{1n-1}	a_{1n}
2	a_{21}	a_{22}	a_{23}	...	a_{2n-1}	a_{2n}
3	a_{31}	a_{32}	a_{33}	...	a_{3n-1}	a_{3n}
...	...	...	...	...	...	...
n-1	a_{n-11}	a_{n-12}	a_{n-13}	...	a_{n-1n-1}	a_{n-1n}
n	a_{n1}	a_{n2}	a_{n3}	...	a_{nn-1}	a_{nn}

Figura 6.10b. Diagonala secundară

Un element al matricei aparţine (sau nu) diagonalelor sau zonelor delimitate de acestea dacă respectă anumite reguli în care se utilizează indicii elementelor şi nu valoarea acestora.

Elementele situate pe diagonala principală au proprietatea că indicele liniei este egal cu indicele coloanei, adică **i = j**.

Elementele situate pe diagonala secundară au proprietatea că suma indicilor liniei şi coloanei este egală cu **n + 1**, adică **i + j = n + 1**;

Elementele situate sub diagonala principală au proprietatea că indicele liniei este strict mai mare decât indicele coloanei, adică **i > j**;

Elementele situate deasupra diagonalei principale au proprietatea că indicele liniei este strict mai mic decât indicele coloanei, adică: **i < j**;

Elementele situate sub diagonala secundară au proprietatea că suma indicilor liniei şi a coloanei este strict mai mare decât **n + 1**;

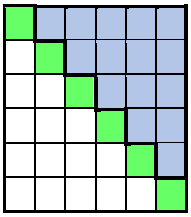
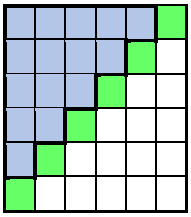
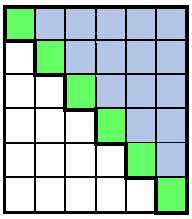
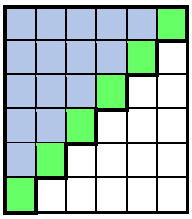
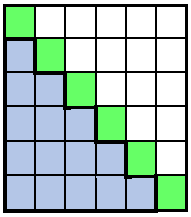
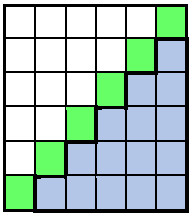
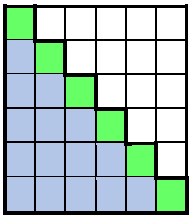
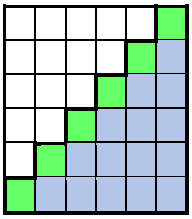
Elementele situate deasupra diagonalei secundare au proprietatea că suma indicilor liniei şi coloanei este strict mai mică decât **n + 1**;

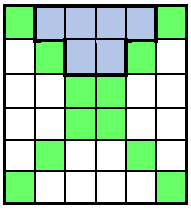
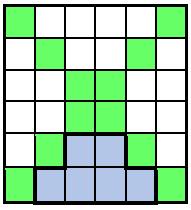
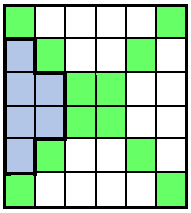
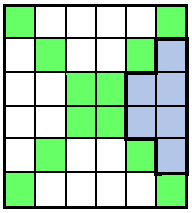
Relaţiile prezentate sunt valabile pentru indexarea de la **1**. În programe se utilizează indexarea de la **1** a tablourilor, relaţiile de mai sus modificându-se corespunzător doar pentru zonele care conţin diagonala secundară.

Pentru utilizarea elementelor dintr-o anumită zonă a matricei există două posibilităţi:

- se parcurge întreaga matrice şi se impun anumite condiţii indicilor elementelor matricei;
- se parcurge doar zona din matrice prin impunerea unor limite de variaţie a valorilor indicilor elementelor matricei;

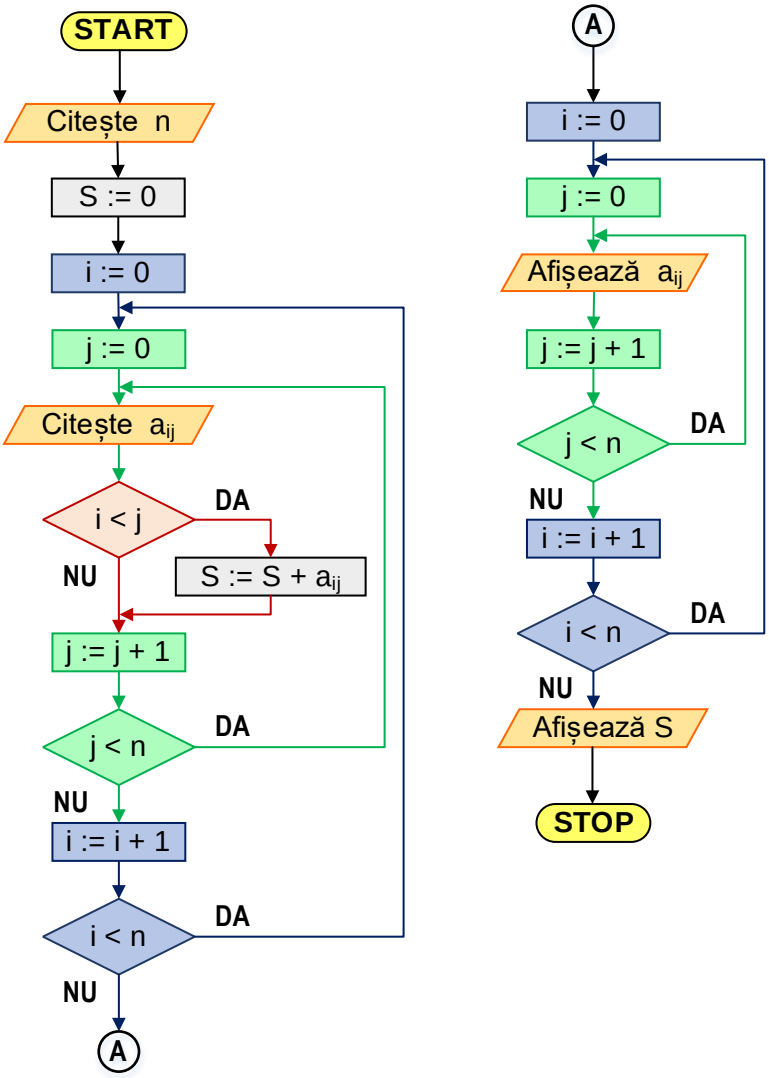
În tabelul 6.10.1 sunt prezentate o serie de exemple de parcurgere a unor zone dintr-o matrice pătratică:

Reprezentare grafică:	**Condiţii impuse indicilor**	**Limite impuse indicilor**		**Reprezentare grafică:**	**Condiţii impuse indicilor**	**Limite impuse indicilor**
Deasupra diagonalei principale				**Deasupra diagonalei secundare**		
	i < j	i = 1, ..., n - 1 j = i + 1, ..., n			i + j < n+1	i = 1, ..., n – 1 j = 1, ..., n - i
Deasupra şi pe diagonala principală				**Deasupra şi pe diagonala secundară**		
	i <= j	i = 1, ..., n j = i, ..., n			i + j <= n+1	i = 1, ..., n j = 1, ..., n – i + 1
Sub diagonala principală				**Sub diagonala secundară**		
	i > j	i = 2, ..., n j = 1, ..., i - 1			i + j > n+1	i = 2, ..., n j = n – i + 2, ..., n
Sub şi pe diagonala principală				**Sub şi pe diagonala secundară**		
	i >= j	i = 1, ..., n j = 1, ..., i			i + j > n+1	i = 1, ..., n j = n – i + 1, ..., n

Deasupra diagonalei principale şi secundare				**Sub diagonala principală şi secundară**		
	i < j şi i + j < n+1	i = 1, ..., (n +1) / 2 - 1 j = i + 1, ..., n - i			i > j şi i + j > n+1	i = (n +1) / 2 + 1, ..., n j = n - i + 2, ..., i - 1
Sub diagonala principală şi deasupra celei secundare				**Deasupra diagonalei principale şi sub cea secundară**		
	i > j şi i + j < n+1	j = 1, ..., (n +1) / 2 - 1 i = j + 1, ..., n - j			i < j şi i + j > n+1	j = (n +1) / 2 + 1, ..,n i = n - j + 2, ..., j - 1

Algoritmul pentru calculul sumei elementelor situate deasupra diagonalei principale poate fi conceput în două variante:

Schema logică



Pseudocod

```
Început
  Citeşte n
  S←0
  Pentru i←1,n
    Pentru j←1,n
      Citeşte aij
      Dacă i<j atunci
        S←S+aij
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Pentru i←1,n
    Pentru j←1,n
      Scrie aij
    Sfârşit pentru
  Sfârşit pentru
Scrie S
Sfârşit
```

Figura 6.10c. Reprezentarea algoritmului pentru calculul sumei elementelor situate deasupra diagonalei principale – varianta 1

Varianta 1: Se parcurge toată matricea şi se impun condiţii indicilor:
- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila n;
- se iniţializează valoarea sumei cu 0;
- utilizând un ciclu cu contor se parcurge matricea linie cu linie, atribuindu-se contorului pentru linii (variabila i) valori de la 1 la n. Pentru fiecare valoare a lui i (adică pentru fiecare linie) se realizează parcurgerea liniei utilizând un ciclu cu contor în care variabila j primeşte valori de la 1 la n. Pentru fiecare element al matricei, se realizează citirea valorii acestuia şi utilizând o instrucţiune de decizie, se verifică dacă se află situat deasupra diagonalei principale. Dacă condiţia este îndeplinită se adaugă valoarea acestui element la sumă;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se afişează suma calculată;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.10c, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.10d.

Programul MATLAB

```
n=input(' Introdu n= ');
S=0;
for i=1:n
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
        if i<j
            S=S+a(i,j);
        end
    end
end
for i=1:n
    s='';
```

Execuția programului

```
Introdu n= 4
 a[1,1] = 1
 a[1,2] = 5
 a[1,3] = 6
 a[1,4] = 2
 a[2,1] = 3
 a[2,2] = 0
 a[2,3] = 2
 a[2,4] = 4
 a[3,1] = 8
 a[3,2] = 7
 a[3,3] = 9
 a[3,4] = 5
 a[4,1] = 2
```

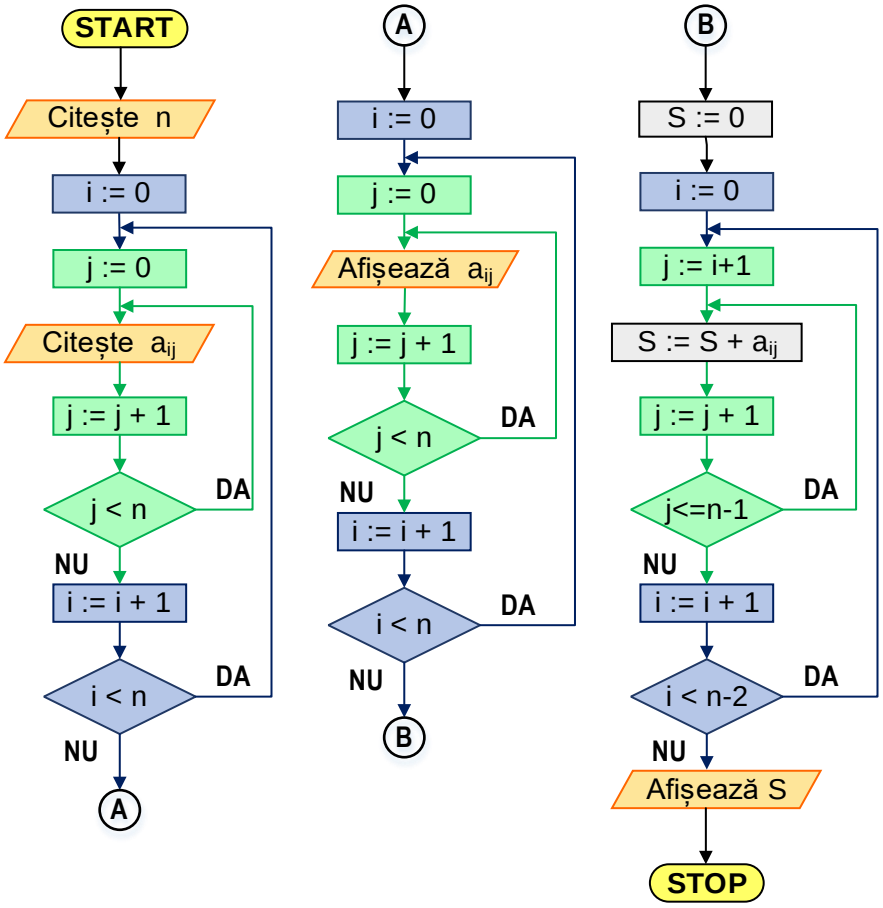
```
    for j=1:n
        s=[s, num2str(a(i,j)), ' '];
    end
    disp(s)
end
s=sprintf('Suma S=%d',S);
disp(s)
```

```
 a[4,2] = 6
 a[4,3] = 3
 a[4,4] = 0
1 5 6 2
3 0 2 4
8 7 9 5
2 6 3 0
Suma S=24
```

Figura 6.10d. Programul MATLAB şi execuția acestuia pentru calculul sumei elementelor situate deasupra diagonalei principale – varianta 1

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

Schema logică



Pseudocod

Început
  Citeşte n
  Pentru i←1,n
    Pentru j←1,n
      Citeşte a_{ij}
    Sfârşit pentru
  Sfârşit pentru
  Pentru i←1,n
    Pentru j←1,n
      Scrie a_{ij}

```
    Sfârşit pentru
  Sfârşit pentru
  S←0
  Pentru i←1,n-1
    Pentru j←i+1,n
      Scrie S←S+aij
    Sfârşit pentru
  Sfârşit pentru
  Scrie S
Sfârşit
```

Figura 6.10e. Reprezentarea algoritmului pentru calculul sumei elementelor situate deasupra diagonalei principale – varianta 2

Varianta 2: După ce citeşte şi se afişează toată matricea, pentru pasul de calculare a sumei se parcurge doar zona de deasupra diagonalei principale:
- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila n;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează valoarea sumei cu 0;
- utilizând două cicluri cu contor suprapuse se realizează calculul sumei impunând indicilor intervalele de variaţie specifice zonei de deasupra diagonalei principale;
- se afişează suma calculată;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.10e, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.10f.

Programul MATLAB	Execuția programului

```
n=input(' Introdu n= ');
for i=1:n
    for j=1:n
        s=sprintf(' a[%d,%d] = ',i,j);
        a(i,j)=input(s);
    end
end
for i=1:n
    s='';
    for j=1:n
        s=[s, num2str(a(i,j)), ' '];
    end
    disp(s)
end
S=0;
for i=1:n-1
    for j=i+1:n
        S=S+a(i,j);
    end
end
s=sprintf('Suma S=%d',S); disp(s)
```

```
Introdu n= 4
 a[1,1] = 1
 a[1,2] = 5
 a[1,3] = 6
 a[1,4] = 2
 a[2,1] = 3
 a[2,2] = 0
 a[2,3] = 2
 a[2,4] = 4
 a[3,1] = 8
 a[3,2] = 7
 a[3,3] = 9
 a[3,4] = 5
 a[4,1] = 2
 a[4,2] = 6
 a[4,3] = 3
 a[4,4] = 0
1 5 6 2
3 0 2 4
8 7 9 5
2 6 3 0
Suma S=24
```

Figura 6.10f. Programul MATLAB şi execuţia acestuia pentru calculul sumei elementelor situate deasupra diagonalei principale – varianta 2

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

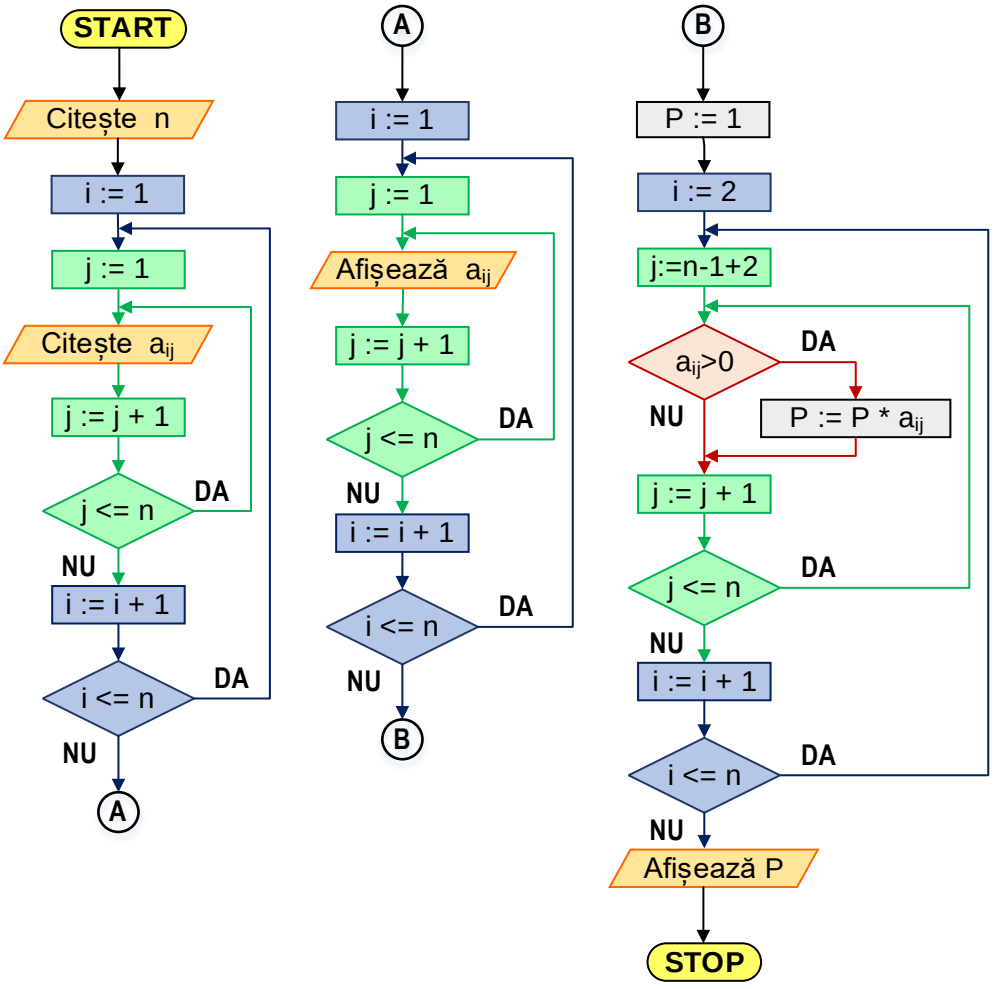
6.11. Produsul elementelor strict pozitive situate sub diagonala secundară (matrice pătratice)

Algoritmul propus în acest subcapitol este similar cu varianta 2 din subcapitolul anterior. Astfel pentru calculul produsului elementelor strict pozitive situate sub diagonala secundară a unei matrice pătratice necesită se parcurg următorii paşi:

- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila n;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează valoarea produsului cu 1;
- utilizând două cicluri cu contor suprapuse se parcurge zona din matrice de sub diagonala secundară, impunând indicilor intervalele de variaţie, adică pentru i de la 2 la n, respectiv pentru j de la n - i + 2 la n. De asemenea, pentru fiecare element al matricei care se află în această zonă se verifică, cu ajutorul unei instrucţiuni de decizie, dacă este strict pozitiv. Dacă elementul este strict pozitiv se înmulţeşte valoarea acestuia cu valoarea produsului;
- se afişează produsul calculat;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.11a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.11b.

Schema logică



Pseudocod

```
Început
  Citeşte n
  Pentru i←1,n
    Pentru j←1,n
      Citeşte aij
    Sfârşit pentru
  Sfârşit pentru
  Pentru i←1,n
    Pentru j←1,n
      Scrie aij
    Sfârşit pentru
  Sfârşit pentru
  P←1
  Pentru i←2,n
    Pentru j←n-i+2,n
      Dacă aij>0 atunci
        P←P*aij
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Scrie P
Sfârşit
```

Figura 6.11a. Reprezentarea algoritmului pentru calculul produsului elementelor strict pozitive situate sub diagonala secundară

Programul MATLAB

```
n=input(' Introdu n= ');
for i=1:n
    for j=1:n
       s=sprintf(' a[%d,%d] = ',i,j);
       a(i,j)=input(s);
    end
end
for i=1:n
    s='';
    for j=1:n
        s=[s, num2str(a(i,j)), ' '];
    end
    disp(s)
end
P=1;
for i=2:n
    for j=n-i+2:n
        if a(i,j)>0    P=P*a(i,j);
        end
    end
end
```

```
Introdu n= 4
 a[1,1] = 1
 a[1,2] = 2
 a[1,3] = 3
 a[1,4] = 4
 a[2,1] = -2
 a[2,2] = 5
 a[2,3] = 6
 a[2,4] = 0
 a[3,1] = -3
 a[3,2] = 2
 a[3,3] = 0
 a[3,4] = 1
 a[4,1] = -5
 a[4,2] = 2
 a[4,3] = 3
 a[4,4] = 0
1 2 3 4
-2 5 6 0
-3 2 0 1
-5 2 3 0
Produsul P=6.000000
```

```
s=sprintf('Produsul P=%f',P);
disp(s)
```

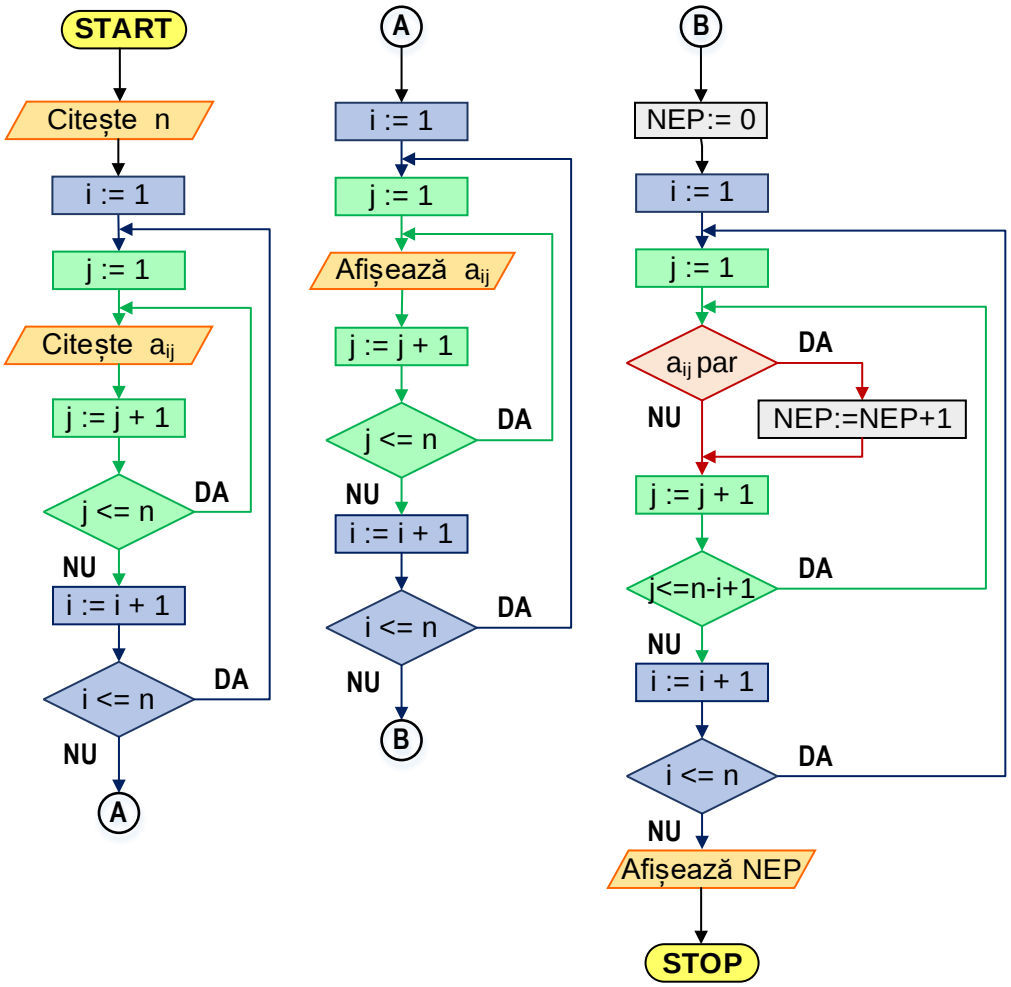
Figura 6.11b. Programul MATLAB şi execuţia acestuia pentru calculul produsului elementelor strict pozitive situate sub diagonala secundară

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

6.12. Numărul de elemente pare de pe şi deasupra diagonalei secundare (matrice pătratice)

Algoritmul propus pentru calculul numărului de elemente pare situate pe şi deasupra diagonalei secundare a unei matrice pătratice implică analizarea elementelor doar din zona dorită.

Schema logică



Pseudocod

```
Început
   Citeşte n
   Pentru i←1,n
      Pentru j←1,n
        Citeşte aij
      Sfârşit pentru
   Sfârşit pentru
   Pentru i←1,n
```

```
    Pentru j←1,n
      Scrie aij
    Sfârşit pentru
  Sfârşit pentru
  NEP←0
  Pentru i←1,n
    Pentru j←1,n-i+1
      Dacă aij par atunci  NEP←NEP+1
      Sfârşit dacă
    Sfârşit pentru
  Sfârşit pentru
  Scrie NEP
Sfârşit
```

Figura 6.12a. Reprezentarea algoritmului pentru determinarea numărului de elemente pare de pe şi deasupra diagonalei secundare

Astfel, acest algoritm necesită parcurgerea următorilor paşi:

- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila n;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează valoarea variabilei care memorează numărul de elemente pare cu 0, adică NEP = 0;
- utilizând două cicluri cu contor suprapuse se parcurge zona din matrice formată din diagonala secundară şi zona de deasupra diagonalei secundare, impunând indicilor intervalele de variaţie, adică pentru i de la **1** la **n**, respectiv pentru **j** de la **1** la **n – i + 1**. De asemenea, pentru fiecare element al matricei care se află în această zonă se verifică, cu ajutorul unei instrucţiuni de decizie, dacă este par, adică dacă restul împărţirii acestuia la 2 este nul. Dacă elementul este par se incrementează numărul de elemente pare;
- se afişează produsul calculat.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.11a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.11b.

Programul MATLAB	Execuția programului

`n=input(' Introdu n= ');`
`for i=1:n`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  end`
`end`
`for i=1:n`
`  s='';`
`  for j=1:n`
`  s=[s, num2str(a(i,j)), ' '];`
`  end`
`  disp(s)`
`end`
`NEP=0;`
`for i=1:n`
`  for j=1:n-i+1`
`  if mod(a(i,j),2)==0  NEP=NEP+1;`
	`Introdu n= 4`
	` a[1,1] = 1`
	` a[1,2] = 2`
	` a[1,3] = 3`
	` a[1,4] = 4`
	` a[2,1] = 5`
	` a[2,2] = 6`
	` a[2,3] = 7`
	` a[2,4] = 8`
	` a[3,1] = 9`
	` a[3,2] = 0`
	` a[3,3] = 2`
	` a[3,4] = 3`
	` a[4,1] = 5`
	` a[4,2] = 6`
	` a[4,3] = 4`
	` a[4,4] = 1`
	`1 2 3 4`

```
        end
    end
end
s=sprintf('NEP = %d',NEP); disp(s)
```

```
5 6 7 8
9 0 2 3
NEP = 4
```

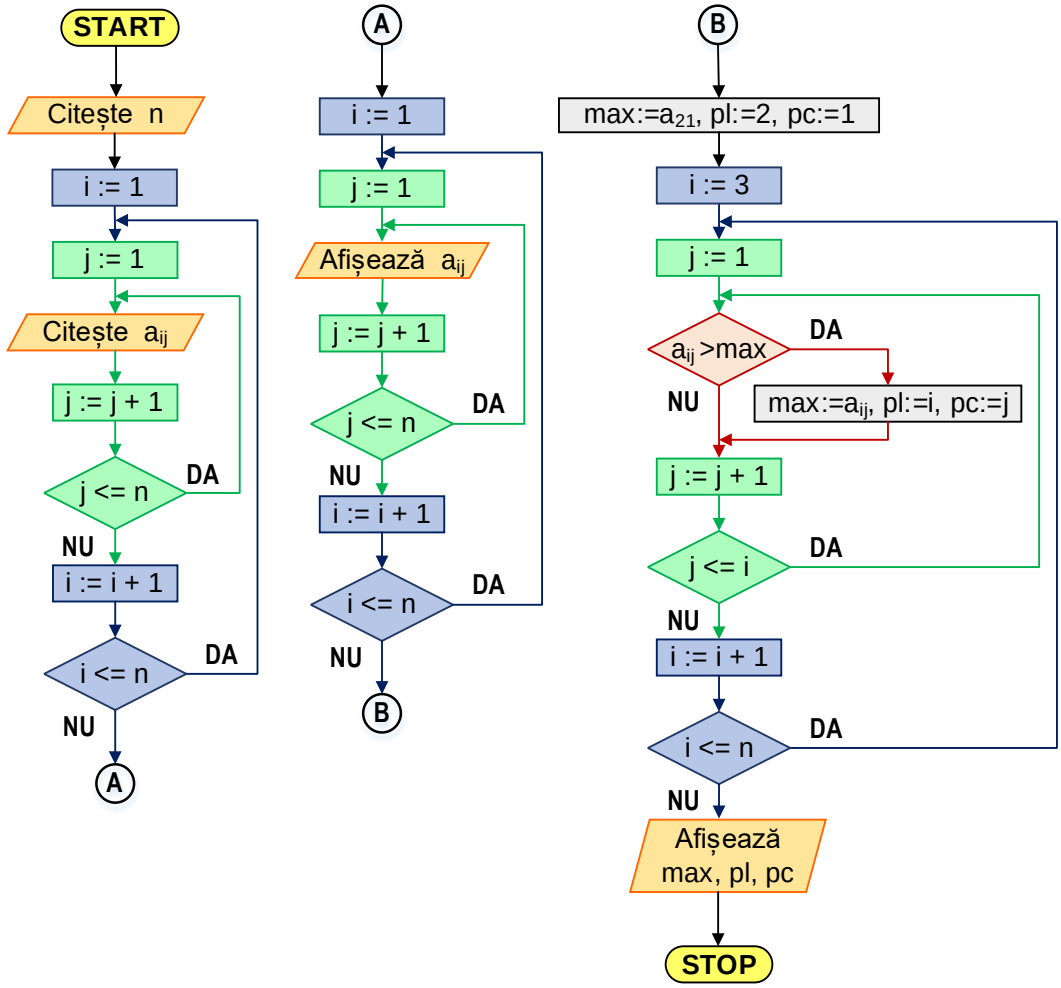
Figura 6.12b. Programul MATLAB şi execuţia acestuia pentru determinarea numărului de elemente pare de pe şi deasupra diagonalei secundare

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

6.13. Determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală

Algoritmul pentru determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală a unei matrice pătratice presupune analizarea elementelor doar din zona dorită.

Schema logică



Pseudocod

Început
  Citeşte n
  Pentru i←1,n
    Pentru j←1,n
      Citeşte a_{ij}
    Sfârşit pentru
  Sfârşit pentru

```
  Pentru i←1,n
    Pentru j←1,n
      Scrie aij
    Sfârşit pentru
  Sfârşit pentru
  Max←a21
  pl←2
  pc←1
  Pentru i←3,n
    Pentru j←1,i
     Dacă aij >max atunci
        Max←aij;   pl←i;  pc←j
    Sfârşit dacă
   Sfârşit pentru
  Sfârşit pentru
  Scrie Max, pl, pc
Sfârşit
```

Figura 6.13a. Reprezentarea algoritmului pentru determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală

Acesta presupune parcurgerea următoarelor etape:

- se citeşte de la tastatură numărul de linii şi de coloane ale matricei, adică variabila n;
- utilizând două cicluri cu contor suprapuse se realizează citirea matricei;
- utilizând două cicluri cu contor suprapuse se realizează afişarea matricei;
- se iniţializează variabila care memorează valoarea elementului maxim cu valoarea elementului situat pe linia a doua şi pe coloana întâi, iar variabila care memorează linia pe care se află elementul maxim primeşte valoarea 2, respectiv variabila care memorează coloana pe care se află elementul maxim primeşte valoarea 1;
- utilizând două cicluri cu contor suprapuse se parcurge zona din matrice de sub diagonala principală, impunând limitele pentru indici, astfel indicele liniei i ia valori de la 3 la n, respectiv indicele coloanei j ia valori de la 1 la i-1. Utilizând o instrucţiune de decizie se verifică pentru fiecare element din această zonă dacă este mai mare decât maximul. Dacă elementul curent este mai mare decât maximul atunci maximul primeşte valoarea elementului curent, variabila care memorează linia pe care se află elementul maxim primeşte valoarea liniei curente, respectiv variabila care memorează coloana pe care se află elementul maxim primeşte valoarea coloanei curente;
- se afişează valoarea elementului maxim, respectiv linia şi coloana pe care se află acesta;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.13a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.13b.

Programul MATLAB	Execuția programului

`n=input(' Introdu n= ');`
`for i=1:n`
`  for j=1:n`
`  s=sprintf(' a[%d,%d] = ',i,j);`
`  a(i,j)=input(s);`
`  end`
`end`
`for i=1:n`
`  s='';`
`  for j=1:n`
	`Introdu n= 5`
	`a[1,1] = 1`
	`a[1,2] = 2`
	`a[1,3] = 3`
	`a[1,4] = 5`
	`a[1,5] = 6`
	`a[2,1] = 4`
	`a[2,2] = 8`
	`a[2,3] = 9`
	`a[2,4] = 2`

```
        s=[s, num2str(a(i,j)), ' '];
    end
    disp(s)
end
Max=a(2,1); pl=2; pc=1;
for i=3:n
    for j=1:i

        if a(i,j)>Max
            Max=a(i,j);   pl=i;    pc=j;
        end
    end
end
s=sprintf('Maximul = %d',Max);
disp(s)
s=sprintf('Se afla pe linia %d si
coloana %d ',pl, pc);
disp(s)
```

```
 a[2,5] = 5
 a[3,1] = 3
 a[3,2] = 0
 a[3,3] = 4
 a[3,4] = 3
 a[3,5] = 7
 a[4,1] = 0
 a[4,2] = 6
 a[4,3] = 5
 a[4,4] = 4
 a[4,5] = 2
 a[5,1] = 3
 a[5,2] = 1
 a[5,3] = 8
 a[5,4] = 7
 a[5,5] = 0
1 2 3 5 6
4 8 9 2 5
3 0 4 3 7
0 6 5 4 2
Maximul = 8
Se afla pe linia 5 si coloana 3
```

Figura 6.13b. Programul MATLAB şi execuţia acestuia pentru determinarea elementului maxim şi a poziţiei acestuia de sub diagonala principală

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

6.14. Generarea unor matrice pătratice după o anumită cerinţă

Se doreşte generarea unor matrice pătratice după o anumită cerinţă. De exemplu, se doreşte elaborarea unui algoritm şi a unui program MATLAB pentru construirea unei o matrice pătratice cu n linii și coloane, numerotate de la 1 la n, după următoarea regulă: fiecare element din matrice aflat pe o linie impară va fi egal cu numărul liniei pe care se află iar fiecare element aflat pe o linie pară va fi egal cu numărul coloanei pe care se află.

Algoritmul pentru rezolvarea cerinței specificate în acest subcapitol presupune parcurgerea următoarelor etape:
- se citește numărul natural n, care reprezintă numărul de linii și coloane ale matricei;
- utilizând două cicluri cu contor suprapuse conduse de variabilele care reprezintă indicii de linie și coloană a elementelor din matrice (i, respectiv j), se generează elementele matricei astfel: se verifică dacă indicele liniei, i, are o valoare pară (mod(i,2) == 0), atunci elementul matricei de pe poziția respectivă va avea valoarea egală cu numărul coloanei pe care se află, adică a[i,j] = j, iar dacă indicele liniei are o valoare impară, elementul matricei va avea valoarea egală cu numărul liniei pe care se află, adică a[i,j] = i;
- utilizând două cicluri cu contor suprapuse se afișează matricea.

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.14a, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.14b.

Schema logică	Pseudocod

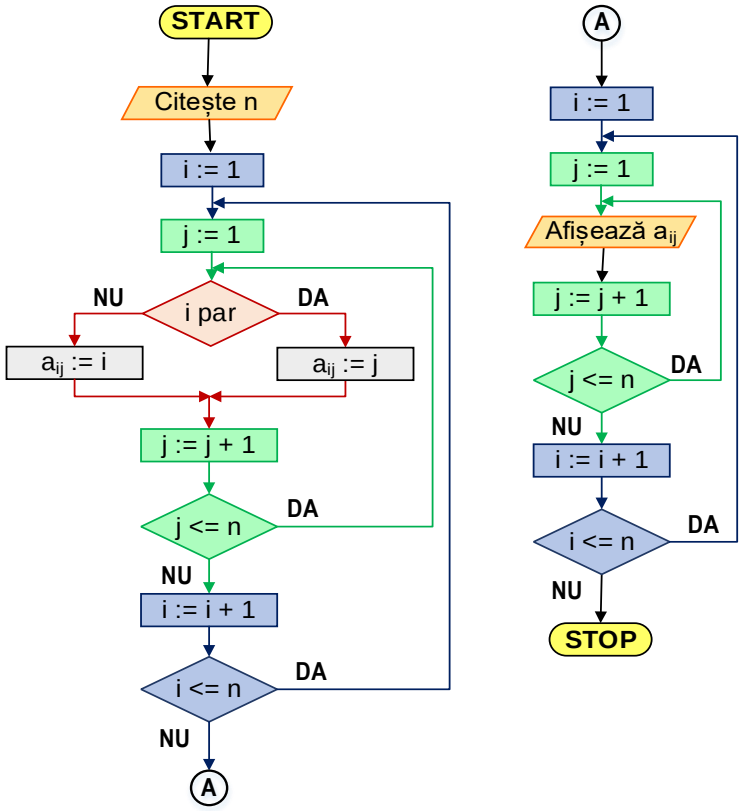
	**Început**
	  Citeşte n
	  Pentru i←1,n
	  Pentru j←1,n
	  Dacă i par **atunci**
	  a_{ij}←j
	  altfel
	  a_{ij}←i
	  Sfârşit dacă
	  Sfârşit pentru
	  Sfârşit pentru
	  Pentru i←1,n
	  Pentru j←1,n
	  Scrie a_{ij}
	  Sfârşit pentru
	  Sfârşit pentru
	Sfârşit

Figura 6.14a. Reprezentarea algoritmului pentru generarea unei matrice particulare

Programul MATLAB	Execuția programului

```
n=input(' Introdu n= ');
for i=1:n
    for j=1:n
        if mod(i,2)==0    a(i,j)=j;
        else              a(i,j)=i;
        end
    end
end
for i=1:n
    s='';
    for j=1:n
        s=[s, num2str(a(i,j)), ' '];
    end
    disp(s)
end
```

```
 Introdu n = 7

     1     1     1     1     1     1     1
     1     2     3     4     5     6     7
     3     3     3     3     3     3     3
     1     2     3     4     5     6     7
     5     5     5     5     5     5     5
     1     2     3     4     5     6     7
     7     7     7     7     7     7     7
```

Figura 6.14b. Programul MATLAB şi execuţia acestuia pentru generarea unei matrice particulare

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

6.15. Generarea unor matrice pătratice după o anumită cerinţă – triunghiul lui Pascal

Triunghiul lui Pascal reprezintă un tablou triunghiular ale cărui elemente sunt numere naturale, putând fi asociat cu o matrice pătratică având elemente în zona de pe şi de sub diagonala principală (figura 6.15a).

	0	1	2	3	4	5	6	7
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
5	1	5	10	10	5	1		
6	1	6	15	20	15	6	1	
7	1	7	21	35	35	21	7	1

Figura 6.15a. Triunghiul lui Pascal

Elementele de pe linia p a matricei ce de dorește a fi generată reprezintă coeficienţii binomiali ai dezvoltării binomului lui Newton:

$$(a+b)^n = C_n^0 \cdot a^n + C_n^1 \cdot a^{n-1} \cdot b^1 + C_n^2 \cdot a^{n-2} \cdot b^2 + \cdots + C_n^k \cdot a^{n-k} \cdot b^k + \cdots + C_n^{n-1} \cdot a^1 \cdot b^{n-1} + C_n^1 b^n$$

Cunoscând că:

$$C_n^k = \begin{cases} 1, & daca\ n = k\ sau\ k = 0 \\ C_{n-1}^{k-1} + C_{n-1}^k, & altfel \end{cases}$$

se poate deduce relaţia de definiţie a valorii elementului matricei, astfel:

$$a_{ij} = \begin{cases} 1, & daca\ i = j\ sau\ j = 0 \\ a_{i-1,j-1} + a_{i-1,j}, & altfel \end{cases}$$

Algoritmul pentru calculul elementelor triunghiului lui Pascal necesită parcurgerea următorilor paşi:

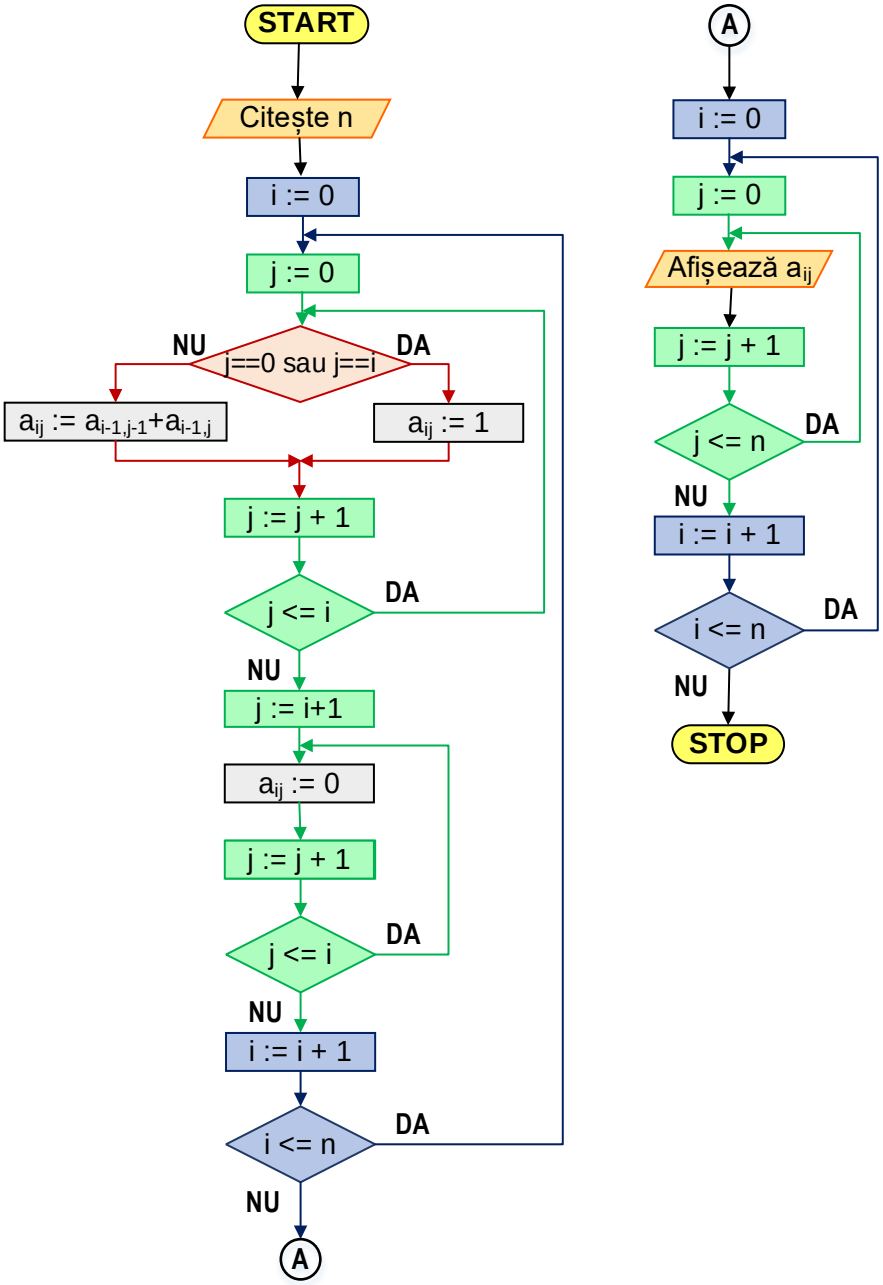
- se citeşte valoarea variabilei **n**;
- cu ajutorul unui ciclu cu contor condus de variabila i se generează fiecare linie a matricei astfel:
  - cu ajutorul unui ciclu cu contor, condus de variabila j, se parcurge linia i element cu element, până când j este egal cu i. Prin intermediul unei instrucţiuni de decizie se verifică dacă j este egal cu 0 sau cu i, situaţie în care elementul matricei va fi egal cu 1, adică a[i,j] = 1. În caz contrar, elementul matricei va fi calculat cu relaţia:

    a[i,j] = a[i – 1,j - 1] + a[i – 1,j]

  - cu ajutorul unui ciclu cu contor, condus tot de variabila j, se parcurge linia i de la elementul cu indicele j + 1 până la cel cu indicele n, aceste elemente primind valoarea 0;
- cu ajutorul a două cicluri cu contor se afişează matricea formată;

Reprezentarea algoritmului prin schemă logică și limbaj pseudocod este prezentată în figura 6.15b, iar programul MATLAB aferent și execuția acestuia sunt ilustrate în figura 6.15c.

Schema logică



Pseudocod

```
Început
  Citeşte n
  Pentru i←0,n
    Pentru j←0,i
      Dacă j=0 SAU j=i atunci
        ai+1j+1←1
      altfel
        ai+1j+1←aij+aij+1
      Sfârşit dacă
    Sfârşit pentru
    Pentru j←i+1,n
```

$a_{i+1j+1} \leftarrow 0$
Sfârşit pentru
Sfârşit pentru
Pentru i←0,n
Pentru j←0,n
Scrie a_{i+1j+1}
Sfârşit pentru
Sfârşit pentru
Sfârşit

Figura 6.15b. Reprezentarea algoritmului pentru generarea unei matrice particulare – triunghiul lui Pascal

Programul MATLAB şi execuţia acestuia

```
n=input(' Introdu n= ');
for i=0:n
    for j=0:i
        if (j==0) || (j == i)
            a(i+1,j+1)=1;
        else
            a(i+1,j+1)=a(i,j)+a(i,j+1);
        end
    end
    for j=i+1:n
        a(i+1,j+1)=0;
    end
end
for i=0:n
    s='';
    for j=0:n
        s=[s, num2str(a(i+1,j+1)), ' '];
    end
    disp(s)
end
```

```
Introdu n = 7

     1     0     0     0     0     0     0     0
     1     1     0     0     0     0     0     0
     1     2     1     0     0     0     0     0
     1     3     3     1     0     0     0     0
     1     4     6     4     1     0     0     0
     1     5    10    10     5     1     0     0
     1     6    15    20    15     6     1     0
     1     7    21    35    35    21     7     1
```

Figura 6.15c. Programul MATLAB şi execuţia acestuia pentru generarea unei matrice particulare – triunghiul lui Pascal

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

Capitolul 7. Şiruri şi serii de numere reale. Dezvoltări în serie de puteri

7.1. Şiruri de numere reale

7.1.1. Noţiuni teoretice

Prin şir de numere reale (denumit simplu şir) se înţelege o funcţie $f: \mathbb{N} \to \mathbb{R}$, care asociază oricărui număr natural *n*, $n \geq 1$, numărul real notat a_n, astfel că: $f(n) = a_n$. Uzual, a_n se numeşte termenul general al şirului, iar *n* se numeşte rangul termenului respectiv.

Un şir poate fi dat fie precizându-se formula termenului general, fie printr-o relaţie de recurenţă.

Un şir este mărginit dacă *Im(f)* este o mulţime mărginită din $\mathbb{R}$, adică dacă şi numai dacă, există $a, b \in \mathbb{R}$ astfel încât: $a \leq a_n \leq b$ pentru orice $n \in \mathbb{N}, n \geq 1$. Dacă $a \leq a_n$, pentru orice $n \in \mathbb{N}, n \geq 1$, se spune că şirul este mărginit inferior, iar dacă $a_n \leq b$, pentru orice $n \in \mathbb{N}, n \geq 1$, se spune că şirul este mărginit superior.

Şirul $a_n, n \geq 1$ este un şir monoton dacă $f(n) = a_n$ este o funcţie monotonă, astfel că şirul este monoton crescător dacă $a_n \leq a_{n+1}$, respectiv şirul este monoton descrescător dacă $a_n \geq a_{n+1}$, oricare ar fi $n \in \mathbb{N}, n \geq 1$.

Numărul real a se numeşte limita şirului a_n, dacă $a_n \to a, n \to \infty$ şi se scrie: $\lim_{n\to\infty} a_n = a$, dacă orice vecinătate a lui a conţine toţi termenii şirului cu excepţia unui număr finit de termeni. Un şir se numeşte convergent dacă are limita un număr real. Un şir care nu este convergent se numeşte divergent.

7.1.2. Determinarea numărului de termeni necesari pentru calculul limitei unui şir, cu o precizie impusă

În acest exemplu se doreşte determinarea numărului de termeni ai şirului şirului: $a_n = \left(1 + \frac{1}{n}\right)^n$, necesari obţinerii valorii limitei acestui şir, cu o precizie de $\varepsilon = 0.000001$. Limita acestui şir este e, baza logaritmului natural, a cărui valoare aproximativă, cu 20 de zecimale este: $e \approx 2{,}71828\ 18284\ 59045\ 23536$.

În acest caz, se calculează elementele şirului până când diferenţa dintre valoarea constantei e şi elementul curent al şirului este mai mică decât precizia impusă.

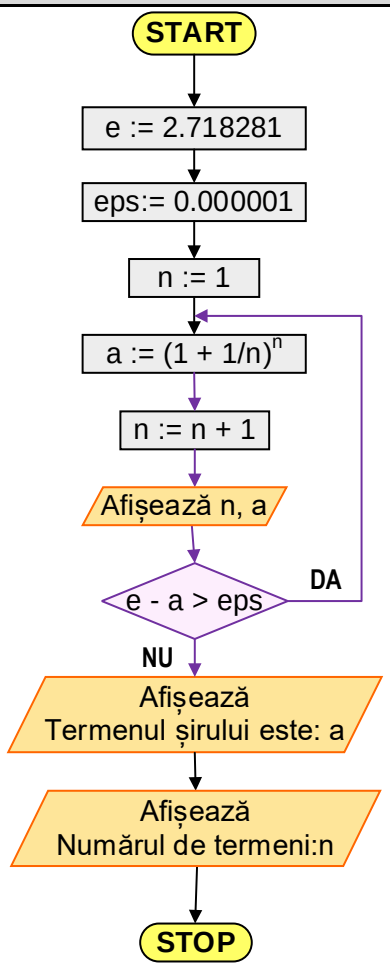
Algoritmul de calcul este următorul:

- se iniţializează variabila **n**, cu valoarea **1**;
- se iniţializează constanta **e**, cu valoarea **2.718281**;
- se iniţializează variabila **eps**, cu valoarea **0.000001**;
- se calculează primul termen în variabila **a**;
- se repetă următoarele operaţii cât timp diferenţa dintre valoarea precizată a constantei **e** şi cea a variabilei **a** (termenul curent) este mai mare decât precizia impusă (valoarea variabilei **eps**):
  - incrementarea valorii variabilei **n**;
  - atribuirea valorii termenului general variabilei **a**;
  - afişarea valorii termenului curent (valoarea variabilei **a**);
- se afişează valoarea ultimului termen calculat;
- se afişează valoarea variabilei **n**, adică numărul de termeni după care se obţine valoarea constantei **e**, cu precizia impusă.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.1a., iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.1b.

Se observă că se obţine valoarea constantei ***e*** cu precizia impusă de **0,000001** după **743255** termeni, valoarea termenului fiind de **2.7182800000259**.

Schema logică	Limbajul pseudocod



Început
n ← 1, e ← 2.718281, eps ← 0.000001
a ← (1 + 1/n)^n
Cât timp (e - a > eps) **execută**
n ← n + 1
a ← (1 + 1/n)^n
Scrie n, a
Sfârșit cât timp
Scrie a
Scrie n
Sfârşit

Figura 7.1a. Reprezentarea algoritmului pentru calculul termenilor şirului $a_n = \left(1+\frac{1}{n}\right)^n$

Programul MATLAB şi execuţia acestuia

```
e=2.718281; eps=0.000001; n=1;
a=(1+1/n)^n;
while e-a>eps
    n=n+1;
    a=(1+1/n)^n;
    s=sprintf('n=%6d \t a=%15.13f',n,a);
    disp(s);
end
s=sprintf(' a= %15.13f',a);
disp(s);
s=sprintf(' n= %d',n);
disp(s);
```

Rularea programului:

```
 ...
n=743253   a=2.7182799996657
n=743254   a=2.7182799998507
n=743255   a=2.7182800000259
a= 2.7182800000259
n= 743255
```

Figura 7.1b. Programul MATLAB şi execuţia acestuia pentru calculul termenilor şirului $a_n = \left(1+\frac{1}{n}\right)^n$

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

7.1.3. Calculul unor sume (produse) de termeni

Se doreşte calculul sumei **S = 1 + 11 + 111 + 1111 + 11111 + ...** utilizând **n** termeni.

Pentru calculul acestor sume (produse) se utilizează o relaţie de forma: **$S = T_1 + T_2 + T_3 + T_4 + T_5 + ...$**, utilizând **n** termeni. Aşa cum s-a arătat anterior, termenul general poate fi precizat fie printr-o formulă, fie printr-o relaţie de recurenţă. În continuare, sunt prezentate ambele variante care pot fi utilizate pentru calculul sumei (produsului).

Varianta I: Termenul general este definit printr-o formulă:

În acest caz, termenul general este definit printr-o relaţie de calcul, de exemplu: $\boldsymbol{T_i = \sum_{j=0}^{i-1} 10^j}$.

Se iniţializează suma **S** cu **0** (produsul **P** cu **1**) şi se utilizează o relaţie de forma: **$S := S + T_i$**, pentru toate elementele de la **T_1** şi până la **T_n**, adică**:** $\boldsymbol{i = \overline{1,n}}$. Calculul sumei **S** se realizează utilizând o instrucţiune repetitivă cu contor, contorul fiind **i**, acesta primind valori de la **1** la **n**, cu pasul **1**. Se observă că fiecare termen este la rândul său o sumă, deci pentru calcul fiecărui termen **T_i** se va realiza iniţializarea acestuia cu **0** şi se va utiliza o instrucţiune repetitivă cu contor, contorul fiind **j**, acesta primind valori de la **0** la **i-1**, conform relaţiei de calcul de mai sus.

Algoritmul de calcul constă în parcurgerea următorilor paşi:

1. Se citeşte numărul de termeni **n**;

2. Se iniţializează suma **S** cu **0** (zero – element neutru pentru operaţia de adunare);

3. Se iniţializează contorul **i** cu **1**, pentru calculul primului termen al sumei;

4. Se verifică condiţia **i ≤ n**, prin care se verifică dacă valoarea contorului este mai mică sau egală cu numărul de termeni, adică dacă nu am depăşit numărul de termeni specificaţi anterior.

Dacă condiţia este **adevărată**, se parcurg în continuare următorii paşi:

1.1. Se iniţializează termenul curent **T_i** cu **0**, calculul acestuia realizându-se ca o sumă de puteri a lui **10**;

1.2. Se iniţializează contorul **j**, cu valoarea **0**;

1.3. Se verifică condiţia **j <= i-1**. Dacă condiţia este **adevărată** se modifică valoarea termenului general, astfel: $T_i := T_i + 10^j$. Dacă condiţia este falsă se trece la pasul **5**.

1.4. Se modifică valoarea contorului **j**, cu pasul **1**, astfel: **j := j + 1** după care se revine la pasul 4.3;

Dacă condiţia este **falsă**, înseamnă că s-au calculat şi adunat la sumă toţi termenii sumei, astfel că se părăseşte instrucţiunea repetitivă utilizată pentru calculul sumei, algoritmul continuând cu pasul **7**;

5. În acest pas, termenul general al sumei de la pasul **i**, adică **T_i**, este calculat astfel că se adaugă valoarea acestuia la suma calculată anterior, adică se utilizează relaţia: **$S := S + T_i$**;

6. Se modifică valoarea contorului **i**, cu pasul **1**, astfel: **i := i + 1** după care se revine la pasul **4**;

7. Se afişează valoarea calculată a sumei **S**;

În tabelul următor (tabelul 7.1) este prezentat algoritmul de calcul al sumei pentru n = 3.

Tabelul 7.1. Algoritmul de calcul al sumei S = 1 + 11 + 111 – varianta I

Etapa:	n	i	i <= n	j	j <= i - 1	T_i	S
0	3						0
1.1		1	DA			$T_1 := 0$	
1.2				0	DA	$T_1 := 0 + 10^0 = 0 + 1 = 1$	
1.3				1	NU		$S := 0 + T_1 = 0 + 1 = 1$
2.1		2	DA			$T_2 := 0$	
2.2				0	DA	$T_2 := 0 + 10^0 = 0 + 1 = 1$	
2.3				1	DA	$T_2 := 1 + 10^1 = 1 + 10 = 11$	
2.4				2	NU		$S := 1 + T_2 = 1 + 11 = 12$
3.1		3	DA			$T_3 := 0$	
3.2				0	DA	$T_3 := 0 + 10^0 = 0 + 1 = 1$	
3.3				1	DA	$T_3 := 1 + 10^1 = 1 + 10 = 11$	
3.4				2	DA	$T_3 := 11 + 10^2 = 11 + 100 = 111$	
3.5				3	NU		$S := 1 + 11 + T_3 = 1 + 11 + 111 = 123$
4.1		4	NU				**S := 1 + 11 + 111 = 123**

Etapa **0**: Se citeşte numărul de termeni, **n**, în acest caz **n = 3** şi se iniţializează suma **S** cu **0**, adică **S := 0**;
Etapa **1.1**: Contorul **i** primeşte valoarea **1**, adică **i := 1**. Se verifică dacă se îndeplineşte condiţia **i <= n**, deoarece **1 < 3**, condiţia este **adevărată**, se continuă pe ramura **DA**. Se iniţializează termenul T_i cu zero, adică T_1 **:= 0** (în acest caz **i = 1**);
Etapa **1.2**: Se iniţializează contorul **j** cu valoarea **0**, adică **j := 0** şi se verifică condiţia **j <= i-1**. Deoarece **0 = 0**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_1 cu relaţia $T_i := T_i + 10^j$, adică: $T_1 := 0 + 10^0 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 0 + 1 = 1**;
Etapa **1.3**: Cu valoarea nouă a contorului **j** se verifică condiţia **j <= i-1**. Deoarece **1 > 0**, condiţia este **falsă** şi se continuă pe ramura **NU**, modificându-se valoarea sumei **S** după relaţia **S := S +** T_i, adică $S := 0 + T_1 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **i**, după relaţia **i := i + 1**, deci **i := 1 + 1 = 2**;
Etapa **2.1**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **2 < 3**, condiţia este **adevărată** şi se continuă pe ramura **DA**. Se iniţializează termenul T_i cu zero, adică T_2 **:= 0** (în acest caz **i = 2**);
Etapa **2.2**: Se iniţializează din nou contorul **j** cu valoarea **0**, adică **j := 0** şi se verifică condiţia **j <= i-1**. Deoarece **0 < 1**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_2 cu relaţia $T_i := T_i + 10^j$, adică: $T_2 := 0 + 10^0 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 0 + 1 = 1**;
Etapa **2.3**: Cu valoarea nouă a contorului **j**, adică **j := 1** şi se verifică condiţia **j <= i-1**. Deoarece **1 = 1**, condiţia este **adevărată** (îndeplinită prin egalitate) şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_2 cu relaţia $T_i := T_i + 10^j$, adică: $T_2 := 1 + 10^1 = 1 + 10 = 11$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 1 + 1 = 2**;
Etapa **2.4**: Cu valoarea nouă a contorului **j** se verifică condiţia **j <= i-1**. Deoarece **2 > 1**, condiţia este **falsă** şi se continuă pe ramura **NU**, modificându-se valoarea sumei **S** după relaţia **S := S +** T_i, adică $S := 1 + T_2 = 1 + 11 = 12$. În continuare, se modifică valoarea contorului **i**, după relaţia **i := i + 1**, deci **i := 2 + 1 = 3**;
Etapa **3.1**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **3 = 3**, condiţia este **adevărată** şi se continuă pe ramura **DA**. Se iniţializează termenul T_i cu zero, adică T_3 **:= 0** (în acest caz **i = 3**);
Etapa **3.2**: Se iniţializează din nou contorul **j** cu valoarea **0**, adică **j := 0** şi se verifică condiţia **j <= i-1**. Deoarece **0 < 2**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_3 cu relaţia $T_i := T_i + 10^j$, adică: $T_3 := 0 + 10^0 = 0 + 1 = 1$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 0 + 1 = 1**;
Etapa **3.3**: Cu valoarea nouă a contorului **j**, adică **j := 1** şi se verifică condiţia **j <= i-1**. Deoarece **1 < 2**, condiţia este **adevărată** şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_3 cu relaţia $T_i := T_i + 10^j$, adică: $T_3 := 1 + 10^1 = 1 + 10 = 11$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 1 + 1 = 2**;
Etapa **3.4**: Cu valoarea nouă a contorului **j**, adică **j := 2** şi se verifică condiţia **j <= i-1**. Deoarece **2 = 2**, condiţia este **adevărată** (îndeplinită prin egalitate) şi se continuă pe ramura **DA**, se calculează valoarea nouă a termenului T_3 cu relaţia $T_i := T_i + 10^j$, adică: $T_3 := 11 + 10^2 = 11 + 100 = 111$. În continuare, se modifică valoarea contorului **j** după relaţia **j := j + 1**, deci **j := 2 + 1 = 3**;
Etapa **3.5**: Cu valoarea nouă a contorului **j** se verifică condiţia **j <= i-1**. Deoarece **3 > 2**, condiţia este **falsă** şi se continuă pe ramura **NU**, modificându-se valoarea sumei **S** după relaţia **S := S +** T_i, adică $S := 12 + T_3 = 12 + 111 = 123$. În continuare, se modifică valoarea contorului **i**, după relaţia **i := i + 1**, deci **i := 3 + 1 = 4**;
Etapa **4.1**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **4 > 3**, condiţia este **falsă** şi se continuă pe ramura **NU**, ceea ce înseamnă că se continuă cu afişarea valorii variabilei suma, adică a variabilei **S**, după care se încheie algoritmul.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.1c, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.1d.

Schema logică	Limbajul pseudocod

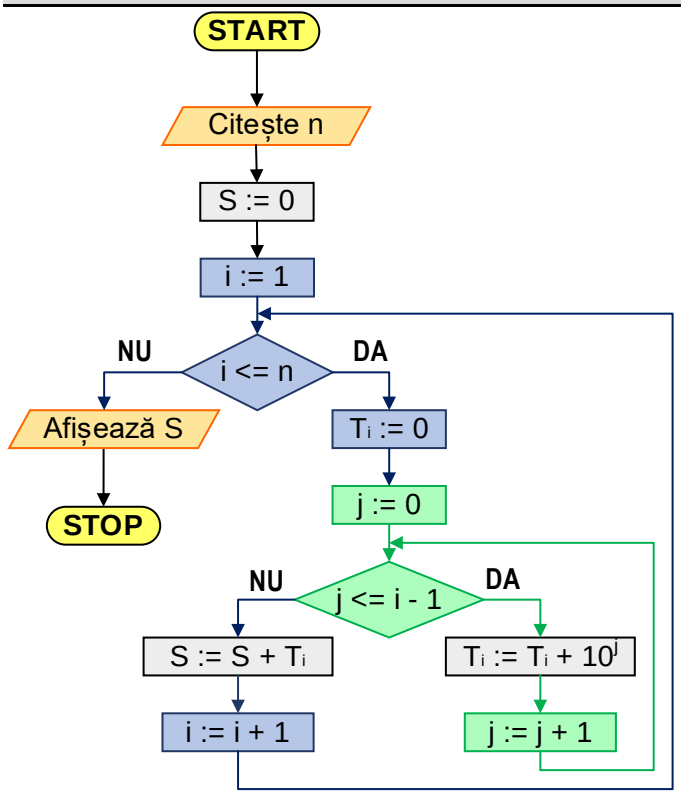
	**Început**
	Citeşte n
	S ← 0
	Pentru i = 1, n
	  T[i] ← 0
	  Pentru j = 0, i-1
	  T[i] ← **T[i]** + 10^j
	  Sfârşit pentru
	  S ← **S** + **T[i]**
	Sfârşit pentru
	Scrie S
	Sfârşit

Figura 7.1c. Reprezentarea algoritmului pentru calculul sumei **S = 1 + 11 + 111 + 1111 + ...**

Programul MATLAB şi execuţia acestuia	

`n=input('Introduceti n=');`
`S=0;`
`for i=1:n`
`  T(i)=0;`
`  for j=0:i-1`
`  T(i)=T(i)+10^j;`
`  end`
`  S=S+T(i);`
`end`
`s=sprintf(' Suma este S = %d',S);`
`disp(s);`
	Rularea programului – cazul 1:
	Introduceti n = **3**
	Suma este S = 123
	Rularea programului – cazul 2:
	Introduceti n = **5**
	Suma este S = 12345

Figura 7.1d. Programul MATLAB şi execuţia acestuia pentru calculul sumei **S = 1 + 11 + 111 + 1111 + ...**

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Varianta II: Termenul general este definit printr-o relaţie de recurenţă:

În acest caz, termenul general T_i este definit printr-o relaţie de recurență în care apare termenul anterior T_{i-1}, de exemplu: $\boldsymbol{T_i = T_{i-1} \cdot 10 + 1}$.

În această variantă, se iniţializează primul termen cu valoarea sa, adică $T_1 = 1$ şi suma **S** cu valoarea primului termen, adică $S = T_1$. Pentru calculul sumei **S** se utilizează o instrucțiune repetitivă cu contor, contorul fiind **i**, acesta primind valori de la **2** la **n**, cu pasul **1**. În cadrul acestei instrucţiuni repetitive se realizează calculul valorii termenului curent T_i cu relaţia $T_i := T_{i-1} * 10 + 1$ (relaţia de recurenţă), precum şi calculul sumei S cu relaţia $S := S + T_i$.

Algoritmul de calcul constă în parcurgerea următorilor paşi:

1. Se citeşte numărul de termeni **n**;

2. Se iniţializează primul termen cu valoarea sa, adică $T_1 = 1$ şi suma **S** cu valoarea primului termen, adică $S = T_1$;

3. Se iniţializează contorul **i** cu **2**;

4. Se verifică condiţia **i ≤ n**, prin care se verifică dacă valoarea contorului este mai mică sau egală cu numărul de termeni, adică dacă nu am depăşit numărul de termeni specificaţi anterior.

Dacă condiţia este **adevărată**, se calculează valoarea termenului curent T_i cu relaţia $T_i := T_{i-1} * 10 + 1$ şi se modifică valoarea sumei, conform relaţiei $S := S + T_i$;

Dacă condiţia este **falsă**, înseamnă că s-au calculat şi adunat la sumă toţi termenii acesteia, astfel că se părăseşte instrucţiunea repetitivă utilizată pentru calculul sumei, algorimul continuând cu pasul **6**;

5. Se modifică valoarea contorului **i**, cu pasul **1**, astfel: **i := i + 1** după care se revine la pasul **4**;

6. Se afişează valoarea calculată a sumei **S**.

Schema logică	Limbajul pseudocod

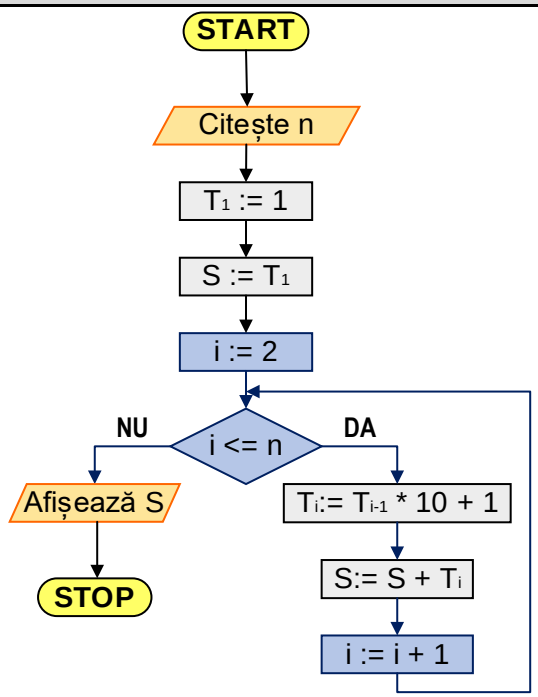
	**Început**
	Citeşte n
	T[1] ← 1, **S** ← **T[1]**
	Pentru i = 2, n
	  T[i] ← **T[i]** * 10 + 1
	  S ← **S** + **T[i]**
	Sfârşit pentru
	Scrie S
	Sfârşit

Figura 7.1e. Reprezentarea algoritmului pentru calculul sumei S = 1 + 11 + 111 + 1111 + ...

Programul MATLAB şi execuţia acestuia	

`n=input('Introduceti n=');`
`T(1)=1;S=T(1);`
`for i=2:n`
`  T(i)=T(i-1)*10+1;  S=S+T(i);`
`end`
`s=sprintf(' Suma este S = %d',S);`
`disp(s);`
	Rularea programului – cazul 1:
	Introduceti n = **3**
	Suma este S = 123
	Rularea programului – cazul 2:
	Introduceti n = **6**
	Suma este S = 123456

Figura 7.1f. Programul MATLAB şi execuţia acestuia pentru calculul sumei S = 1 + 11 + 111 + 1111 + ...

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

În tabelul următor (tabelul 7.2) este prezentat algoritmul de calcul al sumei pentru n = 3.

Tabelul 7.2. Algoritmul de calcul al sumei S = 1 + 11 + 111 – varianta II

Etapa:	n	i	i <= n	T_i	S
0	3	1		T_1 := 1	S := T_1 = 1
1		2	DA	T_2 := T_1 * 10 + 1 = 1 * 10 + 1 = 11	S := 1 + T_2 = 1 + 11 = 12
2		3	DA	T_3 := T_2 * 10 + 1 = 11 * 10 + 1 = 111	S := 12 + T_3 = 12 + 111 = 123
3		4	NU		**S := 1 + 11 + 111 = 123**

Etapa **0**: Se citeşte numărul de termeni, **n**, în acest caz **n = 3** şi se iniţializează primul termen cu valoarea sa, adică **T_1 = 1** şi suma **S** cu valoarea primului termen, adică **S := T_1**;
Etapa **1**: Contorul **i** primeşte valoarea **2**, adică **i := 2**. Se verifică dacă se îndeplineşte condiţia **i <= n** şi deoarece **2 < 3**, condiţia este **adevărată**, se continuă pe ramura **DA**. Se calculează termenul curent **T_i** cu relaţia **T_i := T_{i-1} * 10 + 1**, adică **T_2 := 1 * 10 + 1**, deci **T_2 = 11**. Se calculează valoarea sumei cu ajutorul relaţiei **S := S + T_i**, adică **S : = 1 + 11 = 12** şi se modifică valoarea contorului **i** cu relaţia: **i := i + 1**, deci **i := 2 + 1 = 3**;
Etapa **2**: Cu valoarea nouă a contorul **i**, adică **i := 3** se verifică dacă se îndeplineşte condiţia **i <= n**. Deoarece **3 = 3**, condiţia este **adevărată**, se continuă pe ramura **DA**. Se calculează termenul curent **T_i** cu relaţia **T_i := T_{i-1} * 10 + 1**, adică **T_3 := 11 * 10 + 1**, deci **T_3 = 111**. Se calculează valoarea sumei cu ajutorul relaţiei **S := S + T_i**, adică **S : = 12 + 111 = 123** şi se modifică valoarea contorului **i** cu relaţia: **i := i + 1**, deci **i := 3 + 1 = 4**;
Etapa **3**: Cu valoarea nouă a contorului **i**, se verifică condiţia **i <= n**. Deoarece **4 > 3**, condiţia este **falsă** şi se continuă pe ramura **NU**, ceea ce înseamnă că se continuă cu afişarea valorii variabilei **S**, după care se încheie algoritmul.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.1e, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.1f.

7.2. Serii de numere reale

Se consideră $(a_n)_{n\geq 1}$ un şir de numere reale şi $(s_n)_{n\geq 1}$ şi un şir de numere reale definit prin relaţia: $s_n = \sum_{k=1}^{n} a_k$. Perechea de şiruri ($(a_n)_{n\geq 1}$, $(s_n)_{n\geq 1}$) poartă numele de serie de numere reale şi se notează: $a_1 + a_2 + \cdots + a_n + \cdots$ sau $\sum_{n=1}^{\infty} a_n$. Valorile $a_1,\ a_2, \ldots$ se numesc termenii seriei, a_n se numeşte termenul general al seriei, iar şirul s_n se numeşte şirul sumelor parţiale ale seriei. Dacă şirul $(s_n)_{n\geq 1}$ are limita s (finită sau infinită), deci dacă există $s = \lim_{n\to\infty} s_n$ se scrie $s = \sum_{n=1}^{\infty} a_n$ sau $s = a_1 + a_2 + \cdots + a_n + \cdots$ şi se spune că suma seriei este s.

Egalitatea $s = \sum_{n=1}^{\infty} a_n$ reprezintă verbal: „suma seriei $\sum_{n=1}^{\infty} a_n$ este s şi semnifică faptul că şirul sumelor parţiale ale seriei are limita s.

Dacă şirul sumelor parţiale ale unei serii nu are limită, seria respectivă se numeşte *serie oscilantă*.

Dacă şirul sumelor parţiale ale unei serii este convergent seria se numeşte serie *convergentă*, în caz contrar seria se numeşte *divergentă*.

Fie seria $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$. Termenul general al acestei serii este $a_n = \frac{1}{n(n+1)}, n \geq 1$.

Şirul sumelor parţiale are termenul general $s_n = a_1 + a_2 + \cdots + a_n = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \cdots + \frac{1}{n(n+1)},\ n \geq 1$.

Ţinând cont că: $\frac{1}{k(k+1)} = \frac{1}{k} - \frac{1}{k+1}, k = 1,2, \ldots, n$ se obţine: $s_n = \frac{1}{1} - \frac{1}{2} + \frac{1}{2} - \frac{1}{3} + \cdots + \frac{1}{n} - \frac{1}{n+1} = 1 - \frac{1}{n+1}$.

Aşadar: $\lim_{n\to\infty} s_n = \lim_{n\to\infty} \left(1 - \frac{1}{n+1}\right) = 1$, astfel că: $\sum_{n=1}^{\infty} \frac{1}{n(n+1)} = 1$.

Algoritmul de calcul este următorul:
1. Se citeşte numărul de termeni, **n**;
2. Se iniţializează suma **S** cu **zero** (0 este element neutru pentru operaţia de adunare);
3. Se utilizează o instrucțiune de ciclare cu contor, contorul notat cu **i** ia valori de la **1** la **n**, în cadrul căruia se calculează valoarea termenului curent (în funcţie de i), precum şi calculul sumei pe baza relaţiei **S := S + a_i**.
4. După parcurgerea ciclului cu contor, se realizează afişarea valorii variabilei **S**;

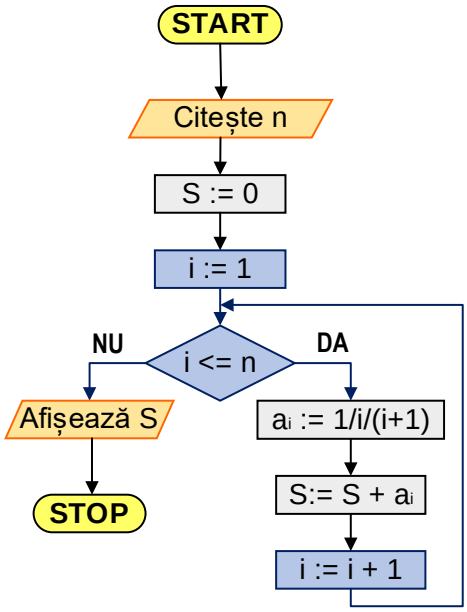
Schema logică	Limbajul pseudocod
	**Început** **Citeşte n** **S** ← 0 **Pentru i** = 1, n **a[i]** ← 1 / (**i** (**i** + 1)) **S** ← **S** + **a[i]** **Sfârşit pentru** **Scrie S** **Sfârşit**

Figura 7.2a. Reprezentarea algoritmului pentru calculul sumei $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.2a, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.2b.

Programul MATLAB şi execuţia acestuia	

`n=input('Introduceti n=');`
`S=0;`
`for i=1:n`
`  a(i)=1/(i*(i+1));`
`  S=S+a(i);`
`end`
`s=sprintf(' Suma este S = %f',S);`
`disp(s);`
	Rularea programului – cazul 1:
	Introduceti n = **500**
	Suma este S = 0.998004
	Rularea programului – cazul 2:
	Introduceti n = **999**
	Suma este S = 0.999000

Figura 7.2b. Programul MATLAB şi execuţia acestuia pentru calculul sumei $\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

7.3. Dezvoltări în serie de puteri

Seriile de puteri reprezintă o extindere a conceptului de funcţii polinomiale, dar şi o clasă particulară de serii de funcţii. Se numeşte serie de puteri o serie de funcţii $\sum_{i=0}^{\infty} f_i$ în care $f_i(x) = a_i \cdot x^i$ sau $f_i(x) = a_i \cdot (x-a)^i$, unde $a_0, a_1, a_2, \dots, a_i, \dots$ reprezintă coeficienţii seriei de puteri, iar $x \in \mathbb{R}$. Numărul a_i se numeşte coeficientul termenului de rang i.

Aşadar, o serie de puteri este o serie de forma:

$$a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_i \cdot x^i + \cdots = \sum_{i=0}^{\infty} a_i \cdot x^i$$

sau de forma:

$$a_0 + a_1 \cdot (x-a) + a_2 \cdot (x-a)^2 + \cdots + a_i \cdot (x-a)^i + \cdots = \sum_{i=0}^{\infty} a_i \cdot (x-a)^i$$

formă care poartă numele de serie de puteri centrată în punctul a.

O reprezentare a unei funcţii ca o sumă infinită de termeni calculaţi cu valorile derivatelor sale într-un punct poartă numele de serie Taylor. Dacă seria utilizează derivatele în zero, atunci ea poartă numele de serie MacLaurin.

Astfel, seria Taylor a unei funcţii reale sau complexe f care este funcţie indefinit derivabilă pe o vecinătate a unui număr real sau complex a, este seria de puteri:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

unde: $n!$ este factorialul lui n, iar $f^n(a)$ este a n-a derivată a lui f în punctul a.

Adesea, $f(x)$ este egală cu seria sa Taylor evaluată în x pentru orice x suficient de apropiat de a.

Exemple de serii de puteri şi serii Taylor remarcabile:

a) $\frac{1}{1-x} = 1 + x + x^2 + \cdots + x^n, x \in (-1,1)$

b) $ln\frac{1+x}{1-x} = 2x\left[1 + \frac{x^2}{3} + \frac{x^4}{5} + \cdots + \frac{x^{2n}}{2n+1}\right]$

c) $arctg\ x = \frac{x}{1} - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots + (-1)^n \frac{x^{2n+1}}{2n+1}$

d) $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{(-1)^n}{2n+1}$

e) $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} = \sum_{i=0}^{n} \frac{x^i}{i!}$

f) $e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} = \sum_{i=0}^{n} \cdot (-1)^i \cdot \frac{x^i}{i!}$

g) $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!} = \sum_{i=0}^{n} \frac{1}{i!}$

h) $\frac{1}{e} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \ldots + \frac{(-1)^n}{n!} = \sum_{i=0}^{n} \cdot (-1)^i \cdot \frac{1}{i!}$

i) $\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots = \sum_{i=0}^{n} (-1)^i \cdot \frac{x^{2i+1}}{(2i+1)!}$

j) $\cos x = x - \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots = \sum_{i=0}^{n} (-1)^i \cdot \frac{x^{2i}}{(2i)!}$

7.3.1. Calculul funcţiei sin(x)

Se consideră dezvoltarea în serie de puteri: $\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$. Se cere să se determine şi să se afişeze valoarea aproximativă calculată pentru **n** termeni şi să se compare valoarea obţinută cu valoarea calculată a funcţiei **sinus** cu ajutorul funcţiei implementate în MATLAB.

Varianta I: Termenul general se calculează cu relaţia: $T_i = (-1)^{i+1} \cdot \frac{x^{2i-1}}{(2i-1)!}$

Se observă că fiecare termen conţine un produs, adică $(\mathbf{2}\boldsymbol{i} - \mathbf{1})!$. Astfel că, pentru fiecare termen, adică pentru fiecare **i**, trebuie să calculăm $(\mathbf{2}\boldsymbol{i} - \mathbf{1})!$, deci un produs.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3a., iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.3b.

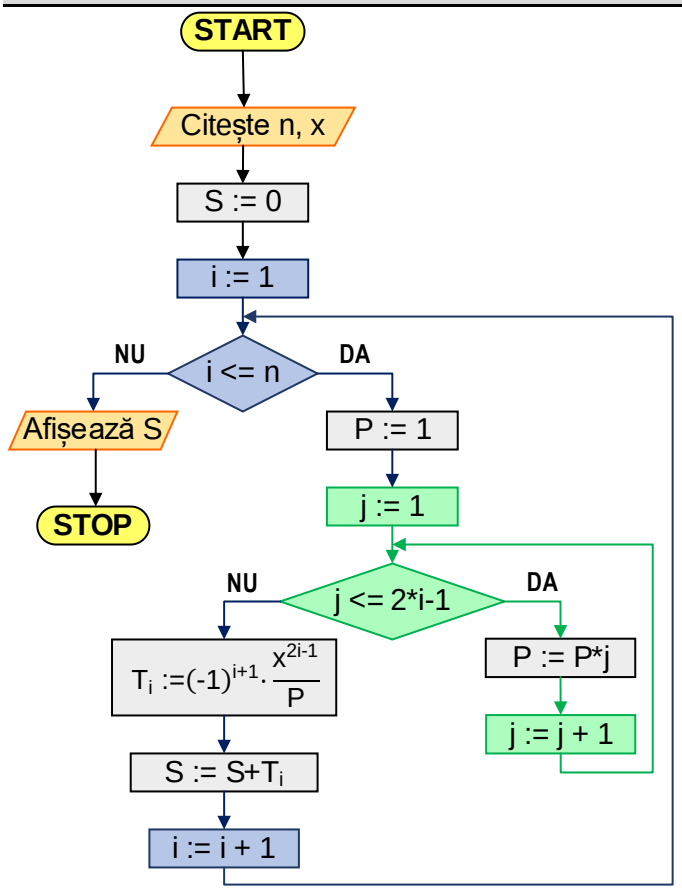
Schema logică	Limbajul pseudocod
	**Început** **Citeşte n** **Citeşte x** **S ← 0** **Pentru i = 1, n** **P ← 1** **Pentru j = 1, 2*i-1** **P ← P * j** **Sfârşit pentru** **T[i] = (-1)$^{i+1}$ * x$^{(2i-1)}$ / P** **S ← S + T[i]** **Sfârşit pentru** **Scrie S, sin(x)** **Sfârşit**

Figura 7.3a. Reprezentarea algoritmului pentru calculul funcţiei **sin(x)** – varianta I

Algoritmul de calcul este următorul:

1. Se citeşte numărul de termeni ai seriei de puteri, adică valoarea variabilei **n**;

2. Se citeşte argumentul funcţiei sinus, adică valoarea variabilei x;

3. Se iniţializează suma **S**, cu valoarea **0** (**zero** – element neutru pentru adunare);

4. Se utilizează o instrucţiune de ciclare cu contor (în acest caz contorul este variabila **i**), pentru calculul fiecărui termen T_i şi adunarea acestuia la sumă astfel:

4.1. Se iniţializează contorul **i**, cu valoarea **1**;

4.2. Se verifică condiţia $i <= n$, adică se verifică dacă valoarea contorului este în intervalul $[1, n]$.

Dacă condiţia este **adevărată**, se trece la pasul următor (pasul 4.3). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare, algoritmul continuă cu pasul **5**;

4.3. Se iniţializează produsul **P** cu valoarea **1** (unu este element neutru pentru adunare);

4.4. Se utilizează o instrucţiune de ciclare cu contor (contorul fiind variabila **j**) pentru calculul factorialului;

4.5. Se calculează termenul T_i al cărui numitor este valoarea produsului **P** calculat anterior;

4.6. Se adună termenul T_i la suma **S**, utilizând o relaţie de forma **S := S + T_i**;

4.7. Se modifică valoarea contorului **i**, utilizând o relaţie de forma: **i := i + 1**. Se revine la pasul 4.2;

5. Se afişează valoarea calculată a sumei, precum şi valoarea calculată a funcţiei **sin(x)** utilizând funcţia implementată în MATLAB.

Programul MATLAB şi execuţia acestuia

```
n=input('Introduceti n=');
x=input('Introduceti x=');
S=0;
for i=1:n
    P=1;
    for j=1:2*i-1
        P=P*j;
    end
    T(i)=(-1)^(i+1)*x^(2*i-1)/P;
    S=S+T(i);
end
s=sprintf(' Suma este S = %11.9f',S);
disp(s);
s=sprintf(' sin(%6.4f) = %11.9f',x,sin(x));
disp(s);
```

```
Rularea programului – cazul 1:
 Introduceti n = 10
 Introduceti x = 0
 Suma este S = 0.000000000
 sin(0.0000) = 0.000000000

Rularea programului – cazul 2:
 Introduceti n = 10
 Introduceti x = 0.523598
 Suma este S = 0.499999328
 sin(0.5236) = 0.499999328

Rularea programului – cazul 3:
Introduceti n = 10
Introduceti x = 1.047
Suma este S = 0.865926611
sin(1.0470) = 0.865926611
```

Figura 7.3b. Programul MATLAB şi execuţia acestuia pentru calculul funcţiei **sin(x)** – varianta I

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Trebuie menționat faptul că al doilea ciclu for (cel scris cu verde) împreună cu instrucțiunea de inițializare **P=1** pot fi înlocuite cu apelul funcției **factorial,** funcție care este implementată în MATLAB. Practic se înlocuiesc cu instrucțiunea **P=factorial(2*i-1)**.

Varianta II: Termenul general se calculează cu relaţia: $T_i = T_{i-1} \cdot \left(-\frac{x^2}{(2\cdot i-1)\cdot(2\cdot i-2)}\right)$

În acest caz, termenul general se exprimă în funcţie de termenul anterior, conform relaţiei de mai sus.

Algoritmul de calcul este următorul:

1. Se citeşte numărul de termeni **n**;

2. Se citeşte argumentul **x**, al funcţiei sinus;

3. Se iniţializează primul termen cu valoarea sa, în acest caz **T[1] = x**;

4. Se iniţializează valoarea sumei cu valoarea primului termen, adică **S = T[1]**;

5. Se utilizează o instrucţiune de ciclare cu contor, acesta luând valori de la **2** la **n**, în cadrul căreia se realizează următoarele etape:

5.1. se iniţializează contorul **i**, cu valoarea **2**, astfel: **i = 2**;

5.2. se verifică condiţia **i <= n** prin care se verifică dacă valoarea curentă a contorului este mai mică decât **n** - numărul de termeni precizat anterior;

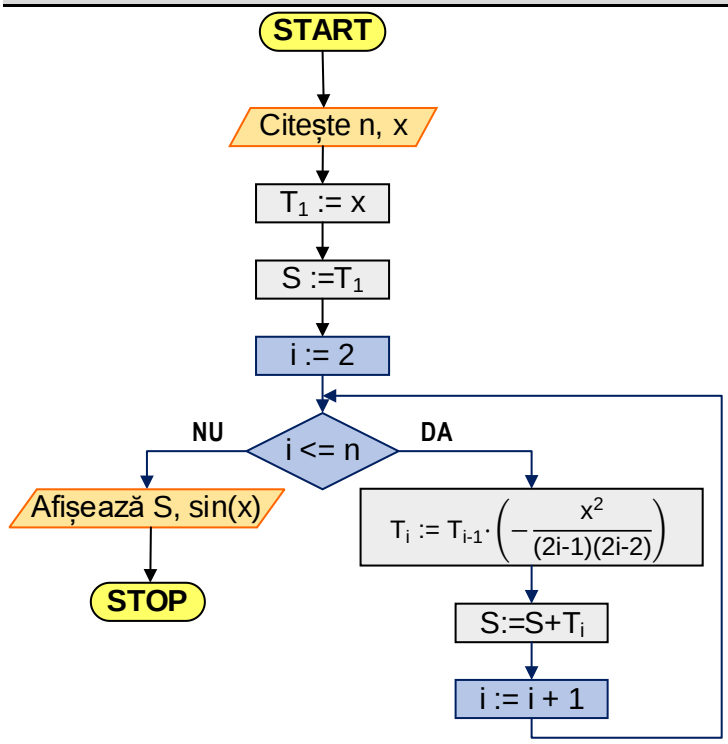
Dacă condiţia este **adevărată**, se calculează valoarea termenului curent cu relaţia de mai sus şi se adaugă la sumă valoarea termenului curent, utilizând relaţia **S = S + T[i]** şi se continuă cu incrementarea valorii contorului (etapa **5.3**). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul **6**;

5.3. Se incrementează valoarea contorului **i**, adică **i = i + 1**, după care se revine la pasul **5.2**;

6. Se afişează valoarea calculată a sumei **S**, respectiv se afişează valoarea funcţiei sinus cu argumentul **x**, adică **sin(x)**;

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3c, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.3d.

Schema logică	Limbajul pseudocod




```
Început
Citeşte n
Citeşte x
T[1] = x
S ← T[1]
Pentru i = 2, n
    T[i] = T[i-1]*(-1*x*x/((2*i-1)*(2*i-2)))
    S ← S + T[i]
Sfârşit pentru
Scrie S, sin(x)
Sfârşit
```

Figura 7.3c. Reprezentarea algoritmului pentru calculul funcţiei **sin(x)** – varianta II

Programul MATLAB şi execuţia acestuia

```
n=input('Introduceti n=');
x=input('Introduceti x=');
T(1)=x; S=T(1);
for i=2:n
    T(i)=T(i-1)*(-1)*x*x/((2*i-1)*(2*i-2));
    S=S+T(i);
end
s=sprintf('Suma este S = %11.9f',S);
disp(s);
s=sprintf('sin(%6.4f) = %11.9f',x,sin(x));
disp(s);
```

```
Rularea programului – cazul 1:
 Introduceti n = 10
 Introduceti x = 0.523598
 Suma este S = 0.499999328
 sin(0.5236) = 0.499999328

Rularea programului – cazul 2:
 Introduceti n = 10
 Introduceti x = 1.047
 Suma este S = 0.865926611
 sin(1.0470) = 0.865926611
```

Figura 7.3d. Programul MATLAB şi execuţia acestuia pentru calculul funcţiei **sin(x)** – varianta II

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

7.3.2. Calculul valorii constantei π

Varianta I: Se consideră următoarea relaţie de calcul pentru constanta **π** (serie Gregory):

$$\pi = 4 \cdot \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 4 \cdot \left(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right)$$

Se doreşte să se calculeze şi să se afişeze valoarea aproximativă a lui π, pentru n termeni luaţi în considerare şi să se compare această valoare cu valoarea constantei dată prin funcția implementată în MATLAB.

Formula termenului general este, în acest caz: $T_i = \frac{(-1)^i}{2i+1}$

Algoritmul de calcul, în acest caz, este următorul:

1. Se citeşte numărul de termeni ai seriei de puteri, adică valoarea variabilei **n**;

2. Se iniţializează suma **S**, cu valoarea **0** (**zero** – element neutru pentru adunare);

3. Se utilizează o instrucţiune de ciclare cu contor (în acest caz contorul este variabila **i**), pentru calculul fiecărui termen T_i şi adunarea acestuia la sumă astfel:

3.1. Se iniţializează contorul **i**, cu valoarea **0**;

3.2. Se verifică condiţia $i <= n$, adică se verifică dacă valoarea contorului este în intervalul $[1, n]$.

Dacă condiţia este **adevărată**, se trece la pasul următor (pasul 3.3). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare, algoritmul continuă cu pasul **4**;

3.3. Se calculează termenul T_i cu ajutorul relaţiei de mai sus;

3.4. Se adună termenul T_i la suma **S**, utilizând o relaţie de forma **S := S + T_i**;

3.5. Se modifică valoarea contorului **i**, utilizând o relaţie de forma: **i := i + 1**. Se revine la pasul 3.2;

4. Se înmulţeşte valoarea calculată a sumei cu 4 şi se afişează. Se afişează valoarea constantei **π** folosind funcția implementată în MATLAB.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3e., iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.3f.

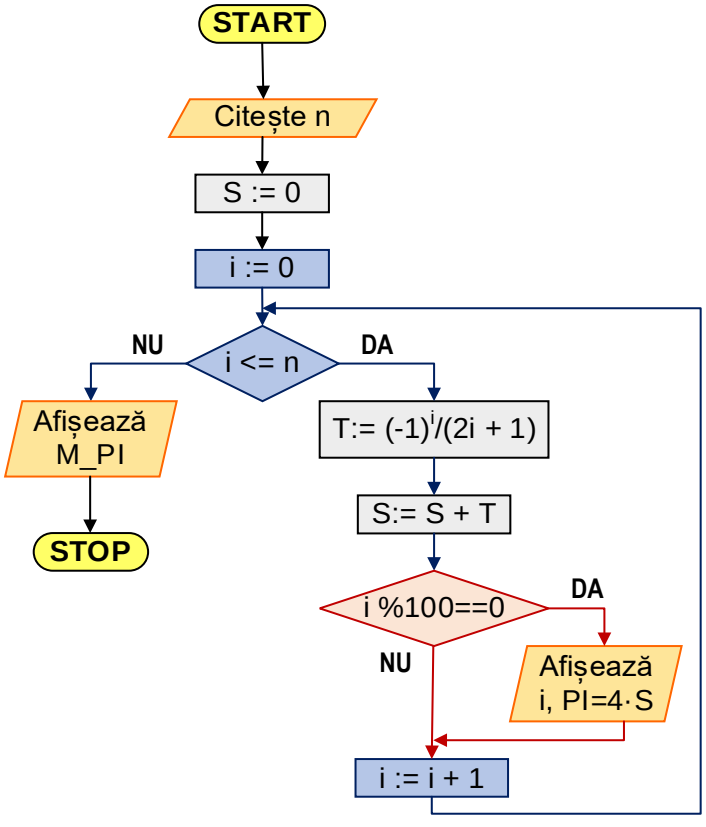
Schema logică	Limbajul pseudocod
	**Început** **Citeşte n** **S ← 0** **Pentru i = 0, n** **T = (-1)^i / (2 * i + 1)** **S ← S + T** **Dacă i DIV 100 = 0 atunci** **Scrie i, 4*S** **Sfârşit dacă** **Sfârşit pentru** **Scrie π** **Sfârşit**

Figura 7.3e. Reprezentarea algoritmului pentru calculul constantei **π** – varianta I

Programul MATLAB	Execuția programului

`n=input('Introduceti n=');`
`S=0;`
`for i=0:n`
`  T=(-1)^i/(2*i+1);`
	`Introduceti n = 1000`
	`i=  0  PI= 4.000000000`
	`i= 100  PI= 3.151493401`
	`i= 200  PI= 3.146567747`

```
    S=S+T;
    if mod(i,100)==0
      s=sprintf('i=%4d \t PI=%11.9f',i, 4*S);
      disp(s);
    end
end
s=sprintf(' Valoarea Pi = %11.9f',pi());
disp(s);
```

```
i= 300      PI= 3.144914904
i= 400      PI= 3.144086415
i= 500      PI= 3.143588660
i= 600      PI= 3.143256546
i= 700      PI= 3.143019186
i= 800      PI= 3.142841093
i= 900      PI= 3.142702531
i=1000      PI= 3.142591654
Valoarea   Pi = 3.141592654
```

Figura 7.3f. Programul MATLAB şi execuţia acestuia pentru calculul constantei **π** – varianta I

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Varianta II: Se consideră următoarea relaţie de calcul pentru constanta **π**:

$$\pi = 2 \cdot \left(1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \frac{1 \cdot 2 \cdot 3 \cdot 4}{3 \cdot 5 \cdot 7 \cdot 9} + \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{3 \cdot 5 \cdot 7 \cdot 9 \cdot 11} + \cdots\right)$$

Care mai poate fi scrisă: $\pi = 2 \cdot (1 + S)$

unde:

$$S = \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \frac{1 \cdot 2 \cdot 3 \cdot 4}{3 \cdot 5 \cdot 7 \cdot 9} + \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{3 \cdot 5 \cdot 7 \cdot 9 \cdot 11} + \cdots$$

Astfel, pentru calculul valorii aproximative a lui **π** trebuie să calculăm suma **S**, formula termenului general fiind:

$$T_i = \sum_{i=1}^{n} \frac{i!}{(2i+1)!!}.$$

Se observă că în formula termenului general apar două produse: factorial de **i** la numărător (notat cu **P1**) şi dublu factorial de **2i+1** la numitor (notat cu **P2**). Deci, pentru fiecare termen T_i este necesar să se calculeze două produse, algoritmul de calcul fiind următorul:

1. Se citeşte numărul de termeni ai sumei **S**, adică valoarea variabilei **n**;

2. Se iniţializează suma **S**, cu valoarea **0** (**zero** – element neutru pentru adunare);

3. Se utilizează o instrucţiune de ciclare cu contor (în acest caz contorul este variabila **i**), pentru calculul fiecărui termen T_i şi adunarea acestuia la sumă astfel:

3.1. Se iniţializează contorul **i**, cu valoarea **1**;

3.2. Se verifică condiţia $i <= n$, adică se verifică dacă valoarea contorului este în intervalul $[1, n]$.

Dacă condiţia este **adevărată**, se trece la pasul următor (pasul 3.3). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare, algoritmul continuă cu pasul **4**;

3.3. Se iniţializează produsele **P1** şi **P2** cu valoarea **1** (unu este element neutru pentru adunare);

3.4. Se utilizează o instrucţiune de ciclare cu contor (contorul fiind variabila **j**) pentru calculul factorialului, astfel:

3.4.1. Se iniţializează variabila **j** cu valoarea **1**;

3.4.2. Se verifică condiţia **j <= i**, dacă este **adevărată** se aplică relaţia **P1 := P1 * j**. Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul 3.5;

3.4.3. Se modifică valoarea contorului **j**, astfel: **j := j + 1**, după care se revine la pasul 3.4.2;

3.5. Se utilizează o instrucţiune de ciclare cu contor (contorul fiind variabila **k**) pentru calculul dublului factorial:

3.5.1. Se iniţializează variabila **k** cu valoarea **1**;

3.5.2. Se verifică condiţia **k <= 2*i+1**, dacă este **adevărată** se aplică relaţia **P2 := P2 * k**. Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul 3.6;

3.5.3. Se modifică valoarea contorului **k**, astfel: **k := k + 2**, după care se revine la pasul 3.5.2;

3.6. Se calculează termenul T_i astfel: T_i **:= P1 / P2** calculat anterior;

3.6. Se adună termenul T_i la suma **S**, utilizând o relaţie de forma **S := S +** T_i;

3.7. Se modifică valoarea contorului **i**, utilizând o relaţie de forma: **i := i + 1**. Se revine la pasul 3.2;

4. Se afişează valoarea relaţiei 2(1+S), precum şi valoarea calculată a constantei **π** folosind funcția implementată în MATLAB.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3g, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.3h

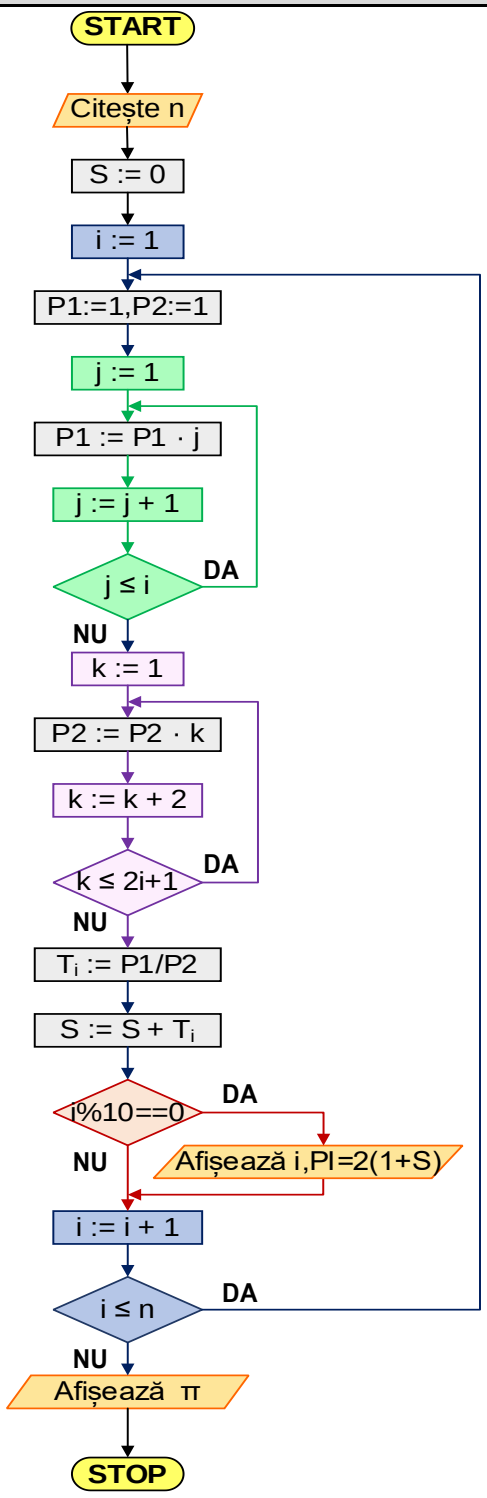
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** n S ← 0 **Pentru** i = 1, n P1 ← 1 P2 ← 1 **Pentru** j = 1, i P1 ← P1 * j **Sfârşit pentru** **Pentru** k = 1, 2*i+1, 2 P2 ← P2 * k **Sfârşit pentru** T[i] = P1 / P2 S ← S + T[i] **Dacă** i MOD 10 = 0 **atunci** **Scrie** i, 2*(1+S) **Sfârşit dacă** **Sfârşit pentru** **Scrie** π **Sfârşit**

Figura 7.3g. Reprezentarea algoritmului pentru calculul constantei **π** – varianta II

Programul MATLAB şi execuţia acestuia

```
n=input('Introduceti n=');
S=0;
for i=1:n
    P1=1; P2=1;
    for j=1:i
        P1=P1*j;
    end
    for k=1:2:2*i+1
        P2=P2*k;
    end
    T(i)=P1/P2;
    S=S+T(i);
    if mod(i,10)==0
      s=sprintf('i=%4d \t PI=%11.9f',i,2*(1+S));
      disp(s);
    end
end
s=sprintf(' Valoarea Pi = %11.9f',pi());
disp(s);
```

```
Rularea programului:
Introduceti n = 50

i= 10   PI= 3.141106022
i= 20   PI= 3.141592299
i= 30   PI= 3.141592653
i= 40   PI= 3.141592654
i= 50   PI= 3.141592654
Valoarea      Pi      =
3.141592654
```

Figura 7.3h. Programul MATLAB şi execuţia acestuia pentru calculul constantei **π** – varianta II

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

Trebuie menționat faptul că al doilea ciclu for (cel scris cu verde) împreună cu instrucțiunea de inițializare **P1=1** pot fi înlocuite cu apelul funcției **factorial,** funcție care este implementată în MATLAB. Practic se înlocuiesc cu instrucțiunea **P1=factorial(i)**.

Varianta III: Termenul general se calculează cu relaţia: $T_i = T_{i-1} \cdot \left(\frac{i}{2i+1}\right)$

În acest caz, termenul general se exprimă în funcţie de termenul anterior, conform relaţiei de mai sus.

Algoritmul de calcul este următorul:

1. Se citeşte numărul de termeni **n**;

2. Se iniţializează primul termen cu valoarea sa, în acest caz **T[1] = 1/3**;

3. Se iniţializează valoarea sumei cu valoarea primului termen, adică **S = T[1]**;

4. Se utilizează o instrucţiune de ciclare cu contor, acesta luând valori de la **2** la **n**, în cadrul căreia se realizează următoarele etape:

4.1. se iniţializează contorul **i**, cu valoarea **2**, astfel: **i = 2**;

4.2. se verifică condiţia **i <= n** prin care se verifică dacă valoarea curentă a contorului este mai mică decât **n** - numărul de termeni precizat anterior;

Dacă condiţia este **adevărată**, se calculează valoarea termenului curent cu relaţia de mai sus şi se adaugă la sumă valoarea termenului curent, utilizând relaţia **S = S + T[i]** şi se continuă cu incrementarea valorii contorului (etapa **4.3**). Dacă condiţia este **falsă** se părăseşte instrucţiunea de ciclare şi se continuă cu pasul **5**;

4.3. Se incrementează valoarea contorului **i**, adică **i = i + 1**, după care se revine la pasul **4.2**;

5. Se afişează valoarea produsului **2*(1 + S)**, respectiv se afişează valoarea constantei **π** folosind funcția implementată în MATLAB.

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 7.3i, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 7.3j.

Schema logică	Limbajul pseudocod

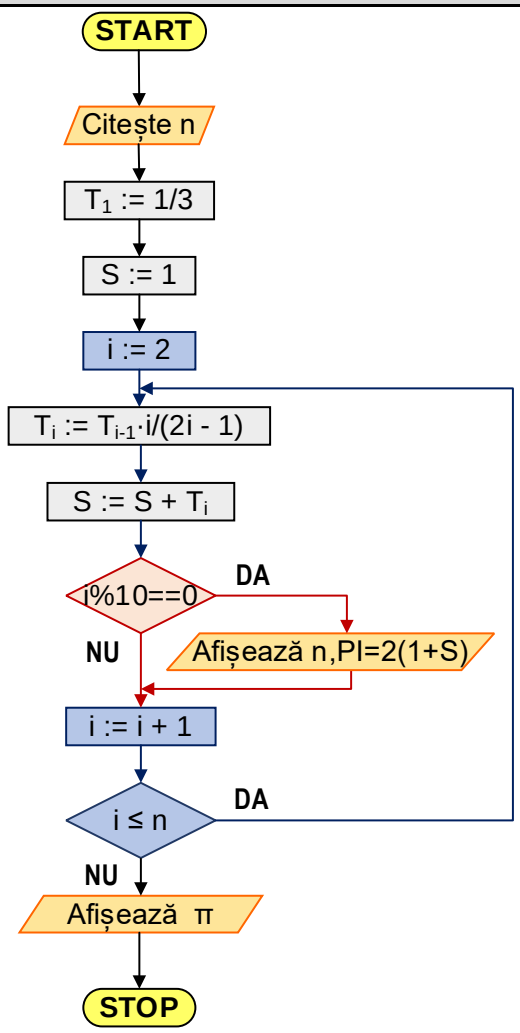
	**Început**
	Citeşte n
	T[1] = 1/3
	S ← T[1]
	Pentru i = 2, n
	  T[i] = T[i-1] * i / (2 * i - 1)
	  S ← S + T[i]
	  Dacă i MOD 10 = 0 **atunci**
	  Scrie i, 2*(1+S)
	  Sfârşit dacă
	Sfârşit pentru
	Scrie π
	Sfârşit

Figura 7.3i. Reprezentarea algoritmului pentru calculul constantei **π** – varianta III

Programul MATLAB şi execuţia acestuia

```
n=input('Introduceti n=');
T(1)=1/3; S=T(1);
for i=2:n
    T(i)=T(i-1)*i/(2*i+1);
    S=S+T(i);
    if mod(i,10)==0
      s=sprintf('i=%4d \t PI=%11.9f',i,2*(1+S));
      disp(s);
    end
end
s=sprintf(' Valoarea Pi = %11.9f',pi());
disp(s);
```

Rularea programului:

```
Introduceti n = 50

i=  10   PI = 3.141106022
i=  20   PI = 3.141592299
i=  30   PI = 3.141592653
i=  40   PI = 3.141592654
i=  50   PI = 3.141592654
Valoarea Pi = 3.141592654
```

Figura 7.3j. Programul MATLAB şi execuţia acestuia pentru calculul constantei **π** – varianta III

Observație: valorile **bolduite** de la execuția programului corespund datelor introduse de utilizator de la tastatură.

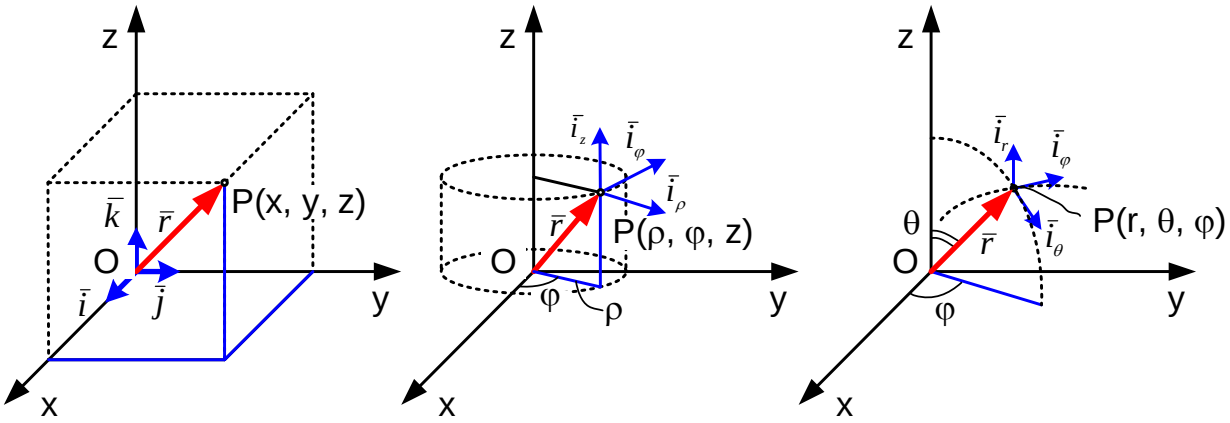
Capitolul 8. Aplicații în domeniul ingineriei mecanice

8.1. Transformări de coordonate

În geometrie, dar şi în alte domenii (mecanica clasică, topografia, etc) se utilizează sisteme de coordonate. Un sistem de coordonate reprezintă o modalitate prin care unui punct i se asociază în mod unic o mulţime ordonată de numere reale, ce poartă denumirea de coordonatele punctului. Cu ajutorul acestora se defineşte poziţia punctului în raport cu originea sistemului de coordonate. În spaţiul euclidian sunt necesare trei coordonate (abscisa, ordonata şi cota), în plan sunt necesare două coordonate (abscisa şi ordonata), iar pentru localizarea punctelor pe o dreaptă este necesară o singură coordonată.

Termenul de coordonată a fost introdus pentru prima oară de **Gottfried Wilhelm Freiherr von Leibniz** în 1692.

În studiul mecanicii clasice (newtoniene) se utilizează trei sisteme de coordonate: **cartezian**, **cilindric** şi **sferic**. În figura 8.1a sunt ilustraţi parametri în cazul celor trei sisteme de coordonate: **cartezian** (stânga), **cilindric** (mijloc), respectiv **sferic** (dreapta).



cartezian cilindric sferic

Figura 8.1a. Sisteme de coordonate

În sistemul de coordonate **cartezian** se aleg trei axe de coordonate, **Ox**, **Oy**, **Oz**, perpendiculare între ele. În acest caz, coordonatele punctului **P** vor fi **x**, **y**, **z**. Notând cu $\bar{\imath}, \bar{\jmath}, \bar{k}$ versorii celor trei axe, orice vector de poziţie $\bar{r}$ poate fi scris sub forma:

$$\bar{r} = x \cdot \bar{\imath} + y \cdot \bar{\jmath} + z \cdot \bar{k} \tag{8.1}$$

În sistemul de coordonate **cilindrice**, coordonatele sunt proiecţia **ρ** a vectorului $\bar{r}$ pe planul **xOy**, unghiul **φ** dintre axa **Ox** şi proiecţia vectorului $\bar{r}$ pe planul **xOy** şi **z**. Coordonatele se numesc **cilindrice** deoarece, păstrând raza cilindrului constantă şi modificând unghiul **φ** (de la 0 la 360°) şi **z**, se generează toate punctele unui cilindru.

În sistemul de coordonate **sferic**, coordonatele sunt raza **r**, unghiul **φ** dintre axa **Ox** şi proiecţia vectorului $\bar{r}$ pe planul **xOy**, respectiv unghiul **θ** dintre raza $\bar{r}$ şi axa **Oz**. Coordonatele se numesc **sferice** deoarece modificând raza de la **0** la **r** şi modificând coordonatele **θ** (de la 0 la 180°), respectiv **φ** (de la 0 la 360°) se pot genera toate punctele unei sfere.

Între coordonatele carteziene, cilindrice şi sferice sunt valabile relaţiile:

$$\begin{cases} x = \rho \cdot cos\varphi = r \cdot sin\theta \cdot cos\varphi; \\ y = \rho \cdot sin\varphi = r \cdot sin\theta \cdot sin\varphi; \\ z = z = r \cdot cos\theta; \end{cases} \tag{8.2}$$

O problemă de interes practic constă în determinarea coordonatelor cilindrice şi sferice atunci când se cunosc coordonatele carteziene.

Se consideră aşadar cunoscute coordonatele carteziene x, y, z şi se doreşte determinarea coordonatelor cilindrice: **ρ**, **φ**, **z**, respectiv a coordonatelor sferice: **r**, **φ**, **θ**.

Pentru determinarea coordonatelor cilindrice se utilizează relaţiile:

$$\begin{cases} \rho = x^2 + y^2 \\ \varphi = arctg\left(\frac{y}{x}\right) \\ z = z \end{cases} \tag{8.3}$$

Pentru determinarea coordonatelor sferice se utilizează relaţiile:

$$\begin{cases} r = x^2 + y^2 + z^2 \\ \varphi = arctg\left(\frac{y}{x}\right) \\ \theta = arccos\left(\frac{z}{r}\right) \end{cases} \tag{8.4}$$

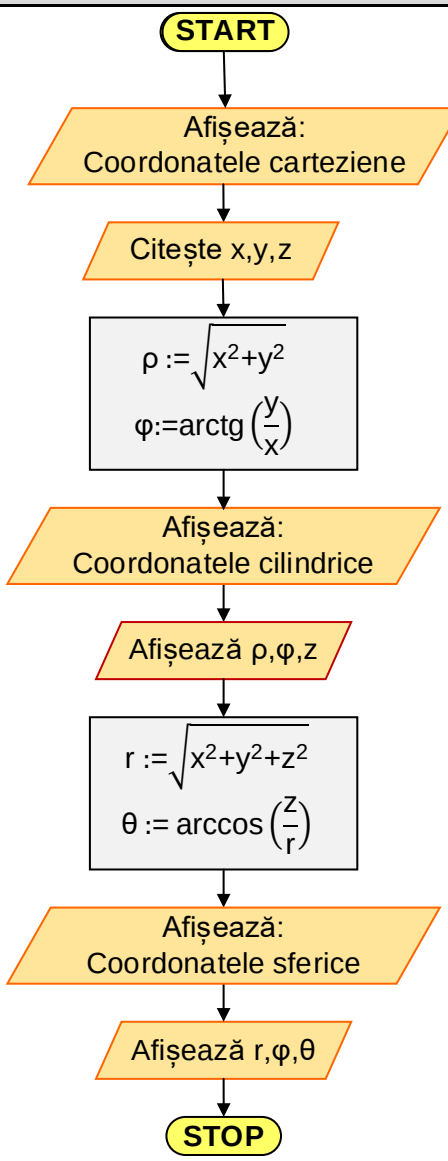
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** **x**, **y**, **z** ro ← sqrt(x^2 + y^2) fi ← arctg(y/x) **Scrie** ro, fi, z r ← sqrt(x^2 + y^2+z^2) fi ← arctg(y/x) teta ← arccos(z/r) **Scrie** r, fi, teta **Sfârşit**

Figura 8.1b. Reprezentarea algoritmului pentru calculul coordonatelor cilindrice şi sferice

Algoritmul de calcul se compune din următorii paşi:
- citirea coordonatelor carteziene: x, y, z;
- calculul şi afişarea coordonatelor cilindrice: **ρ**, **φ**, **z**, pe baza relaţiilor (8.3);
- calculul şi afişarea coordonatelor sferice: **r**, **φ**, **θ**, pe baza relaţiilor (8.4);

Reprezentarea algoritmului prin schemă logică şi pseudocod este ilustrată în figura 8.1b, iar programul MATLAB şi rularea acestuia sunt prezentate în figura 8.1c.

Programul MATLAB şi execuţia acestuia

```
disp('Coordonatele carteziene:');
x=input('Coordonata x = ');
y=input('Coordonata y = ');
z=input('Coordonata z = ');
ro = sqrt( x*x+y*y ); fi = atan( y/x );
disp('Coordonate cilindrice:');
s=sprintf(' Coordonata ro = %6.3f ',ro); disp(s);
s=sprintf(' Coordonata fi = %6.3f [rad] \t %6.3f [grad]',fi,fi*180/pi);
disp(s);
s=sprintf(' Coordonata z = %6.3f ',z); disp(s);
r = sqrt( x*x+y*y+z*z ); teta = acos( z/r );
disp('Coordonate sferice:');
s=sprintf(' Coordonata r = %6.3f ',r); disp(s);
s=sprintf(' Coordonata fi = %6.3f [rad] \t %6.3f [grad]',fi,fi*180/pi);
disp(s);
s=sprintf('    Coordonata    teta    =    %6.3f    [rad]    \t    %6.3f
[grad]',teta,teta*180/pi); disp(s);
```

Rularea programului:

```
Coordonatele carteziene:
 Coordonata x = 3
 Coordonata y = 4
 Coordonata z = 5

 Coordonate cilindrice:
 Coordonata ro =  5.000
 Coordonata fi =  0.927 [rad]    53.130 [grad]
 Coordonata z  =  5.000

 Coordonate sferice:
 Coordonata r    =  7.071
 Coordonata fi   =  0.927 [rad]          53.130 [grad]
 Coordonata teta =  0.785 [rad]          45.000 [grad]
```

Figura 8.1c. Programul MATLAB şi execuţia acestuia pentru calculul coordonatelor cilindrice şi sferice

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.2. Calculul simetricului unui punct faţă de o dreaptă, în plan

Se consideră un punct **P(x_P, y_P)** şi se doreşte determinarea coordonatelor punctului **S(x_S, y_S)** care este simetricul punctului **P** faţă de o dreaptă de ecuaţie **ax + by + c = 0**.

Pentru determinarea coordonatelor punctului **S**, se determină distanţa **d** de la punctul **P** la dreaptă, astfel:

$$d = \frac{|a \cdot x_P + b \cdot y_P + c|}{\sqrt{a^2+b^2}} \tag{8.5}$$

respectiv cu (m_1, m_2) cosinuşii directori ai vectorului n(a, b) normal la dreaptă, conform următoarelor relaţii:

$$m_1 = \pm\frac{a}{\sqrt{a^2+b^2}}, \quad m_2 = \pm\frac{b}{\sqrt{a^2+b^2}} \tag{8.6}$$

În relaţiile (8.6) se alege semnul **minus** dacă $a \cdot x_P + b \cdot y_P + c > 0$, respectiv semnul **plus** dacă $a \cdot x_P + b \cdot y_P + c < 0$.

Punctul **S** va avea coordonatele:

$$\begin{cases} x_S = x_P + 2 \cdot d \cdot m_1 \\ y_S = y_P + 2 \cdot d \cdot m_2 \end{cases} \tag{8.7}$$

Algoritmul de calcul al coordonatelor punctului simetric presupune parcurgerea următoarelor etape:

- citirea coordonatelor punctului **P**: **x_P**, **y_P**;
- citirea coeficienţilor dreptei **d**: **a**, **b**, **c**;
- se calculează variabilele m_1 şi m_2, pe baza relaţiilor (8.6) şi ţinând cont de valoarea expresiei $a \cdot x_P + b \cdot y_P + c$;
- se calculează coordonatele simetricului (punctul **S**) **x_S**, respectiv **y_S** cu ajutorul relaţiilor (8.7).

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.2a, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.2b.

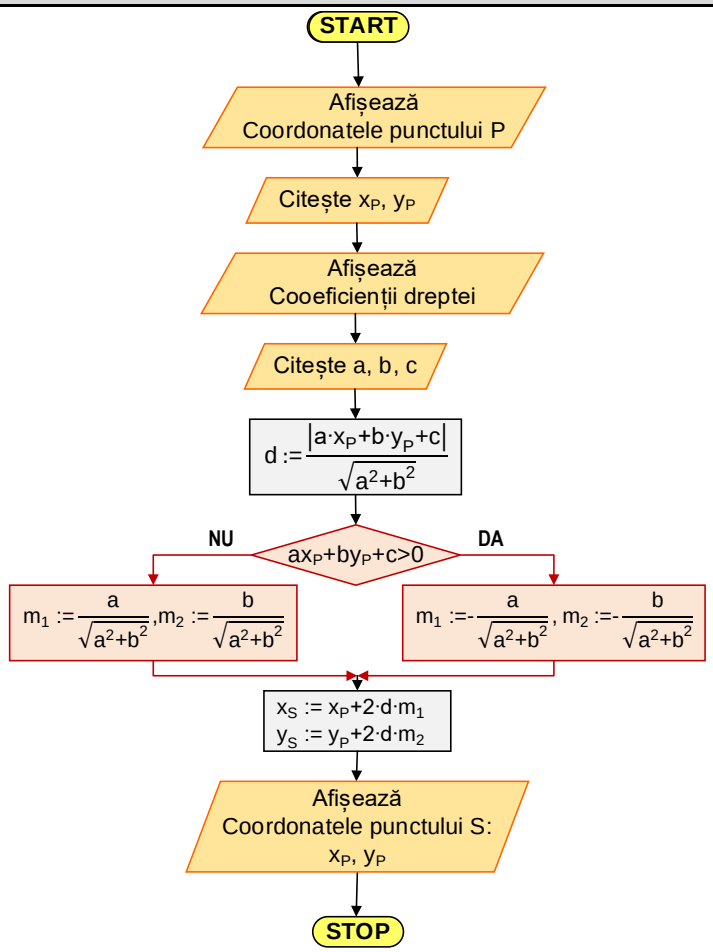
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** x_P, y_P, a, b, c d ← abs(a*x_P+b*y_P+c)/sqrt(a^2+b^2) **Dacă** a*x_P+b*y_P+c > 0 **atunci** $m_1 \leftarrow -\frac{a}{\sqrt{a^2+b^2}}, m_2 \leftarrow -\frac{b}{\sqrt{a^2+b^2}}$ **altfel** $m_1 \leftarrow +\frac{a}{\sqrt{a^2+b^2}}, m_2 \leftarrow +\frac{b}{\sqrt{a^2+b^2}}$ **Sfârşit dacă** $x_S \leftarrow x_P + 2 \cdot d \cdot m_1$ $y_S \leftarrow y_P + 2 \cdot d \cdot m_2$ **Scrie** x_S, y_S **Sfârşit**

Figura 8.2a. Reprezentarea algoritmului pentru calculul simetricului unui punct faţă de o dreaptă

Programul MATLAB şi execuţia acestuia

```
disp('Coordonatele punctului P:');
xp=input('Coordonata xp = ');
yp=input('Coordonata yp = ');
disp('Coeficientii dreptei:');
a=input('Coeficient a = ');
b=input('Coeficient b = ');
c=input('Coeficient b = ');
d = abs(a*xp+b*yp+c)/sqrt(a*a+b*b);
if a*xp+b*yp+c>0
    m1 = -a/sqrt(a*a+b*b);
    m2 = -b/sqrt(a*a+b*b);
else
    m1 = a/sqrt(a*a+b*b);
    m2 = b/sqrt(a*a+b*b);
end
xs = xp + 2*d*m1; ys = yp + 2*d*m2;
disp('Coordonatele punctului S:');
s=sprintf(' Coordonata xs = %6.3f ',xs);
disp(s);
s=sprintf(' Coordonata ys = %6.3f ',ys);
disp(s);
```

Rularea programului:

```
 Coordonatele punctului P:
 Coordonata xp = 1
 Coordonata yp = 3

 Coeficientii dreptei:
 Coeficient a = -1
 Coeficient b = 1
 Coeficient c = 0

 Coordonatele punctului S:
 Coordonata xs =  3.000
 Coordonata ys =  1.000
```

Figura 8.2b. Programul MATLAB şi execuţia acestuia pentru calculul simetricului unui punct faţă de o dreaptă

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.3. Calculul coordonatelor unui punct rotit în sens trigonometric, în plan

Se consideră un punct **P(x_P, y_P)** şi se doreşte determinarea noilor coordonate ale punctului **P(x_{PR}, y_{PR})**, obţinute în urma rotaţiei acestuia, în sens trigonometric, în jurul unui punct **O(x_O, y_O)** (numit centru de rotaţie) după un unghi fix **α** (numit unghi de rotaţie).

În acest caz, coordonatele noi ale punctului **P(x_{PR}, y_{PR})**, rezultate în urma rotaţiei punctului în sens trigonometric în jurul centrului de rotaţie **O(x_O, y_O)**, cu unghiul fix **α** se determină cu ajutorul relaţiilor:

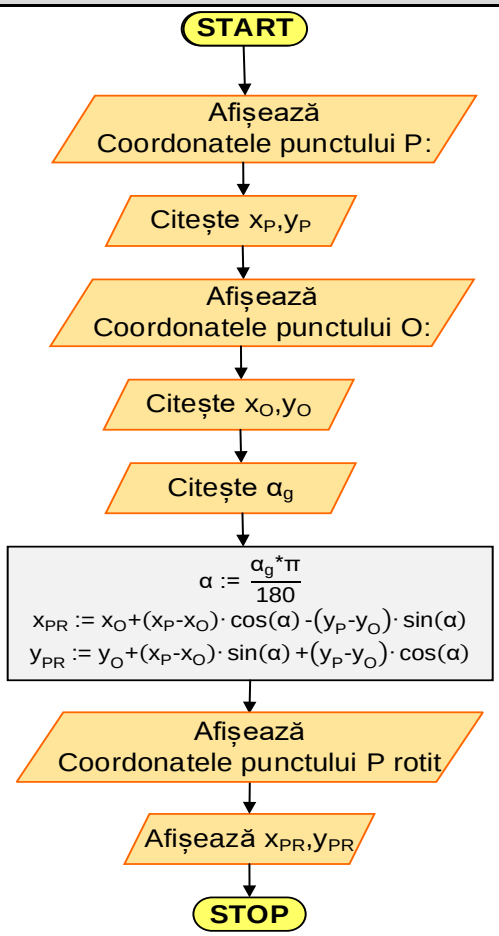
$$\begin{cases} x_{PR} = x_O + (x_P - x_O) \cdot \cos(\alpha) - (y_P - y_O) \cdot \sin(\alpha) \\ y_{PR} = y_O + (x_P - x_O) \cdot \sin(\alpha) + (y_P - y_O) \cdot \cos(\alpha) \end{cases} \tag{8.8}$$

Algoritmul de calcul al coordonatelor obţinute în urma rotaţiei unui punct **P(x_P, y_P)** în sens trigonometric, în jurul unui punct **O(x_O, y_O)**, presupune parcurgerea următoarelor etape:

- citirea coordonatelor punctului **P**: **x_P**, **y_P**;
- citirea coordonatelor punctului **O**: **x_O**, **y_O**;
- citirea unghiului **α**;
- se calculează şi se afişează noile coordonate ale punctului **P(x_{PR}, y_{PR})**, determinate cu ajutorul relaţiilor prezentate în ecuaţia (8.8).

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.3a, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.3b.

Schema logică



Limbajul pseudocod

Început

Citeşte x_P, y_P, x_O, y_O, alfa

alfar ← alfa* π /180.0

$x_{PR} \leftarrow x_O + (x_P - x_O) \cdot \cos(alfar) - (y_P - y_O) \cdot \sin(alfar)$

$y_{PR} \leftarrow y_O + (x_P - x_O) \cdot \sin(alfar) + (y_P - y_O) \cdot \cos(alfar)$

Scrie x_{PR}, y_{PR}

Sfârşit

Figura 8.3a. Reprezentarea algoritmului pentru calculul coordonatelor unui punct rotit cu unghiul **α** în jurul unui punct **O(x_O, y_O)**

Programul MATLAB şi execuţia acestuia

```
disp('Coordonatele punctului P:');
xp=input('Coordonata xp = ');
yp=input('Coordonata yp = ');
disp('Coordonatele punctului O:');
xo=input('Coordonata xo = ');
yo=input('Unghiul a [grade] = ');
alfag=input('Unghiul a [grade] = ');
alfar = alfag*pi/180.0;
xpr=xo+(xp-xo)*cos(alfar)-(yp-yo)*sin(alfar);
ypr=yo+(xp-xo)*sin(alfar)+(yp-yo)*cos(alfar);
disp('Coordonatele punctului P rotit:');
```

Rularea programului:

```
Coordonatele punctului P:
Coordonata xp = 3
Coordonata yp = 0
Coordonatele punctului O:
Coordonata xo = 0
Coordonata yo = 0
Unghiul a [grade] = 90
Coordonatele punctului P rotit:
Coordonata xpr =  0.000
Coordonata ypr =  3.000
```

```
s=sprintf('  Coordonata  xpr  =  %6.3f  ',xpr);
disp(s);
s=sprintf('  Coordonata  ypr  =  %6.3f  ',ypr);
disp(s);
```

Figura 8.3b. Programul MATLAB şi execuţia acestuia pentru calculul coordonatelor unui punct rotit cu unghiul **α** în jurul unui punct **O(x_O, y_O)**

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.4. Calculul ariei unui poligon neregulat

Se consideră un poligon neregulat cu „**n**” laturi (deci „**n**” vârfuri) şi se cere să se calculeze aria acestuia (figura 8.4a).

Pentru calculul ariei unui poligon cu vârfurile $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$, ..., $\mathbf{A_n}$ ($\mathbf{A_1 = A_n}$) se consideră un punct **O** ale cărui coordonate se consideră ($\mathbf{x_O = 0}$, $\mathbf{y_O = 0}$), arbitrar ales în planul poligonului.

Se „împarte” poligonul în triunghiuri $\mathbf{PA_iA_{i+1}}$ şi se calculează „aria cu semn” a fiecărui triunghi, notată cu $\mathbf{T_i}$, fiind posibile două cazuri:

- dacă vârfurile poligonului sunt orientate trigonometric, pentru fiecare latură orientată „spre dreapta”, aria triunghiului va fi **negativă**, iar pentru fiecare latură orientată „spre stânga”, aria triunghiului va fi **pozitivă**;
- dacă vârfurile poligonului sunt orientate în sens orar, pentru fiecare latură orientată „spre dreapta”, aria triunghiului va fi **pozitivă**, iar pentru fiecare latură orientată „spre stânga”, aria triunghiului va fi **negativă**;

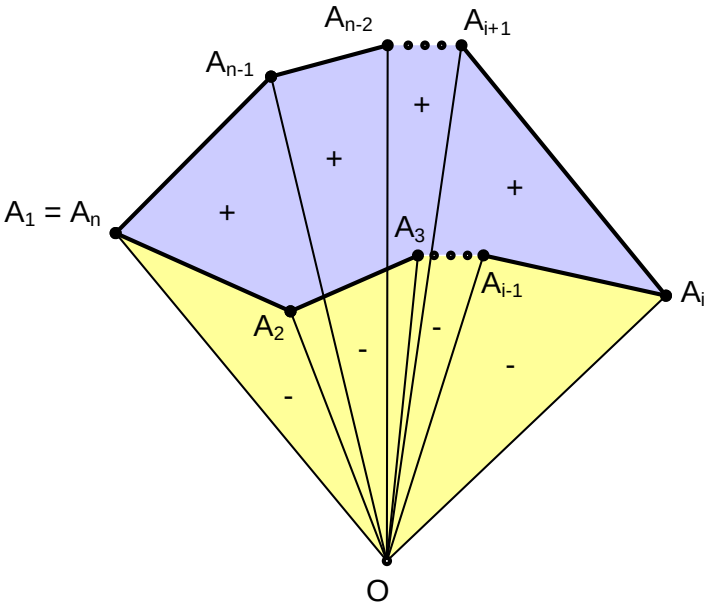


Figura 8.4a. Calculul ariei unui poligon neregulat

Ariile triunghiurilor $\mathbf{T_i}$ se calculează cu relaţia:

$$T_i = \frac{1}{2} \cdot \begin{vmatrix} 0 & 0 & 1 \\ x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \end{vmatrix} = \frac{1}{2} \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \qquad (8.9)$$

Indiferent de situaţie, însumând ariile triunghiurilor $\mathbf{T_i}$ se obţine aria poligonului ale cărui vârfuri sunt punctele $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$, ..., $\mathbf{A_n}$ ($\mathbf{A_1 = A_n}$):

$$A = \sum_{i=1}^{n} T_i = \frac{1}{2} \cdot \sum_{i=1}^{n} (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \qquad (8.10)$$

Relaţia (8.10) mai poate fi scrisă şi sub forma:

$$A = \frac{1}{2} \cdot |[(x_1 - x_2)(y_1 + y_2) + (x_2 - x_3)(y_2 + y_3) + \cdots (x_n - x_1)(y_n + y_1)]| \qquad (8.11)$$

Pe baza relaţiilor prezentate, se poate concepe un algoritm pentru calculul ariei unui poligon oarecare, astfel:

- se citeşte **n** = numărul de puncte – vârfurile poligonului;
- se citesc coordonatele $\mathbf{x_i}$, respectiv $\mathbf{y_i}$ ale fiecărui vârf al poligonului;
- deoarece calculul ariei se reduce la calculul unei sume, se iniţializează variabila care reprezintă suma cu zero, **S = 0**;
- se atribuie coordonatelor punctului $\mathbf{A_n}$ coordonatele punctului $\mathbf{A_1}$, astfel: $\mathbf{x_n = x_1}$, $\mathbf{y_n = y_1}$;
- se repetă operaţia de calcul a sumei, utilizând relaţia: $S = S + \frac{1}{2} \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$, pentru valori ale lui **i** de la **1** la **n**;

- deoarece valoarea sumei obţinute poate fi pozitivă sau negativă, se calculează şi se afişează valoarea modulului sumei obţinute;

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.4b, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.4c.

Schema logică	Limbajul pseudocod

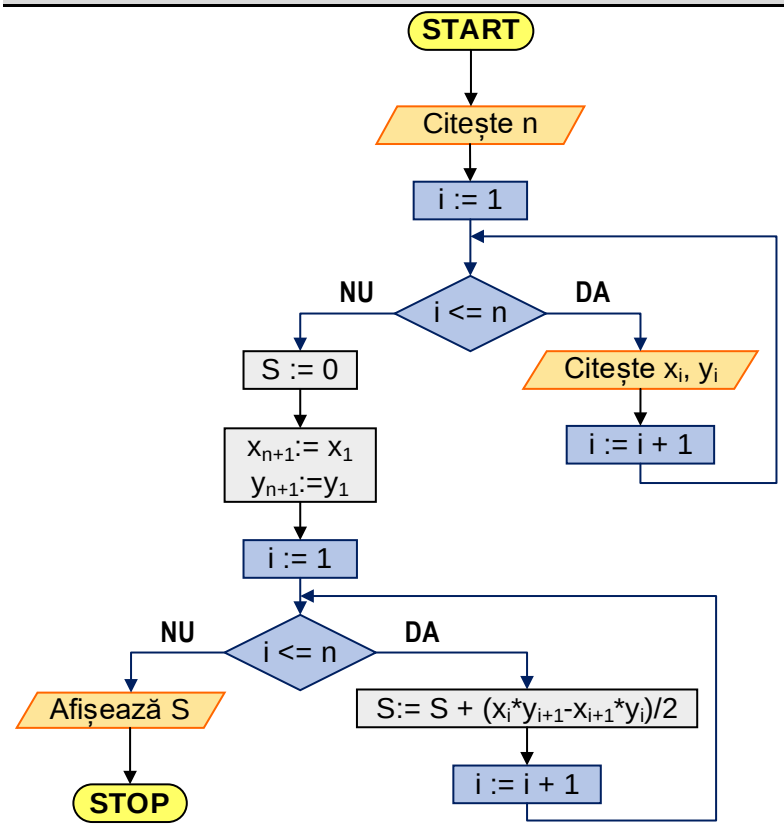
	**Început**
	Citeşte n
	Pentru i = 1, n
	  Citeşte x[i]
	  Citeşte y[i]
	Sfârşit pentru
	  S ← 0
	  x[n+1] ←x[1]
	  y[n+1] ←y[1];
	Pentru i = 1, n
	  S ← S + (x[i]*y[i+1]-x[i+1]*y[i])/2.;
	Sfârşit pentru
	Scrie |S|
	Sfârşit

Figura 8.4b. Reprezentarea algoritmului pentru calculul ariei unui poligon neregulat

Programul MATLAB şi execuţia acestuia	

```
n=input('Introduceti nr. de vf., n = ');
for i=1:n
    s=sprintf(' x(%d) = ',i);
    x(i)=input(s);
    s=sprintf(' y(%d) = ',i);
    y(i)=input(s);
end
S=0; x(n+1)=x(1); y(n+1)=y(1);
for i=1:n
    S=S+(x(i)*y(i+1)-x(i+1)*y(i))/2.;
end
s=sprintf(' Aria: S = %f', abs(S) );
disp(s);
```

```
Rularea programului:

Introduceti nr. de vf., n = 3
 x[1] = 0
 y[1] = 0
 x[2] = 2
 y[2] = 0
 x[3] = 0
 y[3] = 2
 Aria: S = 2.000000
```

Figura 8.4c. Programul MATLAB şi execuţia acestuia pentru calculul ariei unui poligon neregulat

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.5. Calculul centrului de masă al unui poligon neregulat

Centrul de masă, cunoscut şi sub denumirea de **centroid**, sau **centru de greutate** (într-un câmp gravitaţional uniform), al unui corp este definit ca poziţia „medie ponderată" a masei într-un corp. În Mecanică se utilizează centrul de masă, considerându-se masa corpului concentrată în acel punct, ecuaţiile utilizate în dinamică fiind aplicabile în raport cu acest punct.

Între centrul de masă (care este o noţiune mai generală) şi centru de greutate (definit într-un câmp gravitaţional) există o diferenţă. Centrul de greutate se defineşte ca fiind punctul în raport cu care suma momentelor forţelor gravitaţionale este zero (centrul forţelor paralele). În câmp gravitaţional uniform cele două puncte sunt identice.

Printre proprietăţile centrului de masă, amintim:

a) poziţia centrului de masă nu depinde de sistemul de referinţă ales, fiind un punct intrinsec al sistemului material;

b) dacă corpul (sistemul de puncte materiale) are un plan, o axă sau un punct de simetrie, atunci centrul de masă se găseşte în acel plan, pe aceea axă sau în punctul de simetrie.

În plan, în cazul unui poligon, centrul de greutate poate fi calculat cu următoarele formule:

$$XC = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1}) \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \tag{8.12}$$

$$YC = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1}) \cdot (x_i \cdot y_{i+1} - x_{i+1} \cdot y_i) \tag{8.13}$$

A fiind aria poligonului, determinată în subcapitolul precedent.

Se consideră un poligon închis, definit de „n" puncte: $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$, ..., $\mathbf{A_n}$ ($\mathbf{A_1 = A_n}$), figura 8.5.a. Se doreşte să se calculeze coordonatele centrului de masă al poligonului.

Pe baza relaţiilor prezentate, se poate concepe un algoritm pentru calculul coordonatelor, astfel:

- se citeşte **n** = numărul de puncte – vârfurile poligonului;
- se citesc coordonatele $\mathbf{x_i}$, respectiv $\mathbf{y_i}$ ale fiecărui vârf al poligonului;
- deoarece calculul ariei se reduce la calculul unei sume, se iniţializează variabila care reprezintă aria cu zero, **A = 0**;
- deoarece calculul coordonatelor X, respectiv Y se reduce la calculul unei sume, se iniţializează cele două variabile cu zero, astfel: $\mathbf{X_C = 0}$, $\mathbf{Y_C = 0}$;
- se atribuie coordonatelor punctului $\mathbf{A_n}$ coordonatele punctului $\mathbf{A_1}$, astfel: $\mathbf{x_n = x_1}$, $\mathbf{y_n = y_1}$;
- se repetă operaţiile de calcul ale ariei şi ale celor două sume $\mathbf{X_C}$, respectiv $\mathbf{Y_C}$, utilizând relaţiile de mai sus, pentru valori ale lui **i** de la **1** la **n**;
- se calculează valorile celor două coordonate $\mathbf{X_C}$, respectiv $\mathbf{Y_C}$;
- se afişează valorile obţinute.

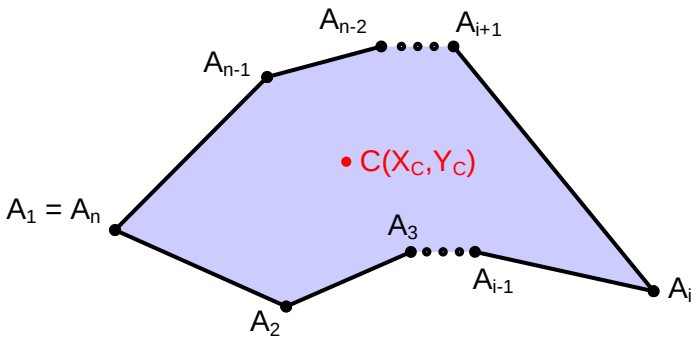


Figura 8.5a. Centrul de masă al unui poligon neregulat

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.5b, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.5c.

Schema logică	Limbajul pseudocod

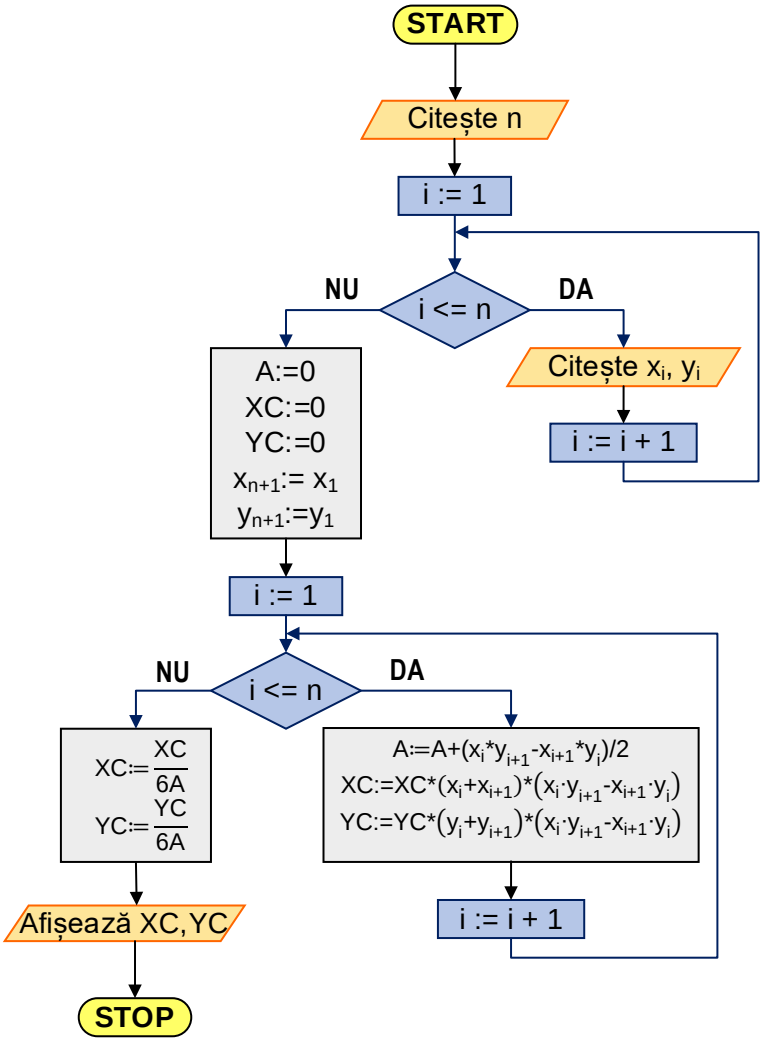
	**Început**
	Citeşte n
	Pentru i = 1, n
	  Citeşte x[i]
	  Citeşte y[i]
	Sfârşit pentru
	  A←0
	  XC←0
	  YC←0
	  x[n+1] ←x[1]
	  y[n+1] ←y[1];
	Pentru i = 1, n
	  A ← A + (x[i]*y[i+1]-x[i+1]*y[i])/2.;
	  XC ← XC + (x[i]+x[i+1])*(x[i]*y[i+1]-x[i+1]*y[i])
	  YC ← YC + (y[i]+y[i+1])*(x[i]*y[i+1]-x[i+1]*y[i])
	Sfârşit pentru
	  XC ← XC/(6*A)
	  YC ← YC/(6*A)
	Scrie XC
	Scrie YC
	Sfârşit

Figura 8.5b. Reprezentarea algoritmului pentru calculul **coordonatelor centrului de masă** al unui poligon neregulat

Programul MATLAB şi execuţia acestuia	

n=**input**(' Introduceti numarul de varfuri, n = ');
for i=1:n
  s=**sprintf**(' x(%d) = ',i); x(i)=input(s);
  s=**sprintf**(' y(%d) = ',i); y(i)=input(s);
end
A=0; XC=0; YC=0; x(n+1)=x(1); y(n+1)=y(1);
for i=1:n
  A = A + (x(i) * y(i+1) - x(i+1) * y(i)) / 2.;
  XC = XC + (x(i)+x(i+1))*(x(i)*y(i+1)-x(i+1)*y(i));
  YC = YC + (y(i)+y(i+1))*(x(i)*y(i+1)-x(i+1)*y(i));
end
XC = XC/6/A; YC = YC/6/A;
s=**sprintf**(' Coordonata X este: XC = %f',XC); disp(s);
s=**sprintf**(' Coordonata Y este: YC = %f',YC); disp(s);
	Rularea programului:
	Introduceti numarul de varfuri, n = **8**
	x[1] = **0**
	y[1] = **0**
	x[2] = **5**
	y[2] = **0**
	x[3] = **5**
	y[3] = **1**
	x[4] = **2**
	y[4] = **1**
	x[5] = **2**
	y[5] = **5**
	x[6] = **5**
	y[6] = **5**
	x[7] = **5**

```
y[7] = 6
x[8] = 0
y[8] = 6
Aria A este: A = 18.000000
Coordonata X este: XC = 1.833333
Coordonata Y este: YC = 3.000000
```

Figura 8.5b. Programul MATLAB şi execuţia acestuia pentru calculul **coordonatelor centrului de masă** al unui poligon neregulat

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

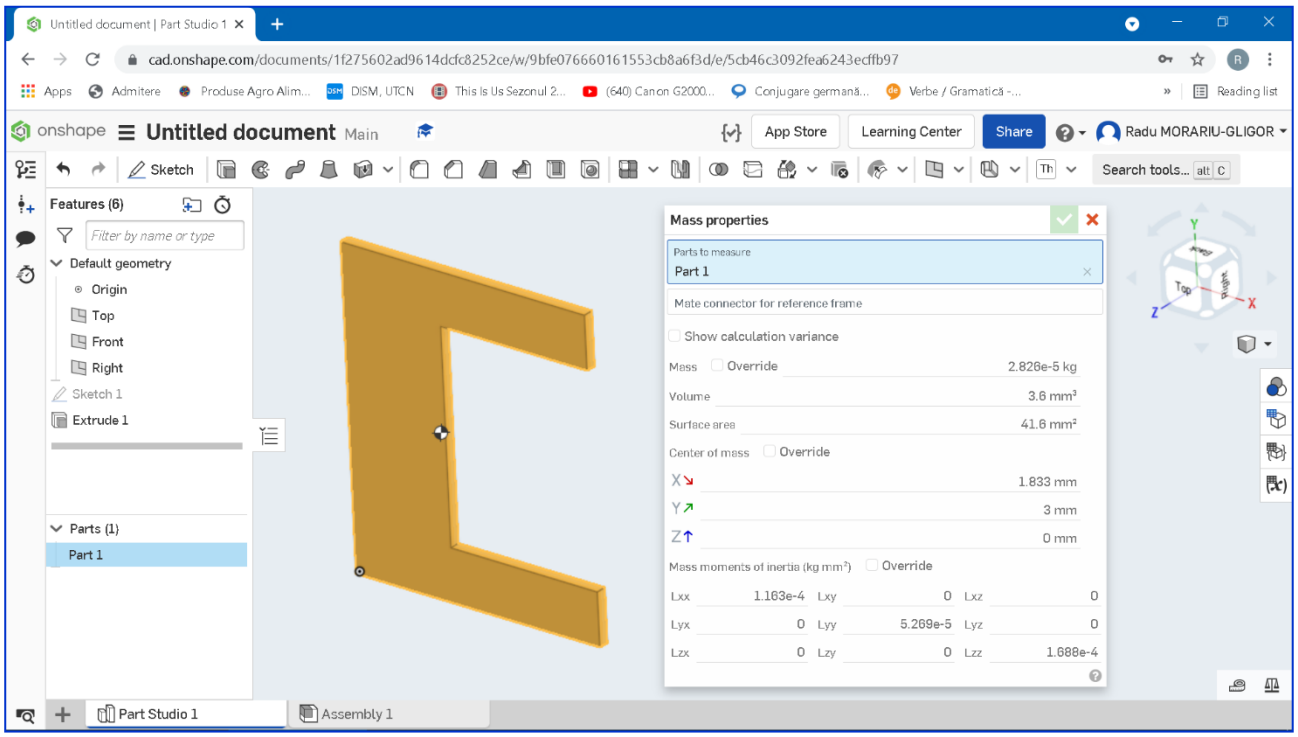


Figura 8.5d. Determinarea coordonatelor centrului de masă pentru un poligon neregulat, utilizând platforma **www.onshape.com**

8.6. Determinarea perioadei oscilaţiilor libere ale unui pendul matematic

Se doreşte să se determine perioadei oscilațiilor libere ale unui pendul matematic având lungimea firului inextensibil ℓ și amplitudinea unghiulară a oscilațiilor α (figura 8.6.a).

În cazul pendulului matematic, perioada oscilațiilor se exprimă cu relația:

$$T = 4\sqrt{\frac{\ell}{g}}\int_0^{\pi/2}\frac{d\varphi}{\sqrt{1-sin^2\frac{\alpha}{2}sin^2\varphi}} \qquad (8.14)$$

unde: g reprezintă accelerația gravitațională.

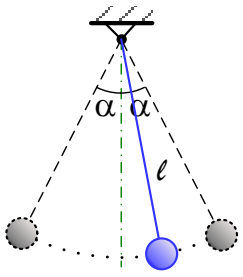


Figura 8.6a. Pendul matematic

În urma dezvoltării în serie a expresiei de sub semnul integralei și a integrării termen cu termen, perioada **T** se poate scrie sub forma:

$$T = 2\pi\sqrt{\frac{\ell}{g}}\cdot\left\{1+\sum_{i=1}^{\infty}\left[\frac{1\cdot3\cdot5\cdot\ldots\cdot(2i-1)}{2\cdot4\cdot6\cdot\ldots\cdot(2i)}\right]^2\cdot sin^{2i}\cdot\frac{\alpha}{2}\right\} \qquad (8.15)$$

Pentru calculul perioadei T trebuie să se calculeze următoarea sumă, cu n termeni:

$$S = \left(\frac{1}{2}\right)^2 \cdot c^2 + \left(\frac{1\cdot 3}{2\cdot 4}\right)^2 \cdot c^4 + \left(\frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\right)^2 \cdot c^6 + \cdots + \left(\frac{1\cdot 3\cdot 5\cdot\ldots\cdot(2i+1)}{2\cdot 4\cdot 6\cdot\ldots\cdot(2i)}\right)^2 \cdot c^{2i} + \cdots \quad (8.16)$$

unde s-a notat $c = sin\frac{\alpha}{2}$.

Dacă se notează t_i termenul de rang **i** al seriei, pentru calculul acestuia se utilizează următoarea relație de recurență:

$$t_i = c^2 \cdot t_{i-1} \cdot \left(\frac{2i-1}{2i}\right)^2 \quad (8.17)$$

Pentru a calcula suma seriei, se inițializează primul termen cu valoarea sa, adică $t_1 = \left(\frac{1}{2}\right)^2 \cdot c^2$, se consideră inițial că $S = t_1$, și apoi se aplică relația de calcul $S = S + t_i$ pentru toate valorile lui **i** de la **2** până la **n**.

Perioada **T** se va calcula cu relația:

$$T = a \cdot (1 + S) \quad (8.18)$$

unde: $a = 2\pi\sqrt{\frac{\ell}{g}}$.

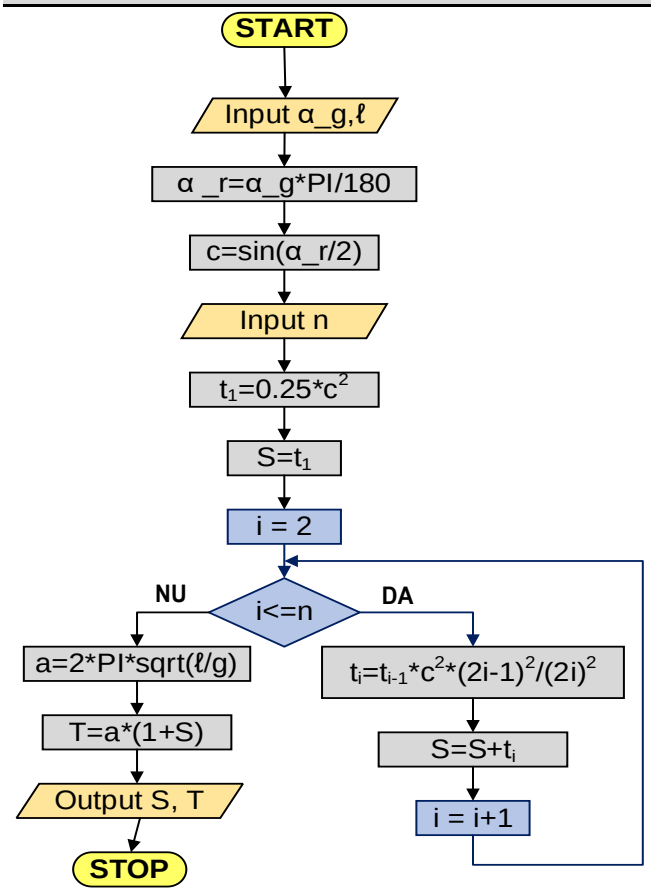
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** alfa_g, l alfa_r ← alfa_g*PI/180 c ← sin(alfa_r/2) **Citeşte** n t[1] ← 0,25*c^2 S ← t[1] **Pentru** i = 2, n t[i] ← t[i-1]*c^2*(2*i-1)^2/(2*i)^2 S ← S + t[i] **Sfârşit pentru** a ← 2*PI*sqrt(l/g) T ← a*(1+S) **Scrie** T **Sfârşit**

Figura 8.5b. Reprezentarea algoritmului pentru determinarea perioadei oscilaţiilor libere ale unui pendul matematic

Programul MATLAB şi execuţia acestuia

```
G=9.81;
alfa_g=input('Unghiul [grade], alfa_g=');
l=input('Lungimea firului, l = ');
alfa_r = (alfa_g*pi)/180; c = sin(alfa_r/2);
n=input('n = ');
```

Rularea programului:

Cazul 1:

```
Unghiul [grade], alfa_g = 30
Lungimea firului, l = 4
```

```
t(1) = 0.25*c*c; S = t(1);
for i=2:n
  t(i)=t(i-1)*c*c*((2.*i-1)/(2*i))*((2.*i-
1)/(2*i));
  S=S+t(i);
end
a=2*pi*sqrt(l/G); T=a*(1+S);
s=sprintf(' Perioada pendulului este:
%6.3f',T );
disp(s);
```

```
n = 4
Perioada pendulului este:
4.082
Cazul 2:
Unghiul [grade], alfa_g = 60
Lungimea firului, l = 10
n = 10
Perioada pendulului este:
6.808
```

Figura 8.5b. Programul MATLAB şi execuţia acestuia pentru determinarea perioadei oscilaţiilor libere ale unui pendul matematic

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

Algoritmul pentru rezolvarea problemei presupune parcurgerea următorilor paşi:

- se citesc valorile amplitudinii unghiulare ale oscilațiilor α, în grade, precum și lungimea firului pendulului matematic;
- se transformă valoarea lui α, din grade în radiani, cu formula: $\alpha = \frac{\alpha^{\circ} \cdot 3.14159}{180.0}$;
- se calculează variabila c, cu relația $c = sin\frac{\alpha}{2}$ și se alege n, numărul de termeni pentru care se calculează suma;
- se inițializează primul termen din serie, $t_1 = \left(\frac{1}{2}\right)^2 \cdot c^2$ și suma, $S = t_1$;
- în mod repetat, cu condiția ca poziția elementului din sumă să fie mai mică sau egală cu numărul maxim de elemente (**i ≤ n**), se calculează ceilalți **n** termeni ai seriei precum și suma lor. Tot în cadrul aceleiași iterații se afișează fiecare element al seriei după ce s-a calculat;
- după calculul sumei celor n termeni, se modifică valoarea variabilei a cu ajutorul relației $a = 2\pi\sqrt{\frac{\ell}{g}}$ și se calculează perioada oscilațiilor pendulului matematic cu formula: $T = a \cdot (1 + S)$.
- se afișează valoarea perioadei **T**.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.6b, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.6c.

8.7. Reducerea unui sistem de forţe coplanare

Se consideră o placă asupra căreia acționează un sistem de forțe coplanare situate în planul plăcii, în acest caz planul **xOy** (figura 8.7.a). Se doreşte să se determine elementele torsorului de reducere în raport cu punctul **O** – originea sistemului de coordonate.

Torsorul de reducere se compune din vectorul **R** – rezultanta forţelor care acţionează asupra plăcii, respectiv vectorul **M** – momentul rezultant, adică suma momentelor fiecărei forţe care acţionează asupra plăcii în raport cu punctul **O** – originea sistemului de coordonate.

În acest caz, deoarece: $z_i = 0, F_{iz} = 0$ relațiile de calcul ale componentelor rezultantei, respectiv momentului rezultant sunt:

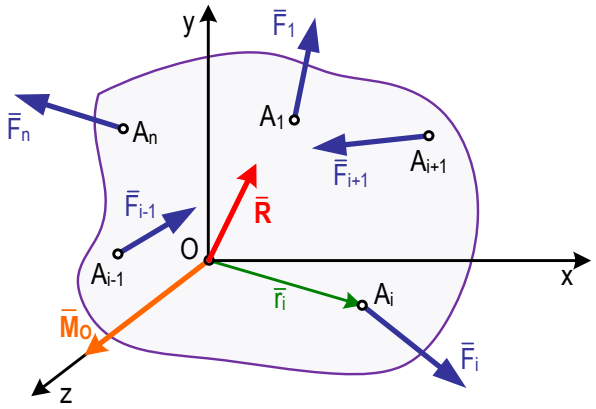


Figura 8.7a. Sistem de forțe coplanare

$$R_x = \sum_{i=1}^{n} F_{ix} = \sum_{i=1}^{n} F_i \cdot cos(\alpha_i) \quad (8.19)$$

$$R_y = \sum_{i=1}^{n} F_{iy} = \sum_{i=1}^{n} F_i \cdot sin(\alpha_i) \quad (8.20)$$

Modulul rezultantei se obține pe baza relației:

$$R = \sqrt{R_x^2 + R_y^2}. \quad (8.21)$$

Componenta momentului rezultant pe axa Oz, se calculează astfel:

$$M_z = \sum_{i=1}^{n} \left(x_i \cdot F_i \cdot sin(\alpha_i) - y_i \cdot F_i \cdot cos(\alpha_i) \right) \quad (8.22)$$

Pentru calculul elementelor torsorului de reducere sunt necesare numărul și valorile modulelor forțelor care acționează asupra plăcii plane, direcția forțelor (unghiurile α), precum și coordonatele punctelor în care acționează aceste forțe asupra plăcii.

Algoritm pentru rezolvarea problemei presupune parcurgerea următoarelor etape:

- citirea valorii variabilei **n**, numărul de forţe;
- deoarece pentru calculul variabilelor **R_x**, **R_y**, respectiv **M_z** se utiizează sume, acestea se iniţializează cu **0**;
- utilizând un ciclu cu contor se citesc valorile modulelor forțelor, unghiurile care indică direcția lor (în grade), precum și coordonatele **x** şi **y** ale punctelor unde acestea acționează și se calculează componentele vectorilor rezultanți **R_x** și **R_y** cu relaţiile (8.19) şi (8.20), respectiv componenta momentului rezultant pe axa **Oz** notat cu **M_z** cu ajutorul relației (8.22);
- se calculează modulul rezultantei, pe baza componentelor acesteia, utilizând relaţia (8.21);
- se afișează valorile obţinute ale variabilelor: **R_x** , **R_y** , **R** și **M_z**.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.7b, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.7c.

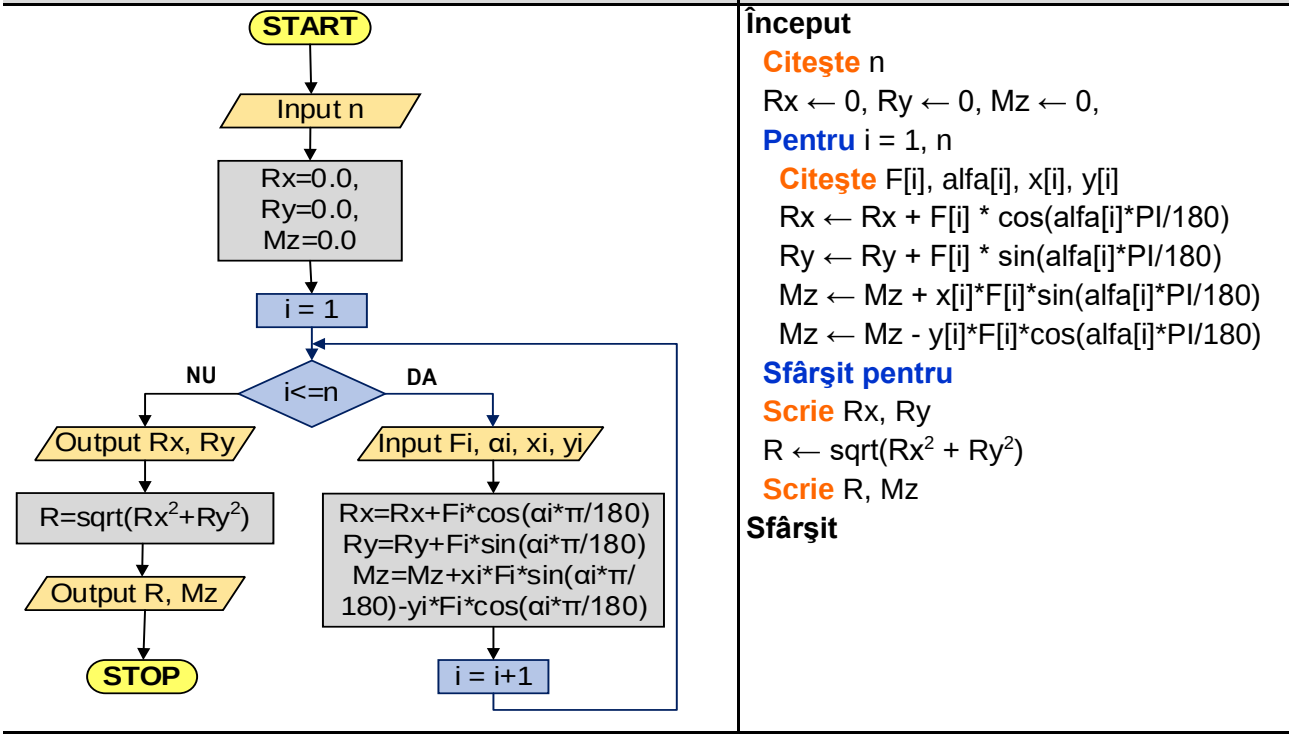


Figura 8.7b. Reprezentarea algoritmului pentru reducerea unui sistem de forţe coplanare

Programul MATLAB şi execuţia acestuia

```
Rx=0;Ry=0; Mz=0;
n=input('Numarul fortelor n = ');
for i = 1 : n
    s=sprintf(' F(%d) = ',i);
    F(i)=input(s);
    s=sprintf(' alfa(%d) = ',i);
    alfa(i)=input(s);
    s=sprintf(' x(%d) = ',i);
    x(i)=input(s);
    s=sprintf(' y(%d) = ',i);
    y(i)=input(s);
    Rx=Rx+F(i)*cos(alfa(i)*pi/180.0);
    Ry=Ry+F(i)*sin(alfa(i)*pi/180.0);
    Mz=Mz+x(i)*F(i)*sin(alfa(i)*pi/180.0);
    Mz=Mz-y(i)*F(i)*cos(alfa(i)*pi/180.0);
end
s=sprintf(' Rx=%6.3f',Rx );disp(s);
s=sprintf(' Ry=%6.3f',Ry );disp(s);
R = sqrt(Rx*Rx+Ry*Ry);
s=sprintf(' R = %6.3f',R );disp(s);
s=sprintf(' Mz = %6.3f',Mz );
disp(s);
```

Rularea programului:

```
Numarul fortelor n = 3
F[1] = 55
alfa[1] = 315
x[1] = 50
y[1] = 0
F[2] = 50
alfa[2] = 15
x[2] = 0
y[2] = 0
F[3] = 60
alfa[3] = 45
x[3] = 100
y[3] = 60
Rx = 129.614
Ry = 16.476
R = 130.657
Mz=-247.487
```

Figura 8.7c. Programul MATLAB şi execuţia acestuia pentru reducerea unui sistem de forţe coplanare

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.8. Mişcarea în vid a punctului material greu

Se consideră un punct material **M**, de masă **m**, lansat din punctul fix **O** în plan vertical cu viteza inițială v_0, înclinată cu unghiul **α** faţă de orizontală. Asupra punctului acționează greutatea sa $\bar{G} = m \cdot \bar{g}$. Pornind de la ecuațiile diferențiale de mișcare în planul **xOy** (figura 8.8a):

$$\begin{cases} m \cdot \ddot{x} = 0 \\ m \cdot \ddot{y} = -m \cdot g \end{cases} \tag{8.23}$$

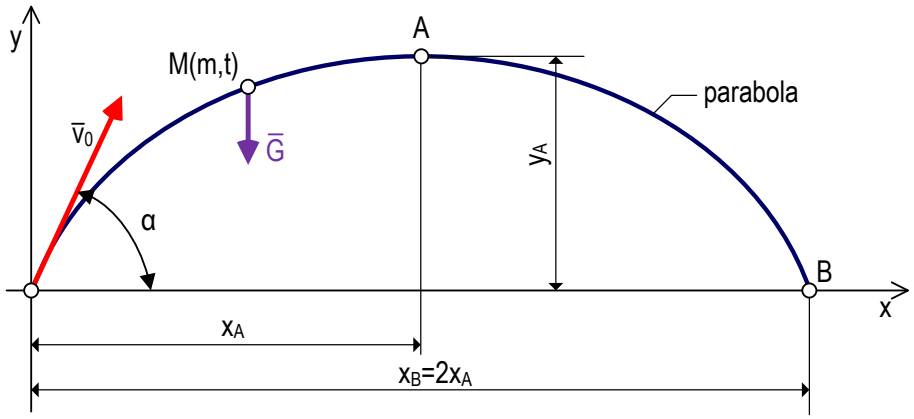


Figura 8.8a. Mişcarea în vid a punctului material greu

prin integrare succesivă se obțin soluțiile generale:

$$\begin{cases} \dot{x} = C_1 \\ x = C_1 \cdot t + C_2 \end{cases}; \quad \begin{cases} \dot{y} = -g \cdot t + C_3 \\ y = -g \cdot \frac{t^2}{2} + C_3 \cdot t + C_4 \end{cases} \tag{8.24}$$

Ţinând cont de condiţiile iniţiale în ceea ce priveşte poziţia şi viteza la momentul **t = 0**, adică:

$$\begin{cases} x = 0 \\ y = 0 \end{cases}; \quad \begin{cases} \dot{x} = v_0 \cdot cos(\alpha) \\ \dot{y} = v_0 \cdot sin(\alpha) \end{cases} \tag{8.25}$$

se determină constantele de integrare: $\begin{cases} C_1 = v_0 \cdot cos(\alpha) \\ C_2 = 0 \end{cases}; \quad \begin{cases} C_3 = v_0 \cdot sin(\alpha) \\ C_4 = 0 \end{cases}$

Astfel, *ecuaţiile parametrice de mişcare* a punctului material greu în vid sunt:

$$\begin{cases} x = v_0 \cdot t \cdot cos(\alpha) \\ y = -g \cdot \frac{t^2}{2} + v_0 \cdot t \cdot sin(\alpha) \end{cases} \tag{8.26}$$

Traiectoria de mişcare a punctului se obţine eliminând timpul din cele două ecuaţii, astfel rezultă :

$$y = -\frac{g}{2 \cdot v_0^2 \cdot cos^2(\alpha)} \cdot x^2 + x \cdot tg(\alpha) \tag{8.27}$$

Relaţia (8.27) reprezintă ecuaţia unei **parabole** având ca axă de simetrie verticala ce trece prin punctul **A** de înălţime maximă.

Coordonatele acestui punct se determină din condiţia ca $\frac{dy}{dx} = 0$ adică:

$$\begin{cases} x_A = \frac{v_0^2 \cdot sin(2\alpha)}{2 \cdot g} \\ y_A = \frac{v_0^2 \cdot sin^2(\alpha)}{2 \cdot g} \end{cases} \tag{8.28}$$

În acest punct, tangenta la traiectorie este orizontală, astfel că viteza nu va avea proiecţie pe axa verticală:

$$v_{Ay} = -g t_A + v_0 \cdot sin\alpha = 0 \tag{8.29}$$

de unde rezultă expresia timpului de urcare: $t_A = \frac{v_0 \cdot sin\alpha}{g}$ (8.30)

Distanţa $\mathbf{OB = x_B = 2 \cdot x_A}$ poartă numele de “bătaie”.

În programul următor se vor introduce ca date de intrare viteza iniţială v_0 şi unghiul de lansare **α** determinându-se pe baza relaţiilor prezentate anterior înălţimea maximă la care ajunge punctul y_A, distanţa la care punctul atinge din nou suprafaţa orizontală (bătaia), adică $2*x_A$ şi timpul [sec] cât punctul s-a aflat în aer, de la momentul lansării, adică $2*t_A$.

De asemenea, după determinarea timpului în care punctul material s-a aflat în aer, se determină poziţia punctului în funcţie de timp utilizând un interval de timp precizat de utilizator.

Algoritmul pentru rezolvarea acestei probleme necesită parcurgerea următorilor paşi :

- citirea de la tastatură a datelor de intrare, adică a vitezei iniţiale v_0 [m/s] şi unghiul de lansare **α** [grade] ;
- se calculează înălţimea maximă, bătaia şi timpul în care punctul material se află în aer, utilizând relaţiile prezentate anterior ;
- se citeşte de la tastatură pasul utilizat pentru calculul poziţiei punctului material de la momentul lansării până la momentul atingerii suprafeţei orizontale (notat cu **pas_t**) ;
- cu ajutorul unui ciclu cu contor se calculează şi se afişează poziţia punctului material în fiecare moment, de la momentul lansării până la momentul atingerii suprafeţei orizontale ;

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.8b, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.8c.

Schema logică	Limbajul pseudocod

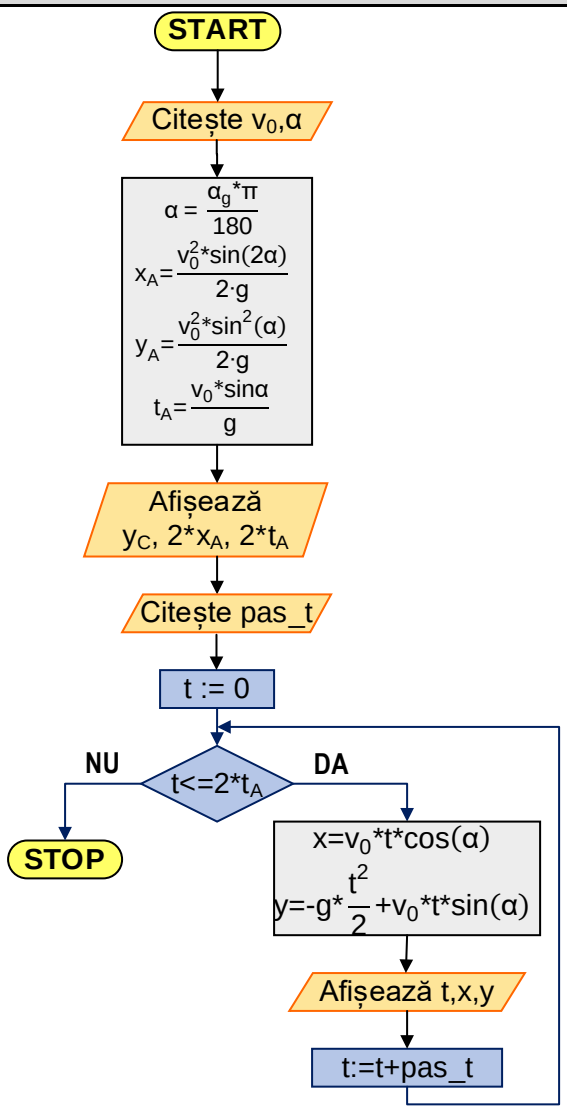
	**Început**
	  Citeşte v0, alfag
	  alfar ← alfag*PI/180.0
	  xA ← v0*v0*sin(2*alfar)/(2*G)
	  yA ← v0*v0*sin(alfar)*sin(alfar)/(2*G)
	  tA ← v0*sin(alfar)/G
	  Scrie yA, 2*xA, 2*tA
	  Citeşte pas_t
	  Pentru t = 0, 2 * tA **pas** pas_t
	  x ← v0 * t * cos(alfar)
	  y ← v0 * t * sin(alfar) – G * t * t / 2
	  Scrie t, x, y
	  Sfârşit pentru
	Sfârşit

Figura 8.8b. Reprezentarea algoritmului pentru studiul mişcării în vid a punctului material greu

Programul MATLAB şi execuţia acestuia	

`G=9.81;`
`v0=input('Viteza initiala [m/s]: v0 = ');`
`alfag=input('Unghiul de lansare [grade]: alfa = ');`
`alfar = alfag * pi/180.0;`
`xA = v0*v0*sin(2*alfar)/(2*G);`
`yA = v0*v0*(sin(alfar))^2/(2*G);`
`tA = v0*sin(alfar)/G;`
`s=sprintf('Inaltimea max = %6.3f [m]',yA );`
`disp(s);`
`s=sprintf('Bataia = %6.3f',2*xA );disp(s);`
	`Viteza initiala [m/s]: v0 = 15`
	`Unghiul de lansare [grade]: alfa = 42`
	
	`Inaltimea max = 5.135 [m]`
	`Bataia = 22.810 [m]`
	`Timpul total = 2.046 [s]`
	
	`Pasul interv. de timp [s]: pas_t = 0.25`
	
	`t= 0.000 x= 0.000 y= 0.000`
	`t= 0.250 x= 2.787 y= 2.203`
	`t= 0.500 x= 5.574 y= 3.792`
	`t= 0.750 x= 8.360 y= 4.769`

```
s=sprintf('Timpul total = %6.3f [s]',2*tA );
disp(s);
pas_t=input(' \n Pasul interv. de timp [s]: pas_t = ');
for t = 0:pas_t: 2*tA
    x=v0*t*cos(alfar);
    y=v0*t*sin(alfar)-G*t*t/2;
    s=sprintf(' t= %6.3f \t x= %6.3f \t y= %6.3f',t,x,y );
    disp(s);
end
```

```
t=   1.000    x= 11.147    y=  5.132
t=   1.250    x= 13.934    y=  4.882
t=   1.500    x= 16.721    y=  4.019
t=   1.750    x= 19.508    y=  2.543
t=   2.000    x= 22.294    y=  0.454
```

Figura 8.8c. Programul MATLAB şi execuţia acestuia pentru studiul mişcării în vid a punctului material greu

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

O problemă de interes practic este aceea de a determina unghiul de lansare **α** astfel încât, pentru o viteză iniţială de modul cunoscut v_0, să fie atinsă o ţintă $P(x_P, y_P)$ situată în planul vertical **xOy**. Pentru ca acest lucru să se întâmple, coordonatele punctului ţintă trebuie să verifice ecuaţia traiectoriei:

$$y_P = -\frac{g}{2\cdot v_0^2\cdot cos^2(\alpha)}\cdot x_P^2 + x_P\cdot tg(\alpha) = -\frac{g}{2\cdot v_0^2}\cdot x_P^2\cdot\left(1+tg^2(\alpha)\right) + x_P\cdot tg(\alpha) \qquad (8.31)$$

După ordonare în funcţie de **tg(α)** rezultă : $tg^2(\alpha) - \frac{2\cdot v_0^2}{g\cdot x_P}\cdot tg(\alpha) + \frac{2\cdot v_0^2}{g\cdot x_P^2}\cdot y_P + 1 = 0$ (8.32)

Soluţiile acestei ecuaţii sunt:

$$tg(\alpha) = \frac{1}{g\cdot x_P}\cdot\left[v_0^2 \pm \sqrt{v_0^4 - (2\cdot v_0^2\cdot g\cdot y_P + g^2\cdot x_P^2)}\right] \qquad (8.33)$$

Se constată astfel că, în general, există două soluţii pentru **tgα**, deci sunt două unghiuri sub care poate fi lansat punctul material greu pentru a atinge ţinta **P** (figura 8.8d):

- α_1 - corespunzător ***traiectoriei razante*** (Γ_1);
- α_2 - corespunzător ***traiectoriei boltite*** (Γ_2) ;

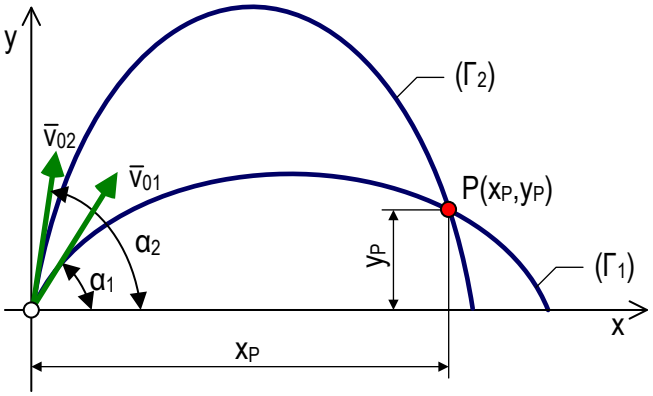


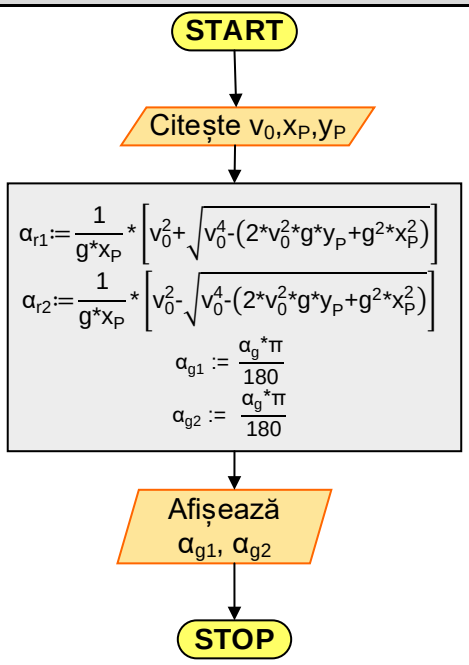
Figura 8.8d.

Algoritmul de calcul al unghiurilor de lansare a punctului material presupune parcurgerea următoarelor etape:

- citirea de la tastatură a vitezei iniţiale v_0;
- citirea de la tastatură a coordonatelor punctului ţintă : x_P, respectiv y_P ;
- calculul şi afişarea celor două valori ale unghiului de lansare **α** pentru care punctul ţintă este atins ;

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.8e, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.8f.

Schema logică



Limbajul pseudocod

Început
  Citeşte v0, xP, yP
  alfar1 ← arctg((v0*v0+sqrt(v0*v0*v0*v0-(2*v0*v0*G*yP+G*G*xP*xP)))/(G*xP))
  alfar2 ← arctg((v0*v0-sqrt(v0*v0*v0*v0-(2*v0*v0*G*yP+G*G*xP*xP)))/(G*xP))
  alfag1 ← alfar1*180/PI
  alfag2 ← alfar2*180/PI
  Scrie alfag1, alfag2
Sfârşit

Figura 8.8e. Reprezentarea algoritmului pentru determinarea unghiului de lansare **α**

Programul MATLAB şi execuţia acestuia

```
G=9.81;
v0=input('Viteza initiala [m/s]: v0 = ');
xP=input('Coordonata x a tintei [m]: xP = ');
yP=input('Coordonata y a tintei [m]: yP = ');
alfar1=atan((v0*v0+sqrt(v0*v0*v0*v0-(2*v0*v0*G*yP+G*G*xP*xP)))/(G*xP));
alfar2=atan((v0*v0-sqrt(v0*v0*v0*v0-(2*v0*v0*G*yP+G*G*xP*xP)))/(G*xP));
alfag1 = alfar1 * 180.0/pi;    alfag2 = alfar2 * 180.0/pi;
s=sprintf(' Unghiul de lansare alfa 1 = %6.3f [grade]',alfag1 );disp(s);
s=sprintf(' Unghiul de lansare alfa 2 = %6.3f [grade]',alfag2 );disp(s);
```

Rularea programului:

```
Viteza initiala [m/s]: v0 = 20
 Coordonata x a tintei [m]: xP = 20
 Coordonata y a tintei [m]: yP = 8
 Unghiul de lansare alfa 1 = 73.015 [grade]
 Unghiul de lansare alfa 2 = 38.787 [grade]
```

Figura 8.8f. Programul MATLAB şi execuţia acestuia pentru determinarea unghiului de lansare **α**

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.9. Studiul mişcării unui corp pe planul înclinat, cu frecare

Se consideră un corp de masă **m**, aflat pe un plan înclinat faţă de orizontală la unghiul **α** (figura 8.9a). Asupra corpului acţionează forţa de greutate **G**, precum şi o forță de tracţiune **F** sub un unghi oarecare **β** față de orizontală, coeficientul de frecare dintre corp şi planul înclinat fiind **μ**. De asemenea, se consideră coeficientul de frecare statică egal cu cel de frecare dinamică iar corpul se deplasează pe planul înclinat cu viteză constantă sau stă în repaus.

Pe baza datelor de intrare (masa corpului **m**, valoarea forţei de tracţiune **F**, unghiurile **α** şi **β**, respectiv coeficientul de frecare dintre corp şi planul înclinat **μ** se doreşte să se determine felul în care se deplasează corpul pe planul înclinat (în jos pe planul înclinat, în repaus faţă de planul înclinat sau se deplasează în sus de-a lungul planului înclinat).

În figura 8.9b este prezentat cazul în care corpul se deplasează în sus pe planul înclinat, forţa de frecare F_R acţionând în acest caz în jos, de-a lungul planului înclinat.

În figura 8.9c este prezentat cazul în care corpul se deplasează în jos pe planul înclinat, situaţie în care forţa de frecare F_R acţionează în sus, de-a lungul planului înclinat.

În primul caz (figura 8.9b), ecuaţiile de echilibru sunt:

$$\begin{cases} \sum F_x = 0: \;\; F_x - G_x - F_R = 0 \\ \sum F_y = 0: \;\; N + F_y - G_y = 0 \\ F_R \le \mu \cdot N \end{cases} \tag{8.34}$$

$$N = G_y - F_y = G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha) \tag{8.35}$$

$$F \cdot \cos(\beta - \alpha) > G \cdot \sin\alpha + \mu \cdot (G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha)) \tag{8.36}$$

$$F \cdot \cos(\beta - \alpha) + \mu \cdot F \cdot \sin(\beta - \alpha) > G \cdot \sin\alpha + \mu \cdot G \cdot \cos\alpha \tag{8.37}$$

$$F > G \cdot \frac{\sin\alpha + \mu \cdot \cos\alpha}{\cos(\beta - \alpha) + \mu \cdot \sin(\beta - \alpha)} \tag{8.38}$$

În al doilea caz (figura 8.9c), ecuaţiile de echilibru sunt:

$$\begin{cases} \sum F_x = 0: \;\; F_x - G_x + F_R = 0 \\ \sum F_y = 0: \;\; N + F_y - G_y = 0 \\ F_R = \mu \cdot N \end{cases} \tag{8.39}$$

$$N = G_y - F_y = G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha) \tag{8.40}$$

$$F \cdot \cos(\beta - \alpha) + \mu \cdot (G \cdot \cos\alpha - F \cdot \sin(\beta - \alpha)) < G \cdot \sin\alpha \tag{8.41}$$

$$F \cdot \cos(\beta - \alpha) - \mu \cdot F \cdot \sin(\beta - \alpha) < G \cdot \sin\alpha - \mu \cdot G \cdot \cos\alpha \tag{8.42}$$

$$F < G \cdot \frac{\sin\alpha - \mu \cdot \cos\alpha}{\cos(\beta - \alpha) - \mu \cdot \sin(\beta - \alpha)} \tag{8.43}$$

Prin urmare, se pot întâlni trei situaţii:

- corpul se deplasează în sus pe planul înclinat dacă:

$$F > G \cdot \frac{\sin\alpha + \mu \cdot \cos\alpha}{\cos(\beta - \alpha) + \mu \cdot \sin(\beta - \alpha)} \tag{8.44}$$

- corpul stă în repaus pe planul înclinat dacă:

$$G \cdot \frac{\sin\alpha - \mu \cdot \cos\alpha}{\cos(\beta - \alpha) - \mu \cdot \sin(\beta - \alpha)} \le F \le G \cdot \frac{\sin\alpha + \mu \cdot \cos\alpha}{\cos(\beta - \alpha) + \mu \cdot \sin(\beta - \alpha)} \tag{8.45}$$

- corpul coboară pe planul înclinat dacă:

$$F < G \cdot \frac{\sin\alpha - \mu \cdot \cos\alpha}{\cos(\beta - \alpha) - \mu \cdot \sin(\beta - \alpha)} \tag{8.46}$$

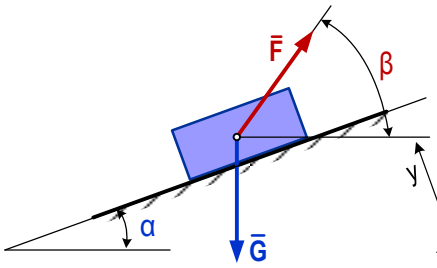


Figura 8.9a

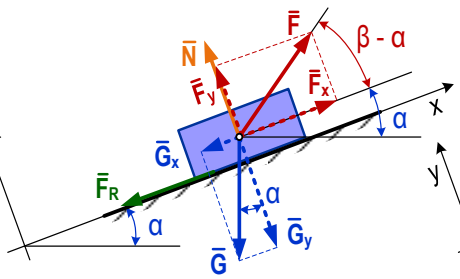


Figura 8.9b

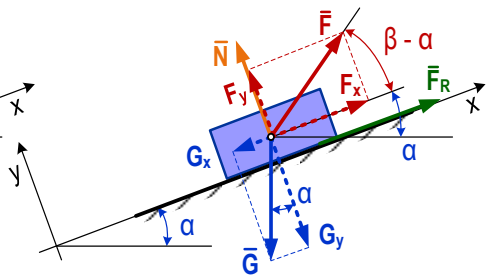


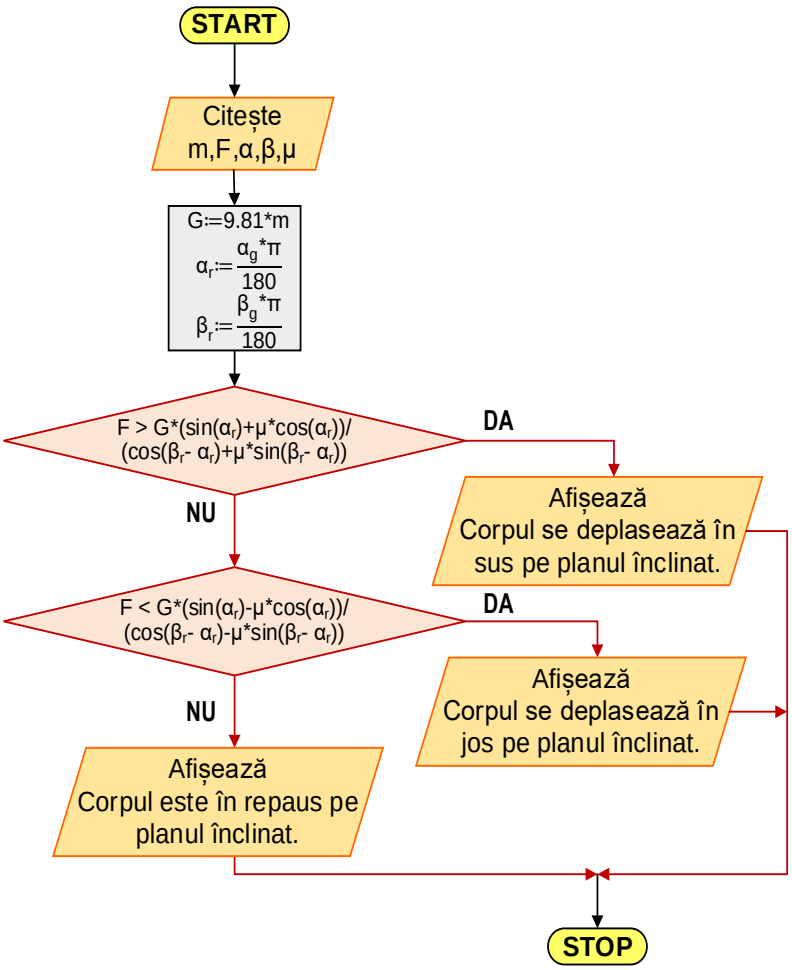
Figura 8.9c

Algoritmul de rezolvare a problemei presupune parcurgerea următorilor paşi:

- se citesc de la tastatură următoarele date de intrare: masa corpului **m**, forţa de tracţiune **F**, unghiul planului înclinat **α**, unghiul sub care acţionează forţa de tracţiune **β**, coeficientul de frecare **μ**;
- se verifică prima condiţie, dată de relaţia (8.44), dacă această condiţie este adevărată se afişează un mesaj prin care se specifică că corpul se deplasează în sus pe planul înclinat;
- dacă condiţia (8.44) nu se îndeplineşte, se verifică condiţia (8.46). Dacă aceasta este adevărată se afişează un mesaj prin care se specifică că corpul se deplasează în jos pe planul înclinat;
- dacă condiţia (8.46) nu este adevărată, înseamnă că corpul stă în repaus pe planul înclinat, pe ecran se afişează un mesaj corespunzător;

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.9d, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.9e.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte m, F, α, β, μ
  G ← 9,81*m
  Dacă F > G * ( sin(α) + μ * cos(α) ) / ( cos(β- α) + μ * sin(β- α) ) atunci
     Scrie „ Corpul se deplasează în sus pe planul înclinat!”
  altfel
     Dacă F < G * ( sin(α) - μ * cos(α) ) / ( cos(β- α) - μ * sin(β- α) ) atunci
          Scrie „ Corpul se deplasează în jos pe planul înclinat!”
     altfel
          Scrie „ Corpul este in repaus pe planul inclinat!”
     Sfârşit dacă
  Sfârşit dacă
Sfârşit
```

Figura 8.9d. Reprezentarea algoritmului pentru studiul mişcării unui corp pe planul înclinat

Programul MATLAB şi execuţia acestuia

```
m=input('Masa corpului [kg], m = ');
F=input('Forta de tractiune [N], F = ');
ag=input('Unghiul alfa [grade], alfa = ');
bg=input('Unghiul beta [grade], beta = ');
u=input('Coeficientul de frecare, miu = ');
G = 9.81*m; ar = ag * pi / 180.0; br = bg * pi / 180.0;
if F > G*(sin(ar)+u*cos(ar))/(cos(br-ar)+u*sin(br-ar))
    disp('Corpul se deplaseaza in sus pe planul înclinat!');
elseif F < G*(sin(ar)-u*cos(ar))/(cos(br-ar)-u*sin(br-ar))
    disp('Corpul se deplaseaza in jos pe planul înclinat!');
else
    disp('Corpul este in repaus pe planul inclinat!');
end
```

Rularea programului:

Cazul 1:

```
Masa corpului [kg], m = 5
Forta de tractiune [N], F = 60
Unghiul alfa [grade], alfa = 30
Unghiul beta [grade], beta = 45
Coeficientul de frecare, miu = 0.3
Corpul se deplaseaza in sus pe planul inclinat!
```

Cazul 2:

```
Masa corpului [kg], m = 5
Forta de tractiune [N], F = 30
Unghiul alfa [grade], alfa = 30
Unghiul beta [grade], beta = 45
Coeficientul de frecare, miu = 0.3
Corpul este in repaus pe planul inclinat!
```

```
Cazul 3:
 Masa corpului [kg], m = 5
 Forta de tractiune [N], F = 10
 Unghiul alfa [grade], alfa = 30
 Unghiul beta [grade], beta = 30
 Coeficientul de frecare, miu = 0.3
 Corpul se deplaseaza in jos pe planul inclinat!
```

Figura 8.9e. Programul MATLAB şi execuţia acestuia pentru studiul mişcării unui corp pe planul înclinat

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.10. Cinematica mecanismului bielă - manivelă

Mecanismul bielă-manivelă (figura 8.10a) se utilizează pentru transformarea mișcării de translație alternativă în mișcare de rotație continuă (motoare cu ardere internă), respectiv pentru transformarea mișcării de rotație continuă în mișcare de translație alternativă (compresoare, pompe, prese).

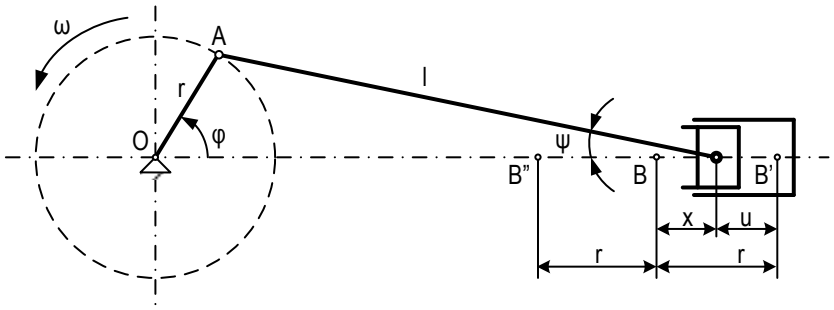


Figura 8.10a. Mecanismul bielă - manivelă

În figură, notaţiile au următoarea semnificaţie: **l** – lungimea bielei, **r** – raza manivelei, **A** – centru de articulaţie dintre bielă şi axa cilindrului, **B'** şi **B"** – poziţiile extreme ale capătului bielei, **x** – deplasarea capătului bielei la un anumit moment. Turaţia manivelei se notează cu „**n**" şi se măsoară în [rot/min].

Manivela se roteşte cu viteza unghiulară constantă: **ω** = **φ'**, iar cupla de translaţie **B** are cursa B'B"=2r.

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

$$\varphi = \omega \cdot t = \frac{\pi \cdot n}{30} \cdot t \tag{8.47}$$

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

$$u = r \cdot (1 - \cos\varphi) + l(1 - \cos\psi); \tag{8.48}$$

Ţinând seama de relaţia geometrică: $\cos\psi = \sqrt{1 - \left(\frac{r}{l}\sin\varphi\right)^2}$ (8.50)

şi dezvoltând în serie de puteri expresia (8.50), şi considerând numai primii doi termeni, se obţine legea de mişcare a pistonului:

$$u \cong r \cdot \left(1 - \cos\varphi + \frac{\lambda}{2}\sin^2\varphi\right) \tag{8.51}$$

În expresia (8.51) au fost neglijaţi termenii cu puteri mai mari decât 2 şi s-a notat $\lambda = \frac{r}{l}$, raport denumit caracteristica structurală a mecansimului.

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

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

rezultă:

$$v(\varphi) = \omega \cdot r \cdot \left(\sin\varphi + \frac{1}{2} \cdot \lambda \cdot \sin 2\varphi \right),$$
$$v(t) = \omega \cdot r \cdot \left(\sin(\omega \cdot t) + \frac{1}{2} \cdot \lambda \cdot \sin(2 \cdot \omega \cdot t) \right) \quad (8.52)$$

Legea de variație a accelerației se determină derivând viteza în raport cu timpul: $a = \frac{dv}{dt} = \frac{dv}{d\phi} \cdot \frac{d\phi}{dt} = \omega \cdot \frac{dv}{d\phi}$

Rezultă:

$$a(\phi) = \omega^2 \cdot r \cdot (\cos\phi + \lambda \cdot \cos 2\phi)$$
$$a(t) = \omega^2 \cdot r \cdot (\cos(\omega \cdot t) + \lambda \cdot \cos(2 \cdot \omega \cdot t)) \quad (8.53)$$

Studiu cinematic presupune trasarea graficelor deplasării (**x**), a vitezei (**v**), respectiv a accelerației (**a**) în funcție de timp (figura 8.10b).

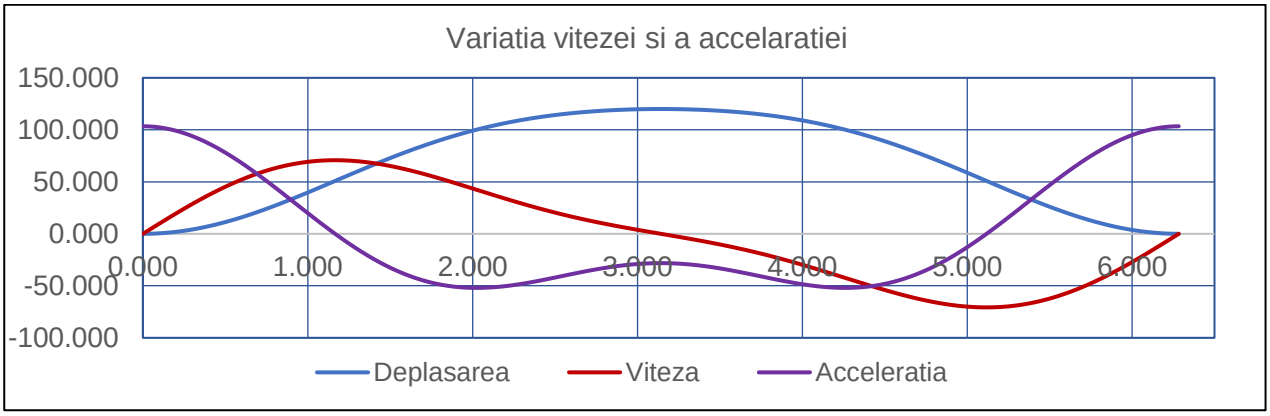


Figura 8.10b. Graficele de variaţie a deplasării, vitezei şi acceleraţiei

În tabelul 8.1 sunt prezentate o parte din valorile unghiului în grade, a unghiului în radiani, a deplasării, vitezei şi acceleraţiei calculate pe baza relaţiilor de mai sus;

Tabelul 8.1

Unghiul în grade φ	Unghiul în radiani φr	Deplasarea x [mm]	Viteza v [mm/s]	Acceleraţia a [mm/s²]
0	0.000	0.000	0.000	103.396
1	0.017	0.014	1.723	103.363
2	0.035	0.057	3.445	103.264
3	0.052	0.129	5.165	103.100
4	0.070	0.230	6.881	102.870
5	0.087	0.359	8.593	102.574
...	...	...	...	...
355	6.196	0.359	-8.593	102.574
356	6.213	0.230	-6.881	102.870
357	6.231	0.129	-5.165	103.100
358	6.248	0.057	-3.445	103.264
359	6.266	0.014	-1.723	103.363
360	6.283	0.000	0.000	103.396

Algoritmul de calcul pentru valorile deplasării, vitezei şi acceleraţiei constă în:

- se citesc variabilele de intrare, adică: lungimea bielei – **l** [mm], raza manivelei – **r** [mm], turaţia manivelei – **n** [rot/min];
- cu ajutorul unui ciclu cu contor se dau valori unghiului în grade, în intervalul [0, 360°], cu pasul de **1°**;
- pentru fiecare valoarea a unghiului în grade se calculează unghiul în radiani, valoarea deplasării – **x**, valoarea vitezei – **v**, valoarea acceleraţiei – **a**, pe baza relaţiilor de mai sus şi se afişează aceste valori;

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.10c, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.10d.

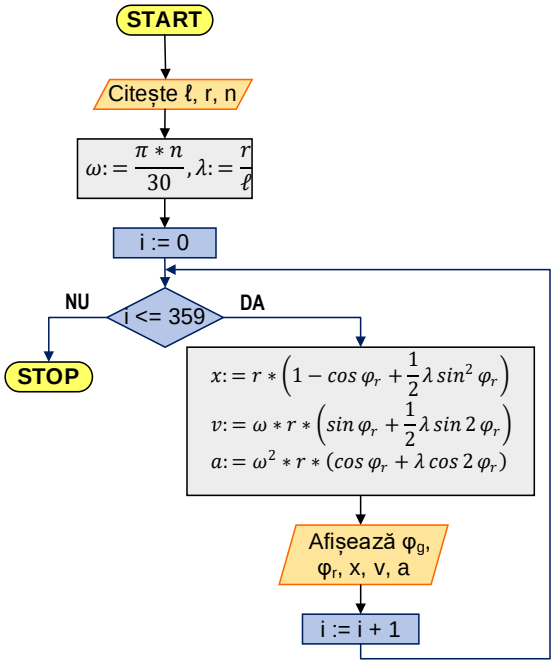
Schema logică	Limbajul pseudocod
	**Început** **Citeşte** l, r, n omg ←π * n / 30.0 lbd←r / l **Pentru** i = 0, 359 fig[i]←i fir[i]←fig[i] * π / 180 x[i]←r*(1-cos(fir[i])+lbd*pow(sin(fir[i]),2)/2.0) v[i]←omg*r*(sin(fir[i])+0.5*lbd*sin(2*fir[i]))/10^3 a[i]←omg^2*r*(cos(fir[i])+lbd*cos(2*fir[i]))/10^6 **Scrie** fig[i], fir[i], x, v, a **Sfârşit pentru** **Sfârşit**

Figura 8.10c. Reprezentarea algoritmului pentru sinteza mecanismului bielă - manivelă

Programul MATLAB şi execuţia acestuia

```
l=input('Lungimea bielei [m], l = ');
r=input('Raza manivelei [m], r = ');
n=input('Turatia manivelei [rot/min], n = ');
omg=pi*n/30.0; lbd=r/l;
for i=1:20:360
    fig(i)=i-1; fir(i)=fig(i)*pi/180;
    x(i)=r*(1-cos(fir(i))+lbd*sin(fir(i))^2/2.0);
    v(i)=omg*r*(sin(fir(i))+0.5*lbd*sin(2*fir(i)))/1000;
    a(i)=omg*omg*r*(cos(fir(i))+lbd*cos(2*fir(i)))/10^6;
    s=sprintf('\n  %9.6f  %9.6f  %9.6f  %9.6f  %9.6f',fig(i),fir(i),x(i),
v(i),a(i));  disp(s);
 end
```

Rularea programului:

```
 Lungimea bielei [m], l = 0.2
 Raza manivelei [m], r = 0.075
```

```
Turatia manivelei [rot/min], n = 3000
   0.000000  0.000000  0.000000  0.000000  0.010178
  20.000000  0.349066  0.006168  0.010898  0.009082
  40.000000  0.698132  0.023357  0.019496  0.006152
  60.000000  1.047198  0.048047  0.024231  0.002313
  80.000000  1.396263  0.075615  0.024715 -0.001323
 100.000000  1.745329  0.101662  0.021693 -0.003894
 120.000000  2.094395  0.123047  0.016579 -0.005089
 140.000000  2.443461  0.138264  0.010795 -0.005188
 160.000000  2.792527  0.147122  0.005219 -0.004829
 180.000000  3.141593  0.150000 -0.000000 -0.004626
 200.000000  3.490659  0.147122 -0.005219 -0.004829
 220.000000  3.839724  0.138264 -0.010795 -0.005188
 240.000000  4.188790  0.123047 -0.016579 -0.005089
 260.000000  4.537856  0.101662 -0.021693 -0.003894
 280.000000  4.886922  0.075615 -0.024715 -0.001323
 300.000000  5.235988  0.048047 -0.024231  0.002313
 320.000000  5.585053  0.023357 -0.019496  0.006152
 340.000000  5.934119  0.006168 -0.010898  0.009082
```

Figura 8.10d. Programul MATLAB şi execuţia acestuia pentru sinteza mecanismului bielă - manivelă

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

8.11. Calculul de dimensionare al capătului de arbore şi alegerea penei

Se consideră un capăt de arbore supus doar solicitării la torsiune. Relația de calcul al diametrului acestuia este:

$$d_p = \sqrt[3]{\frac{16 \cdot M_t}{\pi \cdot \tau_{at}}} \tag{8.54}$$

unde: M_t reprezintă momentul de torsiune [Nm], τ_{at} reprezintă rezistența admisibilă la torsiune (pentru oțeluri obișnuite se recomandă valori mici: τ_{at} = 12 … 25 N/mm^2, valorile mai mari se recomandă în cazul arborilor scurți, respectiv valorile mai mici în cazul arborilor lungi).

Valoarea obținută aplicând relația (8.34) se adoptă din gama de valori standardizate: 10, 11, 12, 14, 16, 18, 19, 20, 22, 24, 25, 28, 30, 32, 35, 38, 40, 42, 45, 48, 50, 55, 60, 65, 70, 71, 75, 80, 85, 90, conform STAS 8724/2-84.

Lungimea tronsonului de arbore se alege în funcție de diametrul acestuia, conform STAS 8724/2-84 (tabelul 8.2).

Pentru montarea roților de curea, a celor dințate sau a cuplajelor pe tronsoanele arborilor, se vor utiliza pene paralele (figura 8.9.a). Dimensiunile penei (**b** x **h**) și ale canalului de pană (t_1 și t_2) se aleg în funcție de diametrul tronsonului de arbore pe care se montează roata sau cuplajul, conform (tabelul 8.3).

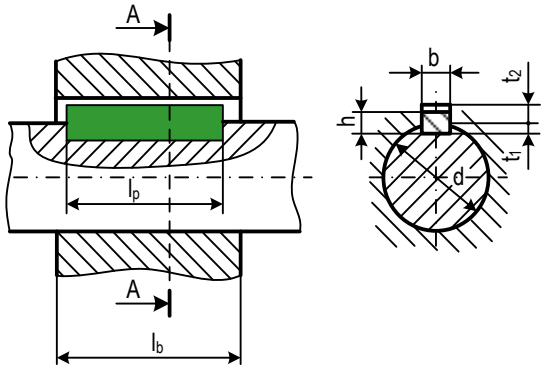


Figura 8.11a

Tabelul 8.2

Valori standardizate ale diametrelor arborilor și a lungimii acestora (serie scurtă)																
Diametrul [mm]	10	11	12	14	16	18	19	20	22	24	25	28	30	32	35	38
Lungimea [mm]	20	20	25	25	28	28	28	36	36	36	42	42	58	58	58	58

Diametrul [mm]	40	42	45	48	50	55	56	60	63	65	70	71	75	80	85	90
Lungimea [mm]	82	82	82	82	82	82	82	105	105	105	105	105	105	130	130	130

Tabelul 8.3

d [mm]		Dimensiunile penei [mm]		Dimensiunile canalului [mm]		Interval de lungimi [mm]
de la	până la	b	h	Adâncimea		
				arbore t_1	butuc t_2	l
10	12	4	4	2,5	1,8	8 … 45
12	17	5	5	3,0	2,3	10 … 56
17	22	6	6	3,5	2,8	14 … 70
22	30	8	7	4,0	3,3	18 … 90
30	38	10	8	5,0	3,3	22 … 110
38	44	12	8	5,0	3,3	28 … 140
44	50	14	9	5,5	3,8	36 … 160
50	58	16	10	6,0	4,3	45 … 180
58	65	18	11	7,0	4,4	50 … 200
65	75	20	12	7,5	4,9	56 … 220
75	85	22	14	9,0	5,4	63 … 250
85	95	25	14	9,0	5,4	70 … 280

Pentru calculul forţei care acţionează în asamblarea cu pană paralelă, se utilizează relaţia:

$$F = \frac{2 \cdot M_t}{d \cdot \left(1 + \mu \cdot \frac{4}{\pi}\right)} \tag{8.35}$$

unde: $\mathbf{M_t}$ reprezintă momentul de torsiune la care este supus tronsonul de arbore, **d** reprezintă diametrul tronsonului de arbore, **μ** reprezintă coeficientul de frecare dintre pană şi butucul roţii.

Lungimea penei se determină din:

a) limitarea presiunii de contact, pe baza relaţiei:

$$l_1 \geq \frac{2 \cdot F}{h \cdot p_a} \tag{8.36}$$

unde: **h** reprezintă înălţimea penei, în mm; $\mathbf{p_a}$ reprezintă presiunea admisibilă de contact, pentru sarcini pulsatorii, $\mathbf{p_a}$ = 65 … 100 [N/mm^2];

b) condiţia de rezistenţă la tensiunea de forfecare, pe baza relaţiei:

$$l_2 \geq \frac{F}{b \cdot \tau_{af}} \tag{8.37}$$

unde: **b** reprezintă lăţimea penei [mm], τ_{af} reprezintă tensiunea admisibilă la forfecare [N/mm^2];

Lungimile tipizate pentru pene paralele sunt (STAS 1004): 8, 10, 12, 14, 16, 18, 20, 22, 25, 28, 32, 36, 40, 45, 50, 56, 63, 70, 80, 90, 100, 110, 125, 140, 160, 180, 200, 220, 250, 280.

Valoarea maximă rezultată din aplicarea celor două relaţii se va standardiza prin adoptarea unei valori conform tabelului 8.4.

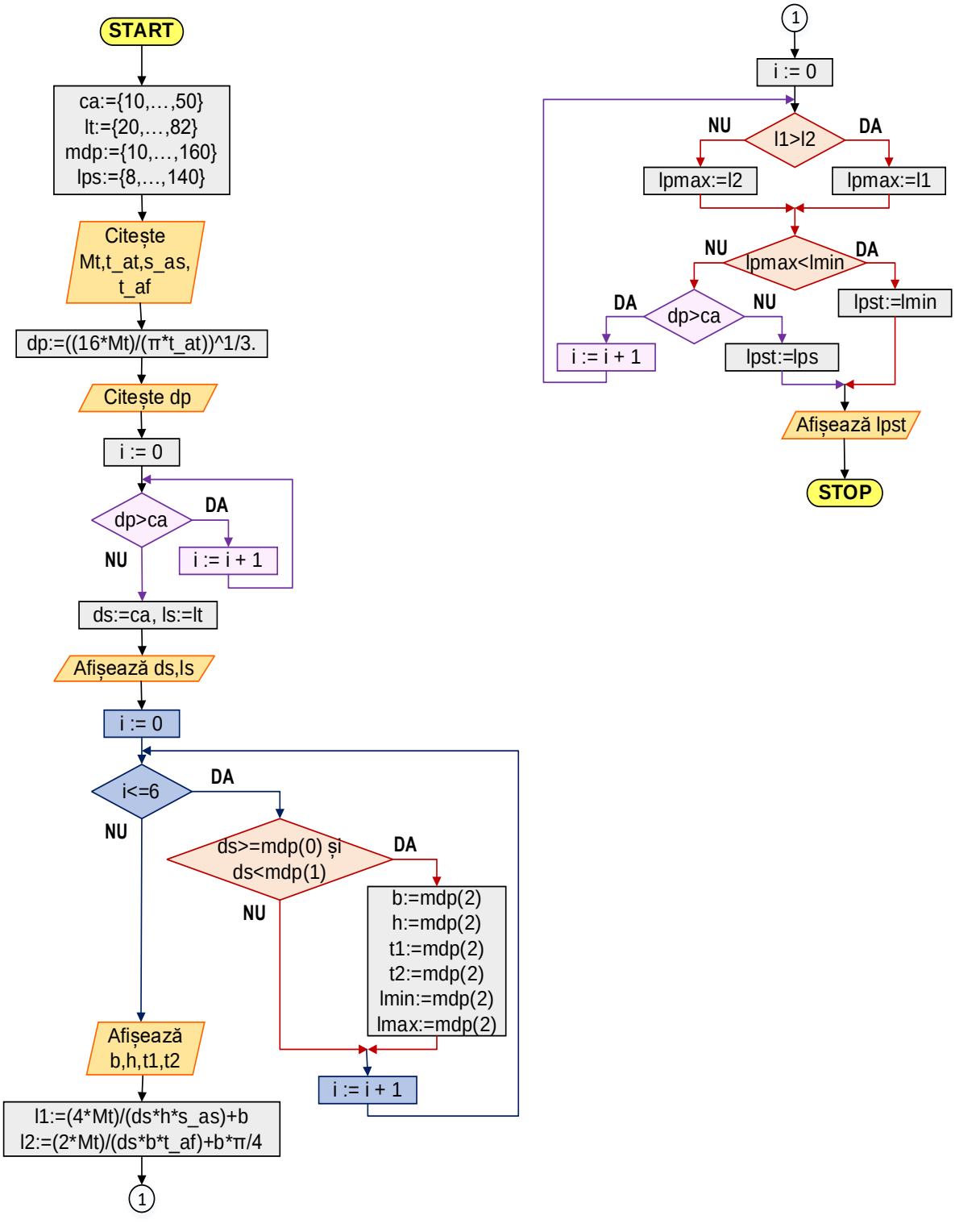
$$l_{st} \geq \max(l_1, l_2) \tag{8.38}$$

Tabelul 8.4. Extras din STAS 1004

b [mm]	h [mm]	Lungimea STAS [mm]													
6	6	16	18	20	22	25	28	32	36	40	45	50	56	63	70
8	7	20	22	25	28	32	36	40	45	50	56	63	70	80	90
10	8	25	28	32	36	40	45	50	56	63	70	80	90	100	110
12	8		28	32	36	40	45	50	56	63	70	80	90	100	110

14	9				36	40	45	50	56	63	70	80	90	100	110
16	10						45	50	56	63	70	80	90	100	110
18	11							50	56	63	70	80	90	100	110
20	12								56	63	70	80	90	100	110
22	14									63	70	80	90	100	110
25	14										70	80	90	100	110

Schema logică



Limbajul pseudocod

```
Început
  ca[21] ← {10,11,12,14,16,18,19,20,22,24,25,28,30,32,35,38,40,42,45,48,50}
  lt[21] ← {20,20,25,25,28,28,28,36,36,36,42,42,58,58,58,58,82,82,82,82,82}
  mdp[7][8] ← { {10,12,4,4,2.5,1.8,8,45},{12,17,5,5,3.0,2.3,10,56},{17,22,6,6,3.5,2.8,14,70},
{22,30,8,7,4.0,3.3,18,90},{30,38,10,8,5.0,3.3,18,90},{38,44,12,8,5.0,3.3,28,140},{44,50,14,9,5.5,
3.8,36,160} }
  lps[25] ← {8,10,12,14,16,18,20,22,25,28,32,36,40,45,50,56,63,70,80,90,100,110,125,140,160}
  Citeşte Mt, t_at, s_as, t_af
  dp ← (16*Mt/(PI*t_at))^1/3.
  i ← 0
  Cât timp dp > ca[i] execută
     i ← i + 1
  Sfârşit cât timp
  ds ← ca[i]
  ls ← lt[i]
  Scrie ds, ls
  Pentru i = 0, 6
        Dacă ds >= mdp[i][0] ŞI ds < mdp[i][1] atunci
              b ← mdp[i][2], h ← mdp[i][3], t1 ← mdp[i][4],
              t2 ← mdp[i][5], lmin ← mdp[i][6], lmax ← mdp[i][7]
        Sfârşit dacă
  Sfârşit pentru
  Scrie b, h, t1, t2
  l1 ← (4 * Mt) / (ds * h * s_as) + b
  l2 ← (2 * Mt) / (ds * b * t_af) + b * π / 4
  i ← 0
  Dacă l1 >= l2 atunci
        lpmax ← l1
  altfel
        lpmax ← l2
  Sfârşit dacă
  Dacă lpmax < lmin atunci
        lpst ← lmin
  altfel
          Cât timp lpmax > lps[i] execută
               i ← i + 1
           Sfârşit cât timp
          lpst ← lps[i]
  Sfârşit dacă
  Scrie lpst
Sfârşit
```

Figura 8.11b. Reprezentarea algoritmului calculul de dimensionare al capătului de arbore şi alegerea penei

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

Programul MATLAB şi execuţia acestuia

```
ca=[10,11,12,14,16,18,19,20,22,24,25,28,30,32,35,38,40,42,45,48,50];
lt=[20,20,25,25,28,28,28,36,36,36,42,42,58,58,58,58,82,82,82,82,82];
mdp=[10,12,4,4,2.5,1.8,8,45;12,17,5,5,3.0,2.3,10,56;17,22,6,6,3.5,2.8,1
4,70;22,30,8,7,4.0,3.3,18,90;30,38,10,8,5.0,3.3,18,90;38,44,12,8,5.0,3.
3,28,140;44,50,14,9,5.5,3.8,36,160];
lps=[8,10,12,14,16,18,20,22,25,28,32,36,40,45,50,56,63,70,80,90,100,110
,125,140];
Mt=input('Momentul de torsiune [N*mm], Mt = ');
t_at=input('Rezistenta admisibila la torsiune [15 ... 45 N/mm2], t_at =
');
s_as=input('Rezistenta admisibila la strivire [100 ... 120 N/mm2], s_as
= ');
t_af=input('Tensiunea admisibila la forfecare, [< 100 N/mm2] t_af = ');
dp = (16 * Mt / pi / t_at)^(1/3);
s=sprintf('Diametrul primitiv calculat, dp = %7.3f [mm]',dp); disp(s);
i = 1;
while dp > ca(i)
   i=i+1;
end
ds = ca(i); ls = lt(i);
s=sprintf('Diametrul standardizat al capatului de arbore, ds = %7.3f
[mm]',ds); disp(s);
s=sprintf('Lungimea standardizata a capatului de arbore, ls = %7.3f
[mm]]',ls); disp(s);
for i = 1:7
    if ds >= mdp(i,1) & ds < mdp(i,2)
      b = mdp(i,3); h = mdp(i,4);
      t1 = mdp(i,5); t2 = mdp(i,6);
      lmin = mdp(i,7); lmax = mdp(i,8);
    end
end
s=sprintf('Latimea penei, b = %7.3f [mm]',b); disp(s);
s=sprintf('Inaltimea penei, h = %7.3f [mm]',h); disp(s);
s=sprintf('Adancimea canalului in arbore, t1 = %7.3f [mm]',t1); disp(s);
s=sprintf('Adancimea canalului in butuc, t2 = %7.3f [mm]',t2); disp(s);
l1 = (4 * Mt) / (ds * h * s_as) + b;  l2 = (2 * Mt) / (ds * b * t_af) +
b * pi / 4;
i = 1;
if l1 > l2
    lpmax = l1;
else
    lpmax = l2;
end
if lpmax < lmin
    lpst = lmin;
else
    while  lpmax > lps(i)
```

```
        i=i+1;
    end
    lpst = lps(i);
end
s=sprintf('Lungimea standardizata a penei, lpst = %7.3f [mm]',lpst);
disp(s);
```

Rularea programului:

```
 Momentul de torsiune [N*mm], Mt = 20000
 Rezistenta admisibila la torsiune [15 ... 45 N/mm2], t_at = 25
 Rezistenta admisibila la strivire [100 ... 120 N/mm2], s_as = 110
 Tensiunea admisibila la forfecare, [< 100 N/mm2] t_af = 50
 Diametrul primitiv calculat, dp =  15.972 [mm]
 Diametrul standardizat al capatului de arbore, ds =  16.000 [mm]
 Lungimea standardizata a capatului de arbore, ls =  28.000 [mm]]
 Latimea penei, b =   5.000 [mm]
 Inaltimea penei, h =   5.000 [mm]
 Adancimea canalului in arbore, t1 =   3.000 [mm]
 Adancimea canalului in butuc, t2 =   2.300 [mm]
 Lungimea standardizata a penei, lpst =  16.000 [mm]
```

Figura 8.11c. Programul MATLAB şi execuţia acestuia pentru calculul de dimensionare al capătului de arbore şi alegerea penei

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

Algoritmul de rezolvare presupune parcurgerea următoarelor etape:

- citirea de la tastatură a momentului de torsiune la care este supus arborele M_t, în N*mm;
- citirea de la tastatură a rezistenţei admisibile la torsiune τ_{at}, se alege o valoare din intervalul [15, 45], în N/mm^2;
- citirea de la tastatură a valorii rezistenţei admisibile la strivire, σ_{as}, introducându-se o valoare din intervalul [100, 120] în N/mm^2;
- citirea de la tastatură a valorii tensiunii admisibile la forfecare, τ_{af}, introducându-se o valoare ≤ 100 [N/mm^2];
- se calculează diametrul preliminar **dp**, al capătului de arbore, cu relaţia (8.54);
- cu ajutorul unui ciclu cu test iniţial, se parcurge şirul valorilor standardizate ale diametrelor capetelor de arbore (notat cu **ca**) şi se alege cea mai apropiată valoare din şir, mai mare decât valoarea rezultată din calcul, reţinându-se de asemenea şi valoarea indicelui acesteia. Se afişează valoarea standardizată a diametrului capătului de arbore;
- din şirul valorilor standardizate ale lungimilor tronsoanelor (notat cu **lt**) se alege valoarea al căui indice este egal cu cel determinat la pasul anterior şi se afişează;
- cu ajutorul unui ciclu cu contor, se parcurge matricea care conţine dimensiunile penei (notată cu **mdp**) şi se determină rândul în care se încadrează diametrul capătului de arbore, se identifică indicele acestui rând şi cu ajutorul acestuia se culeg dimensiunile penei (lăţimea **b** şi înălţimea **h**), adâncimea canalului de pană în arbore **t1**, respectiv în butuc **t2**), precum şi limitele **lmin**, respectiv **lmax** ale lungimii penei;
- se determină lungimile preliminarii ale penei pe baza relaţiilor (8.55) şi (8.56) şi se determină valoarea maximă dintre cele două;
- cu ajutorul unui ciclu cu contor se parcurge şirul valorilor standardizate ale lungimilor penelor **lps**, între limitele stabilite anterior şi se determină cea mai apropiată valoare din şir care este mai mare decât valoarea preliminară. Se afişează valoarea determinată.

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.11b, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.11c. Din considerente legate de spaţiu, s-au utilizat valorile standardizate din şiruri corespunzătoare până la diametrul arborelui de maxim 50 mm.

8.12. Grinda cu forţe

Se consideră o grindă (figura 8.12a), sprijinită pe două reazeme, unul fix şi unul mobil, asupra căruia pot să acţioneze:

- **forţe punctiform aplicate**, pentru care se cunosc: modulul, direcţia (dată de unghiul de înclinare al forţei, măsurat faţă de direcţia orizontală, în sens trigonometric) şi sensul, respectiv punctul de aplicaţie;
- **forţe distribuite**, pentru care se cunosc: modulul pe unitate de lungime, sensul, punctul de început şi punctul de sfârşit. În calcule forţele distribuite se înlocuiesc cu forţe punctiform aplicate al căror punct de aplicaţie se consideră la jumătatea intervalului de acţiune, modulul acestora rezultând din produsul dintre modulul pe unitatea de lungime şi lungimea de acţiune a forţei distribuite;
- **momente**, care acţionează asupra grinzii şi pentru care se cunosc: punctul de aplicaţie, modulul şi sensul.

În reazemul fix apar două reacţuni:

- **reacţiunea orizontală**, notată cu **H**, care acţionează în lungul grinzii;
- **reacţiunea verticală**, notată cu **V**, care acţionează perpendicular pe grindă;

În reazemul mobil apare o **reacţiune normală**, notată cu **N**, care acţionează perpendicular pe grindă.

În calcule, pentru forţe se consideră că acestea au valori **pozitive** dacă acţionează în sus, respectiv au valori **negative** dacă acţionează în jos (figura 8.12a). Pentru momente, se consideră că acestea au valori **pozitive** dacă acţionează în sens trigonometric (sens antiorar), respectiv au valori **negative** dacă acţionează în sens orar.

Pentru determinarea reacţiunilor este necesar să se rezolve un sistem format din trei ecuaţii cu trei necunoscute:

- o ecuaţie de forţe, scrisă pe direcţie orizontală, în care avem o singură necunoscută: **reacţiunea orizontală H**;
- o ecuaţie de forţe, scrisă pe direcţie verticală, în care apar două necunoscute: **reacţiunea verticală V** şi **reacţiunea normală N**;
- o ecuaţie de momente, scrisă în raport cu punctul de aplicaţie al reazemului fix, în care apare o singură necunoscută: **reacţiunea normală N**;

Se observă că din prima ecuaţie se poate determina direct **reacţiunea orizontală H**. De asemenea, din ultima ecuaţie se poate determina direct **reacţiunea normală N**. Cunoscând valoarea **reacţiunii normale N**, din cea de a doua ecuaţie se determină **reacţiunea verticală V**.

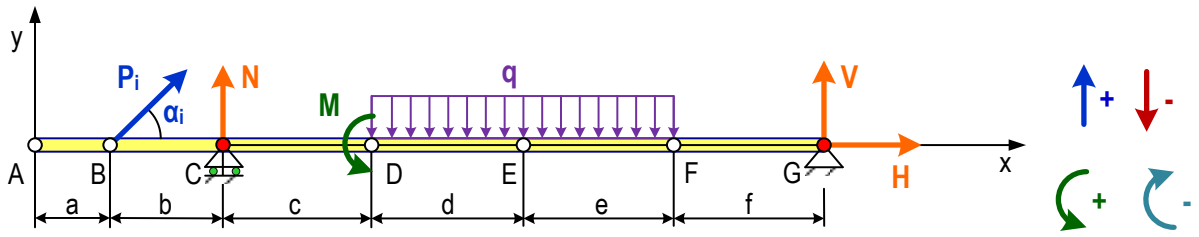


Figura 8.12a. Grindă cu forţe

Algoritmul de calcul al reacţiunilor din reazeme presupune parcurgerea următorilor paşi:

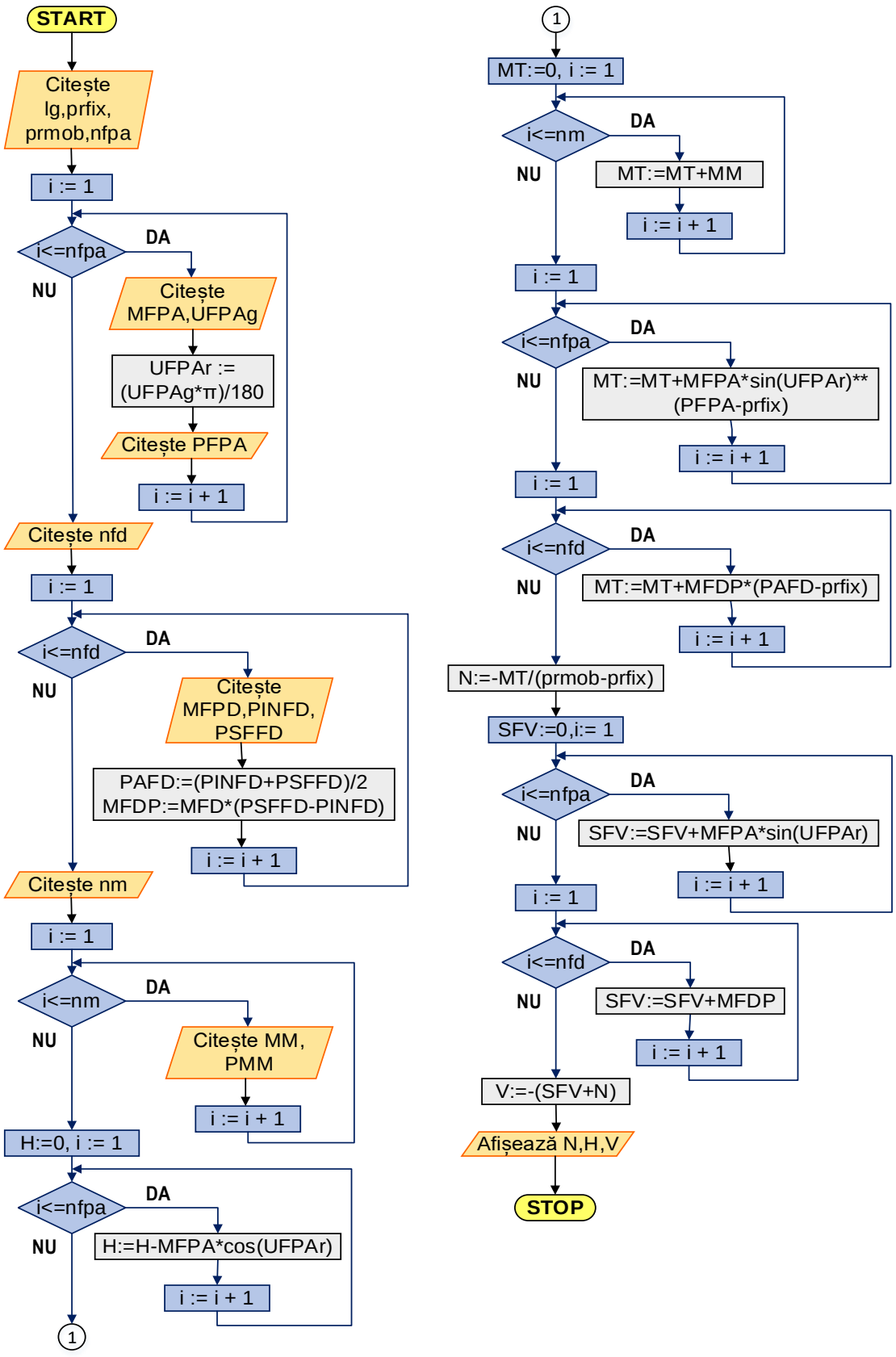
A. Se citeşte de la tastatură lungimea grinzii;
B. Se citeşte de la tastatură poziţia reazemului fix;
C. Se citeşte de la tastatură poziţia reazemului mobil;
D. Se citeşte de la tastatură numărul de forţe punctiform aplicate;
E. Utilizând un ciclu cu contor, se citeşte de la tastatură pentru fiecare forţă aplicată punctiform: **modulul**, **direcţia** (unghiul pe care îl face forţa cu orizontala, măsurat în sens trigonometric), respectiv coordonata **x** a punctului de aplicaţie al forţei;
F. Se citeşte de la tastatură numărul de forţe distribuite;
G. Utilizând un ciclu cu contor, se citeşte pentru fiecare forţă distribuită: modulul (dacă forţa acţionează în sus se introduce o valoare pozitivă, iar dacă forţa acţionează în jos se introduce o valoare negativă), coordonata x a punctului de unde începe acţiunea forţei distribuite, respectiv coordonata x a punctului unde se sfârşeşte acţiunea forţei distribuite;

H. Se citeşte de la tastatură numărul de momente;
I. Utilizând un ciclu cu contor, pentru fiecare moment se introduce modulul (dacă momentul acţionează în sens trigonometric – antiorar, se introduce o valoare pozitivă, iar dacă momentul acţionează în sens orar se introduce o valoare negativă);
J. Se iniţializează valoarea reacţiunii orizontale (notată cu **H** în program) cu zero;
K. Utilizând un ciclu cu contor, se calculează valoarea reacţiunii orizontale, scăzând din valoarea acesteia valoarea componentei orizontale ale fiecărei forţe punctiform aplicate;
L. Se iniţializează valoarea momentului total (notat cu **MT** în program) cu zero;
M. Utilizând un ciclu cu contor, se adaugă la valoarea momentului total valoarea fiecărui moment care acţionează asupra grinzii;
N. Utilizând un ciclu cu contor, se adaugă la valoarea momentului total valoarea momentelor date de fiecare forţă punctiform aplicată în raport cu punctul în care se găseşte reazemul fix;
O. Utilizând un ciclu cu contor se adaugă la momentul total valoarea momentelor date de forţele distribuite, în raport cu punctul în care se găseşte reazemul fix. Pentru a calcula aceste momente se înlocuiesc forţele distribuite cu forţe punctiform aplicate al căror modul este dat de produsul dintre modulul forţei distribuite şi lungimea pe care acţionează aceasta, iar punctul de aplicaţie se află la jumătatea distanţei pe care acţionează forţa distribuită. Momentul dat de o forţă distribuită este dat de produsul dintre modulul forţei punctiform aplicată care înlocuieşte forţa distribuită şi distanţa dintre poziţia reazemului fix şi cea a punctului de aplicaţie a forţei punctiform aplicată care înocuieşte forţa distribuită;
P. Se calculează reacţiunea normală **N** prin împărţirea momentului total calculat anterior la distanţa dintre cele două reazeme;
R. Se iniţializează valoarea variabilei **SFV** (suma forţelor verticale) cu zero;
S. Utilizând un ciclu cu contor, se adaugă variabilei **SFV** componenta verticală a fiecărei forţe punctiform aplicate;
T. Utilizând un ciclu cu contor se adaugă variabilei **SFV** valoarea forţei punctiform aplicate care înlocuieşte forţa distribuită, pentru fiecare forţă distribuită în parte;
U. Se calculează reacţiunea verticală V;
V. Se afişează cele trei reacţiuni calculate: **orizontală H**, **verticală V** şi **normală N**;

În cadrul programului s-au făcut următoarele notaţii:
nfpa – numărul de forţe punctiform aplicate;
nfd – numărul de forţe distribuite;
nm – numărul de momente aplicate;
lg – lungimea grinzii [cm];
prfix – poziţia reazemului fix, adică coordonata x a punctului unde este plasat reazemul fix, considerând ca origine extremitatea stângă a grinzii [cm];
prmob – poziţia reazemului mobil, adică coordonata x a punctului unde este plasat reazemul mobil, considerând ca origine extremitatea stângă a grinzii [cm];
MFPA – modulul forţei punctiform aplicate [N];
UFPAg – unghiul pe care îl face forţa punctiform aplicată cu direcţia orizontală, măsurat în sens trigonometric [grade];
UFPAr - unghiul pe care îl face forţa punctiform aplicată cu direcţia orizontală, măsurat în sens trigonometric [radiani];
PFPA – poziţia punctului în care acţionează forţa punctiform aplicată [cm];
MFD – modulul forţei distribuite [N/cm];
PINFD – poziţia punctului de început a lungimii pe care acţionează forţa distribuită [cm];
PSFFD – poziţia punctului de sfârşit a lungimii pe care acţionează forţa distribuită [cm];
MFDP – modulul forţei punctiform aplicate, care înlocuieşte forţa distribuită [N];
PAFD – poziţia punctului în care acţionează forţa punctiform aplicată care înlocuieşte forţa distribuită [cm];
MM – modulul unui moment aplicat grinzii [N*cm];
PMM – poziţia punctului unde este aplicat momentul [cm];

Schema logică şi reprezentarea în limbaj pseudocod sunt ilustrate în figura 8.12b, iar programul în limbajul MATLAB şi rularea acestuia sunt prezentate în figura 8.12c.

Schema logică



Limbajul pseudocod

```
Început
  Citeşte lg, prfix, prmob
  Citeşte nfpa
  Pentru i = 1, nfpa
     Citeşte MFPA[i], UFPAg[i], PFPA[i]
     UFPAr[i] ← UFPAg[i] * PI/180
  Sfârşit pentru
  Citeşte nfd
  Pentru i = 1, nfd
     Citeşte MFD[i], PINFD[i], PSFFD[i]
     PAFD[i] ← (PINFD[i]+PSFFD[i]) / 2
     MFDP[i] ← MFD[i] * (PSFFD[i] - PINFD[i])
  Sfârşit pentru
  Citeşte nm
  Pentru i = 1, nm
     Citeşte MM[i], PMM[i]
  Sfârşit pentru
  H ← 0
  Pentru i = 1, nfpa
     H ← H - MFPA[i] * cos(UFPAr[i])
  Sfârşit pentru
  MT ← 0
  Pentru i = 1, nm
     MT ← MT + MM[i]
  Sfârşit pentru
  Pentru i = 1, nfpa
     MT ← MT + MFPA[i] * sin(UFPAr[i]) * (PFPA[i] - prfix)
  Sfârşit pentru
  Pentru i = 1, nfd
     MT ← MT + MFDP[i] * (PAFD[i] - prfix)
  Sfârşit pentru
  N ← - MT / (prmob - prfix)
  SFV ← 0
  Pentru i = 1, nfpa
     SFV ← SFV + MFPA[i] * sin(UFPAr[i])
  Sfârşit pentru
  Pentru i = 1, nfd
     SFV ← SFV + MFDP[i]
  Sfârşit pentru
  V ← -(SFV + N)
  Scrie N, H, V
Sfârşit
```

Figura 8.12b. Reprezentarea algoritmului calculul reacţiunilor din reazeme pentru o grindă încărcată cu sarcini

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

Programul MATLAB şi execuţia acestuia

```
lg=input('Lungimea grinzii [cm], lg = ');
prfix=input('Pozitia reazemului fix [cm], prfix = ');
prmob=input('Pozitia reazemului mobil [cm], prmob =  ');
nfpa=input('Numarul de forte punctiform aplicate, nfpa = ');
for i = 1: nfpa
    s=sprintf('Modulul fortei pct. aplic. [N], MFPA[%d] = ',i);
MFPA(i)=input(s);
    s=sprintf('Unghiul fortei pct. aplic. [grade], UFPA[%d] =',i);
UFPAg(i)=input(s);
    UFPAr(i) = UFPAg(i) * pi/180.0;
    s=sprintf('Pozitia fortei pct. aplic. [cm], PFPA[%d] =',i);
PFPA(i)=input(s);
end
nfd=input('Numarul de forte distribuite, nfd =');
for i = 1: nfd
    s=sprintf('Modulul fortei distribuite [N/cm], MFD[%d] = ',i);
MFD(i)=input(s);
    s=sprintf('Punctul de start al fortei distrib. [cm], PINFD[%d] =
',i); PINFD(i)=input(s);
    s=sprintf('Punctul final al fortei distrib. [cm], PSFFD[%d] = ',i);
PSFFD(i)=input(s);
    PAFD(i)=(PINFD(i)+PSFFD(i))/2;  MFDP(i)=MFD(i)*(PSFFD(i)-PINFD(i));
end
nm=input('Numarul de momente, nm = ');
for i = 1: nm
    s=sprintf('Modulul momentului [N*cm], MM[%d] = ',i); MM(i)=input(s);
    s=sprintf('Punctul unde se aplica momentului [cm], PMM[%d] = ',i);
PMM(i)=input(s);
end
% Ecuatia de forte pe directie orizontala
  H = 0;
  for i = 1: nfpa
    H = H - MFPA(i) * cos(UFPAr(i));
  end
 %Ecuatia de momente
  MT = 0;
  for i = 1 : nm
    MT = MT + MM(i);
  end
  for i = 1: nfpa
    MT = MT + MFPA(i) * sin(UFPAr(i)) * (PFPA(i) - prfix);
  end
  for i = 1: nfd
    MT = MT + MFDP(i) * (PAFD(i) - prfix);
  end
  N = - MT / (prmob - prfix);
% Ecuatia de forte pe directie verticala
```

```
  SFV = 0;
  for  i = 1: nfpa
    SFV = SFV + MFPA(i) * sin(UFPAr(i));
  end
  for i = 1: nfd
    SFV = SFV + MFDP(i);
  end
  V = - (SFV + N);
s=sprintf('Reactiunea normala, N = %9.3f [N]',N); disp(s);
s=sprintf('Reactiunea orizontala, H = %9.3f [N]',H); disp(s);
s=sprintf('Reactiunea verticala, V = %9.3f [N]',V); disp(s);
```

Rularea programului:

Cazul 1:

```
 Lungimea grinzii [cm], lg = 100
 Pozitia reazemului fix [cm], prfix = 10
 Pozitia reazemului mobil [cm], prmob = 90
 Numarul de forte punctiform aplicate, nfpa = 2
 Modulul fortei pct. aplic. [N], MFPA[1] = 300
 Unghiul fortei pct. aplic. [grade], UFPA[1] = 270
 Pozitia fortei pct. aplic. [cm], PFPA[1] = 0
 Modulul fortei pct. aplic. [N], MFPA[2] = 500
 Unghiul fortei pct. aplic. [grade], UFPA[2] = 60
 Pozitia fortei pct. aplic. [cm], PFPA[2] = 50
 Numarul de forte distribuite, nfd = 1
 Modulul fortei distribuite [N/cm], MFD[1] = -10
 Punctul de start al fortei distrib. [cm], PINFD[1] = 30
 Punctul final al fortei distrib. [cm], PSFFD[1] = 90
 Numarul de momente, nm = 1
 Modulul momentului [N*cm], MM[1] = 2000
 Punctul unde se aplica momentului [cm], PMM[1] = 10

 Reactiunea normala,    N =    95.994 [N]
 Reactiunea orizontala, H =  -250.000 [N]
 Reactiunea verticala,   V =   370.994 [N]
```

Cazul 2:

```
 Lungimea grinzii [cm], lg = 100
 Pozitia reazemului fix [cm], prfix = 90
 Pozitia reazemului mobil [cm], prmob = 10
 Numarul de forte punctiform aplicate, nfpa = 2
 Modulul fortei pct. aplic. [N], MFPA[1] = 400
 Unghiul fortei pct. aplic. [grade], UFPA[1] = 90
 Pozitia fortei pct. aplic. [cm], PFPA[1] = 100
 Modulul fortei pct. aplic. [N], MFPA[2] = 500
 Unghiul fortei pct. aplic. [grade], UFPA[2] = 240
 Pozitia fortei pct. aplic. [cm], PFPA[2] = 20
 Numarul de forte distribuite, nfd = 1
 Modulul fortei distribuite [N/cm], MFD[1] = -15
 Punctul de start al fortei distrib. [cm], PINFD[1] = 20
```

```
Punctul final al fortei distrib. [cm], PSFFD[1] = 60
Numarul de momente, nm = 1
Modulul momentului [N*cm], MM[1] = -3000
Punctul unde se aplica momentului [cm], PMM[1] = 90

Reactiunea normala,     N =   766.386 [N]
Reactiunea orizontala, H =   250.000 [N]
Reactiunea verticala,    V =  -133.373 [N]
```

Figura 8.12c. Programul MATLAB şi execuţia acestuia pentru calculul reacţiunilor din reazeme pentru o grindă încărcată cu sarcini

Observaţie: valorile **bolduite** de la execuţia programului corespund datelor introduse de utilizator de la tastatură.

Cazul 1: Se consideră grinda din figura 8.12d, pentru care avem:
- lungimea grinzii: **100** [cm];
- poziţia reazemului fix: **10** [cm];
- poziţia reazemului mobil: **90** [cm];
- o forţă punctiform aplicată P_1, având: poziţia punctului de aplicaţie **0** [cm], modulul **300** [N], unghiul de orientare **270** [°];
- o forţă punctiform aplicată P_2 având: poziţia punctului de aplicaţie **50** [cm], modulul **500** [N], unghiul de orientare **60** [°];
- o forţă distribuită **q** având: modulul pe unitatea de lungime **-10** [N/cm], poziţia punctului de început al forţei **30** [cm], poziţia punctului final al forţei **90** [cm]. În calcule această forţă se înlocuieşte cu o forţă punctiform aplicată, notată Q, al cărei modul este dat de produsul dintre modulul pe unitatea de lungime şi lungimea ocupată pe grindă, adică **-600** [N], punctul de aplicaţie al acestei forţe fiind la jumătatea distanţei ocupate de forţa distribuită, adică **60** [cm];
- un moment **M** având: modulul de **2000** [N*cm] şi punctul de aplicaţie **10** [cm];

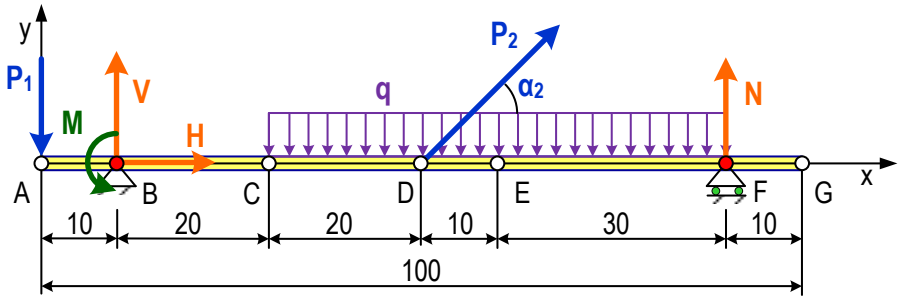


Figura 8.12d. Grindă încărcată – **cazul 1**

Rezolvare:

A. Se scrie mai întâi ecuaţia de forţe pe direcţie orizontală, cu ajutorul căreia se determină **reacţiunea orizontală H**:

$$H + P_2 \cdot cos(\alpha_2) = 0$$

de unde rezultă:

$$H = -P_2 \cdot cos(\alpha_2) = -500 \cdot cos(60) = \mathbf{-250.000}\ [\text{N}]$$

B. Se scrie ecuaţia de momente în raport cu punctul de aplicaţie al reazemului fix, rezultând **reacţiunea normală N**:

$$P_1 \cdot AB + M + P_2 \cdot sin(\alpha_2) \cdot BD - q \cdot CF \cdot BE + N \cdot BF = 0$$

de unde rezultă:

$$N = \frac{-P_1 \cdot AB - M - P_2 \cdot sin(\alpha_2) \cdot BD + q \cdot CF \cdot BE}{BF}$$

$$N = \frac{-300 \cdot 10 - 2000 - 500 \cdot 0.866025 \cdot 40 + 10 \cdot 60 \cdot 50}{80} = \frac{-3000 - 2000 - 17320.508 + 30000}{80}$$

$$N = \frac{-3000 - 2000 - 17320.508 + 30000}{80} = \frac{7679.492}{80} = \mathbf{95.994}\ [\text{N}]$$

C. Se scrie ecuaţia de forţe pe direcţie verticală, cu ajutorul căreia se determină **reacţiunea verticală V**:

$$-P_1 + V + q * CF + P_2 \cdot sin(60) + N = 0$$

de unde rezultă:

$$V = P_1 - q * CF - P_2 \cdot sin(60) - N = 300 - (-10) \cdot 60 - 500 \cdot 0.866025 - 95.994 = \mathbf{370.994}\ [\text{N}]$$

Cazul 2: Se consideră grinda din figura 8.12e, pentru care avem:
- lungimea grinzii: **100** [cm];
- poziţia reazemului fix: **90** [cm];
- poziţia reazemului mobil: **10** [cm];
- o forţă punctiform aplicată P_1, având: poziţia punctului de aplicaţie **100** [cm], modulul **400** [N], unghiul de orientare **90** [°];
- o forţă punctiform aplicată P_2 având: poziţia punctului de aplicaţie **20** [cm], modulul **500** [N], unghiul de orientare **240** [°];
- o forţă distribuită **q** având: modulul pe unitatea de lungime **-15** [N/cm], poziţia punctului de început al forţei **20** [cm], poziţia punctului final al forţei **60** [cm]. În calcule această forţă se înlocuieşte cu o forţă punctiform aplicată, notată **Q**, al cărei modul este dat de produsul dintre modulul pe unitatea de lungime şi lungimea ocupată pe grindă, adică **-600** [N], punctul de aplicaţie al acestei forţe fiind la jumătatea distanţei ocupate de forţa distribuită, adică **40** [cm];
- un moment **M** având: modulul de **-3000** [N*cm] şi punctul de aplicaţie **90** [cm];

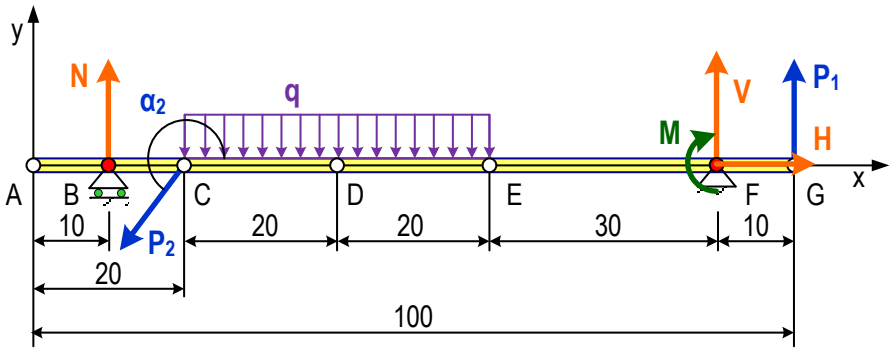


Figura 8.12e. Grindă încărcată – **cazul 2**

Rezolvare:

A. Se scrie mai întâi ecuaţia de forţe pe direcţie orizontală, cu ajutorul căreia se determină **reacţiunea orizontală H**:

$$H + P_2 \cdot cos(\alpha_2) = 0$$

de unde rezultă:

$$H = -P_2 \cdot cos(\alpha_2) = -500 \cdot cos(240) = \mathbf{250.000}\ [\text{N}]$$

B. Se scrie ecuaţia de momente în raport cu punctul de aplicaţie al reazemului fix, rezultând **reacţiunea normală N**:

$$-N \cdot BF - P_2 \cdot sin(\alpha_2) \cdot CF + q \cdot CE \cdot DF - M + P_1 \cdot FG = 0$$

de unde rezultă:

$$N = \frac{-P_2 \cdot sin(\alpha_2) \cdot CF - q \cdot CE \cdot DF - M + P_1 \cdot FG}{BF}$$

$$N = \frac{-500 \cdot (-0.866025) \cdot 70 - (-15) \cdot 40 \cdot 50 - 3000 + 400 \cdot 10}{80}$$

$$N = \frac{30310.875 + 30000 - 3000 + 4000}{80} = \frac{61310.875}{80} = \mathbf{766.386}\ [\text{N}]$$

C. Se scrie ecuaţia de forţe pe direcţie verticală, cu ajutorul căreia se determină **reacţiunea verticală V**:

$$N + P_2 \cdot sin(\alpha_2) + q * CE + V + P_1 = 0$$

de unde rezultă:

$$V = -P_1 - q * CE - P_2 \cdot sin(\alpha_2) - N = -400 - (-15) \cdot 40 - 500 \cdot (-0.866025) - 766.386$$

$$V = \mathbf{-133.373}\ [\text{N}]$$

Bibliografie

[1] Vaida C., Gherman B., Pisla D., Programare in MATLAB cu aplicaţii în inginerie, Mediamira, 2014, 380 pp., ISBN 978-973-713-312-0

[2] Vaida, C., Pisla, D., Utilizarea calculatoarelor. Aplicaţii. Vol. I, Editura MEDIAMIRA, 2009

[3] Curteanu, S., Initiere in MatLab, Editura POLIROM, 2008, ISBN 978-973-46-0920-8, 315 pg.

[4] Gilat, A., MATLAB An Introduction with Applications, J Wiley & Sons, Inc., 2004, ISBN 0-471-43997-5

[5] Hahn, B.D., Valentine, D.T., Essential MATLAB for Engineers and Scientists, Elsevier, 2007

[6] Hunt, B.R., Lipsman, R.L., Rosenberg, J.M., A Guide to MATLAB - For Beginners and Experienced Users, Second Edition, Cambridge University Press, Cambridge UK, 2006, ISBN-13 978-0-521-61565-5.

[7] Chapman, S.J., Essentials MATLAB Programming 2nd Edition, Cengage Learning, USA, 2006, ISBN-13: 978-0-495-29568-6, 429 pg.

[8] Kalechman, M., Practical MATLAB basics for engineers, CRC Press, Boca Raton, Florida, 2007, ISBN 978-1-4200-4774-5, 736 pg.

[9] Quarteroni, A., Saleri, F., Gervasio, P., Scientific Computing eith MATLAB and OCTAVE, Third Edition, Springer – Verlag, 2010, ISSN 1611-0994, ISBN 978-3-642-12429-7, e-ISBN 978-3-642-12430-3

[10] Smith, D.M., Engineering Computation with MATLAB 2nd Edition, Pearson Education, Inc., 2010.

[11] Cleve Moler, Experiments with MATLAB, Free E-book, 2011

Anexa A. Mediul de programare Octave

Mediul de programare Octave permite scrierea şi executarea unor programe ale căror cod sursă este scris în limbajul MATLAB și împreună cu FreeMat sunt variante gratuite ale mediului de programare MATLAB.

A.1. Instalarea mediului de programare

Pentru a realiza instalarea mediului de programare Octave pe calculatorul dvs. trebuie să parcurgeţi următoarele etape:

A. Obţinerea fişierului **octave-6.3.0-w64-installer.exe** sau **octave-6.3.0-w32-installer.exe** care conţine kit-ul de instalare, de pe site-ul https://www.gnu.org/software/octave/download (în funcție de versiunea dvs. de Windows pe 64 de biți sau pe 32 de biți).

B. Se lansează în execuţie fişierul descărcat anterior. Pe ecran se deschide fereastra din figura A.1.

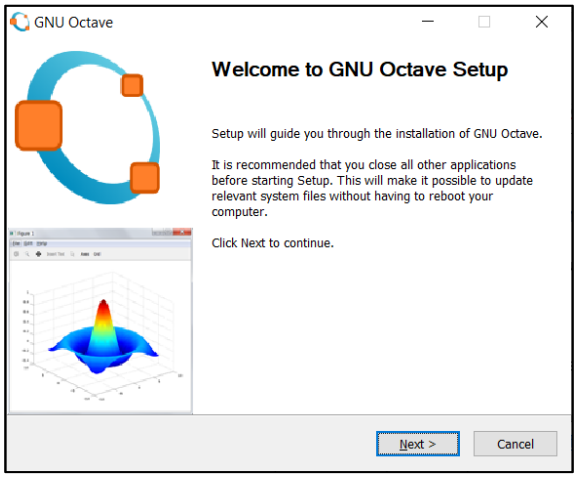


Figura A.1.

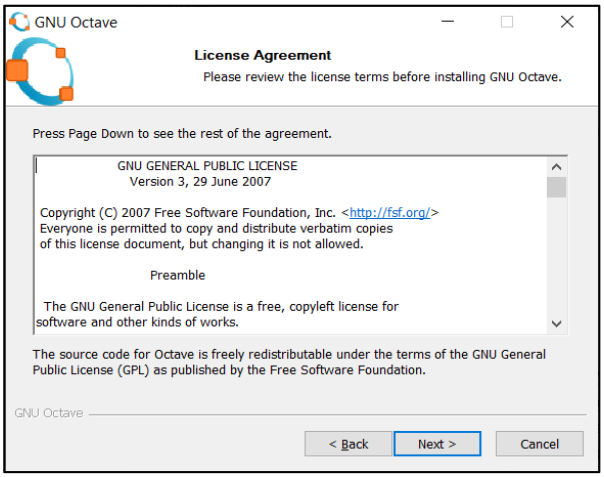


Figura A.2.

Se acţionează butonul **Next**. Pe ecran se deschide fereastra ***License Agreement*** (figura A.2). Din nou se apasă butonul **Next**.

C. **Alegerea utilizatorilor**. După parcurgerea etapei anterioare, pe ecran se deschide fereastra din figura A.3, unde se pot alege utilizatorii mediului de programare Octave. Apoi se apasă butonul **Next.**

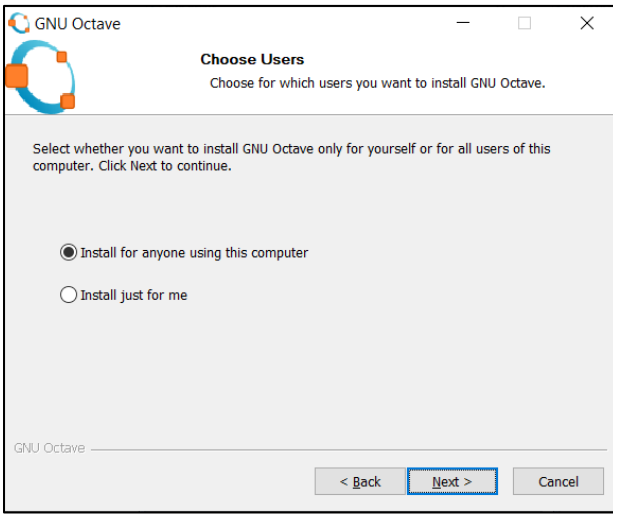


Figura A.3.

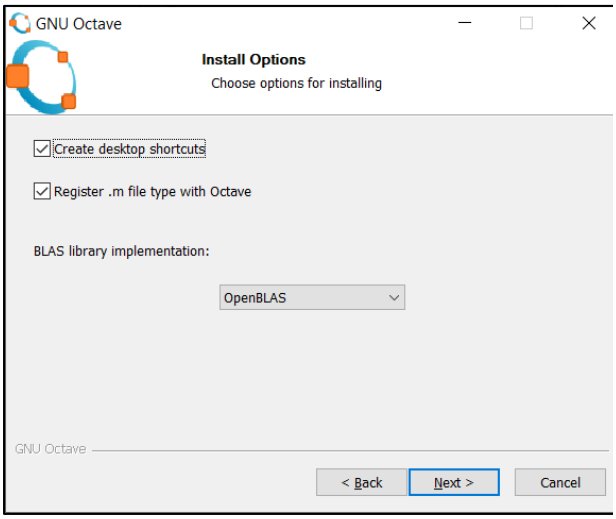


Figura A.4.

D. **Selectarea opțiunilor de instalare**. După alegerea utilizatorilor pe ecran se deschide fereastra din figura A.4 care permite setarea opțiunilor de instalare (crearea unui shortcut pe Desktop, asocierea fișierelor cu extensia .m cu mediul de programare Octave). Apoi se apasă butonul **Next.**

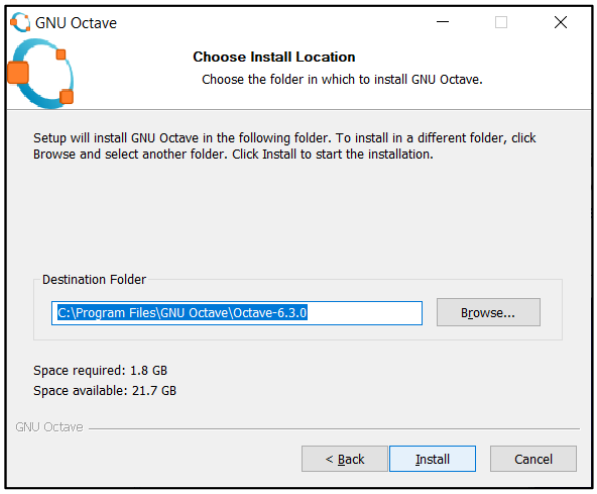


Figura A.5.

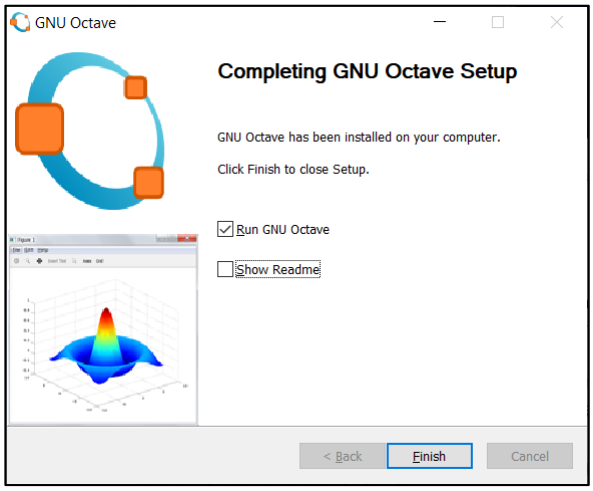


Figura A.6.

E. **Selectarea directorului destinaţie**. După selectarea opțiunilor de instalare, pe ecran se deschide fereastra ***Choose install location*** (figura A.5). Dacă se doreşte se poate modifica directorul unde să fie instalate fişierele. Se acţionează butonul **Install**. În câteva minute se realizează instalarea mediului de programare.

F. **Finalizarea instalării**. După instalarea fişierelor pe calculatorul dvs. în directorul specificat, pe ecran se deschide fereastra prezentată în figura A.6. Pentru finalizarea instalării se acţionează butonul **Finish**.

G. **Lansarea în execuţie a aplicaţiei**. După finalizarea instalării, dacă este activată opţiunea ***Run GNU Octave*** pe ecran se deschide aplicaţia ***Octave***, a cărei interfaţă este prezentată în figura A.7.

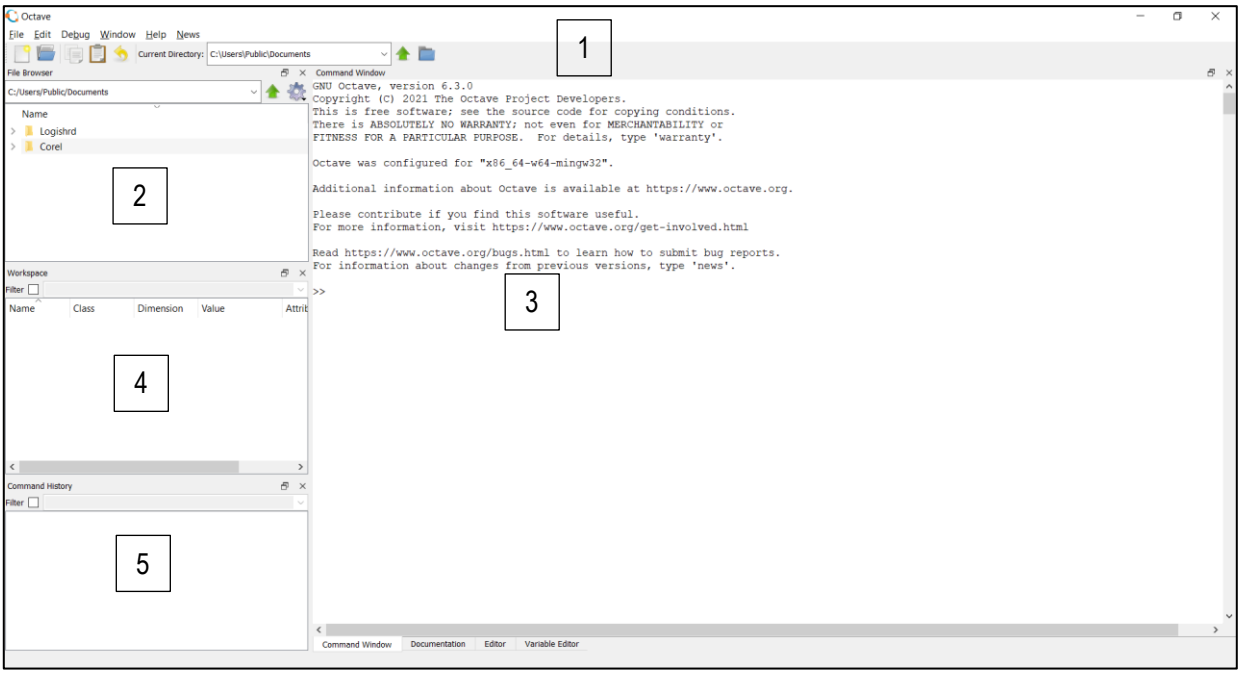


Figura A.7. Interfaţa mediului de programare Octave

A.2. Prezentarea mediului de programare

Interfaţa mediului de programare este prezentată în figura A.7. Elementele componente ale acesteia sunt:
1 – meniul principal și bara cu unelte;
2 – browser-ul de fișiere: permite vizualizarea directoarelor și a fișierelor curente;
3 – fereastra de comenzi: permite introducerea comenzilor, definirea variabilelor sau a constantelor, precum și definirea și apelarea funcțiilor;
4 – fereastra “Workspace”: în această fereastră apar toate variabilele utilizate;
5 – fereastra de istoric: ilustrează în ordine cronologică toate comenzile introduse precum şi rezultatele executării acestora.

Meniul principal conţine 6 meniuri, în continuare fiind prezentare cele mai importante.

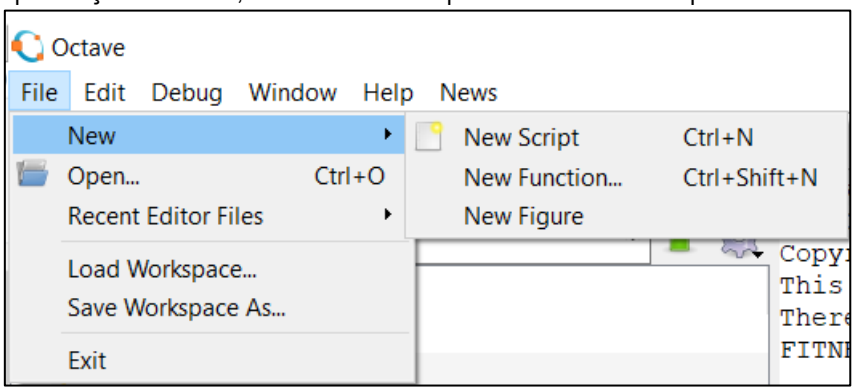


Figura A.8. Meniul **File**

Meniul **File** (figura A.8): se utilizează pentru gestionarea fişierelor, având opţiunile:
New : permite crearea unui element nou (fişier script, fișier de tip funcție, figură);
Open … : deschide un fişier existent;
Recent Editor Files : afişează o listă cu ultimele fişiere deschise (istoric) şi permite deschiderea acestora;
Load Workspace : importarea variabilelor și valorilor acestora care au fost salvate anterior;
Save Workspace As … : salvarea variabilelor utilizate precum și a valorilor și proprietăților acestora ;
Exit : părăsirea mediului de programare;

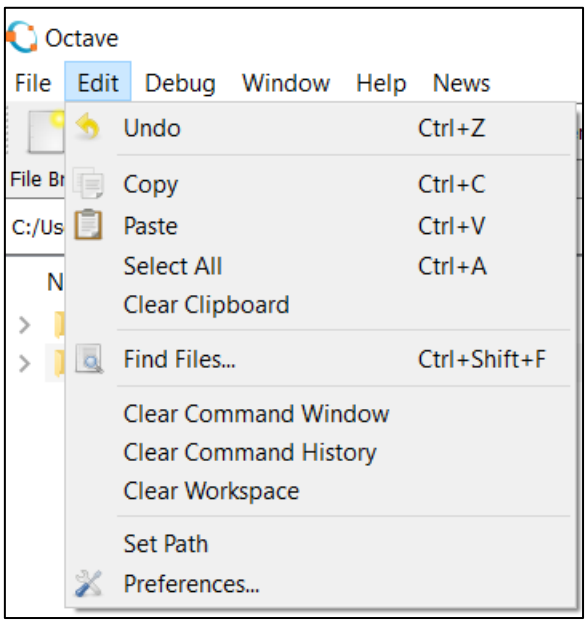


Figura A.9. Meniul **Edit**

Meniul **Edit**: se utilizează pentru operaţii în cadrul editorului de texte și a componentelor interfeței mediului de programare Octave (figura A.9):

Undo : anulează ultima comandă de editare;
Copy : copiază zona de text selectată şi o plasează în memoria tampon;
Paste : inserează în poziţia curentă a cursorului, conţinutul memoriei tampon;
Select All : realizează selectarea întregului conţinut al fişierului curent**;**
Clear Clipboard : șterge conţinutul memoriei tampon;
Find Files... : permite căutarea de fișiere care conțin un anumit text;
Clear Command Window : șterge conținutul ferestrei de comenzi;
Clear Command History : șterge conținutul ferestrei de istoric;
Clear Workspace : șterge conținutul ferestrei cu variabile;
Set Path : permite setarea fișierelor din calea de căutare;
Preferences : permite setarea unor opțiuni legate de interfața aplicației, a editorului de comenzi etc., cum ar fi de exemplu modificarea fontului și a dimensiunii textului;

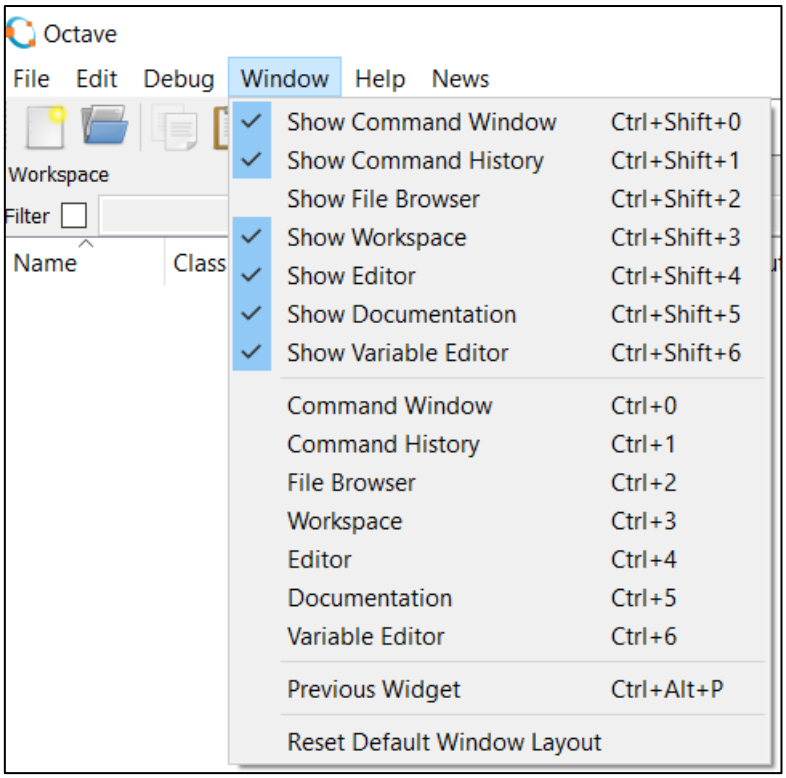


Figura A.10. Meniul **Window**

Meniul **Window**: se utilizează pentru particularizarea interfeței (figura A.10):
Show Command Window : include în interfața mediului de programare fereastra de comenzi;
Show Command History : include în interfața mediului de programare fereastra cu istoricul comenzilor;
Show File Browser : include browser-ul de fișiere în interfața mediului de programare;
Show Workspace : include în interfața mediului de programare fereastra cu variabile;
Show Editor : include în interfața mediului de programare fereastra cu editorul de fișiere;
Show Documentation : include în interfața mediului de programare fereastra cu documentația;
Show Variable Editor : include în interfața mediului de programare editorul de variabile;
Command Window : selectează/face vizibilă (și include dacă nu a fost inclusă anterior) în interfața mediului de programare fereastra cu comenzi;
Command History : selectează/face vizibilă (și include dacă nu a fost inclusă anterior) în interfața mediului de programare fereastra cu istoricul comenzilor;
File Browser : selectează/face vizibil (și include dacă nu a fost inclus anterior) în interfața mediului de programare browser-ul de fișiere;

Workspace : selectează/face vizibilă (și include dacă nu a fost inclusă anterior) în interfața mediului de programare fereastra cu variabile;
Editor : selectează/face vizibil (și include dacă nu a fost inclus anterior) în interfața mediului de programare editorul de fișiere;
Documentation : selectează/face vizibilă (și include dacă nu a fost inclusă anterior) în interfața mediului de programare fereastra cu documentația;
Variable Editor : selectează/face vizibilă (și include dacă nu a fost inclus anterior) în interfața mediului de programare editorul de variabile;
Previous Widget : selectează/face vizibilă (și include dacă nu a fost inclusă anterior) în interfața mediului de programare fereastra folosită/activată anterior;
Reset Default Window Layout : resetează interfața mediului de programare la forma inițială.

Anexa B. Tutorial de instalare al mediului de programare MATLAB

Pentru a realiza instalarea mediului de programare MATLAB pe calculatorul dvs. trebuie să parcurgeţi următoarele etape:

B.1. Crearea contului de student pe Intranet

A. Intrați pe pagina https://intranet.utcluj.ro/login și accesați opțiunea **Student nou** (figura B.1).

Figura B.1 Pagina de Intranet

B. În câmpurile **Creare user student** introduceți datele de identificare care le-ați utilizat pentru aplicația SINU (user name și parola, email-ul alternativ și telefonul mobil exact așa cum apar în SINU) (figura B.2).

Figura B.2 Pagina de *Creare user student*

B.2. Instalarea mediului de programare

A. Intrați pe pagina www.mathworks.com și accesați **Create Account** (Figura B.3).

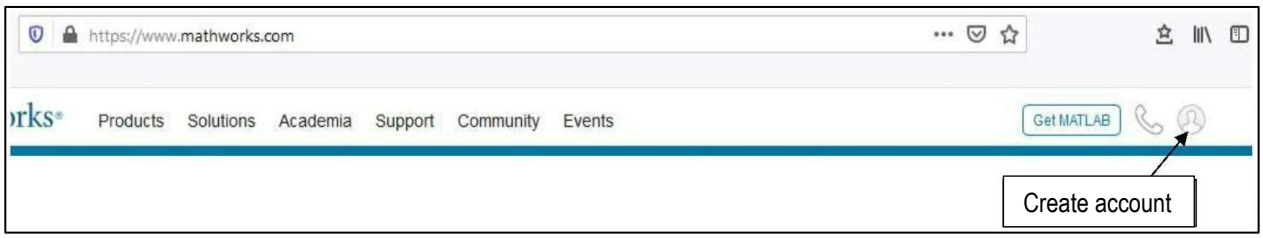


Figura B.3 Butonul pentru deschiderea paginii de *Login/Create account*

C. Apăsați **Create one** (Figura B.4).

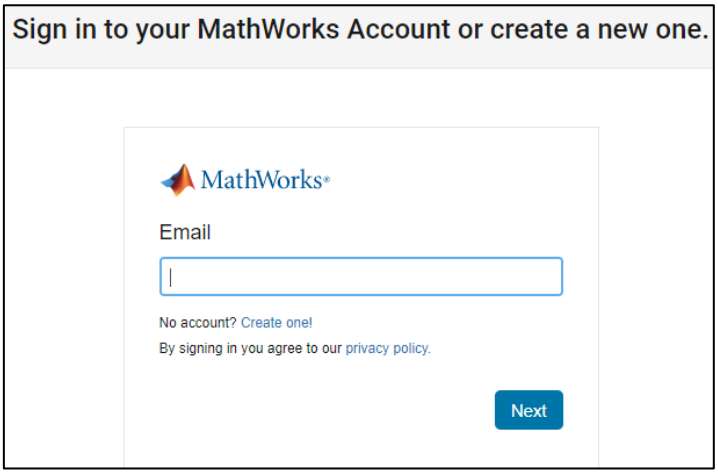


Figura B.4 Pagina de *Login/Create account*

D. Introduceți adresa de email UTCN (de ex. Nume.prenume@student.utcluj.ro), aceasta vă permite să instalați și să folosiți softul gratuit. Completați câmpurile prezentate în Figura B.5 cu informațiile specifice și apoi accesați butonul **Create**.

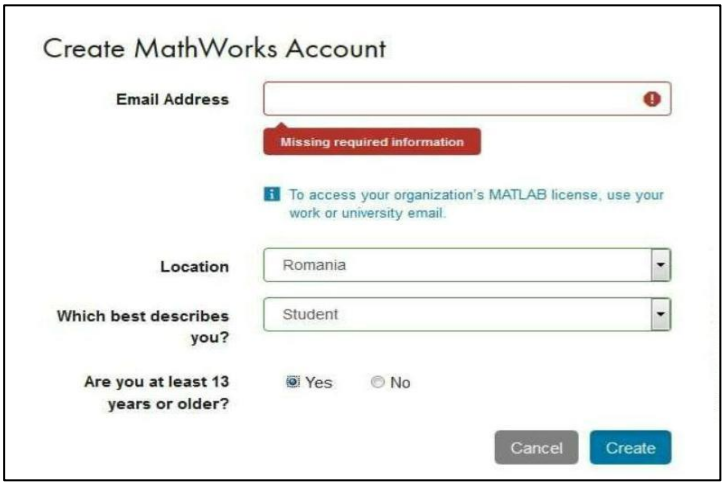


Figura B.5 Pagina de *Create MathWorks account*

E. Verificați email-ul dacă ați primit un link pe care va trebui să îl accesați. În cazul în care nu ați primit, aveți opțiunea *Send me the email again* pentru a retrimite link-ul (Figura B.6).

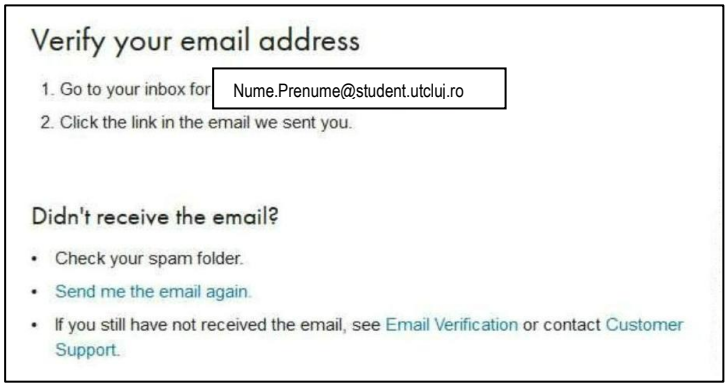


Figura B.6 Pașii pentru verificarea adresei de e-mail

F. Din căsuța de email apăsați **Verify your email**.

Figura B.7 Verificarea adresei de e-mail

G. Pentru a finaliza crearea contului, completați câmpurile cerute (Figura B.8).

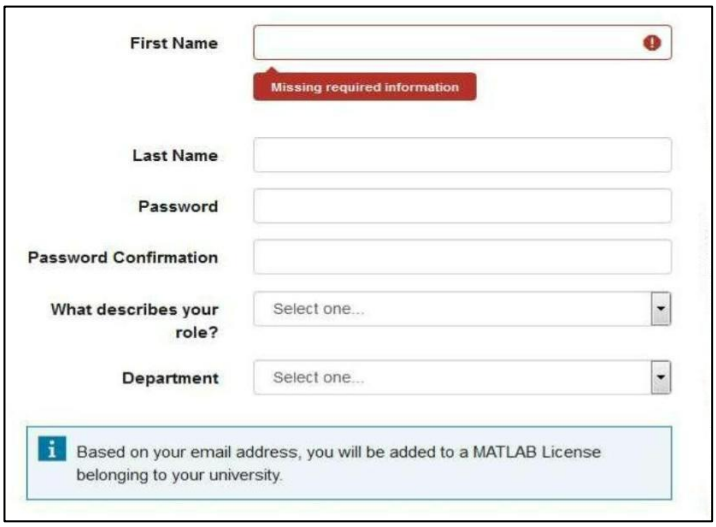


Figura B.8 Completarea profilului

H. Bifați opțiunea *I accept the Online Services Agreement* și apoi butonul **Create** (Figura B.9).

Figura B.9 Finalizarea creării contului

I. Contul a fost creat, aveți posibilitatea să descărcați softul MATLAB, accesând butonul indicat în Figura B.10. În căsuța de email veți primi un mesaj de confirmare a contului creat.

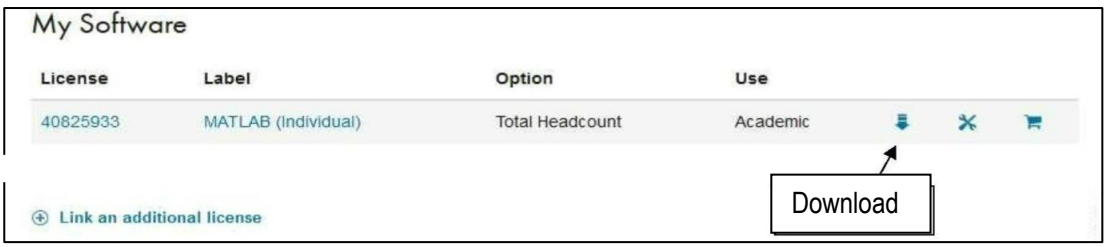


Figura B.10 Accesarea paginii de download a softului MATLAB

J. Vă puteți alege din lista de softuri disponibile, diferite versiuni de MATLAB (Figura B.11). Dacă doriți instalarea unei versiuni începând cu anul 2016 până la versiunea curentă, va trebui să aveți un sistem de operare pe 64 de biți. Versiunea R2015b permite instalarea și pe sistem de operare cu 32 de biți.

Figura B.11 Alegerea versiunii de MATLAB dorite

K. După ce s-a descărcat softul cu versiunea dorită, trebuie să va autentificați din nou folosind user-ul și parola personală pe care le-ați utilizat atunci când v-ați creat contul pe MathWorks, apoi apăsați butonul **Next,** respectiv **Sign In**.

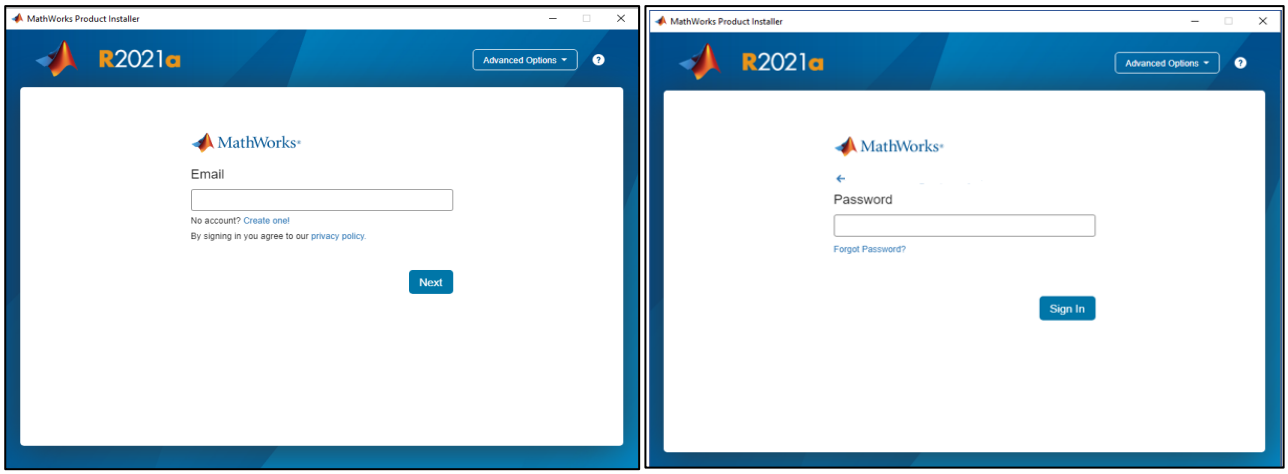


Figura B.12 Autentificare înainte de instalarea softului MATLAB

L. Bifați acceptarea termenilor de licențiere (Figura B.13), apoi apăsați butonul **Next**.

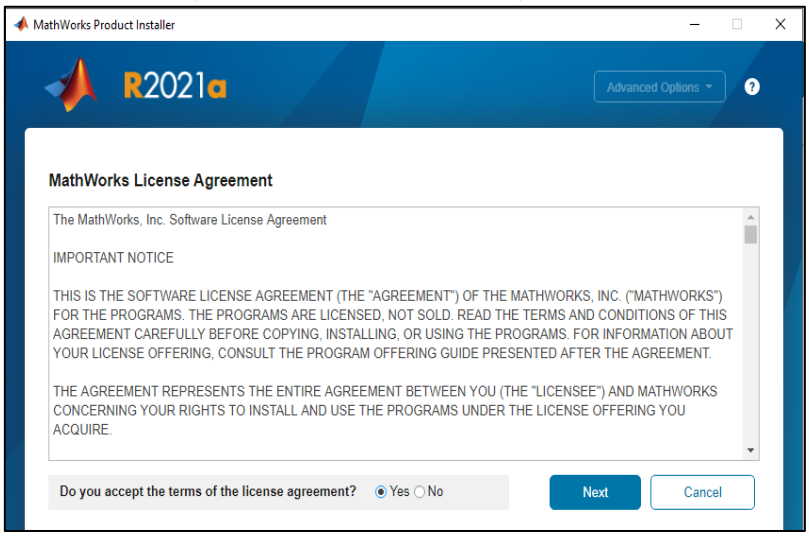


Figura B.13 Acceptarea termenilor de licențiere a softului MATLAB

M. Aveți selectată licența pentru student (Figura B.14), apoi apăsați butonul **Next**.

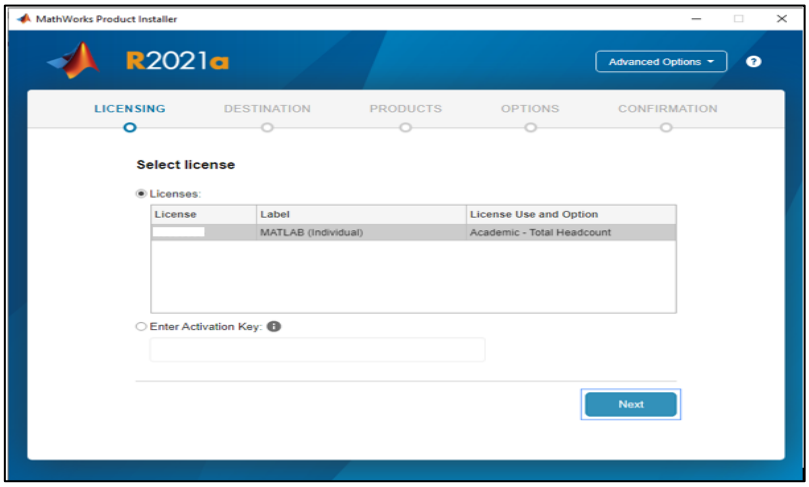


Figura B.14 Logarea în contul de MathWorks pentru a selecta licența

N. Confirmați numele și prenumele precum și adresa de email (Figura B.15), apoi apăsați butonul **Next**.

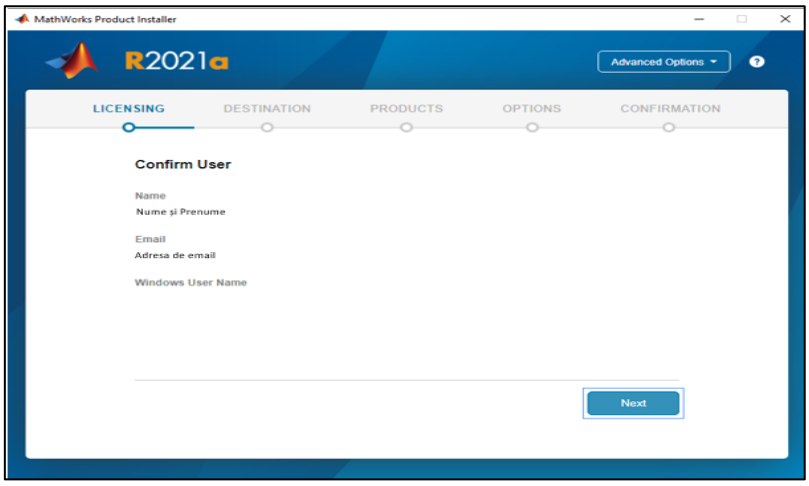


Figura B.15 Confirmarea datelor personale

O. Selectați directorul pentru instalarea soft-ului (Figura B.16), apoi apăsați butonul **Next**.

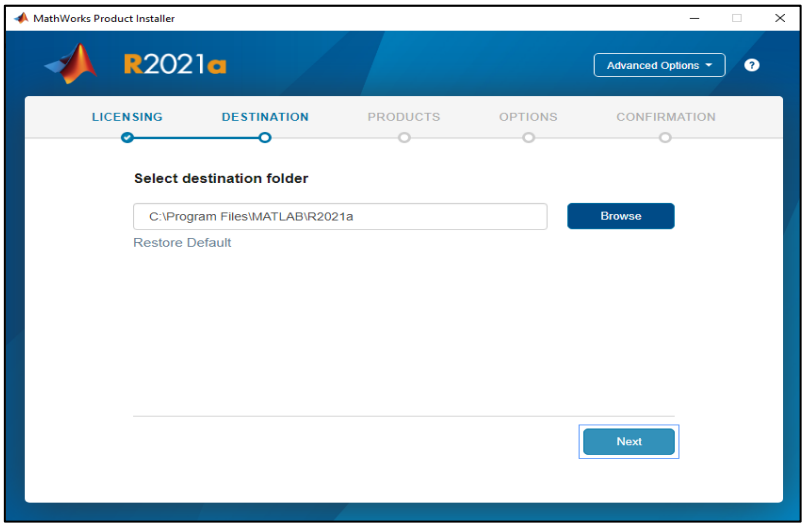


Figura B.16 Selectarea directorului pentru instalarea soft-ului

P. Selectați produsele (Matlab, Simulink, Symbolic Math Toolbox, etc.) pe care doriți să le instalați (Figura B.17), apoi apăsați butonul **Next**.

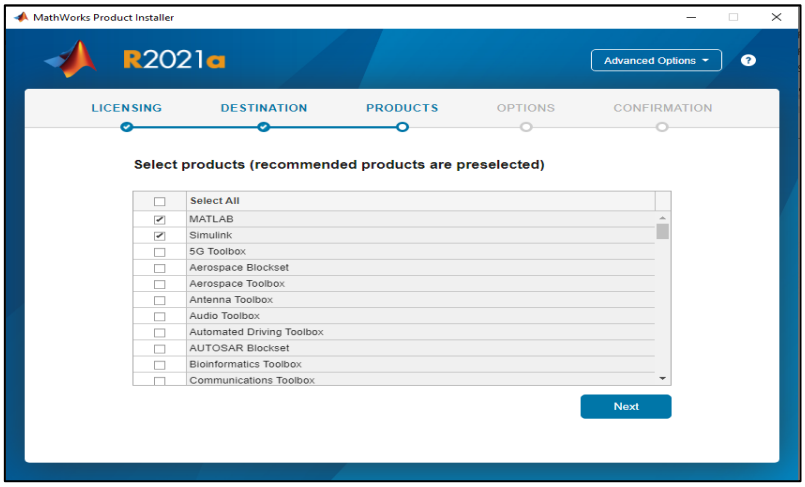


Figura B.17 Selectarea produselor dorite

Q. Selectați opțiunile dorite (Figura B.18), apoi apăsați butonul **Next**.

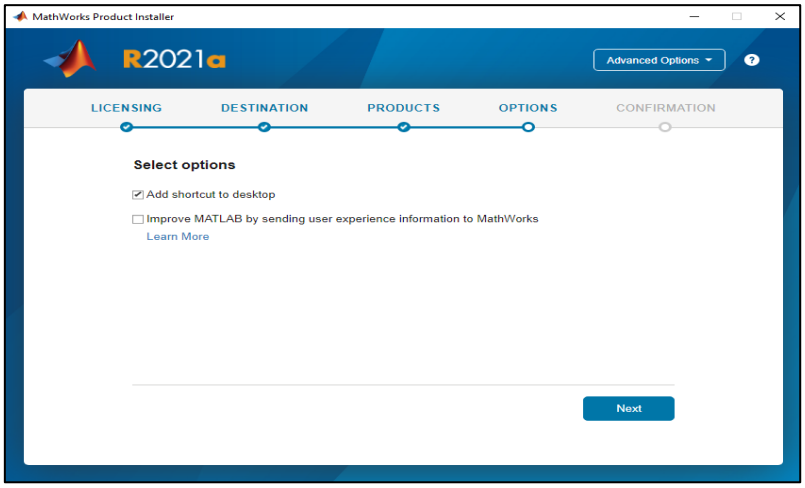


Figura B.18 Selectarea opțiunilor dorite

R. În fereastra următoare sunt prezentate pe scurt informațiile selectate anterior (Figura B.19). Confirmați selecția apăsând butonul **Begin Install**.

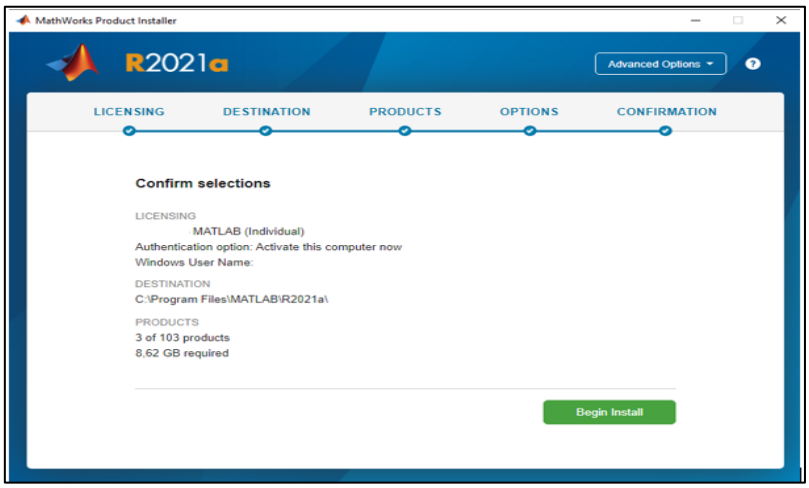


Figura B.19 Confirmarea opțiunilor dorite

S. Instalarea este completă (Figura B.20), apăsați butonul **Close**. În acest moment soft-ul poate fi utilizat prin accesarea scurtăturii de pe *Desktop*.

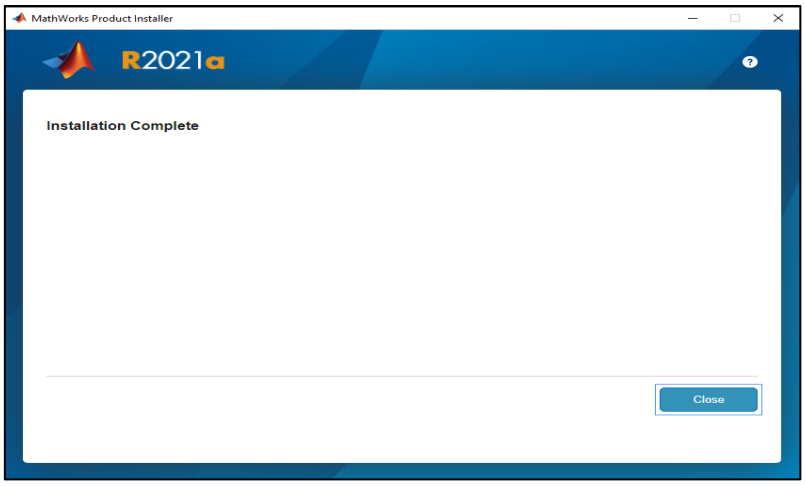


Figura B.20 Instalarea completă a soft-ului