Gabriela Ciuprina

Algoritmi numerici

- prin exerciţii şi implementări în Matlab -

2011-2012

Cuprins

Capitolul 1

Familiarizarea cu mediul de lucru: Matlab

În cadrul laboratorului aferent disciplinei *Algoritmi numerici* se va folosi limbajul de programare ***Matlab*** pentru rezolvarea numerică a unor probleme de circuite electrice şi de câmp electromagnetic.

Această temă are ca scop familiarizarea cu mediul de lucru, în vederea rezolvării temelor ulterioare. Deoarece acest limbaj de programare l-ati mai folosit şi la alte discipline, acestă temă prezinta doar câteva aspecte ale lucrului cu ***Matlab***. Cei pe deplin familiarizaţi cu acest mediu, pot sări peste exerciţiile de început. Cei cu mai puţină experienţă trebuie să citească cu atenţie documentul *Matlab - getting started*.

Lansaţi aplicaţia ***Matlab*** şi parcurgeţi exerciţiile indicate de profesor. Notaţi răspunsurile pe scurt într-un document şi, la sfârşitul şedinţei, trimiteţi documentul generat pe adresa de email a profesorului îndrumător.

1.1 Variabile şi constante.

În rezolvarea temelor veţi folosi în mare măsură matrice cu elemente reale. Un număr poate fi considerat o matrice cu un singur element.

- *Dimensiunea unei matrice* nu trebuie declarată explicit.

Exerciţiul 1.1:
Care este efectul comenzii:

```
>> a(10,5) = 1
```

- *Introducerea unei matrice* se poate face natural astfel:

```
>> A = [  1  2  3
          4  5  6
          7  8  9  ]
```

Într-o scriere compactă, liniile matricei pot fi separate prin ";" astfel:

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

Pentru separarea elementelor unei linii se poate folosi caracterul blank (ca mai sus) sau virgula:

```
>> A = [1,2,3;4,5,6;7,8,9]
```

Exerciţiul 1.2:
Ştiind că vectorii sunt un caz particular de matrice, care sunt comenzile cu care se va introduce un vector linie, respectiv coloană cu elementele 1, 2, 3?

Exerciţiul 1.3:
Ce reprezintă e în următoarea instrucţiune?

```
>> u = 12.4e-3
```

Observaţie: variabilele utilizate într-o sesiune de lucru ocupă memoria sistemului pe măsură ce sunt definite. Pentru a vizualiza lista variabilelor existente la un moment dat şi memoria disponibilă se foloseşte comanda `who`. Pentru a vizualiza câtă memorie ocupă variabilele existente la un moment dat, se foloseşte comanda `whos`. Dacă se doreşte eliberarea memoriei de toate variabilele generate se foloseşte comanda `clear`.

Exerciţiul 1.4:
Executaţi instrucţiunea `clear`. Acum nu mai există nici o variabilă în mediul de lucru (verificaţi cu `who`). Executaţi totuşi, următoarele instrucţiuni. Comentaţi rezultatul lor.

```
>> i
>> j
>> pi
>> eps
```

1.2 Atribuiri şi expresii

Instrucţiunea de atribuire are sintaxa:

```
variabila = expresie
```

sau simplu:

```
expresie
```

în care `variabila` este numele unei variabile, iar `expresie` este un şir de operatori şi operanzi care respectă anumite reguli sintactice. În a doua formă, după evaluarea expresiei, rezultatul este atribuit variabilei predefinite `ans`.

- *Operatorii aritmetici*[1] recunoscuţi de ***Matlab*** sunt:

`+` adunare;
`-` scădere;
`*` înmulţire;
`/` împărţire la dreapta;
`\` împărţire la stânga;
`^` ridicare la putere.

Pentru transpunerea unei matrice se foloseşte operatorul "apostrof" ca în exemplul:

```
>> B = A'
```

în care matricea `B` se calculează ca transpusa matricei `A` dacă aceasta are elemente reale, sau ca transpusa şi conjugata dacă aceasta are elemente complexe.

Observaţii:

1. Adunarea şi scăderea pot fi efectuate:
   - între două matrice cu aceleaşi dimensiuni;
   - între o matrice şi un număr (caz în care numărul este adunat, respectiv scăzut din fiecare din elementele matricei).

2. Înmulţirea poate fi efectuată:
   - între două matrice dacă lungimea liniei primei matrice este egală cu lungimea coloanei celei de a doua matrice;

[1] Operatorii aritmetici se aplică unor operanzi aritmetici, iar rezultatul este aritmetic.

- între un număr şi o matrice (caz în care numărul este înmulţit cu fiecare din elementele matricei);
- între două matrice cu aceleaşi dimensiuni (element cu element), caz în care operatorul * este precedat de un punct, ca în exemplul:

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

3. Împărţirea matricelor poate fi făcută în mai multe feluri:
   - la dreapta (pentru matrice pătrate şi nesingulare):

```
--> X = B / A
```

   echivalent cu:

```
--> X = B * inv(A)
```

   sau cu:

```
--> X = B * A^(-1)
```

   - la stânga:

```
--> X = A \ B
```

   echivalent cu:

```
--> X = inv(A) * B
```

   sau cu:

```
--> X = A^(-1) * B
```

   Dacă A este o matrice dreptunghiulară de dimensiuni $m \times n$, iar b este un vector coloană cu m elemente, atunci împărţirea la stânga $x = A \setminus b$ calculează soluţia ecuaţiei $Ax = b$ în sensul celor mai mici pătrate. - împărţirea unei matrice la un număr (să îl notăm cu `u`):

```
--> Y = A / u
```

   - împărţirea element cu element a matricelor de dimensiuni egale:

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

sau

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

4. Ridicarea la putere `A^p` se face astfel [2]:
   - dacă `p` este un întreg pozitiv: dacă matricea `A` este pătrată atunci `A` se înmulţeşte cu ea însăşi de `p` ori; dacă matricea `A` este dreptunghiulară atunci se ridică la puterea `p` fiecare element din matricea `A`
   - dacă `p` este un întreg negativ: dacă matricea `A` este pătrată atunci inversa ei se înmulţeşte cu ea însăşi de $-p$ ori; dacă matricea `A` este dreptunghiulară atunci se ridică la puterea `p` fiecare element din matricea `A`
   - dacă `p` este un număr real (dar nu întreg) pozitiv: dacă matricea `A` este pătrată atunci calculul se face cu ajutorul vectorilor şi valorilor proprii ale matricei; dacă matricea `A` este dreptunghiulară, atunci se ridică la puterea `p` fiecare element din matricea A.
   - dacă `p` este un număr real (dar nu întreg) negativ: dacă matricea `A` este pătrată atunci calculul se face cu ajutorul vectorilor şi valorilor proprii ale inversei matricei; dacă matricea `A` este dreptunghiulară, atunci se ridică la puterea `p` fiecare element din matricea A.

- *Operatorii de relaţie*[3] recunoscuţi de ***Scilab*** sunt:

`<` mai mic decât;
`<=` mai mic sau egal cu;
`>` mai mare decât;
`>=` mai mare sau egal cu;
`==` egal cu;
`~=` diferit de.

Aceştia permit testarea unor condiţii, rezultatul având valoarea de adevăr fals (0) sau adevărat (1). Dacă operanzii sunt matrice de dimensiuni egale, atunci operaţiile logice se fac între elementele de pe aceleaşi poziţii, iar rezultatul este o matrice cu elementele 0 şi 1.

- *Operatori logici*[4] recunoscuţi de ***Matlab*** sunt:

`&` conjuncţia logică;

[2] Nu sunt descrise toate situaţiile posibile.
[3] Operatorii de relaţie se aplică unor operanzi aritmetici iar rezultatul este logic.
[4] Operatorii logici se aplică unor operanzi logici iar rezultatul este logic.

| disjuncţia logică;
~ negaţia logică.

Dacă operanzii sunt matrice (logice) cu aceleaşi dimensiuni, atunci operaţia se face element cu element. Dacă unul din operanzi este o valoare logică, atunci acesta se combină cu fiecare din elementele celuilalt operand. Alte situaţii nu sunt permise.

• *Funcţii elementare.* Operanzii unor expresii pot fi şi apeluri de funcţii elementare (de exemplu trigonometrice), sau alte funcţii cunoscute. Aceste funcţii aplicate unei matrice acţionează asupra fiecărui element în mod independent.

Exerciţiul 1.5:
Fie $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$ şi $v = \begin{bmatrix} 4 & 3 \end{bmatrix}$. Care sunt comenzile ***Matlab*** pentru rezolvarea ecuaţiei $A\left(v^T + x\right) = b$.

1.3 Generarea vectorilor şi matricelor

• *Vectorii ai căror elemente formează o progresie aritmetică* pot fi generaţi cu construcţia:

valin : pas : valfin

Exerciţiul 1.6:
Comentaţi rezultatele următoarelor instrucţiuni:

```
>> x = 1:10
>> y = 1:2:10
>> z = 1:3:10
>> t = 1:-3:10
>> w = -1:-3:-10
>> u = -1:-10
>> v = -10:-1
>> a = 10:-2:-3
```

• *Vectorii ai căror elemente formează o progresie geometrică* pot fi generaţi cu construcţia:

```
logspace(d1, d2, n)
```

Exerciţiul 1.7:
a) Care este semnificaţia mărimilor $d1, d2, n$ din comanda `logspace` ?

b) Ce generează comanda `linspace` ?

• *Descrierea vectorilor şi matricelor pe blocuri.* Vectorii şi matricele pot fi descrise pe blocuri, folosind notaţii de forma:

```
A = [X Y; U V];
```

cu semnificaţia $A = \begin{bmatrix} X & Y \\ U & V \end{bmatrix}$, în care X, Y, U, V sunt matrice sau vectori.

Exerciţiul 1.8:

Care este rezultatul comenzii:

```
>> A = [1:3 ; 1:2:7]
```

• *Referirea la elementele unei matrice.* Pentru a obţine valoarea unui element, se folosesc construcţii de forma $a(1,1), a(1,2)$. Se pot obţine valorile mai multor elemente prin construcţii de forma $a(u,v)$ unde u şi v sunt vectori. De exemplu a(2,1:3) reprezintă primele trei elemente din linia a doua a matricei a. Pentru a obţine toate elementele liniei 2 se scrie $a(2,:)$.

Exerciţiul 1.9:

Fie $A = \begin{bmatrix} 1 & 10 & 100 & 1000 \\ 2 & 20 & 200 & 2000 \\ 3 & 30 & 300 & 3000 \end{bmatrix}$.

Notaţi rezultatele şi comentaţi următoarele comenzi:

```
>> A(0,1)
>> A(2,3)
>> A(:,3)
>> A(:,:)
>> A(3,:)
>> A(2,2:4)
>> A(2:3,2:4)
```

• *Generarea unor matrice particulare utile* se poate face cu ajutorul funcţiilor:

`eye` matrice cu elementele unitare pe diagonală şi nule în rest;

`zeros` matrice nulă;

`ones` matrice cu toate elementele unitare;

`rand` matrice cu elemente aleatoare în intervalul (0,1);

`diag` construieşte o matrice cu o anumită diagonală, sau extrage diagonala dintr-o matrice.

Exerciţiul 1.10:

Comentaţi următoarele comenzi (unde `A` este matricea de la exerciţiul 9 iar $v = [\ 1 \quad 2 \quad 3 \quad 4 \quad 5\ ]$):

```
>> eye(3)
>> eye(3,3)
>> eye(3,4)
>> diag(A)
>> diag(v)
```

Exerciţiul 1.11:

Comentaţi următoarele comenzi:

```
>> A = diag(1:3)
>> B = [A, eye(size(A)); ones(size(A)) zeros(size(A))]
>> C = diag(B)
>> D = C'*C
>> E = (D == 14)
```

• *Dimensiunile matricelor (vectorilor)* pot fi modificate în timpul execuţiei unui program. Pentru a obţine dimensiunea unei matrice X se foloseşte instrucţiunea:

```
[m, n] = size(X)
```

în care m reprezintă numărul de linii şi n numărul de coloane. Dimensiunea unui vector v se obţine cu:

```
lenght(v)
```

care are semnificaţia `max(size(v))`.

Exerciţiul 1.12:

Definiţi o matrice oarecare B (de exemplu cu 3 linii şi 4 coloane). Executaţi şi comentaţi următoarele instrucţiuni:

```
>> [m,n] = size(B)
>> B = [B; zeros(1, n)]
>> B = [B zeros(m+1,1)]
```

• *Matricea vidă.* ***Matlab*** operează şi cu conceptul de matrice vidă, notată cu [] şi care este o matrice de dimensiune nulă, fără elemente. Aceasta se dovedeşte utilă în eliminarea unor linii sau coloane dintr-o matrice dată. De exemplu, instrucţiunea

```
>> B(:, [2 4]) = []
```

are ca efect eliminarea coloanelor 2 şi 4 din matricea B. În acest fel, dimensiunea unei matrice poate să şi scadă în timpul execuţiei unui program, nu numai să crească prin adăugarea de noi elemente.

1.4 Instrucţiuni grafice

Funcţia principală pentru reprezentări grafice este:

`plot`

Ea are diferite variante, printre care:

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

în care x este vectorul variabilei independente, iar y este vectorul variabilei dependente. Instrucţiunea:

```
plot(A)
```

în care A este o matrice are ca efect reprezentarea grafică a variaţiei elementelor coloanelor matricei A în funcţie de indexul lor. Numărul de grafice este egal cu numărul de coloane.

Funcţiile auxiliare ca `title` şi `grid` permit completarea graficului cu un titlu şi respectiv adăugarea unui rastru. Completarea graficului poate fi făcută şi cu ajutorul interfeţei grafice, apăsând *Show plot tools*.

Exerciţiul 1.13:
Realizaţi graficul din figura 1.1. El reprezintă funcţia $y(t) = 10 * \sin(t)$ pentru $t \in [0, 4\pi]$. Puneţi etichete axelor, rastrul şi legenda ca în figură.

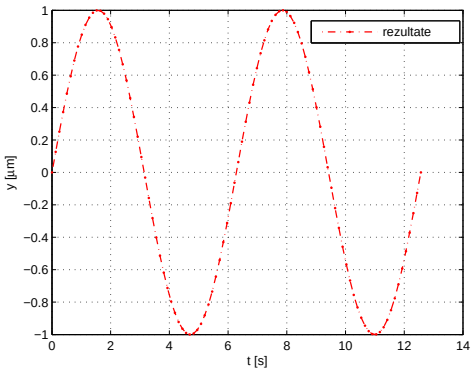

Figura 1.1: Acest grafic trebuie obţinut la exerciţiul 13.

1.5 Programare în *Matlab*

Matlab permite utilizarea unor structuri de control (decizii, cicluri, definiri de funcţii) ca orice limbaj de programare de nivel înalt. Se pot scrie programe în ***Matlab***, ca în orice limbaj de programare.

Avantajul folosirii ***Matlab*** constă, mai ales, în uşurinţa cu care se pot postprocesa rezultatele, ţinând seama de facilităţile grafice ale sale. Un alt avantaj deosebit de important este acela că se pot efectua operaţii cu vectori şi matrici, operaţii care sunt deosebit de eficiente mai ales pentru vectori şi matrici rare. În acest fel, ***Matlab*** este un mediu foarte potrivit pentru testarea unor noi algoritmi.

1.5.1 Editarea programelor

Până acum, aţi lucrat la consola ***Matlab***. Comenzile introduse pot fi scrise într-un fişier (numit script)şi apoi executate fie prin tastarea numelui fişierului script în consolă, fie prin apăsarea butonului *Run*.

Un script în ***Matlab*** are următoarea structură posibilă:

```
% comentarii
.....
instrucţiune;   % fară afişarea rezultatului
instrucţiune    % cu afişarea rezultatului
```

Exerciţiul 1.14:
Scrieţi comenzile cu care aţi rezolvat exerciţiul 9, într-un fişier numit `mytest.m`, din directorul ~/AlgoritmiNumerici/Tema1 şi apoi executaţi-l fie cu comanda:

```
>> mytest
```

fie apăsând butonul *Run* din editor. Observaţi ce se întâmplă dacă la sfârşitul fiecărei instrucţiuni adăugaţi terminatorul ";".

IMPORTANT: Comenzile necesare rezolvării temelor ce vor urma vor fi scrise în fişiere.

1.5.2 Operaţii de intrare/ieşire

- *Introducerea datelor.*

Cea mai simplă metodă constă în utilizarea instrucţiunii de atribuire, ca în exemplul:

$a = 5$

În cazul unui program scris într-un fişier, este mai convenabil să se folosească funcţia `input`. Funcţia input se utilizează în atribuiri de forma:

variabila = `input`('text')

în care "variabila" este numele variabilei a cărei valoare va fi citită de la consolă, iar "text" este un şir de caractere ce va fi afişat, ajutând utilizatorul la identificarea informaţiei ce trebuie introdusă. De exemplu:

```
a = input('Introduceti valoarea variabilei a. a = ');
```

• *Inspectarea şi afişarea rezultatelor*

Pentru inspectarea valorilor variabilelor este suficient să fie invocat numele lor:

```
>> a
```

pentru ca interpretorul să afişeze valoarea lor.

Dacă se doreşte eliminarea afişării numelui variabilelor din faţa valorii sale, atunci se foloseşte funcţia `disp`:

```
>> disp(a)
```

Această funcţie poate fi folosită şi pentru afişarea textelor, de exemplu:

```
disp('Acest program calculeaza ceva ');
```

Formatul în care sunt afişate valorile numerice poate fi modificat de utilizator cu ajutorul instrucţiunii:

```
format
```

Operaţia de ieşire se poate realiza şi prin apelul funcţiei `sprintf` în instrucţiuni de forma:

`sprintf`('format',variabile)

în care "variabile" sunt variabilele care vor fi scrise în formatul corespunzător instrucţiunii, iar "format" este un şir de caractere ce descrie formatul de afişare. Sunt recunoscute următoarele construcţii, similare celor din limbajul C:

`%f` scrierea numărului în format cu virgulă fixă;
`%e` scrierea numărului în format cu exponent;
`%g` scrierea numărului în formatul cel mai potrivit (%f sau %e).

Celelalte caractere întâlnite în şirul "format" sunt afişate ca atare, de exemplu:

```
sprintf(' Rezultatul este a = %g', a);
```

Afişarea rezultatelor se poate face şi grafic (vezi paragraful 1.4).

Exerciţiul 1.15:
Scrieţi, într-un fişier, un program prietenos care va permite introducerea de la consola ***Matlab*** a două numere reale, va calcula suma lor, şi va afişa rezultatul în formatul cu exponent.

1.5.3 Structuri de control

- *Decizii*

Decizia simplă:

Pseudolimbaj	***Matlab***	Observaţii
dacă condiţie **atunci** instrucţiuni	`if` condiţie instrucţiuni `end`	"condiţie" este o expresie care este evaluată, iar dacă rezultatul este adevărat (T), atunci se execută "instrucţiuni", altfel se execută prima instrucţiune ce urmează după `end`.

Decizia cu alternativă:

Pseudolimbaj	***Matlab***	Observaţii

dacă condiţie **atunci**
instrucţiuni1
altfel
instrucţiuni2
	`if` condiţie
	instrucţiuni1
	`else`
	instrucţiuni2
	`end`
		"condiţie" este o expresie care este evaluată, iar dacă rezultatul este adevărat (1), atunci se execută "instrucţiuni1", iar dacă rezultatul este fals (0), se execută "instrucţiuni2".

Decizia de tip selecţie:

Pseudolimbaj	***Matlab***	Observaţii

dacă condiţie1 **atunci**
instrucţiuni1
altfel dacă condiţie2
instrucţiuni2
altfel
instrucţiuni3
	`if` condiţie1
	instrucţiuni1
	`elseif` condiţie2
	instrucţiuni2
	`else`
	instrucţiuni3
	`end`
		Pot exista oricâte alternative de selecţie.

	`switch` expresie, `case` expresie1 instrucţiuni1, `case` expresie2 instrucţiuni2, `otherwise` instrucţiuni, `end`	Pot exista oricâte cazuri. "instrucţiuni1" sunt executate dacă expresie == expresie1, etc.

- *Cicluri*

Ciclul cu test iniţial:

Pseudolimbaj	***Matlab***	Observaţii
cât timp condiţie instrucţiuni	`while` condiţie instrucţiuni `end`	Se repetă corpul ciclului, adică "instrucţiuni", cât timp "condiţie" este adevărată. S-ar putea ca "instrucţiuni" să nu fie executate niciodată în timpul rulării programului.

Ciclul cu contor:

Ciclul cu contor are două forme, din care a doua este cea generală. Dacă "expresie" este o matrice, atunci "variabila" ia succesiv valorile coloanelor matricei. "instrucţiuni" nu sunt executate niciodată dacă vectorul "valin:pas:valfin" este incorect definit (vid) sau dacă "expresie" este matricea vidă.

Pseudolimbaj	***Matlab***	Observaţii
pentru contor = valin, valfin, pas instrucţiuni	`for` contor = valin : pas : valfin, instrucţiuni `end`	Forma a doua este cea generală.
	`for` variabila = expresie, instrucţiuni `end`	

Ieşirile forţate din cicluri se pot face cu instrucţiunea `break`.

Exerciţiul 1.16:

Scrieţi un program care să determine cel mai mare număr întreg n pentru care 10^n poate fi reprezentat în ***Matlab***. Indicaţie: folosiţi un ciclu cu test, în care condiţia de intrare în ciclu testează egalitatea dintre 10^n şi constanta `Inf`.

1.5.4 Funcţii

Funcţiile sunt rutine ***Matlab*** care accepta parametri de intrare şi întorc parametri de ieşire.

Este bine ca fiecare funcţie să fie definită într-un fişier separat, care are acelaşi nume ca numele funcţiei şi extensia *.m. Un fişier conţinând o funcţie trebuie să înceapă astfel:

`function`[$y_1, \ldots, y_n$] = nume_functie ($x_1, \ldots, x_m$)

unde y_i sunt variabilele de ieşire, calculate în funcţie de variabilele de intrare x_j şi, eventual, de alte variabile existente în ***Matlab*** în momentul execuţiei funcţiei. Se recomandă ca acest fişier să se numeasca `nume_functie.m`.

Exerciţiul 1.17:
Editaţi un fişier numit "combin.m" cu următorul conţinut:

```
function [x,y] = combin(a,b)
x = a + b;
y = a - b;
```

şi un fişier numit "main_combin.m" cu următorul conţinut:

```
clear all;
a = input('Introduceti a. a = ');
b = input('Introduceti b. b = ');
[c,d] = combin(a,b);
disp(sprintf(' Suma numerelor  a = %g si b = %g este a + b = %g',a,b,c));
disp(sprintf(' Diferenta numerelor a = %g si b = %g este a - b  = %g',a,b,d));
```

Executaţi în ***Matlab*** comenzile din "main_combin.m":

```
>> main_combin;
```

a) Explicaţi comanda `disp(sprintf(....));`
b) Comentaţi necesitatea instrucţiunii `clear all`.

c) Rulaţi programul pas cu pas şi urmăriţi fereastra *Workspace*
d) Adaugaţi după citirea valorii b, instrucţiunea inutilă $z = 7$. Rulaţi programul pas cu pas şi urmăriţi fereastra *Workspace*. Comentaţi.
e) Pe exemplul de la punctul c), daţi în consola Matlab comanda

```
>> mlint main_combin
```

Comentaţi rezultatul ei, după ce vă informaţi asupra comenzii `mlint`.

IMPORTANT: Folosirea comenzii `mlint` trebuie să fie o practică obişnuită în cazul folosirii limbajului Matlab.

1.6 Implementări eficiente ale operaţiilor cu matrici

Unul din avantajele lucrului în Matlab este acela că el permite implementarea operaţiilor cu matrice. Aceasta nu numai că simplifică scrierea programelor, dar conduce la implementări mai eficiente deoarece intern, Matlab, are proceduri optimizate pentru aceste operaţii. Următoarele exerciţii ilustrează acest lucru.

Exerciţiul 1.18:
Scrieţi o funcţie `ps_v1` care să calculeze produsul scalar a doi vectori a şi b de dimensiune $n \times 1$ prin implementarea formulei $\sum_{i=1}^{n}$ şi o altă funcţie `ps_v2` care să implementeze calculul produsului scalar folosind operaţiile cu matrice $a * b'$. Verificaţi corectitudinea funcţiile scrise cu ajutorul unui script în care să comparaţi rezultatele.

Exerciţiul 1.19:
Comentaţi conţinutul scriptului de mai jos. Executaţi-l şi comparaţi rezultatul cu cel din figura 1.2.

```
clear all;
nn = linspace(1e6,1e7,10);
N = length(nn);
t1 = zeros(1,N);
t2 = zeros(1,N);

for i = 1:length(nn);
    n = floor(nn(i));
    a = rand(1,n);
    b = rand(1,n);
```

```
    tic;
    rez1 = ps_v1(n,a,b);
    t1(i) = toc;
    tic;
    rez2 = ps_v2(a,b);
    t2(i) = toc;
end

plot(nn,t1,'bo-');
hold on;
plot(nn,t2,'r*-');
leg{1} = 'implementare cu for';
leg{2} = 'implementare a*b^\prime';
legend(leg);
xlabel('n');
ylabel('t [s]');
title('Timp de calcul al produsului scalar');
```

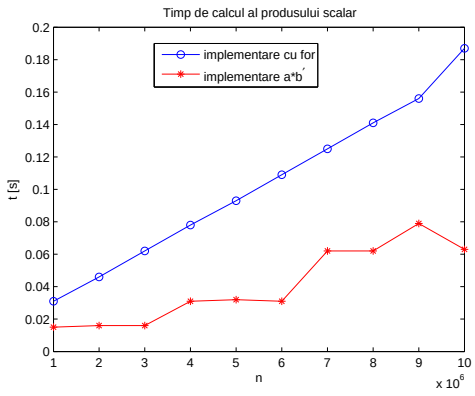

Figura 1.2: Timpul de calcul al produsului vectorial în funcţie de dimensiunea vectorilor. Comparaţie între cele două implementări propuse la exerciţiul 19.

În concluzie, pentru a scrie programe eficiente, în Matlab trebuie folosit calculul matriceal ori de câte ori este posibil.

Exerciţiul 1.20:

Scrieţi o funcţie `pmv_v1` care să calculeze produsul dintre o matrice pătrată a de dimensiune nşi un vector coloană b prin implementarea formulei $c_i = \sum_{j=1}^{n} a_{ij} b_j$ şi o altă funcţie `pmv_v2` care să implementeze calculul aceluiaşi produs folosind operaţiile cu matrice $c = a * b$. Comparaţi eficienţa celor două implementări. Comparaţi rezultatele

obţinute cu graficele din figura 1.3.

Exerciţiul 1.21:

Scrieţi o funcţie `pmm_v1` care să calculeze produsul dintre două matrice pătrate a şi b de dimensiune $n$rin implementarea formulei $c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$ şi o altă funcţie `pmm_v2` care să implementeze calculul aceluiaşi produs folosind operaţiile cu matrice $c = a * b$. Comparaţi eficienţa celor două implementări. Comparaţi rezultatele obţinute cu graficele din figura 1.4.

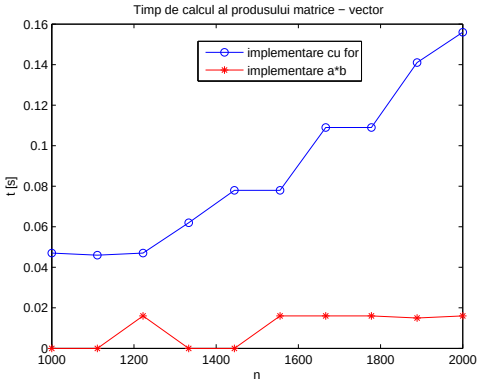

Figura 1.3: Timpul de calcul al produsului dintre o matrice pătrată şi un vector în funcţie de dimensiunea problemei. Comparaţie între cele două implementări propuse la exerciţiul 20.

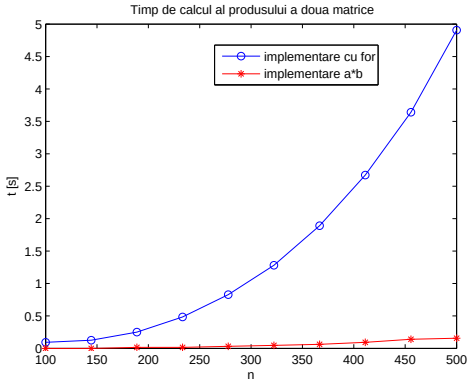

Figura 1.4: Timpul de calcul al produsului dintre două matrice pătrate în funcţie de dimensiunea problemei. Comparaţie între cele două implementări propuse la exerciţiul 21.

1.7 Facilităţi de post-procesare

Acest exerciţiu urmăreşte exersarea facilităţilor de postprocesare ale programului, necesare temelor viitoare. Exerciţiul propus se referă la reprezentarea grafică a câmpului electric produs de o sarcină punctiformă plasată în vid.

Exerciţiul 1.22:

Fie o sarcină punctiformă $q = 10^{-10}$ C, situată în vid ($\varepsilon_0 = 8.8 \cdot 10^{-12}$ F/m), în punctul de coordonate $(x_0, y_0, z_0) = (0, 0, 0)$.

a) Să se reprezinte grafic liniile echipotenţiale în planul $z = 0$;

b) Să se reprezinte grafic potenţialul $V(x, y, 0)$;

c) Să se reprezinte grafic potenţialul $V(x, 0, 0)$;

c) Să se reprezinte grafic vectorul câmp electric în planul $z = 0$;

d) Să se reprezinte grafic componenta după x a câmpului electric $E_x(x, 0, 0)$.

Reamintim că o sarcină q plasată în punctul având vectorul de poziţie

$$\mathbf{r}_0 = x_0\mathbf{i} + y_0\mathbf{j} + z_0\mathbf{k},$$

într-un mediu omogen de permitivitate ε produce un câmp electric

$$\mathbf{E}(\mathbf{r}) = \frac{q}{4\pi\varepsilon}\frac{\mathbf{R}}{R^3},$$

unde $\mathbf{r} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ este vectorul de poziţie în punctul in care se calculează câmpul iar $\mathbf{R} = \mathbf{r} - \mathbf{r}_0 = (x - x_0)\mathbf{i} + (y - y_0)\mathbf{j} + (z - z_0)\mathbf{k}$ este un vector ce uneşte punctul în care se afla sarcină cu punctul în care se află câmpul. Modulul acestui vector este

$$R = \|\mathbf{R}\| = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}.$$

Acest câmp poate fi descris şi cu ajutorul potenţialului electric V, unde $\mathbf{E} = -\text{grad}V$. În acest caz simplu, V este dat de expresia

$$V(\mathbf{r}) = \frac{q}{4\pi\varepsilon R}.$$

Linii echipotenţiale şi vectorii se pot trasa cu ajutorul comenzilor `contour` şi `quiver`. Pentru aceasta, se vor determina valorilor potenţialului şi ale componentelor câmpului într-o mulţime discretă de puncte din spaţiu, plasate în nodurile unui grid generat de un vector de abscise şi un vector de ordonate.

Pentru rezolvarea problemei se va scrie o funcţie care calculează, într-un punct oarecare, potenţialul şi câmpul unei sarcini punctuale situată într-un punct oarecare din spaţiu, de exemplu de următoarea formă.

```
function [V,E] = camp_sarcina(q,epsilon,x0,y0,z0,x,y,z)
% Calculeaza campul unei sarcini in vid
% Date de intrare
%       q - valoarea sarcinii
%       epsilon - permitivitatea absoluta a mediului
%       x0, y0, z0 - coordonatele punctului unde se afla sarcina
%       x, y, z - coordonatele punctului unde se calculeaza campul
% Date de iesire
%       V - potentialul
%       E - vectorul camp electric (vector cu 3 componente)

......
```

Se va scrie un program principal care să apeleze funcţia definită mai sus. Un exemplu este următorul.

```
clear all;
q = 1e10;
epsilon = 8.8e-12;
x0 = 0;
y0 = 0;
z0 = 0;

xmin = -0.5;
xmax = 0.5;
ymin = -0.5;
ymax = 0.5;
nx = 4;
ny = 4;
x = linspace(xmin,xmax,nx);
y = linspace(ymin,ymax,ny);

for i = 1:nx
    for j = 1:ny
        xi = x(i);
        yj = y(j);
        [v,e] = camp_sarcina(q,epsilon,x0,y0,z0,xi,yj,0);
        V(i,j) = v;
        Ex(i,j) = e(1);
        Ey(i,j) = e(2);
    end
end
[X,Y] = meshgrid(x,y);
X = X'; Y = Y';
figure(1);
contour(X,Y,V);
hold on;
quiver(X,Y,Ex,Ey);
```

Exerciţiul 1.23:

a) Observaţi şi comentaţi alura echipotenţialelor pentru diferite grade de fineţe ale grilei

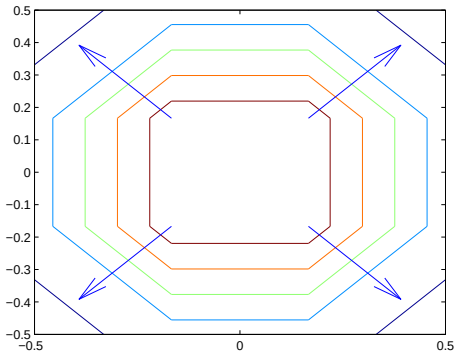

Figura 1.5: Efectul comenzilor `contour` şi `quiver`. Curbele au această formă datorită gridului extrem de rar folosit.

folosite.

b) Cum trataţi cazul în care unul din punctele gridului coincide cu punctul în care se află sarcina?

c) Propuneţi un algoritm care să genereze un grid adaptat problemei, astfel încât liniile echipotenţiale să aibă racordări cât mai "dulci".

1.8 Referinţe utile

- Documentaţia Matlab - disponibilă online la http://www.mathworks.com/access/helpdesk/help/helpdesk.html
- Clever Moler - Numerical Computing with Matlab, SIAM, 2004, disponibilă online la http://www.mathworks.com/moler/
- Pascal Getreuer - Writing fast Matlab code, 2009 `http://www.math.ucla.edu/~getreuer/matopt.pdf`

Capitolul 2

Evaluarea algoritmilor

În general, o problemă admite mai multe metode de rezolvare, şi în consecinţă, vor exista mai mulţi algoritmi diferiţi pentru rezolvarea ei cu ajutorul calculatorului. De aceea, este absolut necesar să se facă o evaluare a algoritmilor pentru a stabili care dintre ei este cel mai bun pentru o problemă dată.

Nu există un singur criteriu de evaluare a algoritmilor. Algoritmul ideal trebuie să fie simplu, să dea o soluţie corectă într-un timp scurt şi să ocupe o zonă mică de memorie. Nu există însă nici un algoritm care să rezolve perfect o problemă, fără nici un fel de eroare şi atunci vom urmări să avem erori rezonabile pentru o anumită aplicaţie. O analiză a erorilor posibile dintr-un algoritm este prezentată în paragraful următor. De asemenea, criteriul referitor la timpul de calcul intră de multe ori în contradicţie cu criteriul referitor la memoria necesară algoritmului.

În consecinţă, la alegerea unui algoritm potrivit pentru o aplicaţie dată, trebuie făcut un compromis între trei criterii: timpul de calcul necesar obţinerii soluţiei, necesarul de memorie şi acurateţea soluţiei. Exerciţiile propuse în această temă ilustrează aceste trei criterii.

2.1 Timpul de calcul

Timpul de calcul al unui algoritm depinde de complexitatea problemei de rezolvat, de performanţele intrinseci ale calculatorului şi limbajului de programare folosit şi evident, de algoritm. Este util însă să poată fi apreciată doar calitatea algoritmului şi nu a problemei sau a mediului în care se lucrează. De aceea s-a inventat conceptul de complexitate a unui algoritm din punct de vedere al timpului de calcul.

Complexitatea unui algoritm din punct de vedere al timpului de calcul este relaţia dintre timpul de calcul exprimat în număr de operaţii elementare şi dimensiunea problemei. Dimensiunea problemei depinde de problema studiată. De exemplu, pentru calculul produsului scalar a doi vectori, dimensiunea problemei este dimensiunea vectorilor. Pentru rezolvarea unui sistem de ecuaţii algebrice liniare, dimensiunea problemei este dimensiunea sistemului. Uneori, este potrivit să exprimăm dimensiunea problemei în funcţie de două numere în loc de unul. De exemplu, dacă datele de intrare reprezintă graful unui circuit, dimensiunea problemei este reprezentată de perechea alcătuită din numărul de noduri şi numărul de laturi.

Timpul de calcul este de fapt suma timpilor necesari pentru executarea tuturor instrucţiunilor algoritmului. Nu toate instrucţiunile durează la fel de mult, de aceea, estimarea complexităţii nu se poate face foarte precis, dar nici nu este necesar acest lucru. Se alege o *operaţie elementară*, considerată a fi cea care durează cel mai mult (de exemplu evaluarea unei anumite funcţii) şi se numără câte astfel de operaţii elementare sunt executate.

Să considerăm următorul fragment de pseudocod, corespunzător calculului unui produs scalar $p = \sum_{i=1}^{n} a_i b_i$.

$p = 0$
pentru $i = 1, n$
    $p = p + a_i b_i$
•

Considerând ca operaţie elementară orice operaţie algebrică (adunare, scădere, înmulţire, împărţire) şi neglijând timpul de calcul petrecut în declaraţii şi atribuiri, rezultă că în algoritmul de mai sus se fac $2n$ operaţii elementare, câte două (o adunare şi o înmulţire) pentru fiecare valoare a contorului i. Timpul de calcul este proporţional cu dimensiunea problemei, în acest caz n. Un astfel de algoritm se spune că este un *algoritm liniar, sau de ordinul 1* şi se notează $T = \mathrm{O}(n)$. În cazul produsului scalar a doi vectori de dimensiune n, putem scrie $T = \mathrm{O}(2n) \approx \mathrm{O}(n)$. Constanta 2 nu este atât de relevantă, ceea ce este important este dependenţa de dimensiunea problemei.

Fie acum cazul înmulţirii unei matrice pătrate a de dimensiune n cu un vector coloană x. Rezultatul este un vector coloană ale cărui componente se calculează ca $b_i = \sum_{j=1}^{n} a_{ij} x_j$.

pentru $i = 1, n$
    $b_i = 0$
    pentru $j = 1, n$
        $b_i = b_i + a_{ij} x_j$
    •
•

În acest caz, pentru fiecare i se fac $2n$ operaţii (judecând exact ca la produsul scalar), deci pentru toate cele n valori ale lui i se vor face $2n^2$ operaţii elementare. Un algoritm pentru care timpul de calcul T este proporţional cu pătratul dimensiunii problemei n se spune că este un *algoritm pătratic, sau de ordinul 2* şi se notează $T = \mathrm{O}(n^2)$. Înmulţirea dintre o matrice şi un vector are deci complexitatea $T = \mathrm{O}(2n^2) \approx \mathrm{O}(n^2)$.

Exerciţiul 2.1:
Scrieţi pseudocodul pentru înmulţirea a două matrice pătrate de dimensiune n şi evaluaţi ordinul de complexitate din punct de vedere a timpului de calcul.

Exerciţiul 2.2:
Fie pseudocodul

```
procedură gaxpy(m, n, a, x, y, b)
; calculează b = ax + y
; date de intrare
întreg m, n              ; dimensiunile problemei
tablou real a[m][n]; matrice dreptunghiulara cu m linii şi n coloane
tablou real x[n]
tablou real y[m]
; rezultat
tablou real b[m]
; alte declaraţii
...
pentru i = 1, m
      b_i = y_i
      pentru j = 1, n
            b_i = b_i + a_ij x_j
      •
•
retur
```

a) Estimaţi ordinul de complexitate din punct de vedere al timpului de calcul. Particularizaţi rezultatul pentru cazul $m = n$.
b) Implementaţi procedura `gaxpy` în Matlab.
c) Cum veţi apela această funcţie în Matlab pentru ca rezultatul să se scrie tot în variabila y? d) Scrieţi un script care să contorizeze timpul de calcul necesar acestei proceduri, pentru cazul $n = m$. Observaţi că rulări distincte ale acestuia nu conduc la exact aceleaşi valori numerice pentru timpul de calcul. Comparaţi timpul obţinut cu cel necesar

instrucţiunii Matlab scrise cu vectori. Pentru aceasta, realizaţi un grafic de tipul celui din figura 2.1. Comentaţi rezultatele obţinute.

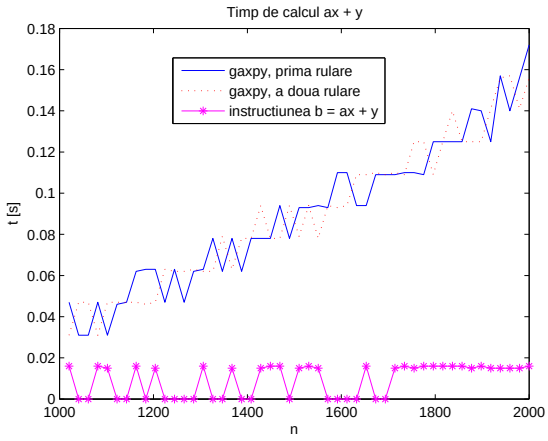

Figura 2.1: Un grafic de acest tip trebuie obţinut la exerciţiul 2.

În mod riguros, notaţia $T = \mathrm{O}(\cdots)$ folosită mai sus, defineşte o margine asimptotică superioară pentru dependenţa timpului de calcul de dimensiunea problemei. În general, se spune că un algoritm are *ordinul de complexitate* $O(g(n))$ din punct de vedere al timpului de calcul dacă şi numai dacă există o constantă pozitivă $C > 0$ şi un număr n_0 astfel încât $T \leq Cg(n)$ pentru orice $n \geq n_0$. Se poate defini şi o margine inferioară pentru această dependenţă. Un algoritm are *ordinul de complexitate* $\Omega(g(n))$ din punct de vedere al timpului de calcul dacă şi numai dacă există o constantă pozitivă $C > 0$ şi un număr n_0 astfel încât $Cg(n) \leq T$ pentru orice $n \geq n_0$.

În exemplul următor, se va analiza complexitatea unui algoritm care va căuta o valoare reala $xcrt$ în interiorul unui şir de n valori x ordonate crescător. Dacă valoarea $xcrt$ se află în afara intervalului $[x_1, x_n]$ atunci funcţia va întoarce valoarea -1, alftel ea va întoarce indexul din stânga al sub-intervalului în care se află valoarea. Fie pseudocodul următor:

`funcţie` caută_v1$(n, x, xcrt)$
; caută valoarea $xcrt$ în şirul ordonat x cu n valori
`întreg` n
`tablou real` $x[n]$; şirul este presupus ordonat
`real` $xcrt$
`logic` $flag$
...
`dacă` $(xcrt < x_1)$ `sau` $(xcrt > x_n)$
$rez = -1$

```
altfel
      flag = 0
      k = 1
      cât timp ((k <= n − 1) şi (flag = 0))
            dacă xcrt ≤ x_{k+1}
                  rez = k
                  flag = 1
            altfel
                  k = k + 1
            •
      •
•
întoarce rez
```

Exerciţiul 2.3:

a) Explicaţi cum se face căutarea în funcţia `caută_v1`.

b) Arătaţi că ordinul de complexitate în cazul cel mai defavorabil este $T = \mathrm{O}(n)$ şi în cazul cel mai favorabil $T = \Omega(1)$. Se va considera ca operaţie de referinţă compararea a două numere reale (instrucţiunea $k \leq n-1$ este o comparaţie între două numere întregi, iar $flag = 0$ este o comparaţie între două variabile logice, timpul necesar acestor instrucţiuni se va neglija).

c) Implementaţi funcţia în Matlab şi scrieţi un script pentru testarea corectitudinii ei.

Căutarea se face mai eficient dacă se adoptă o strategie bazată pe înjumătăţirea numărului de subintervale în care are loc căutarea, ca în pseudocodul de mai jos.

```
funcţie caută_v2(n, x, xcrt)
...
dacă (xcrt < x_1) sau (xcrt > x_n)
      rez = −1
altfel
      k1 = 1
      k2 = n
      cât timp k2 − k1 ≠ 1
            km = [(k1 + k2)/2]      ; [···] este partea întreagă
            dacă xcrt < x_{km}
                  k2 = km
            altfel
                  k1 = km
```

-
-
  $rez = k1$
-

întoarce rez

Exerciţiul 2.4:

a) Explicaţi cum se face căutarea în funcţia `caută_v2`.

b) Arătaţi că ordinul de complexitate este $T = O(\log(n))$, unde log este logaritmul în baza 2.

c) Implementaţi funcţia în Matlab şi completaţi script-ul scris la exerciţiul anterior pentru testarea corectitudinii ei.

O altă posibilitate de a implementa căutarea binară este prin folosirea unei algoritm recursiv. Un algoritm recursiv este un algorim care se apelează pe el însuşi. Ideea principală este aceea de a împărţi problema în subprobleme de acelaşi tip. Pseudocodul căutării binare pentru problema de mai sus este următorul.

funcţie caută_v3($n, x, xcrt$)
...
dacă ($xcrt < x_1$) sau ($xcrt > x_n$)
    $rez = -1$
altfel
    $rez =$ binary_search($x, xcrt, 1, n$);
-
întoarce rez

funcţie binary_search($x, xcrt, idx\_start, idx\_stop$)
; presupuneri:
; x - ordonate, $idx\_start < idx\_stop$
; $xcrt$ se afla in interiorul vectorului x
...
$mijloc = [(idx\_start + idx\_stop)/2]$
dacă $idx\_stop = idx\_start + 1$
    $rez = mijloc$
altfel dacă $x_{mijloc} > xcrt$
    $rez =$ binary_search($x, xcrt, idx\_start, mijloc$)
altfel
    $rez =$ binary_search($x, xcrt, mijloc, idx\_stop$)

•
întoarce *rez*

Evaluarea complexităţii algoritmilor recursivi se face prin scrierea unei relaţii de recurenţă. De exemplu, pentru algoritmul recursiv de mai sus, la fiecare iteraţie dimensiunea problemei se reduce la jumătate, iar timpul necesar pentru a decide care jumătate să fie eliminată este constant, nu depinde de dimensiunea problemei. De aceea putem scrie

$$T(n) = T(n/2) + \mathrm{O}(1).$$

Pentru a simplifica raţionamentul, vom presupune că n este o putere a lui 2: $n = 2^k$. Urmează că

$$T(n) \leq T(n/2)+C \leq T(n/4)+2C \leq \cdots \leq T(n/2^k)+kC = T(1)+C\log(n) = \mathrm{O}(\log(n)).$$

Se poate arăta că acest algoritm are această complexitate pentru orice valoare a lui n, nu numai pentru valori de tipul $n = 2^k$.

Exerciţiul 2.5:
Implementaţi funcţia `cautã_v3` în Matlab şi completaţi script-ul scris la exerciţiul anterior pentru testarea corectitudinii ei.

2.2 Necesarul de memorie

Complexitatea unui algoritm din punct de vedere al necesarului de memorie este dependenţa dintre necesarul de memorie exprimat în număr de locaţii elementare de memorie şi dimensiunea problemei. De obicei, o locaţie elementară de memorie este cea corespunzătoare unui număr real.

Exerciţiul 2.6:
Executaţi următoarele comenzi în consola Matlab:

```
>> clear all;
>> a = 2.3;
>> whos
```

Caţi bytes ocupă un număr real?

Exerciţiul 2.7:
Executaţi următoarele comenzi în consola Matlab:

```
>> clear all;
>> i = 7;  % Numar intreg?
>> j = int8(7);
>> whos
```

Comentaţi rezultatul. Pentru înţelegerea detaliilor, căutaţi în documentaţia Matlab cuvintele cheie "Integer data types".

În cazul codului ce calculează produsul scalar a doi vectori, descris la pagina 22, este necesară memorarea a doi vectori de dimensiune n şi a unui scalar real pentru rezultat. Ordinul de complexitate este deci $M = \mathrm{O}(2n+1) = \mathrm{O}(2n) \approx \mathrm{O}(n)$.

Exerciţiul 2.8:
Care este ordinul de complexitate din punct de vedere al necesarului de memorie pentru algoritmul implementat la exerciţiul 2.1? Dar la exerciţiul 2.2?

Exerciţiul 2.9:
Ce spaţiu de memorie este necesar pentru stocarea unei matrice pătrate de dimensiune 100000?

În rezolvarea numerică a problemelor reale din inginerie, un sistem de ecuaţii cu 100000 ecuaţii cu tot atâtea necunoscute poate reprezenta un model mediu sau chiar grosier al problemei de analizat. Din fericire însă, în majoritatea cazurilor, matrice de astfel de dimensiuni au foarte multe elemente nule. Astfel de matrice nu se memorează în forma lor totală, ca tablouri bidimensionale, ci se memorează doar elementele lor nenule. Mai mult, algoritmii de calcul se vor adapta acestei scheme de memorare, rezultând atât avantaje din punct de vedere al necesarului de memorie dar şi al timpului de calcul. *O matrice care conţine un număr foarte mare de elemente nenule se numeşte matrice rară.* Despre o matrice care nu este rară se spune că este *matrice densă* sau *plină*.

Se defineşte *densitatea unei matrice* ca fiind raportul dintre numărul de elemente nenule şi numărul total de elemente al matricei. De obicei, algoritmii care exploatează raritatea matricelor devin avantajoşi pentru valori mai mici ale densităţii. Nu se poate preciza o valoare exactă a densităţii care să demarcheze matricile rare de matricile pline. Dacă, pentru o anumită matrice care are şi elemente nule, se poate elabora un algoritm care exploatează această structură şi care, este mai eficient decât algoritmul gândit pentru matricea plină, atunci aceasta este o matrice rară.

Există mai multe metode de a memora matricele rare. Cele mai generale nu fac nici o presupunere asupra structurii matricei, respectiv asupra poziţiilor în care se află elemente nenule.

Cea mai naturală metodă ar fi cea în care se memorează doar valorile nenule şi "coordonatele" lor în matrice. Astfel, pentru un tablou bidimensional de dimensiune $m \times n$, în loc să se memoreze toate cele mn valori (inclusiv zerouri) într-un spaţiu de $M_{plin} = 8mn$ B, sunt necesare doar n_{nz} locaţii de memorie pentru numere reale şi $2n_{nz}$ locaţii de memorie pentru numere întregi, deci în total $M_{rar,coord} = 8 * n_{nz} + 4 * 2n_{nz} = 16n_{nz}$ B.

De exemplu, matricea $\mathbf{M}$ de dimensiune 3×4, având 6 elemente nenule, va fi memorată într-un vector *val* de dimensiune 6, şi doi vectori de numere întregi, de dimensiune 6, astfel:

$$\mathbf{M} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 \\ 2 & 3 & 0 & 7 \end{bmatrix} \quad \Rightarrow \quad \begin{cases} val & = & [\ 4 & 2 & 3 & 5 & 1 & 7\ ] \\ r\_idx & = & [\ 1 & 3 & 3 & 2 & 2 & 3\ ] \\ c\_idx & = & [\ 1 & 1 & 2 & 3 & 4 & 4\ ] \end{cases}$$

În acest exemplu, valorile nenule au fost citite pe coloană, de sus în jos şi de la stânga la dreapta, dar ele pot fi memorate în orice ordine, evident cu permutarea corespunzătoare a "coordonatelor".

Memorarea poate fi şi mai eficientă decât atât. Informaţia din vectorul *c_idx* poate fi comprimată, indicându-se doar locul unde se schimbă valoarea lui *c_idx*. Acesta este formatul cunoscut sub numele de *CCS* - Compressed Column Storage. O matrice $\mathbf{M}$, de dimensiuni $m \times n$ şi având un număr nnz de elemente nenule, va fi stocată cu ajutorul a trei tablouri unidimensionale astfel:

- un tablou unidimensional *val*, de dimensiune nnz, care conţine toate valorile nenule, de sus în jos şi de la stânga la dreapta;
- un talou unidimensional *c_ptr*, de dimensiune $n + 1$ (câte un element pentru fiecare coloană, plus un element adiţional care marchează sfârşitul tabloului), care conţine indecşii ce indică în tabloul *val* locurile în care încep coloanele. Mai precis, coloana j a matricei iniţiale are valorile în *val* de la poziţia $c\_ptr(j)$ la poziţia $c\_ptr(j+1)-1$;
- un talou unidimensional *r_idx*, de dimensiune nnz, care conţine, pentru fiecare element nenul din *val*, indexul liniei pe care se află.

Ordinul de complexitate din punct de vedere al necesarului de memorie este deci $M_{rar,CCS} = 8n_{nz} + 4(n + 1) + 4n_{nz} = 12n_{nz} + 4(n + 1)$ B.

Acelaşi exemplu, memorat în format CCS este:

$$\mathbf{M} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 \\ 2 & 3 & 0 & 7 \end{bmatrix} \quad \Rightarrow \quad \begin{cases} val & = & [\ 4 & 2 & 3 & 5 & 1 & 7\ ] \\ c\_ptr & = & [\ 1 & 3 & 4 & 5 & 7\ ] \\ r\_idx & = & [\ 1 & 3 & 3 & 2 & 2 & 3\ ] \end{cases}$$

Formatul CCS este şi modul de stocare a matricilor rare în Matlab. Puteţi încerca în Matlab urmaţorul exerciţiu demonstrativ (matricea folosită este aceeaşi ca în exemplele de mai sus):

```
>> M = [4 0 0 0; 0 0 5 1; 2 3 0 7];
>> Ms = sparse(M);
>> whos
```

Răspunsul ultimei comenzi va fi

```
  Name        Size                    Bytes  Class
  M           3x4                        96  double array
  Ms          3x4                        92  double array (sparse)
```

Într-adevăr, `M` este o matrice plină cu 12 elemente, deci are nevoie de 8*12 = 96 B, `Ms` este o matrice rară de dimensiune 3×4, cu 6 elemente nenule, în format CCS, deci are nevoie de $8 * 6 + 4 * (4 + 1) + 4 * 6 = 92$B. Câştigul nu este aici foarte mare pentru că matricea aceasta are o densitate $d = 6/(3 * 4) = 50\%$, ea nefiind în realitate o matrice rară.

Important: În Matlab, comanda `sparse` afişează matricile în triplete de tip coordonate - valoare, dar memorarea internă este în format CCS.

Exerciţiul 2.10:

Executaţi în Matlab următoarele comenzi şi comentaţi rezultatul.

a) Efectul comenzii `sparse` fără argumente de ieşire:

```
>> clear all;
>> M = [4 0 0 0; 0 0 5 1; 2 3 0 7];
>> sparse(M)
```

b) Efectul comenzii `find` cu trei argumente de ieşire:

```
>> clear all;
>> M = [4 0 0 0; 0 0 5 1; 2 3 0 7];
>> [r_idx,c_idx,val] = find(M);
```

c) Ce se întâmplă în cazul în care, într-o memorare pe coordonate, există intrări multiple, cu linii şi coloane identice? Pentru a răspunde, analizaţi următorul cod Matlab.

```
>> clear all;
>> M = [4 0 0 0; 0 0 5 1; 2 3 0 7];
>> [r_idx,c_idx,val] = find(M);
>> Ms = sparse(r_idx,c_idx,val);
>> r_idx(7) = 1;
>> c_idx(7) = 1;
>> val(7) = 10;
>> Ps = sparse(r_idx,c_idx,val)
```

Exerciţiul 2.11:

Executaţi în Matlab următoarele comenzi:

```
>> clear all;
>> M = [4 0 0 0; 0 0 5 1; 2 3 0 7];
>> Ms = sparse(M);
>> A = [0 0 0 0; 0 0 5 1; 2 3 0 7];
>> As = sparse(A);
>> As2 = Ms;
>> As2(1,1) = As2(1,1) - 4;
>> whos
```

Ce se întâmplă în Matlab dacă într-o matrice rară, un element care era nenul devine nul în urma unor calcule?

Exerciţiul 2.12:

a) Ce spaţiu de memorie este necesar pentru stocarea în format CCS a unei matrice pătrate de dimensiune 100000, ştiind că pe fiecare coloană sunt exact patru elemente nenule? Comparaţi rezultatul cu cel obţinut la exerciţiul 2.8

b) Calculaţi densitatea acestei matrice.

c) Scrieţi un script matlab care să rezolve punctele a) şi b).

De altfel, şi matricile pline sunt memorate în Matlab pe coloane. Aceasta se datorează faptului că Matlab a fost scris iniţial în Fortran şi acesta este modul de stocare al tablourilor bidimensionale în aceste limbaj. De aceea, în scrierea funcţiilor în Matlab, acesarea elementelor unei matrice pe coloane este mai rapidă decât accesarea elementelor pe linii.

Exerciţiul 2.13:

a) Scrieţi în Matlab o funcţie care să adune elementele unei matrice pătrate de dimensiune n, parcurgând elementele pe coloane:

```
funcţie suma_acces_coloane(n, a)
întreg n
tablou real a[n, n]
real s
...
s = 0
pentru j = 1, n
      pentru i = 1, n
            s = s + a_{i,j}
      •
• întoarce s
```

b) Scrieţi în Matlab o funcţie care să adune elementele unei matrice pătrate de dimensiune n, parcurgând elementele pe linii:

```
funcţie suma_acces_linii(n, a)
...                                   ; completaţi acest pseudocod
```

c) Apelaţi următorul script şi comentaţi rezultatul (fig. 2.2).

```
clear all;
nn = 1000:100:3000;
N = length(nn);

for i = 1:N
    n = nn(i);
    a = rand(n,n);
    tic;
    s = suma_acces_coloane(n,a);
    t2 = toc;
    t_col(i) = t2;

    tic;
    s = suma_acces_linii(n,a);
    t1 = toc;
    t_lin(i) = t1;
end
figure(1);
```

```
plot(nn,t_col,'r*-');
hold on;
plot(nn,t_lin,'bo-');
leg{1} = 'acces pe coloane';
leg{2} = 'acces pe linii';
legend(leg);
xlabel('n');
ylabel('t [s]');
```

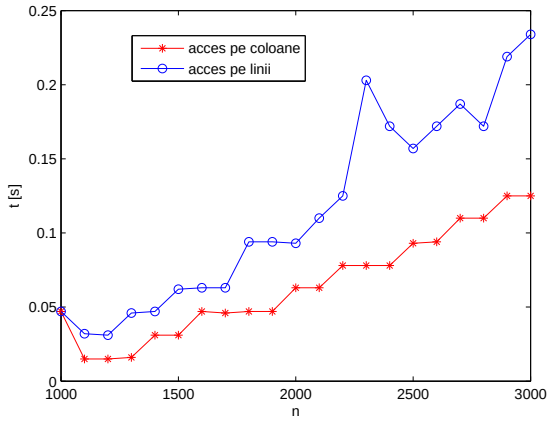

Figura 2.2: Timpul de calcul depinde şi de modul în care se accesează elementele unei matrice. Un astfel de grafic trebuie obţinut la exerciţiul 13.

În cazul în care matricile sunt memorate ca structuri rare, algoritmii ce operează cu ele sunt adaptaţi acestor structuri. De exemplu, algoritmul pentru procedura `gaxpy` în care matricea A este rară, memorată în format CCS `val`, `c_ptr`, `r_idx` este

`procedură` sparse_gaxpy$(m, n, val, col_ptr, row_idx, x, y, b)$
; calculează $b = ax + y$
; a este matrice rară, de dimensiune $m \times n$ memorată în format CCS
; declaraţii
...
`pentru` $i = 1, n$
    $b_i = y_i$
    `pentru` $p = col\_ptr(i), col\_ptr(i+1) - 1$
        $b_{row\_idx(p)} = b_{row\_idx(p)} + val_p x_i$
    •
•
`retur`

Exerciţiul 2.14:
a) Arătaţi că ordinul de complexitate al procedurii `sparse_gaxpy` este O(2**nnz*).
b)[1] Implementaţi procedura în Matlab şi testaţi corectitudinea ei.

2.3 Erori în calculele numerice

Unul din criteriile de evaluare a unui algoritm îl reprezintă eroarea cu care se obţine rezultatul. Erorile nu pot fi înlăturate total dintr-un calcul deoarece, de exemplu, chiar numerele reale nu pot fi reprezentate în calculator cu o precizie infinită. Numărul real π are o infinitate de cifre semnificative, iar orice sistem de calcul lucrează cu un număr finit de cifre semnificative. Pe de altă parte, erorile se propagă în calcule, astfel încât, chiar dacă datele de intrare ale unui program sunt foarte precise, rezultatul poate avea doar câteva cifre semnificative corecte. De aceea este foarte important ca pentru orice algoritm să se estimeze eroarea rezultatului.

Pentru analiza cantitativă a acestor erori, este utilă definirea următoarelor mărimi.

Eroarea absolută e_x a unei mărimi este diferenţa dintre valoarea aproximativă $\bar{x}$ şi valoarea exactă x a mărimii

$$e_x = \bar{x} - x. \tag{2.1}$$

Este evident că o astfel de mărime nu se poate calcula deoarece valoarea exactă nu este cunoscută. De aceea, este mai utilă aflarea unei *margini a erorii absolute* a_x, adică a unei mărimi care satisface

$$\|e_x\| \leq a_x. \tag{2.2}$$

Dacă presupunem că mărimea este scalară, atunci din (2.1) şi (2.2) rezultă că

$$\bar{x} - a_x \leq x \leq \bar{x} + a_x. \tag{2.3}$$

Altfel spus, cunoaşterea marginii erorii absolute permite definirea intervalului în care este plasată soluţia exactă. Relaţia (2.3) se mai scrie şi sub forma

$$x = \bar{x} \pm a_x. \tag{2.4}$$

Dezavantajul folosirii erorii absolute este acela că valoarea numerică depinde de sistemul de unităţi de măsură folosit, făcând dificilă aprecierea gradului de acurateţe a soluţiei. De aceea, se preferă folosirea unei mărimi relative, invariantă la sistemul de unităţi de măsură.

[1] Facultativ

Eroarea relativă ε_x a unei mărimi se defineşte ca fiind raportul dintre eroarea absolută şi norma mărimii

$$\varepsilon_x = \frac{e_x}{\|x\|}. \tag{2.5}$$

Nici această mărime nu se poate calcula deoarece nici eroarea absolută, nici mărimea exactă nu sunt cunoscute. De aceea se preferă folosirea unei *margini a erorii relative* r_x, mărime care satisface

$$\|\varepsilon_x\| \leq r_x. \tag{2.6}$$

Cel mai adesea, marginea erorii relative se exprimă în procente, iar relaţia (2.6) se scrie şi sub forma

$$x = \bar{x} \pm r_x\%. \tag{2.7}$$

Erorile dintr-un algoritm se pot clasifica în funcţie de tipul cauzelor care le generează în: erori inerente, erori de rotunjire şi erori de trunchiere.

Erorile inerente sunt erorile datorate reprezentării imprecise a datelor de intrare. Datele de intrare ale unui algoritm pot proveni de exemplu din măsurători, şi de aceea ele sunt afectate de erori. Aceste erori nu pot fi eliminate şi de aceea este important să putem evalua modul în care se propagă ele în calculele numerice, astfel încât să poată fi făcută o estimare a erorii rezultatului în funcţie de erorile datelor de intrare.

Erorile de rotunjire se datorează reprezentării finite a numerelor reale în calculator. Sistemele de calcul nu pot lucra decât cu aproximări raţionale ale numerelor reale. Numerele reale sunt reprezentate cu un număr finit de cifre semnificative. Nici aceste erori nu pot fi eliminate şi, ca şi în cazul erorilor inerente, este importantă evaluarea modului în care ele afectează rezultatul final.

Erorile de trunchiere provin din reprezentarea finită a algoritmilor. Există metode matematice a căror soluţie exactă ar necesita efectuarea unui număr infinit de calcule. Un astfel de exemplu este sumarea unei serii. Deoarece implementările nu pot fi făcute decât pentru algoritmi care fac un număr finit de calcule, nu are sens să permitem în pseudocod cicluri cu contor cu un număr infinit de paşi. În acest caz este importantă evaluarea erorii care se face datorită trunchierii procesului infinit.

Exerciţiile care urmează ilustrează aceste tipuri de erori.

2.3.1 Erori de rotunjire

Erorile de rotunjire se datorează reprezentării finite a numerelor reale în calculator. Sistemele de calcul nu pot lucra cu numere reale exacte, ci doar cu aproximări raţionale

ale acestora. În consecinţă, numerele reale nu pot fi reprezentate în calculator decât cu un număr finit de cifre semnificative.

În cele ce urmează, este util să reprezentăm numerele reale în baza 10, cu ajutorul unei părţi fracţionare f şi a unui exponent n.

$$\bar{x} = f \cdot 10^n. \tag{2.8}$$

Mai mult, pentru orice număr în afara lui 0, prin alegerea convenabilă a lui n, partea fracţionară satisface $0.1 \leq |f| < 1$. De exemplu $3.14 = 0.314 \cdot 10^1$, $-0.007856 = -0.7856 \cdot 10^{-2}$. *Cifrele părţii fracţionare se numesc cifre semnificative.*

Exerciţiul 2.15:
Câte cifre semnificative are numărul 3.14? Dar numărul -0.007856?

În calculator se pot memora un număr finit k de cifre semnificative. Se poate demonstra că eroarea relativă datorată acestui proces de rotunjire este majorată de $|\varepsilon_x| \leq 10^{-k+1}$. Evident că dacă cifrele pierdute sunt toate zero, atunci numărul poate fi reprezentat exact. Calculele în care intervin numere reale, chiar dacă ele sunt reprezentate exact, pot fi afectate însă la rândul lor de procesul de rotunjire. De exemplu, adunarea a două numere reale se face adunând părţile fracţionare după ce, în prealabil, numărul mai mic a fost rescris astfel încât să aibă exponentul numărului mai mare. La rescrierea numărului mai mic se pot pierde cifre semnificative. Pentru înţelegerea acestei afirmaţii să ne imaginăm că lucrăm pe un calculator ipotetic în care numărul de cifre semnificative este 3. Într-un astfel de calculator vrem să adunăm $x_1 = 3.73 = 0.373 \cdot 10$ cu $x_2 = 0.004 = 4 \cdot 10^{-3}$. Numărul x_2, de modul mai mic, se rescrie astfel încât să aibă exponentul 1, ca al numărului mai mare. $x_2 = 4 \cdot 10^{-4} \cdot 10 = 0.0004 \cdot 10$. Deoarece calculatorul permite memorarea doar a 3 cifre după virgulă, iar cifra 4 ar fi a patra, însemnă că de fapt cifra 4 este pierdută. La adunare x_2 este "văzut" a fi zero şi rezultatul adunării este de fapt x_1. Evident că un astfel de calculator nu există, în mod uzual se lucrează cu mai mult de 12 cifre semnificative, acest lucru depinzând de configuraţia hard, de limbajul de programare folosit şi de declaraţia făcută pentru variabile, dar ideea este exact cea descrisă de acest exemplu simplu.

Se defineşte zeroul (acurateţea, precizia, "epsilon-ul") maşinii ca fiind cel mai mic număr real care adunat la unitate îi modifică valoarea. Am putea spune că zeroul maşinii `eps` este cel mai mic număr pentru care `1 + eps > 1`. Pentru orice număr a mai mic decât zeroul maşinii $1 + a = 1$, unde această din urmă relaţie o considerăm efectuată în calculator şi nu în matematica exactă.

Într-un mediu în care zeroul maşinii nu este cunoscut, el poate fi calculat cu uşurinţă cu ajutorul unui program ce implementează următorul pseudocod:

`funcţie` zeroul_maşinii ()
`real` *epsilon*
$epsilon = 1$
`cât timp` $(1 + epsilon > 1)$
$\quad epsilon = epsilon/2$
•
$epsilon = epsilon * 2$
`întoarce` *epsilon*

Ca o consecinţă a celor discutate mai sus, rezultă că, spre deosebire de matematica exactă, în calculator adunarea numerelor reale nu este asociativă. Dacă trebuie adunate mai multe numere reale, pentru a obţine un rezultat afectat cât mai puţin de erori de rotunjire, trebuie ca numerele să fie adunate în ordinea crescătoare a valorii lor.

Exerciţiul 2.16:
a) Implementaţi în Matlab funcţia `zeroul_maşinii`.
b) Comparaţi rezultatul obţinut cu cel al funcţiei Matlab `eps`.

2.3.2 Erori inerente

Se poate demonstra că, în cazul operaţiilor algebrice elementare, propagarea erorilor inerente se face conform formulelor din tabelele de mai jos.

Erori	**Adunare** $y = x_1 + x_2$	**Scădere** $y = x_1 - x_2$
Eroare absolută: $e_y =$	$e_{x_1} + e_{x_2}$	$e_{x_1} - e_{x_2}$
majorată de: $a_y =$	$a_{x_1} + a_{x_2}$	$a_{x_1} + a_{x_2}$
Eroare relativă: $\varepsilon_y =$	$\frac{x_1}{x_1+x_2}\varepsilon_{x_1} + \frac{x_2}{x_1+x_2}\varepsilon_{x_2}$	$\frac{x_1}{x_1-x_2}\varepsilon_{x_1} - \frac{x_2}{x_1+x_2}\varepsilon_{x_2}$
majorată de $r_y =$	$\left\|\frac{x_1}{x_1+x_2}\right\| r_{x_1} + \left\|\frac{x_2}{x_1+x_2}\right\| r_{x_2}$	$\left\|\frac{x_1}{x_1-x_2}\right\| r_{x_1} + \left\|\frac{x_2}{x_1-x_2}\right\| r_{x_2}$

Tabelul 2.1: Erorile rezultatului adunării şi scăderii a două numere reale în funcţie de erorile datelor de intrare.

Precizăm că operaţia $x_1 + x_2$ este considerată adunare dacă ambii operanzi au acelaşi semn. În caz contrar, operaţia este de fapt o scădere. Similar, $x_1 - x_2$ este o scădere dacă ambii operanzi au acelaşi semn, în caz contrar fiind de fapt efectuată o adunare.

Dacă analizăm marginea erorii relative de la adunare, se observă că erorile relative ale datelor de intrare sunt ponderate cu $|x_1/(x_1 + x_2)|$ şi $|x_2/(x_1 + x_2)|$, ambele ponderi fiind

Erori	**Înmulţire** $y = x_1 x_2$	**Împărţire** $y = \frac{x_1}{x_2}$
Eroare absolută: $e_y =$	$x_2 e_{x_1} + x_1 e_{x_2}$	$\frac{1}{x_2} e_{x_1} - \frac{x_1}{x_2^2} e_{x_2}$
majorată de: $a_y =$	$\|x_2\| a_{x_1} + \|x_1\| a_{x_2}$	$\frac{1}{\|x_2\|} a_{x_1} + \frac{\|x_1\|}{x_2^2} a_{x_2}$
Eroare relativă: $\varepsilon_y =$	$\varepsilon_{x_1} + \varepsilon_{x_2}$	$\varepsilon_{x_1} - \varepsilon_{x_2}$
majorată de $r_y =$	$r_{x_1} + r_{x_2}$	$r_{x_1} + r_{x_2}$

Tabelul 2.2: Erorile rezultatului înmulţirii şi împărţirii a două numere reale în funcţie de erorile datelor de intrare.

subunitare. Aceasta înseamnă că eroarea rezultatului adunării a două numere reale nu este amplificată, ea rămâne de acelaşi ordin de mărime cu erorile datelor de intrare. Se spune că *adunarea este o operaţie stabilă numeric*. Nu acelaşi lucru se poate spune despre scădere. Ponderile în acest caz sunt $|x_1/(x_1 - x_2)|$ şi $|x_2/(x_1 - x_2)|$, care pot fi oricât de mari pentru că diferenţa $x_1 - x_2$ poate fi oricât de mică. În consecinţă, rezultatul unei scăderi poate fi afectat de erori mult mai mari decât erorile datelor de intrare. Se spune că *scăderea este o operaţie instabilă numeric*. Instabilitatea la scădere apare mai ales atunci când numerele sunt foarte apropiate, efect cunoscut şi sub numele de *efect de anulare prin scădere*.

Efectuarea unor calcule de acest tip, în care se urmăreşte nu numai valoarea rezultatului ci şi modul în care acesta este afectat de erorile datelor de intrare se numeşte *calcul cu intervale*. Este acum mai evident faptul că adunarea numerelor reale în calculator nu este o operaţie asociativă, afirmaţie făcută cu ocazia discuţiei referitoare la erorile de rotunjire.

Un ultim exemplu interesant este cel al funcţiei $y = \sqrt{x}$ pentru care rezultă o eroare absolută $e_y = 1/(2\sqrt{x})e_x$ şi o eroare relativă egală cu jumătate din eroarea relativă a datei $\varepsilon_y = \varepsilon_x/2$. Am putea deduce de aici, în mod eronat, că în acest caz, dacă aplicăm radicalul în mod repetat rezultatului, eroarea tinde către zero. Toate calculele prezentate în acest paragraf (inclusiv acesta din urmă) au presupus un calcul exact, adică un calcul fără erori de rotunjire. Rotunjirea nu poate fi însă ignorată. Nu putem separa proprietăţile erorilor inerente de efectul rotunjirii. De aceea, vom accepta un fel de superpoziţie a erorilor în sensul că eroarea relativă într-un calcul aproximativ este egală cu eroarea relativă produsă de calculul aproximativ cu numere exacte (adică eroarea de rotunjire) plus eroarea relativă produsă de calculul exact cu numere aproximative (afectate deci de erori inerente). Cu această precizare, formulelor deduse trebuie să li se adauge o eroare de rotunjire, de exemplu

$$\varepsilon_{\sqrt{x}} = \frac{\varepsilon_x}{2} + eps. \tag{2.9}$$

Exerciţiul 2.17:
Scrieţi un program Matlab care să calculeze soluţiile ecuaţiei $ax^2 + bx + c = 0$, unde $a = 1 \pm 0.1\%$, $b = -100.01 \pm 0.1\%$, $c = 1 \pm 0.1\%$.

2.3.3 Erori de trunchiere

Erorile de trunchiere provin din reprezentarea finită a algoritmilor. Există metode matematice a căror soluţie exactă ar necesita efectuarea unui număr infinit de calcule. Estimarea erorilor de trunchiere se poate face de multe ori apriori, folosind rezultatele teoretice ale matematicii.

Ca exemplu, să considerăm dezvoltarea în serie Taylor a unei funcţii:

$$f(x) = f(x_0) + \frac{x - x_0}{1!} f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \cdots \tag{2.10}$$

Exerciţiul 2.18:
Scrieţi seria alternantă obţinută din dezvoltarea în serie Taylor a funcţiei $\sin(x)$ în jurul punctului $x_0 = 0$.

Conform teoriei seriilor alternante, se ştie că eroarea absolută făcută prin această trunchiere este mai mică decât ultimul termen considerat. Cunoaşterea acestui rezultat teoretic ne permite să elaborăm un algoritm de evaluare a funcţiei sinus care să stabilească singur când se va opri. Algoritmul va aduna termeni la suma parţială până când termenul curent sumat este mai mic decât o anumită valoare. Această valoare nu trebuie să fie prea mare căci atunci soluţia va fi nesatisfăcătoare. De asemenea, o valoare mai mică decât zeroul maşinii este lipsită de sens deoarece un termen curent cu o astfel de valoare, adunat la suma parţială acumulată până în acel moment, practic nu mai are nici o influenţă asupra valorii sumei parţiale. Într-un astfel de moment, continuarea sumării seriei este inutilă, ea nu mai îmbunătăţeşte rezultatul, fenomen total diferit de teoria matematică în care rezultatul este cu atât mai precis cu cât suma conţine mai mulţi termeni. Pseudocodul următor implementează calculul funtiei sinus într-un punct x, cu o eroare impusă err.

`funcţie` sinus(x, err)
; întoarce valoarea funcţiei sinus in punctul x
; prin trunchierea seriei Taylor dezvoltată in 0
`real` x
`real` err
`real` t, s
`întreg` k

$t = x$
$s = t$
$k = 0$
`cât timp` $(|t| > err)$
    $k = k + 1$
    $t = (-1) * t * \frac{x^2}{(2k)(2k+1)}$
    $s = s + t$

•

`intoarce` s

Exerciţiul 2.19:

a) Implementaţi în Matlab pseudocodul de mai sus.

b) Scrieţi un program care să reprezinte grafic variaţia modulului termenului curent (ca în figura 2.3) şi modulul diferenţei dintre sumele parţiale consecutive (figura 2.4) în funcţie de iteraţie. Comentaţi rezultatul.

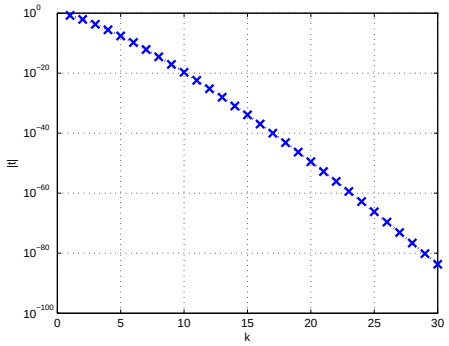

Figura 2.3: Modulul termenului curent al dezvoltării în serie Taylor a funcţiei sinus. Un astfel de grafic trebuie obţinut la exerciţiul 18.

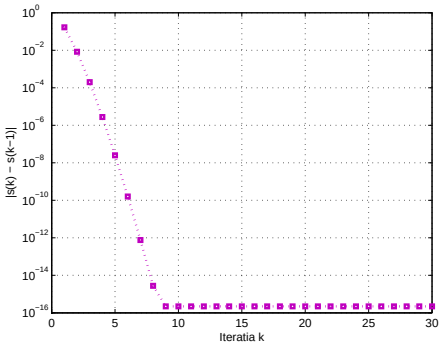

Figura 2.4: Modulul diferenţei dintre sume parţiale consecutive la dezvoltarea în serie Taylor a funcţiei sinus. Un astfel de grafic trebuie obţinut la exerciţiul 18.

Capitolul 3

Analiza circuitelor electrice rezistive liniare

Această temă reprezintă cel mai simplu exemplu de aplicaţie din ingineria electrică, care conduce la rezolvarea unui sistem de ecuaţii algebrice liniare.

Un circuit rezistiv liniar este un circuit ce conţine rezistoare, surse ideale de tensiune şi curent şi surse comandate liniar. Problema fundamentală a analizei acestor circuite are ca date: topologia circuitului, valorile parametrilor (rezistenţele, valorile surselor) şi urmăreşte calculul curenţilor şi tensiunilor din fiecare latură.

Există mai multe metode de rezolvare a unor astfel de circuite: metoda ecuaţiilor Kirchhoff, metoda potenţialelor nodurilor, metoda curenţilor ciclici. Atât metoda potenţialelor nodurilor cât şi metoda curenţilor ciclici generează un sistem de ecuaţii mai mic decât metoda Kirchhoff, având o matrice a coeficienţilor cu proprietăţi mai bune (de exemplu simetrie, diagonal dominanţă). Metoda curenţilor ciclici necesită şi alegerea unui sistem convenabil de bucle independente, fiind mai dificil de transpus într-un algoritm decât metoda potenţialelor nodurilor (numită mai scurt *metoda sau tehnica nodală*).

Exerciţiile din acest capitol vă vor ajuta să înţelegeţi modul în care se concepe şi se testează un algoritm folosind tehnica nodală pentru asamblarea sistemului de ecuaţii, metode directe de rezolvare a sistemului rezultat şi tehnici de matrice rare pentru a obţine un algoritm cu o complexitate cât mai bună.

3.1 Metoda potenţialelor nodurilor

Primul pas către elaborarea unui algoritm îl constituie înţelegerea teoriei metodei de rezolvare şi pregătirea unui exemplu de dimensiuni mici pentru primele teste care vor fi făcute viitorului program.

Vom considera un circuit electric alcătuit numai din laturi standard de tip sursă reală de tensiune (fig.3.1). În particular, aceste laturi pot fi rezistoare ideale, dar nu surse ideale de tensiune.

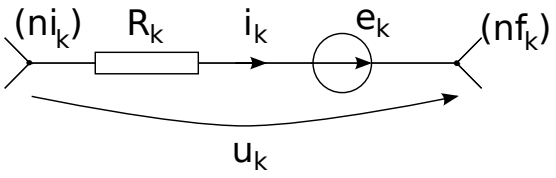

Figura 3.1: Latura standard.

Datele problemei sunt

- topologia: numărul de noduri N, numărul de laturi L şi modul în care sunt conectate laturile adică graful circuitului,
- toate rezistenţele laturilor R_k, $k = 1, \ldots, L$,
- toate tensiunile electromotoare e_k, $k = 1, \ldots, L$

Se cere să se calculeze

- toate tensiunile u_k la bornele laturilor,
- toţi curenţii i_k ce străbat laturile,
- puterea consumată şi puterea generată în circuit.

Deşi alegerea sensurilor de referinţă poate fi arbitrară, în vederea transformării metodei nodale într-un algoritm, este mult mai uşor dacă se aleg sensuri de referinţă în mod sistematic, ca de exemplu: sensul de referinţă al curentului este dat de sensul săgeţii interioare al sursei de tensiune şi sensul de referinţă al tensiunii se alege astfel încât să se respecte convenţia de la receptoare (aşa cum sunt reprezentate în fig. 3.1). Sensul de referinţă al curentului ce străbate o latură k stabileşte şi o relaţie de ordonare între cele două noduri ale laturii respective, el fiind sensul de la nodul iniţial notat (ni_k) la nodul final notat (nf_k).

Conform teoriei circuitelor, fenomenele sunt complet descrise de un număr de $N-1$ ecuaţii Kirchhoff I

$$\sum_{k\in(n)}^{A} i_k = 0, \quad n = 1, \ldots, N, \tag{3.1}$$

un număr de $L - N + 1$ ecuaţii Kirchoff II scrie pe un sistem de bucle independente

$$\sum_{k\in[b]}^{A} u_k = 0, \quad b = 1, \ldots, L - N + 1, \tag{3.2}$$

şi L relaţii Joubert, scrise pentru toate laturile

$$u_k = R_k i_k - e_k, \quad k = 1, \ldots, L, \tag{3.3}$$

în total un număr de $2L$ ecuaţii cu $2L$ necunoscute (toţi curenţii şi toate tensiunile).

În tehnica nodală necunoscutele principale ale problemei sunt potenţialele nodurilor v_k, $k = 1, \ldots, N$, unde unul din noduri are, în mod convenţional, potenţialul nul. Vom presupune că numerotarea nodurilor este făcută astfel încât nodul de indice maxim este cel de referinţă $v_N = 0$. Prin exprimarea tensiunilor ca diferenţe de potenţial, teorema Kirchhoff II este identic satisfăcută. Altfel spus, relaţia (3.2) este satisfăcută dacă se scriu toate relaţiile

$$u_k = v_{ni_k} - v_{nf_k}, \quad k = 1, \ldots, L. \tag{3.4}$$

Pentru a face scrierea mai compactă este util să folosim următoarele notaţii:

$\mathbf{u}$	$=$	$[\ u_1 \quad u_2 \quad \ldots \quad u_L\ ]^T \in \mathbb{R}^{L\times 1}$	vectorul tensiunilor laturilor,
$\mathbf{i}$	$=$	$[\ i_1 \quad i_2 \quad \ldots \quad i_L\ ]^T \in \mathbb{R}^{L\times 1}$	vectorul curenţilor prin laturi,
$\mathbf{v}$	$=$	$[\ v_1 \quad v_2 \quad \ldots \quad v_N\ ]^T \in \mathbb{R}^{N\times 1}$	vectorul potenţialelor nodurilor,
$\mathbf{e}$	$=$	$[\ e_1 \quad e_2 \quad \ldots \quad e_L\ ]^T \in \mathbb{R}^{L\times 1}$	vectorul tensiunilor electromotoare,
$\mathbf{R}$	$=$	$\mathrm{diag}([\ R_1 \quad R_2 \quad \ldots \quad R_L\ ]) \in \mathbb{R}^{L\times L}$	matricea diagonală a rezistenţelor laturilor.

Cu aceste notaţii, relaţiile Kirchhoff I (3.1) se scriu în mod compact

$$\mathbf{Ai} = \mathbf{0}, \tag{3.5}$$

unde $\mathbf{A} = (a_{ij})_{i=1,N-1;j=1,L}$ este *matricea incidenţelor laturi-noduri*, o matrice topologică de dimensiune $(N-1)\times L$, un element al ei fiind definit astfel

$$a_{ij} = \begin{cases} 0 & \text{dacă nodul } i \text{ nu aparţine laturii } j; \\ +1 & \text{dacă nodul } i \text{ este nod iniţial pentru latura } j; \\ -1 & \text{dacă nodul } i \text{ este nod final pentru latura } j. \end{cases}$$

Exerciţiul 3.1:

Fie circuitul din fig.3.2.

a) Orientaţi laturile în conformitate cu convenţia descrisă mai sus;

b) Numerotaţi laturile;

c) Numerotaţi nodurile;

d) Scrieţi matricea incidenţelor laturi noduri $\mathbf{A} \in \mathbb{Z}^{4\times 9}$.

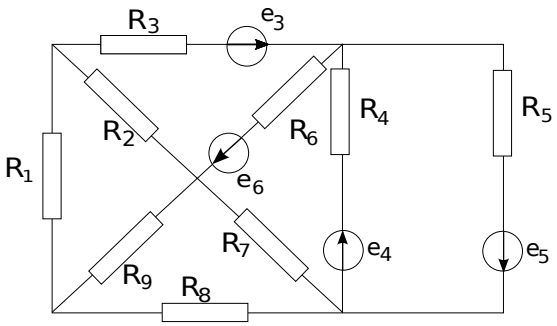

Figura 3.2: Circuit simplu de test.

Relaţia Kirchhoff II în forma (3.4) se scrie

$$\mathbf{u} = \mathbf{A}^T\mathbf{v}, \tag{3.6}$$

iar relaţiile lui Joubert (3.3) se scriu compact

$$\mathbf{u} = \mathbf{R}\mathbf{i} - \mathbf{e}. \tag{3.7}$$

Exerciţiul 3.2:

Verificaţi relaţiile (3.6) şi (3.7) pentru circuitul de test din fig. 3.2.

Dacă matricea $\mathbf{R}$ este inversabilă (lucru adevărat dacă toate rezistenţele sunt pozitive) atunci din (3.7) rezultă că

$$\mathbf{i} = \mathbf{R}^{-1}(\mathbf{u} + \mathbf{e}). \tag{3.8}$$

Înlocuind (3.8) şi (3.6) în (3.5), rezultă ecuaţia matriceală satisfăcută de vectorul potenţialelor

$$\mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T\mathbf{v} = -\mathbf{A}\mathbf{R}^{-1}\mathbf{e}. \tag{3.9}$$

Matricea coeficienţilor sistemului algebric liniar de rezolvat

$$\mathbf{G} = \mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T \quad \in \mathbb{R}^{(N-1)\times(N-1)} \tag{3.10}$$

este numită *matricea conductanţelor nodale*.

Exerciţiul 3.3:

Calculaţi matricea $\mathbf{G}$ dată de relaţia (3.10) pentru circuitul simplu de test.

	ni_k				nf_k		
	$*$	$*$	$*$	$*$	$*$	$*$	$*$
ni_k	$*$	$+1/R_k$	$*$	$*$	$-1/R_k$	$*$	$*$
	$*$	$*$	$*$	$*$	$*$	$*$	$*$
	$*$	$*$	$*$	$*$	$*$	$*$	$*$
nf_k	$*$	$-1/R_k$	$*$	$*$	$+1/R_k$	$*$	$*$
	$*$	$*$	$*$	$*$	$*$	$*$	$*$
	$*$	$*$	$*$	$*$	$*$	$*$	$*$

Figura 3.3: Contribuţia unei laturi k la matricea conductanţelor nodale.

	$*$
ni_k	$-e_k/R_k$
	$*$
	$*$
nf_k	$+e_k/R_k$
	$*$
	$*$

Figura 3.4: Contribuţia unei laturi k la vectorul injecţiilor de curent.

Este uşor de observat că termenii ei au următoarea semnificaţie: orice termen diagonal G_{ii} reprezintă suma conductanţelor laturilor care concură la nodul i, iar orice termen nediagonal G_{ij} cu $i \neq j$ reprezintă suma conductanţelor laturilor care unesc direct nodul i cu nodul j, luată cu semnul minus. Altfel scris,

$$G_{ii} = \sum_{k \in (i)} \frac{1}{R_k}, \tag{3.11}$$

$$G_{ij} = -\sum_{k \in (i); k \in (j)} \frac{1}{R_k} \qquad \text{pentru} \quad i \neq j. \tag{3.12}$$

Termenul liber al sistemului de ecuaţii

$$\mathbf{t} = -\mathbf{A}\mathbf{R}^{-1}\mathbf{e} \quad \in \mathbb{R}^{(N-1)\times 1} \tag{3.13}$$

este numit *vectorul injecţiilor de curent.*

Exerciţiul 3.4:
Calculaţi vectorul $\mathbf{t}$ dat de relaţia (3.13) pentru circuitul simplu de test.

Un termen al acestui vector t_k reprezintă suma algebrică a unor termeni de tipul e_m/R_m pentru toate laturile m care concură la nodul k, cu plus dacă săgeata internă a sursei de tensiune electromotoare de pe latura m intra în nodul k, şi cu semnul minus în caz contrar.

Este important să remarcăm că matricea $\mathbf{G}$ este simetrică şi pozitiv definită. În relaţia

(3.10) $\mathbf{R}$ este o matrice diagonala, având pe diagonală valorile rezistenţelor laturilor

$$\mathbf{R} = \begin{bmatrix} R_1 & 0 & \cdots & 0 \\ 0 & R_2 & \cdots & 0 \\ \cdots & & & \\ 0 & 0 & \cdots & R_L \end{bmatrix}. \tag{3.14}$$

Matricea $\mathbf{R}^{-1}$ este tot o matrice diagonală, având pe diagonală valorile conductanţelor laturilor

$$\mathbf{R}^{-1} = \begin{bmatrix} 1/R_1 & 0 & \cdots & 0 \\ 0 & 1/R_2 & \cdots & 0 \\ \cdots & & & \\ 0 & 0 & \cdots & 1/R_L \end{bmatrix}. \tag{3.15}$$

Atunci

$$\mathbf{G}^T = \left(\mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T\right)^T = \left(\mathbf{A}^T\right)^T \left(\mathbf{R}^{-1}\right)^T (\mathbf{A})^T = \mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T = \mathbf{G}, \tag{3.16}$$

deci matricea conductanţelor nodale este simetrică. Mai mult, ea este diagonal dominantă.

Pentru demonstrarea proprietăţii de pozitiv definire, să considerăm un vector coloană $\mathbf{x}$ arbitrar, nenul. Atunci

$$\mathbf{x}^T\mathbf{G}\mathbf{x} = \mathbf{x}^T\mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T\mathbf{x} = \mathbf{y}^T\mathbf{R}^{-1}\mathbf{y} = \sum_{k=1}^{L} \frac{y_k^2}{R_k} > 0, \tag{3.17}$$

unde am notat $\mathbf{y} = \mathbf{A}^T\mathbf{x}$ un vector coloană de componente y_k, $k = 1, L$. Inegalitatea demonstrată în (3.17) este strictă, ea ar putea fi zero doar dacă vectorul $\mathbf{y}$, şi în consecinţă vectorul $\mathbf{x}$ este nul, ceea ce contrazice ipoteza făcută. În concluzie, matricea conductanţelor nodale este pozitiv definită.

Există mai multe variante posibile de concepere a algoritmului acestei metode. Toate au trei etape principale: *etapa de preprocesare* în care se descrie problema şi se asamblează sistemul de ecuaţii de rezolvat, *etapa de rezolvare* în care se apelează o procedură propriu-zisă de rezolvare a sistemului de ecuaţii rezultat (procedură numită foarte adesea "solver") şi *etapa de postprocesare* în care se calculează alte mărimi de interes.

3.2 Structuri de date

În conceperea unui algoritm, stabilirea structurilor de date ce vor fi folosite este de asemenea o etapă foarte importantă.

Numele datelor ce le vom folosi în algoritm vor fi alese în concordanţă cu teoria prezentată în paragraful anterior. Reamintim că trebuie să ştim informaţiile legate de topologie:

numărul de noduri N, numărul de laturi L şi, pentru fiecare latură, care este nodul iniţial ni_k şi care este nodul final nf_k. De asemenea, trebuie să ştim parametrii fiecărei laturi: rezistenţa R_k şi tensiunea electromotoare e_k.

Declaraţii posibile pentru aceste date ar putea fi

```
; declaratii date - varianta A
întreg N                    ; număr de noduri
întreg L                    ; număr de laturi
tablou întreg ni[L]         ; noduri iniţiale ale laturilor
tablou întreg nf[L]         ; noduri finale ale laturilor
tablou real R[L]            ; rezistenţe
tablou real e[L]            ; tensiuni electromotoare
```

Exerciţiul 3.5:

Pentru circuitul simplu de test, consideraţi următoarele valori numerice: $R_1 = 1\Omega$, $R_2 = 1/2\Omega$, $R_3 = 1/3\Omega$, $e_3 = 30$ V, $R_4 = 1/4\Omega$, $e_4 = 40$ V, $R_5 = 1/5\Omega$, $e_5 = 50$ V, $R_6 = 1/6\Omega$, $e_6 = 60$ V, $R_7 = 1/7\Omega$, $R_8 = 1/8\Omega$, $R_9 = 1/9\Omega$, $e_9 = 90$ V. Reamintim că acest exemplu nu are importanţă practică, el va fi folosit exclusiv pentru prima validare a programului ce va fi implementat. Completaţi apoi un tabel de tipul:

k	ni_k	nf_k	R_k	e_k
1				
...				
9				

Pentru a evita însă transmiterea unui număr mare de parametri ca argumente de funcţii, recomandăm însă agregarea datelor, ca de exemplu

```
; declaraţii date - varianta B
înregistrare circuit
    întreg N                ; număr de noduri
    întreg L                ; număr de laturi
    tablou întreg ni[L]     ; noduri iniţiale ale laturilor
    tablou întreg nf[L]     ; noduri finale ale laturilor
    tablou real R[L]        ; rezistenţe
    tablou real e[L]        ; tensiuni electromotoare
```

De asemenea, un alt aspect important este acela că matricea conductanţelor nodale şi vectorul injecţiilor de curent sunt rare. Totuşi, pentru a păstra simplă descrierea algoritmului, în cele ce urmează vom păstra acelaşi tip de declaraţii şi pentru matricile rare şi vectorii rari, urmând a discuta exploatarea rarităţii ulterior. În consecinţă, variabile mai importante care vor apărea în algoritm sunt:

; declaraţii variabile utile
tablou real $G[N, N]$; matricea conductanţelor nodale (în final va fi stocată rar)
tablou real $t[N]$; vectorul injecţiilor de curent (în final va fi stocat rar)
tablou real $v[N]$; vectorul potenţialelor

3.3 Etapa de preprocesare

Etapa de preprocesare constă în citirea datelor (de exemplu de la tastatură sau din fişiere) şi în asamblarea sistemului de ecuaţii de rezolvat.

În cazul variantei A de declaraţii, procedura de citire a datelor poate fi

procedură citire_date_A (N, L, ni, nf, R, e)
; declaraţii
...
citeşte N, L
pentru $k = 1, L$
    citeşte ni_k, nf_k, R_k, e_k
•
retur

În cazul variantei B de declaraţii, rutina de citire a datelor poate fi o funcţie, de tipul

funcţie citire_date_B ()
; declaraţii
...
citeşte circuit.N, circuit.L
pentru $k = 1$,circuit.L
    citeşte circuit.ni_k, circuit.nf_k, circuit.R_k, circuit.e_k
•
întoarce circuit

Exerciţiul 3.6:

Pentru primele teste vom prefera chiar o procedură de "citire" rapidă a circuitului simplu de test. Pentru aceasta, scrieţi un fişier `citire_circuitRE_simplu.m` ca mai jos. Completaţi datele circuitului în conformitate cu tabelul de la exerciţiul 3.5.

```
function [circuit] = citire_circuitRE_simplu()

circuit.N = 5;
circuit.L = 9;
circuit.ni = [?; ?; ?; ?; ?; ?; ?; ?; ?];  % completati
circuit.nf = .........                      % completati
circuit.R = [1/1; 1/2; 1/3; 1/4; 1/5; 1/6; 1/7; 1/8; 1/9];
circuit.e = ..........                      % completati
```

Există mai multe variante şi de a asambla sistemul de ecuaţii. O variantă ar fi cea în care se parcurg nodurile şi se scriu ecuaţiile una câte una. Aşa ar face şi o persoană care ar aplica această metoda, cu creionul pe hârtie, pentru a rezolva o problemă de dimensiuni extrem de mici. În această abordare trebuie identificate care sunt laturile ce ating un nod. Numărul lor variază de la nod la nod, iar algoritmul corespunzător este destul de costisitor necesitând analiza grafului circuitului. O altă abordare se bazează pe parcurgerea laturilor şi adunarea contribuţiilor acestora la sistem. Această variantă este mult mai simplu de conceput deoarece fiecare latură are exact două noduri. Din acelaşi motiv şi descrierea circuitului a fost orientată pe laturi şi nu pe noduri.

Algoritmul metodei nodale este următorul

`procedură` nodalRE_v1 (circuit, G, t)
; asamblează sistemul de ecuaţii pentru un circuit cu laturi de tip
; sursă reală de tensiune, folosind tehnica nodală
; parametri de intrare:
; circuit - structură de date ce descrie circuitul
; parametri de ieşire:
; G - matricea conductanţelor nodale şi
; t - vectorul injecţiilor de curent
; declaraţii
....
L =circuit.L ; pentru simplificarea scrierii algoritmului
N =circuit.N
ni = circuit.ni

nf = circuit.nf
R = circuit.R
e = circuit.e
; anulează componentele matricei G şi a vectorului termenilor liberi t
G = 0
t = 0
; asamblează sistem
pentru $k = 1, L$; parcurge laturi
    i =ni$_k$; nodul iniţial al laturii k
    j =nf$_k$; nodul final al laturii k
    $G_{ii} = G_{ii} + 1/R_k$
    $G_{jj} = G_{jj} + 1/R_k$
    $G_{ij} = G_{ij} - 1/R_k$
    $G_{ji} = G_{ji} - 1/R_k$
    $t_i = t_i - e_k/R_k$
    $t_j = t_j + e_k/R_k$
•

Exerciţiul 3.7:
a) Implementaţi procedura `nodalRE_v1` ca o funcţie Matlab.
b) Scrieţi un script `main_nodal` cu următorul conţinut:

```
clear all;
[circuit] = citire_circuitRE_simplu();
[G,t] = nodalRE_v1(circuit)
```

Verificaţi că rezultatul este cel aşteptat.
c) Arătaţi că această procedură de asamblare are complexitatea $T = \mathrm{O}(12L)$.

Procedura de mai sus poate fi îmbunătăţită. În primul rând, calcule identice nu trebuie să se repete. O variantă îmbunătăţită este

procedură nodalRE_v2 (circuit, G, t)
....
pentru $k = 1, L$; parcurge laturi
    i =ni$_k$; nodul iniţial al laturii k
    j =nf$_k$; nodul final al laturii k
    Glat $= 1/R_k$
    Isc $= e_k$*Glat

$G_{ii} = G_{ii}$+Glat
$G_{jj} = G_{jj}$+Glat
$G_{ij} = G_{ij}$−Glat
$G_{ji} = G_{ji}$−Glat
$t_i = t_i$−Isc
$t_j = t_j$+Isc

•

Exerciţiul 3.8:

a) Implementaţi procedura `nodalRE_v2` ca o funcţie Matlab.

b) Adăugaţi scriptului `main_nodal` apelul acestei proceduri. Verificaţi corectitudinea rezultatului.

c) Care este complexitatea procedurii `nodalRE_v2`?

d) Contorizaţi timpul de calcul pentru cele două proceduri şi comparaţi.

O altă idee de îmbunătăţire este cea care se bazează pe faptul că matricea este simetrică şi putem asambla doar jumătate din ea. Procedura următoare asamblează triunghiul inferior.

`procedură` L_nodalRE_v3 (circuit, G, t)
....
`pentru` $k = 1, L$; parcurge laturi
    i =ni$_k$; nodul iniţial al laturii k
    j =nf$_k$; nodul final al laturii k
    Glat $= 1/R_k$
    Isc $= e_k$*Glat
    $G_{ii} = G_{ii}$+Glat
    $G_{jj} = G_{jj}$+Glat
    `dacă` $i > j$
        $G_{ij} = G_{ij}$−Glat
    `altfel` ; acesta este cazul $i < j$,
        $G_{ji} = G_{ji}$−Glat ; cazul $i = j$ nu este posibil pentru date de intrare corecte
    •
    $t_i = t_i$−Isc
    $t_j = t_j$+Isc

•

Exerciţiul 3.9:

a) Implementaţi procedura `L_nodalRE_v3` ca o funcţie Matlab.

b) Adăugaţi scriptului `main_nodal` apelul acestei proceduri. Verificaţi corectitudinea rezultatului.
c) Care este complexitatea procedurii `L_nodalRE_v3`? Care ar putea fi avantajul folosirii ei?

3.4 Etapa de rezolvare

În etapa de preprocesare asamblarea sistemului nu a fost făcută la dimensiunea $(N-1)\times(N-1)$ ci la dimensiunea $N\times N$, nodul de referintă nefiind tratat special. Acest lucru a făcut extrem de uşoară scrierea pseudocodului pentru asamblarea matricei. Pentru rezolvare însă, sistemul trebuie să fie de dimensiune $N-1$. Presupunând că nodul N este nodul de referinţă ($v_N = 0$), atunci matricea coeficienţilor se obţine eliminând ultima linie şi ultima coloană din matricea asamblată, iar vectorul termenilor liberi se obţine eliminând ultima componentă.

Exerciţiul 3.10:
Adăugaţi scriptului `main_nodal` următoarele comenzi matlab.

```
% rezolvare
N = circuit.N;
G(N,:) = [];
G(:,N) = [];
t(N) = [];
v = G\t;
v(N) = 0;
```

Comentaţi aceste comenzi.

Operatorul *backslash* în Matlab implementează într-un mod eficient metoda Gauss pentru rezolvarea sistemelor de ecuaţii algebrice liniare. Într-o temă viitoare vom analiza şi alte posibilităţi de rezolvare a sistemului de ecuaţii provenind din analiza circuitelor.

Vectorul soluţie întors reprezintă potenţialele a $N-1$ noduri. Potenţialul ultimului nod a fost considerat zero şi de aceea, pentru completitudine, este adaugată atribuirea $v_N = 0$ imediat după apelul procedurii de rezolvare.

3.5 Etapa de postprocesare

După rezolvarea sistemului putem calcula orice alte mărimi de interes: tensiuni, curenţi prin laturi, puteri. Următorul pseudocod ilustrează acest calcul

`procedură` postprocesare_crl (circuit, v)
; declaraţii
...
L =circuit.L
ni = circuit.ni
nf = circuit.nf
R =circuit.R
e = circuit.e
Pc = 0 ; puterea consumată
Pg = 0 ; puterea generată
`pentru` $k = 1, L$; parcurge laturi
$u = v_{\text{ni}_k} - v_{\text{nf}_k}$; tensiunea laturii
$c = (u + e_k)/R_k$; curentul prin latură
`scrie` "Latura" k "are tensiunea" u "şi curentul" c
Pc = Pc + $R_k c^2$; adaugă contribuţia laturii la puterea consumată
Pg = Pg + $e_k c$; adaugă contribuţia laturii la puterea generată
•
`scrie` Pc, Pg
`retur`

Este posibil ca, datorită erorilor de rotunjire, valorile Pc, Pg afişate să nu fie identice. Acest lucru se întâmplă mai ales dacă matricea sistemului este prost condiţionată numeric, caz ce poate apărea dacă valorile rezistenţelor sunt foarte diferite.

Exerciţiul 3.11:
a) Implementaţi în Matlab procedura de postprocesare;
b) Completaţi scriptul principal cu apelul ei;
c) Rulaţi programul şi verificaţi că bilanţul de puteri este verificat.

3.6 Analiza complexităţii. Optimizarea algorimului.

Până acum, ne-am concentrat asupra înţelegerii teoriei şi testării programului implementat pe un exemplu foarte simplu.

Ne propunem acum să analizăm complexitatea algoritmului şi să vedem ce îmbunătăţiri îi putem aduce.

Pentru aceasta, este util să avem o problemă de test "scalabilă", adică dependentă de un parametru care să reprezinte dimensiunea ei. Vom considera circuitul din fig. 3.5 şi ne propunem să determinăm conductanţa echivalentă între nodul marcat cu A şi masă. Dat fiind modul de asamblare al matricei conductanţelor nodale, am preferat să marcăm pe rezistoare valorile conductanţelor. Se poate verifica cu uşurinţă faptul că valoarea conductanţei echivalente între nodul A şi masă este $G_{\text{ech}} = 2G$.

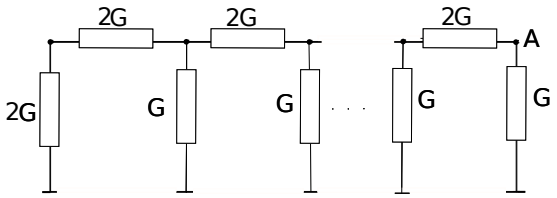

Figura 3.5: Circuit de test pentru analiza complexităţii. Pe rezistoare sunt marcate valorile conductanţelor.

Se observă că circuitul este pasiv, nu există surse. Pentru a determina conductanţa echivalentă între nodul A şi masă, trebuie să impunem o excitaţie în nodul A. De exemplu, dacă considerăm o sursă de curent J conectată între nodul A şi masă (fig. 3.6), atunci

$$J = G_{\text{ech}} U, \tag{3.18}$$

unde U este tensiunea între nodul A şi masă, adică potenţialul nodului A: $U = v_A$. Dacă se alege $J = 1\text{A}$, atunci

$$G_{\text{ech}} = 1/v_A. \tag{3.19}$$

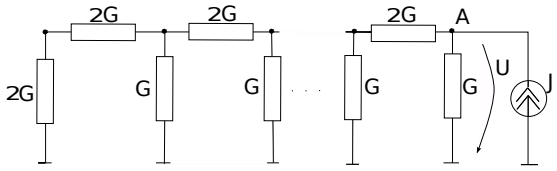

Figura 3.6: Excitarea nodului A în vederea determinării conductanţei echivalente.

Algoritmul ce va fi implementat poate fi descris pe scurt astfel:

1. Preprocesare:
   - Asamblează matricea conductanţelor nodale;
   - Asamblează vectorul injecţiilor de curent. Acesta va avea toate componentele 0, cu excepţia poziţiei corespunzătoare nodului A, unde valoarea va fi 1.

2. Rezolvare: se va rezolva sistemul de ecuaţii.

3. Postprocesare: conductanţa echivalentă va fi inversul potenţialului nodului A.

În vederea implementării etapei de preprocesare, trebuie să pregătim circuitul, numerotându-i nodurile şi laturile. Pentru un circuit cu o structură regulată, este util dacă numerotarea se face ordonat, de exemplu aşa cum este arătat în fig. 3.7: nodurile sunt numerotate de la stânga la dreapta, de la 1 la n, nodul de masă este numerotat ultimul, ca fiind nodul numărul $n+1$. Nodul A este deci nodul n. Laturile orizontale sunt numerotate cu numere pare, având drept index dublul nodului din stânga, iar laturile verticale sunt numerotate cu numere impare, având drept index dublul nodului de sus minus 1.

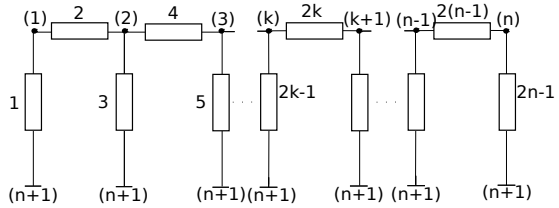

Figura 3.7: Numerotarea nodurilor şi laturilor.

Structura de date ce descrie acum circuitul o vom adapta acestei probleme. Nu vom mai defini tensiuni electromotoare pentru laturi deoarece toate sunt zero. De asemenea, în loc de rezistenţe vom lucra cu conductanţe. Astfel, structura de date ce descrie circuitul poate fi asamblată astfel:

`funcţie` citire_circuit_1d (n,valG)
; declaraţii
...
$L = 2n - 1$; numărul total de laturi
ground $= n + 1$; indexul nodului de masă
doivalG = 2*valG
`pentru` $k = 1, n - 1$
    ; latura impară, de index $2k - 1$
    idx $= 2k - 1$
    $\text{ni}_{\text{idx}} = k$
    $\text{nf}_{\text{idx}} =$ ground
    $G_{\text{idx}} =$ valG

    ; latura pară, de index $2k$
    idx $= 2k$

$\text{ni}_{\text{idx}} = k$
$\text{nf}_{\text{idx}} = k + 1$
$G_{\text{idx}} =$ doivalG

•

; corecţie conductanţa latura 1
$G_1 =$ doivalG
; ultima latură
$\text{ni}_L = n$
$\text{nf}_L =$ground
$G_L =$valG
; asamblare structură circuit
circuit.$N = n + 1$
circuit.$L = L$
circuit.ni = ni
circuit.nf = nf
circuit.$G = G$
`întoarce` *circuit*

Exerciţiul 3.12:
a) Implementaţi în Matlab funcţia `citire_circuit_1d`;
b) Verificaţi funcţia scrisă cu comanda `mlint`;
c) Observaţi necesarul de memorie pentru stocarea circuitului, executând următoarele comenzi în Matlab:

```
>> clear all;
>> circuit_10 = citire_circuit_1d(10,2.1);
>> circuit_100 = citire_circuit_1d(100,2.1);
>> circuit_1000 = citire_circuit_1d(1000,2.1);
>> circuit_10000 = citire_circuit_1d(10000,2.1);
>> circuit_100000 = citire_circuit_1d(100000,2.1);
>> whos
```

Considerând dimensiunea problemei ca fiind dată de numărul de noduri din circuit, dat de variabila n, cât este ordinul de complexitate din punct de vedere al necesarului de memorie pentru memorarea circuitului $\text{M}_{\text{circuit}} = \text{O}(?)$. Comparaţi estimarea teoretică cu cea experimentală.

Vom implementa acum asamblarea matricei conductanţelor nodale, modificând extrem de puţin funcţia `nodalRE_v2` implementată pentru exemplul simplu discutat anterior.

```
procedură nodal_circuit_1d_v1 (circuit,G,t)
; parametri de intrare - circuit
; parametri de ieşire - G, t
N = circuit.N
L = circuit.L
ni = circuit.ni
nf = circuit.nf
Glat = circuit.G

G = zeros(N,N)                  ; anulează componentele matricei G şi
t = zeros(N,1)                  ; a vectorului termenilor liberi t

pentru k = 1,L
      i = ni(k)
      j = nf(k)
      Gk = Glat(k)
      G(i,i) = G(i,i) + Gk
      G(j,j) = G(j,j) + Gk
      G(i,j) = G(i,j) - Gk
      G(j,i) = G(j,i) - Gk
•
t(N-1) = 1
retur
```

Exerciţiul 3.13:

a) Implementaţi procedura de mai sus ca o funcţie Matlab
`[G,t] = function nodal_circuit_1d_v1 (circuit)`. Verificaţi-o cu `mlint`.
b) Scrieţi următorul script Matlab pentru a testa corectitudinea programului.

```
clear all;
n = 10;
valG = 2.1;
% preprocesare
[circuit] = citire_circuit_1d(n,valG);
[G,t] = nodal_circuit_1d_v1(circuit);
% rezolvare
G(n+1,:) = [];
G(:,n+1) = [];
```

```
t(n+1) = [];
v = G\t;
% postprocesare
disp(sprintf('n = %d, G ech = %e',n,1/v(n)));
```

Verificaţi că rezultatul este cel aşteptat, pentru diferite dimensiuni ale problemei (de exemplu $n = 10, 20, 100$).

Exerciţiul 3.14:
Observaţi structura matricei conductanţelor nodale după asamblare şi înainte de rezolvarea propriu-zisă. Pentru aceasta adăugaţi

```
figure(1);
spy(G);
```

imediat după asamblare şi

```
figure(2);
spy(G);
```

chiar înainte de rezolvarea propriu-zisă. Observaţi că matricea sistemului de rezolvat pentru acestă problemă este tridiagonală.

Exerciţiul 3.15:
a) Estimaţi ordinul de complexitate din punct de vedere al necesarului de memorie. Puteţi experimenta numeric şi cu ajutorul următoarelor comenzi executate în consola Matlab:

```
>> clear all;
>> [circuit_10] = citire_circuit_1d(10,2.1);
>> [G10,t10] = nodal_circuit_1d_v1(circuit_10);
>> [circuit_100] = citire_circuit_1d(100,2.1);
>> [G100,t100] = nodal_circuit_1d_v1(circuit_100);
>> whos
```

c) Estimaţi teoretic (nu încercaţi experimental!) necesarul de memorie pentru stocarea matricei conductanţelor nodale în cazul $n = 10000$.

Matricea conductanţelor nodale este o matrice care are foarte multe elemente nule. În cazul problemei studiate, densitatea matricei, definită ca numărul de elemente nenule raportat la numărul total de elemente este aproximativ

$$d_G = \frac{n_{nz}}{n^2} \approx \frac{3n}{n^2} = \frac{3}{n}. \tag{3.20}$$

Exerciţiul 3.16:

a) Cât este densitatea matricei coeficienţilor sistemului în cazul $n = 10$? Estimaţi teoretic cu formula de mai sus şi numeric folosind funcţia Matlab `nnz`. Explicaţi diferenţa.
b) Estimaţi teoretic densitatea matricei pentru $n = 10000$.

Implementarea făcută până acum a folosit numai matrice pline. Funcţia Matlab `zeros` crează matrice pline.

Exerciţiul 3.17:

Observaţi diferenţa dintre funcţiile `zeros` şi `sparse` executând următoarele comenzi:

```
>> clear all;
>> a = zeros(10,10)
>> b = sparse(10,10)
>> whos
```

Vom rescrie acum procedura de asamblare având grijă ca matricile să fie definite rar.

Exerciţiul 3.18:

a) Copiaţi fişierul `nodal_circuit_1d_v1.m` în `nodal_circuit_1d_v2.m`. Pentru aceasta, puteţi da comanda Matlab:

```
>> !copy nodal_circuit_1d_v1.m nodal_circuit_1d_v2.m
```

b) Editaţi fişierul `nodal_circuit_1d_v2.m`.

```
>> edit nodal_circuit_1d_v2.m
```

Corectaţi numele funcţiei şi înlocuiţi funcţia `zeros` cu `sparse`.

Exerciţiul 3.19:

a) Comparaţi necesarul de memorie pentru stocarea matricei conductanţelor nodale în cele două implementări.

```
>> circuit_100 = citire_circuit_1d(100,2.1);
>> [G_plin,t_plin] = nodal_circuit_1d_v1(circuit_100);
>> [G_rar,t_rar] = nodal_circuit_1d_v2(circuit_100);
>> whos
```

Exerciţiul 3.20:

Estimaţi necesarul de memorie pentru stocarea matricei conductanţelor nodale în cazul folosirii structurilor rare. Puteţi experimenta cu următoarele comenzi.

```
>> clear all;
>> circuit_100 = citire_circuit_1d(100,2.1);
>> circuit_1000 = citire_circuit_1d(1000,2.1);
>> circuit_10000 = citire_circuit_1d(10000,2.1);
>> [G_100,t_100] = nodal_circuit_1d_v2(circuit_100);
>> [G_1000,t_1000] = nodal_circuit_1d_v2(circuit_1000);
>> [G_10000,t_10000] = nodal_circuit_1d_v2(circuit_10000);
>> whos
```

Având în vedere structura regulată a circuitului, putem evita memorarea explicită a circuitului şi "dizolva" informaţiile legate de topologia circuitului în procedura nodal.

procedură nodal_circuit_1d_v3(n,valG,G,t)
; parametri de intrare - n, valG
; parametri de ieşire - G,t
$N = n + 1$; numărul total de noduri
ground $= N$; indexul nodului de masă
$G =$ sparse(N, N) ; se aloca memorie pentru o matrice rară de dimensiune N
$t =$ sparse$(N, 1)$
doivalG = 2*valG
pentru $k = 1 : n - 1$
    ; latura impară, de index $2k - 1$
    $i = k$
    $j =$ground
    $G_{ii} = G_{ii}$+valG
    $G_{jj} = G_{jj}$+valG
    $G_{ij} = G_{ij}$−valG
    $G_{ji} = G_{ji}$−valG
    ; latura pară, de index $2k$
    $i = k$
    $j = k + 1$
    $G_{ii} = G_{ii}$+doivalG
    $G_{jj} = G_{jj}$+doivalG
    $G_{ij} = G_{ij}$−doivalG
    $G_{ji} = G_{ji}$−doivalG
•
; corecţie conductanţa latura 1
$G_{11} = G_{11}$+valG

; ultima latură
$i = n$
j =ground
$G_{ii} = G_{ii}$+valG
$G_{jj} = G_{jj}$+valG
$G_{ij} = G_{ij}$−valG
$G_{ji} = G_{ji}$−valG
$t_{N-1} = 1$
retur

Exerciţiul 3.21:
a) Observaţi diferenţa între modul de asamblare al sistemului în cele două variante.
b) Implementaţi această variantă în Matlab şi testaţi-i corectitudinea.
c) Comparaţi necesarul de memorie şi timpul de calcul necesar etapei de preprocesare în variantele 2 şi 3 pentru $n = 10000$. Pentru aceasta puteţi folosi următorul script:

```
clear all;
n = 10000;
valG = 2.1;
% preprocesare v2
tic;
[circuit] = citire_circuit_1d(n,valG);
[G2,t2] = nodal_circuit_1d_v2(circuit);
t = toc;
disp(sprintf('timp preproc v2 = %e',t));
whos

clear G2 t2 circuit;
% preprocesare v3
tic
[G3,t3] = nodal_circuit_1d_v3(n,valG);
t = toc;
disp(sprintf('timp preproc v3 = %e',t));
whos
```

O altă variantă de implementare a procedurii nodal poate folosi asamblarea matricei incidenţelor laturi-noduri şi calculul matricei conductanţelor nodale în acord cu formula (3.10). Pseudocodul acestei proceduri este următorul.

procedură nodal_circuit_1d_v4(n,valG,G,t)
; parametri de intrare - n, valG
; parametri de ieşire - G,t
$N = n + 1$; numărul total de noduri
$L = 2n - 1$; numărul total de laturi
ground $= N$; indexul nodului de masă
; alocare spaţie de memorie pentru:
$A =$ sparse(N, L) ; - matricea incidenţelor laturi-noduri,
Glat $=$ sparse$(L, 1)$; - vectorul conductanţelor laturilor,
$t =$ sparse$(N, 1)$; - vectorul injecţiilor de curent.
doivalG = 2*valG;

pentru $k = 1, n - 1$
; latura impară, de index $2k - 1$
idx $= 2k - 1$
$i = k$; nodul iniţial al laturii
$j =$ground ; nodul final al laturii
$A_{i,\text{idx}} = 1$
$A_{j,\text{idx}} = -1$
$\text{Glat}_{\text{idx}} =$valG

; latura pară, de index $2k$
idx $= 2k$
$i = k$
$j = k + 1$
$A_{i,\text{idx}} = 1$
$A_{j,\text{idx}} = -1$
$\text{Glat}_{\text{idx}} =$doivalG
•
; corecţie conductanţă latura 1
$\text{Glat}_1 =$ doivalG
; ultima latură
$i = n$
$j =$ground
$A_{iL} = 1$
$A_{jL} = -1$
$\text{Glat}_L =$ valG

$G = A{*}\text{diag}(\text{Glat}){*}A^T$; G calculat ca produs de matrice rare

$t_{N-1} = 1$
retur

Important: în algoritmul de mai sus instrucţiunea în care se calculează matricea conductanlor nodale foloseşte operaţii cu matrice rare.

Exerciţiul 3.22:
a) Observaţi rezultatul comenzii `diag` în Matlab. Comentaţi următoarele instrucţiuni

```
>> v = [1 2 3];
>> M = diag(v)
>> Ms = diag(sparse(v))
```

Ce efect (intern) în Matlab ar avea comanda `sparse(diag(v))` ?
b) Implementaţi variantă v4 a procedurii nodal în Matlab şi testaţi-i corectitudinea.
c) Comparaţi necesarul de memorie necesar etapei de preprocesare în variantele 3 şi 4 pentru diferite valori ale lui n. Pentru aceasta puteţi folosi următorul script:

```
clear all;
nn = linspace(10000,20000,10);
valG = 2.1;
for i = 1:length(nn)
    n = floor(nn(i));

    % preprocesare v3
    tic;
    [G3,t3] = nodal_circuit_1d_v3(n,valG);
    t = toc;
    disp(sprintf('n = %d, timp preproc v3 = %e',n,t));

    % preprocesare v4
    tic
    [G4,t4] = nodal_circuit_1d_v4(n,valG);
    t = toc;
    disp(sprintf('n = %d, timp preproc v4 = %e',n,t));
end
```

c) Modificaţi acest script pentru a obţine grafic această dependenţă (fig.3.8).

În Matlab, comanda `sparse` dată doar cu două argumente care reprezintă dimensiunea matricei face o alocare aproximativă a spaţiului de memorie necesar. În cazul în care se cunoaşte exact câte elemente nenule are matricea rară, este mult mai eficient să se construiască matricea pe coordonate şi apoi să se folosească comanda `sparse` cu 5 argumente.

Exerciţiul 3.23:

a) Copiaţi funcţia `nodal_circuit_1d_v4` în `nodal_circuit_1d_v5` şi deschideţi fişierul în editorul Matlab. Corectaţi numele funcţiei;
b) Înlocuiţi alocarea de memorie pentru matricea de incidenţe cu alocările de memorie necesare memorării ei pe coordonate:

```
%A = sparse(N,L); % matricea inciden'telor laturi-noduri
no_nnz = 2*L;
r_idx = zeros(no_nnz,1);
c_idx = zeros(no_nnz,1);
val = zeros(no_nnz,1);
m = 0;
```

c) Înlocuiţi toate atribuirile pentru componentele matricei incidenţelor cu atribuiri pentru cei trei vectori în care se memorează matricea, ca de exemplu:

```
%A(i,idx) = 1;
m = m + 1;
r_idx(m) = i;
c_idx(m) = idx;
val(m) = 1;
```

Trebuie să faceî şase astfel de modificări.
d) Asamblaţi matricea astfel

```
A = sparse(r_idx,c_idx,val,N,L);
```

e) Testaţi corectitudinea noii funcţii.
f) Comparaţi timpul necesar preprocesării, completând scripturile pe care deja le-aţi folosit în exerciţiile anterioare. Trebuie să obţineţi un grafic de tipul celui din fig.3.9.

În concluzie, această temă ilustrează următoarele:

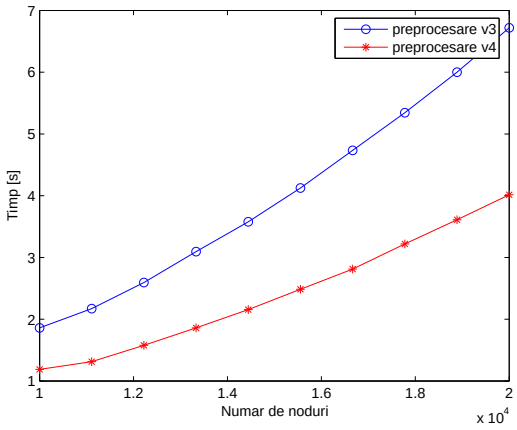

Figura 3.8: Comparaţie între două imple-

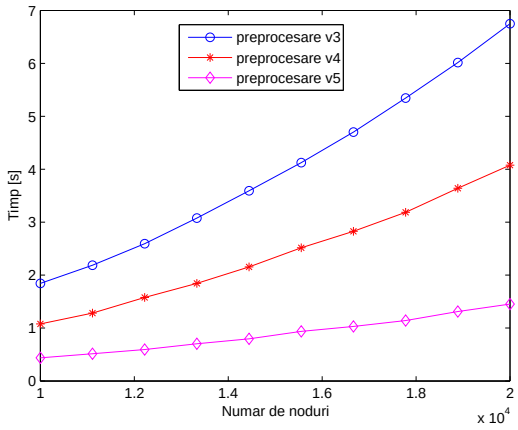

Figura 3.9: Comparaţie între trei imple-

mentări ale etapei de preprocesare. Un astfel de grafic trebuie să obţineţi la exerciţiul 3.22.

mentări ale etapei de preprocesare. Un astfel de grafic trebuie să obţineţi la exerciţiul 3.23.

1. Etapele conceperii unui algoritm dedicat rezolvării unei probleme sunt:
   - înţelegerea perfectă a bazei teoretice a algoritmului;
   - alegerea structurilor de date pentru datele de intrare şi de ieşire;
   - implementarea algoritmului într-un program;
   - testarea şi depanarea programului pentru o problemă simplă, cu rezultat cunoscut;
   - rularea programului pe probleme de dimensiuni crescute, în vederea stabilirii complexităţii sale şi măsurarea performanţei;
   - optimizarea codului.
2. Dacă implementaţi algoritmi în Matlab, este mai bine să profitaţi, acolo unde este posibil, de funcţiile Matlab care folosesc operaţii cu vectori şi matrice. Chiar şi în cazul structurilor pline, funcţiile sunt optimizate (aşa cum aţi vazut într-un capitol anterior). În cazul în care structurile sunt rare, folosirea unor proceduri adaptate acestor structuri este foarte uşoară. Practic, nu este nevoie să scrieţi cod care să lucreze cu formatul CCS. Acest lucru este deja implementat în Matlab.

Capitolul 4

Interpolarea polinomială

Conceperea unor algoritmi pentru probleme de inginerie electrică mai complicate ca de exemplu analiza unor circuite în regim tranzitoriu, sau a unor probleme de câmp electromagnetic, necesită înţelegerea algoritmilor numerici pentru interpolarea funcţiilor, derivarea funcţiilor şi integrarea lor. Exerciţiile din această temă ilustrează interpolarea funcţilor.

Problema interpolării are ca date un tabel de valori cunoscute care reprezintă date şi rezultate ale unei probleme şi are ca scop estimarea rezultatelor pentru un set de alte date, care nu se găsesc în tabelul iniţial de valori. Ideea metodei este aceea de a găsi o funcţie aproximativă care poate fi evaluată cu uşurinţă şi care satisface condiţiile impuse de valorile din tabel. Estimarea rezultatelor corespunzătoare unor date noi se face prin evaluarea acestei funcţii aproximative.

4.1 Cazul 1D

4.1.1 Formularea problemei

Cazul cel mai simplu al interpolării este cel numit *unidimensional* (1D) în care atât datele cât şi rezultatele sunt scalari. Cazul în care datele sunt scalari iar rezultatele sunt vectori se reduce tot la acest caz, interpolarea 1D realizându-se pe fiecare componentă în parte.

Problema interpolării 1D se formulează, în principiu, astfel.

- Date: tabelul de valori format din n perechi (x_k, y_k) pe care le vom numi "puncte",

unde x_k sunt datele iar y_k sunt rezultatele:

x	x_0	x_1	$\cdots$	x_n
y	y_0	y_1	$\cdots$	y_n

- Se cere: să se estimeze valoarea rezultatelor **yi** pentru alte date **xi** care nu se regăsesc în tabelul de valori.

Pentru a simplifica scrierea în cele ce urmează vom presupune că în acest tabel de valori, vectorul **x** este ordonat crescător.

Această formulare nu este însă riguroasă din punct de vedere matematic. Pentru o problemă bine formulată matematic, soluţia există şi este unică. De aceea, în acord cu teoria interpolării, vom presupune că:

- valorile vectorului **x** sunt distincte;
- vom căuta o funcţie aproximativă g care să satisfacă *condiţiile de interpolare*:

$$g(x_k) = y_k, \quad k = 0, \ldots, n, \tag{4.1}$$

- funcţia g se va căuta în spaţiul polinoamelor generalizate, adică g se caută de forma

$$g(x) = \sum_{k=0}^{n} c_k \varphi_k(x), \tag{4.2}$$

unde $\varphi_k(x)$ sunt $n+1$ funcţii de bază, liniar independente, definite explicit.

Cu alte cuvinte, într-o reprezentare grafică, funcţia de intepolare g trece prin punctele ce reprezintă tabelul de valori dat.

4.1.2 Interpolarea globală

În interpolarea polinomială globală, fiecare din funcţiile de bază φ_k din relaţia (4.2) sunt definite explicit printr-o singură expresie compactă. De exemplu, ele pot fi polinoame algebrice ca în

- metoda clasică:

$$\begin{aligned} \varphi_0(x) &= 1, \\ \varphi_1(x) &= x, \\ &\vdots \\ \varphi_k(x) &= x^k, \\ &\vdots \\ \varphi_n(x) &= x^n, \end{aligned} \tag{4.3}$$

- metoda Lagrange:

$$\varphi_k(x) = l_k(x), \quad k = 1, \ldots, n, \tag{4.4}$$

  unde cu l sunt notate polinoamele Lagrange,

- metoda Newton:

$$\begin{aligned} \varphi_0(x) &= 1, \\ \varphi_1(x) &= (x - x_0), \\ &\vdots \\ \varphi_k(x) &= (x - x_0)(x - x_1) \cdots (x - x_{k-1}), \\ &\vdots \\ \varphi_n(x) &= (x - x_0)(x - x_1) \cdots\cdots (x - x_{n-1}). \end{aligned} \tag{4.5}$$

Din punct de vedere matematic, toate aceste trei metode sunt echivalente, ele conduc la acelaşi polinom algebric de grad n care trece prin cele $n+1$ puncte din tabel. Din punct de vedere numeric, metoda clasică este cea mai slabă datorită proastei sale condiţionări numerice, iar metoda Newton cea mai bună datorită bunei condiţionări numerice, posibilităţii de a estima eroarea de interpolare şi faptului că, la adaugărea unor puncte noi în tabel, efortul de calcul anterior nu se pierde.

Metoda Lagrange furnizează însă o metodă de calcul rapid al polinomului de interpolare deoarece, după impunerea condiţiilor de interpolare rezultă că valorile coeficienţilor c_k sunt exact valorile y_k din tabelul de valori: $c_k = y_k$. Reamintim expresia *polinomului Lagrange*, asociat unei diviziuni în n intervale definite prin punctele

$$x_0 < x_1 < \ldots < x_n$$

şi unui punct k:

$$l_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i} \tag{4.6}$$

Expresia *polinomului de interpolare* este deci

$$g(x) = \sum_{k=0}^{n} y_k l_k(x). \tag{4.7}$$

Exerciţiul 4.1:

Fie tabelul de valori

x	0	1	3
y	4	3	7

a) Care sunt cele trei polinoame Lagrange asociate diviziunii pe x?

b) Calculaţi polinomul de interpolare globală cu formula (4.7).
c) Verificaţi că polinomul obţinut la punctul anterior satisface condiţiile de interpolare.

Exerciţiul 4.2:

a) Care este legătura dintre numărul de puncte din tabelul de valori şi gradul polinoamelor Lagrange?
b) Ce valori ia un polinom Lagrange în nodurile reţelei de interpolare $\varphi_k(x_j) =$?
c) Care este legătura între numărul de puncte din tabelul de valori şi gradul polinomului de interpolare?
d) Scrieţi expresia polinoamelor Lagrange asociate unei diviziuni formate din două puncte.

Ne propunem să scriem o funcţie Matlab care să implementeze evaluarea polinomului Lagrange asociat unei diviziuni. În Matlab, vectorii sunt indexaţi de la 1. De aceea, vom considera tabelul de valori având n puncte notate

x	x_1	x_2	$\cdots$	x_n
y	y_1	y_2	$\cdots$	y_n

Acestui tabel îi sunt asociate n funcţii Lagrange de grad $n-1$

$$l_k(x) = \prod_{\substack{i=1 \\ i\neq k}}^{n} \frac{x - x_i}{x_k - x_i}, \quad k = 1, \ldots, n, \tag{4.8}$$

iar polinomul de intepolare, de grad $n-1$ este

$$g(x) = \sum_{k=1}^{n} y_k l_k(x). \tag{4.9}$$

Exerciţiul 4.3:

a) Să se scrie o funcţie Matlab având argumentele definite ca mai jos.

```
function [yi] = lagrange(x,k,xi)
% evalueaza functia Lagrange asociata nodului nr k al unei diviziuni x
% in punctele xi
% date de intrare:
%   x - vector cu n componente, presupus sortat crescator
%   k - indexul nodului, poate fi intre 1 si n
%   xi - vector cu oricate componente, in care va fi evaluat polinomul
%        Lagrange
% date de iesire
%   yi - valorile polinomului l_k(x) evaluat in punctele xi
```

```
n = length(x); % numarul de puncte din tabel
if or(k<1,k>n)
    error('al doilea arg trebuie sa fie un intreg cuprins intre 1 si %d', n);
end

m = length(xi); % numarul de puncte in care se va evalua polinomul Lagrange
yi = zeros(m,1);

%.........................completati.....................
```

b) Verificaţi codul scris cu comanda `mlint`.

Exerciţiul 4.4:

a) Scrieţi un script Matlab `main_lagrange.m` cu următorul conţinut.

```
clear all;
x = [0 1 3];
xi = linspace(0,3,100);
yi_1 = lagrange(x,1,xi);
yi_2 = lagrange(x,2,xi);
yi_3 = lagrange(x,3,xi);
figure(1);
plot(xi,yi_1,'b-','LineWidth',3);
leg{1} = 'l_1(x)';
hold on;
plot(xi,yi_2,'r--','LineWidth',3);
leg{2} = 'l_2(x)';
hold on;
plot(xi,yi_3,'k:','LineWidth',3);
leg{3} = 'l_3(x)';
hold on;
grid on;
legend(leg);
title('Polinoamele Lagrange asociate diviziunii [0 1 3]');
xlabel('x');
ylabel('l_k(x)');
```

Verificaţi că rularea lui furnizează figura 4.1.

b) Copiaţi acest script într-un alt fişier. Modificaţi-l astfel încât să obţineţi figura 4.2.

Care este acum diviziunea folosită pe x?

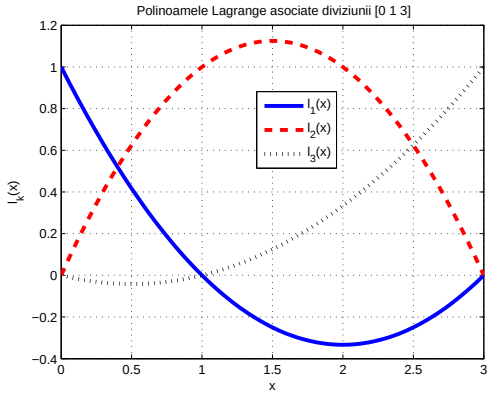

Figura 4.1: Polinoamele Lagrange asociate diviziunii [0 1 3].

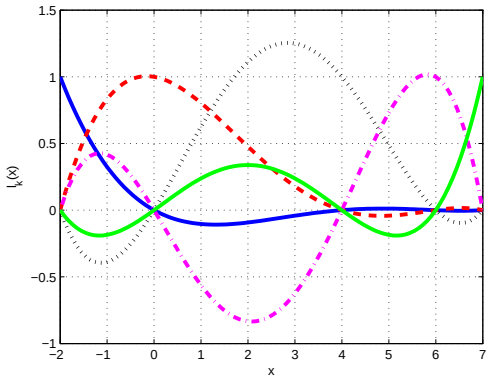

Figura 4.2: Pentru exerciţiul 4.4: Ce reprezintă această figură?

Până acum aţi văzut cum arată funcţiile de bază ale interpolării Lagrange. Să construim acum polinomul de interpolare.

Exerciţiul 4.5:

a) Scrieţi o funcţie Matlab `my_interp1.m` care să construiască polinomul de interpolare, având următoarele argumente:

```
function yi = my_interp1(x,y,xi,method)
% evalueaza polinomul de interpolare
% date de intrare:
%   x - vector cu n componente, presupus sortat crescator
%   y - valorile corespunzatoare vectorului x
%   xi - vector cu oricate componente, in care va fi evaluat polinomul
%        de interpolare
%   method - sir de caractere care determina tipul de intepolare;
%        'lagrange' - intep. globala prin evaluarea polinomului Lagrange
% date de iesire
%   yi - valorile polinomului de interpolare evaluat in punctele xi

n = length(x);
m = length(xi);
yi = zeros(m,1);

switch method
```

```
    case 'lagrange'
%.................. completati ..............................
    otherwise
        error('unknown method');
end
```

b) Verificaţi codul scris cu comanda `mlint`.

Exerciţiul 4.6:

a) Verificaţi corectitudinea funcţiei scrise mai sus, rulând următorul script Matlab.

```
clear all;
x = [0 1 3];
y = [4 3 7];
xi = linspace(0,3,50);
yi = my_interp1(x,y,xi,'lagrange');
figure(1);
plot(x,y,'bo','MarkerSize',8,'LineWidth',2);
hold on;
plot(xi,yi,'r-','LineWidth',2);
grid on;
```

Prin rularea acestui script trebuie să obţineţi rezultatul din fig.4.3.

b) Copiaţi acest script în alt fişier. Modificaţi-l astfel încât să obţineţi rezultatul din fig. 4.4. Ce grad are polinomul de interpolare?

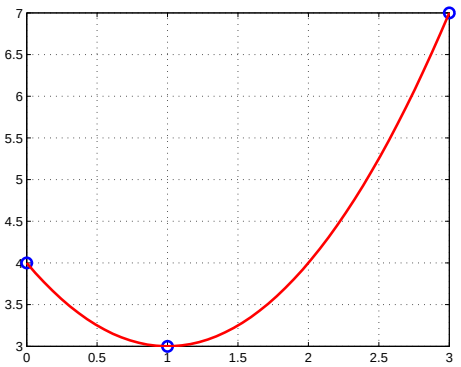

Figura 4.3: Polinomul de interpolare pentru tabelul $\mathbf{x} = [0\ 1\ 3]$, $\mathbf{y} = [4\ 3\ 7]$.

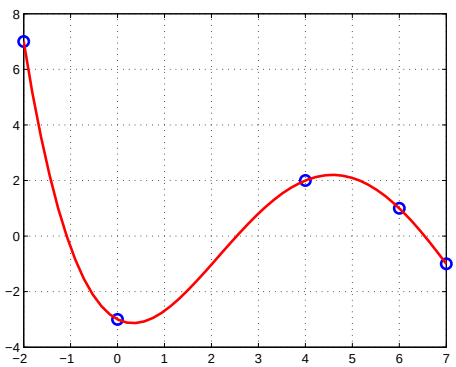

Figura 4.4: Pentru exerciţiul 4.6: Ce grad are polinomul de interpolare?

Interpolarea polinomială globală are dezavantajul că poate fi afectată de efectul Runge, care constă în apariţia unor oscilaţii ale polinomului de interpolare la capetele intervalului. Acest fenomen a fost pus în evidenţă pentru prima dată de Runge, pentru funcţia $f : [-5, 5] \to \mathbb{R}$, $f(x) = 1/(1 + x^2)$. Exerciţiul următor ilustrează acest fenomen.

Exerciţiul 4.7:

a) Scrieţi un script Matlab cu următorul conţinut

```
clear all;
N = 11;
x = linspace(-5,5,N);
y = 1./(1+x.*x);
xi = linspace(-5,5,100);
yi = my_interp1(x,y,xi,'lagrange');
yRunge = 1./(1+xi.*xi);
figure(1);
plot(xi,yRunge,'k-','LineWidth',2);
leg{1} = 'Runge(x)';
hold on;
plot(x,y,'bo','MarkerSize',8,'LineWidth',2);
leg{2} = 'Puncte folosite pentru interpolare';
hold on;
plot(xi,yi,'r-','LineWidth',2);
leg{3} = 'Polinomul de interpolare';
grid on;
legend(leg);
```

Comentaţi rezultatul obţinut (fig. 4.5). Experimentaţi pentru alte grade ale polinomului de interpolare.

b) Încercaţi să daţi o explicaţie acestui fenomen.

c) Acest fenomen ar putea fi evitat dacă punctele diviziunii sunt dispuse conform interpolării Cebyshev. Descrieţi pe scurt ideea interpolării Cebyshev. Care este dezavantajul ei?

d) Adăugaţi scriptului comenzile care permit trasarea graficului erorii absolute şi calculul normei Cebyshev al acestei erori.

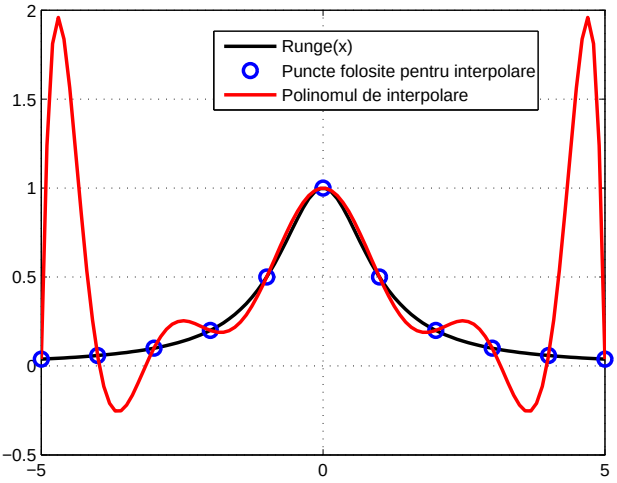

Figura 4.5: Fenomenul Runge.

4.1.3 Interpolarea pe porţiuni

Interpolarea prezentată până acum a folosit un singur polinom care să aproximeze funcţia pe **întreg** domeniul dorit. Prin creşterea fineţei grilei de discretizare a domeniului, nu este îmbunătăţită eroarea de aproximare, ba chiar dimpotrivă, pot apărea fenomene nedorite chiar în cazul în care funcţia de aproximat are o alură cuminte, ca funcţia lui Runge. Dacă funcţia de aproximat are un relief foarte accidentat, atunci de asemenea interpolările polinomiale globale nu sunt precise.

Din aceste motive, mult mai necesare pentru rezolvarea numerică a problemelor de inginerie electrică sunt interpolările pe porţiuni. Ele corespund folosirii unor funcţii de bază $\varphi_k(x)$ care nu mai sunt definite printr-o expresie compactă pe întreg domeniul funcţiei, ci prin "expresii cu acoladă".

De exemplu, polinoamele Lagrange definite pe porţiuni satisfac de asemenea condiţiile $l_k(x_k) = 1$ şi $l_k(x_j) = 0$ pentru $j \neq k$, dar variaţia între nodurile grilei de discretizare este liniară:

$$l_1(x) = \begin{cases} \frac{x-x_2}{x_1-x_2} & \text{dacă} \quad x \in [x_1, x_2] \\ 0 & \text{dacă} \quad x \in (x_2, x_n] \end{cases} \tag{4.10}$$

$$l_k(x) = \begin{cases} \frac{x-x_{k-1}}{x_k-x_{k-1}} & \text{dacă} \quad x \in [x_{k-1}, x_k) \\ \frac{x-x_{k+1}}{x_k-x_{k+1}} & \text{dacă} \quad x \in [x_k, x_{k+1}] \\ 0 & \text{dacă} \quad x \in [x_1, x_{k-1}) \cup (x_{k+1}, x_n] \end{cases} \qquad k = 2, n-1 \tag{4.11}$$

$$l_n(x) = \begin{cases} \frac{x-x_{n-1}}{x_n-x_{n-1}} & \text{dacă} \quad x \in [x_{n-1}, x_n] \\ 0 & \text{dacă} \quad x \in [x_1, x_{n-1}) \end{cases} \tag{4.12}$$

Exerciţiul 4.8:

a) Scrieţi o funcţie Matlab care să evalueze polinoamele Lagrange pe porţiuni, de tipul

```
function [yi] = lagrangep(x,k,xi)
% evalueaza functia lagrange pe portiuni asociata nodului nr k al
% unei diviziuni x in punctele xi
% date de intrare:
%   x - vector cu n componente, presupus sortat crescator
%   k - indexul nodului, poate fi intre 1 si n
%   xi - vector cu oricate componente, in care va fi evaluat polinomul
%        Lagrange
% date de iesire
%   yi - valorile polinomului l_k(x) evaluat in punctele xi

n = length(x); % num'arul de puncte din tabel
if or(k<1,k>n)
    error('al doilea arg trebuie sa fie un intreg cuprins intre 1 si %d', n);
end

m = length(xi); % numarul de puncte in care se va evalua polinomul Lagrange
yi = zeros(m,1);

if k == 1
    for j = 1:m
        xcrt = xi(j);
        if and((x(1) <= xcrt),xcrt<= x(2))
            yi(j) = (xcrt - x(2))/(x(1)-x(2));
        else
            yi(j) = 0;
        end
    end
elseif k == n
% .................. completati .......................
else
    for j = 1:m
        xcrt = xi(j);
        if and((x(k-1) <= xcrt),xcrt<= x(k))
            yi(j) = (xcrt - x(k-1))/(x(k)-x(k-1));
```

```
        elseif and((x(k) < xcrt),xcrt<= x(k+1))
            yi(j) = (xcrt - x(k+1))/(x(k)-x(k+1));
        else
            yi(j) = 0;
        end
    end
end
```

b) Verificaţi codul scris cu comanda `mlint`.
c) Modificaţi scriptul `main_lagrange.m`, apelând funcţia `lagrangep` în loc de `lagrange`. Prin rularea lui trebuie să obţineţi graficul din fig. 4.6.

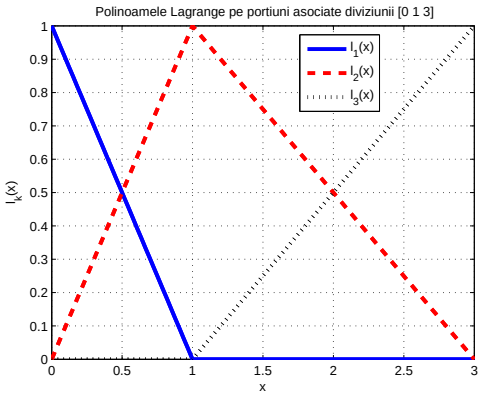

Figura 4.6: Funcţii Lagrange pe porţiuni.

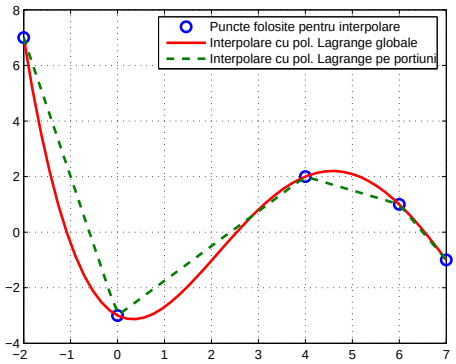

Figura 4.7: Interpolare globală şi interpolare pe porţiuni.

Exerciţiul 4.9:
a) Completaţi funcţia `my_interp1` cu cazul "lagrangep".
b) Adaugaţi la scriptul `main_interp1` cazul interpolării pe porţiuni. Rularea lui trebuie să vă conducă la un rezultat similar cu cel din fig. 4.7.

Interpolarea obţinută cu ajutorul polinoamelor Lagrange pe porţiuni este cunoscută sub numele de *interpolare liniară pe porţiuni*. Evaluarea polinomului de interpolare liniară pe porţiuni poate fi implementată cu un algoritm mai simplu, bazat pe căutarea intervalului în care se află punctul în care se doreşte să se facă evaluarea polinomului:

```
k = cauta(n,x,xcrt);
ycrt = y(k) + (xcrt - x(k))*(y(k+1) - y(k))/(x(k+1) - x(k));
```

Exerciţiul 4.10:
a) Copiaţi o funcţie de căutare binară implementată anterior, într-un fişier `cauta.m`.

Modificaţi numele funcţiei şi modificaţi sintaxa acesteia astfel încât să poată fi realizată extrapolarea funcţiei (în afara domeniului de definiţie funcţia se va aproxima prin prelungirea primului şi, respectiv, ultimului segment).
b) Adaugaţi la funcţia `my_interp1` cazul `lpp` al interpolării pe porţiuni implementată prin căutare binară. Verificaţi corectitudinea codului implementat pe un exemplu simplu.

Interpolarea liniară pe porţiuni, este o metodă de interpolare în care nu apare efectul Runge şi poate fi aplicată şi în cazul în care funcţia de interpolat nu este cunoscută prin cod. Ea are dezavantajul că polinomul de interpolare generat nu este neted, are "colţuri" în nodurile reţelei de interpolare. Matematic spus, funcţia obţinută nu este derivabilă în nodurile grilei de discretizare.

Ne dorim însă uneori ca polinomul de intepolare să fie nu numai continuu ci şi derivabil. Soluţia se numeşte *interpolare Hermite*. Problema interpolării Hermite are ca date nu numai valorile funcţiei într-o mulţime discretă de puncte ci şi a derivatelor sale:

$\mathbf{x}$	x_1	x_2	$\cdots$	x_n
$\mathbf{y}$	y_1	y_2	$\cdots$	y_n
$\mathbf{y}'$	y_1'	y_2'	$\cdots$	y_n'

Pe fiecare interval $[x_k, x_{k+1}]$ condiţiile de interpolare impun acum atât valorile funcţiei cât şi valorile derivatei. Impunând patru condiţii de interpolare pe fiecare interval, rezultă că pentru fiecare astfel de interval vor exista patru coeficienţi ai polinomului local de interpolare, care va avea, în consecinţă, gradul trei. Este util dacă acest polinom se scrie sub forma

$$g(x) = c_{0k} + c_{1k}(x - x_k) + c_{2k}(x - x_k)^2 + c_{3k}(x - x_k)^3, \quad x \in [x_k, x_{k+1}], \quad k = 1, \ldots, n-1. \tag{4.13}$$

Impunerea celor patru *condiţii de interpolare Hermite*:

$$\begin{aligned} g(x_k) &= y_k, & g(x_{k+1}) &= y_{k+1}, \\ g'(x_k) &= y_k', & g'(x_{k+1}) &= y_{k+1}', \end{aligned} \tag{4.14}$$

permite calculul coeficienţilor de interpolare pentru fiecare polinom de interpolare pe porţiuni:

$$c_{0k} = y_k, \tag{4.15}$$

$$c_{1k} = y_k', \tag{4.16}$$

$$c_{2k} = \frac{1}{(x_{k+1} - x_k)^2} \left[3(y_{k+1} - y_k) - (x_{k+1} - x_k)(2y_k' + y_{k+1}') \right], \tag{4.17}$$

$$c_{3k} = \frac{1}{(x_{k+1} - x_k)^2} \left[(y_{k+1}' + y_k') - \frac{2}{x_{k+1} - x_k}(y_{k+1} - y_k) \right]. \tag{4.18}$$

Exerciţiul 4.11:
Demonstraţi relaţiile (4.15) ÷ (4.18).

De regulă însă derivatele funcţiei nu sunt cunoscute şi atunci ele trebuie estimate. O variantă este aceea de a folosi pentru estimarea lor formule de derivare numerică (caz în care interpolarea se numeşte Bessel). O metodă mai avantajoasă este cea în care derivatele necesare sunt determinate din condiţii suplimentare de continuitate pentru derivata a doua a polinomului de interpolare. Această variantă este cunoscută sub numele de *interpolarea spline cubică clasică*. Ea are avantajul că funcţia de interpolare obţinută minimizează curbura pătratică medie a funcţiei. Este util de precizat că *spline* este un cuvânt preluat din limba engleză, care înseamnă o bucată de lemn flexibil, ce era folosit pentru desenarea curbelor. Pentru o astfel de aplicaţie, acest obiect avea tendinţa de a se orienta astfel încât curbura să fie minimă.

Derivata a doua a polinomului de interpolare dat de (4.13) este

$$g''(x) = 2c_{2k} + 6c_{3k}(x - x_k) \quad x \in [x_k, x_{k+1}], \quad k = 1, \ldots, n-1. \tag{4.19}$$

Condiţia de continuitate pentru derivata a doua poate fi pusă numai în nodurile interioare:

$$g''(x_k + 0) = g''(x_k - 0), \quad k = 2, \ldots, n-1. \tag{4.20}$$

Înlocuind expresia (4.19) precum şi expresiile coeficienţilor (4.15) ÷ (4.18) în (4.20) rezultă următoarele relaţii satisfăcute de derivatele funcţiei:

$$\begin{aligned} &\frac{1}{x_k - x_{k-1}} y'_{k-1} + 2\left(\frac{1}{x_k - x_{k-1}} + \frac{1}{x_{k+1} - x_k}\right) y'_k + \frac{1}{x_{k+1} - x_k} y'_{k+1} = \\ &= 3\frac{y_k - y_{k-1}}{(x_k - x_{k-1})^2} + 3\frac{y_{k+1} - y_k}{(x_{k+1} - x_k)^2}, \quad k = 2, \ldots, n-1. \end{aligned} \tag{4.21}$$

Exerciţiul 4.12:
Demonstraţi relaţia (4.21).

Relaţiile (4.21) sunt în număr de $n-2$ şi au n necunoscute. Cele două relaţii lipsă se obţin din impunerea condiţiilor la capete. Fie se impun valorile derivatelor y'_1 şi y'_n ca la interpolarea Bessel, caz în care se spune că interpolarea spline are condiţii la capete *forţate*, fie se impune ca valorile derivatei de ordinul doi să fie nule la capete, caz în care se spune că interpolarea spline are condiţii *naturale* la capete. Vom considera în cele ce urmează condiţiile naturale:

$$g''(x_1) = 0, \tag{4.22}$$

$$g''(x_n) = 0. \tag{4.23}$$

Din impunerea condiţiei (4.22) rezultă următoarea relaţie ce trebuie să existe între derivatele funcţiei:

$$2y_1' + y_2' = 3\frac{y_2 - y_1}{x_2 - x_1}. \tag{4.24}$$

Exerciţiul 4.13:

a) Demonstraţi relaţia (4.24).

b) Deduceţi relaţia ce trebuie impusă între derivate pentru a fi satisfăcută condiţia naturală (4.23) la capătul din dreapta.

Exerciţiul 4.14:

a) Scrieţi o funcţie Matlab cu următorul conţinut:

```
function yi = my_spline(x,y,xi)
% evalueaza polinomul de interpolare cubica spline
% date de intrare:
%   x - vector cu n componente, presupus sortat crescator
%   y - valorile corespunzatoare vectorului x
%   xi - vector cu oricate componente, in care va fi evaluat polinomul
%        de interpolare
% date de iesire
%   yi - valorile polinomului de interpolare evaluat in punctele xi

n = length(x);
m = length(xi);
yi = zeros(m,1);

%% calcul derivate
A = sparse(n,n);
b = zeros(n,1);
% g"(x_1) = 0
% ......................... completati ..........................
for k = 2:n-1
    % g"(x_k - 0) = g"(x_k + 0)
    A(k,k-1) = 1/(x(k)-x(k-1));
    % ......................... completati ......................
end
% g"(x_n) = 0
A(n,n-1) = 1;
A(n,n) = 2;
```

```
b(n) = 3*(y(n)-y(n-1))/(x(n)-x(n-1));
yder = A\b;

%% evaluare
for j = 1:m
    xcrt = xi(j);
    k = cauta_v4(n,x,xcrt);
    c0k = y(k);
    c1k = yder(k);
    h = x(k+1) - x(k);
    c2k = (3*(y(k+1) - y(k)) - h*(2*yder(k) + yder(k+1)))/h^2;
    %c3k =   ..... completati .....
    %yi(j) = ..... completati .....
end
```

b) Verificaţi funcţia scrisă cu comanda `mlint`.

Exerciţiul 4.15:

a) Scrieţi un script `main_spline.m` cu următorul conţinut:

```
x = [-2 0 4 6 7];
y = [7 -3 2 1 -1];
xi = linspace(-2,7,50);
yi = my_interp1(x,y,xi,'lagrange');
yip = my_interp1(x,y,xi,'lpp');
yis = my_spline(x,y,xi);
figure(4);
plot(x,y,'bo','MarkerSize',8,'LineWidth',2);
hold on;
plot(xi,yi,'r-','LineWidth',2);
hold on;
plot(xi,yip,'g--','LineWidth',2);
hold on;
plot(xi,yis,'m*','LineWidth',2);
leg{1} = 'Puncte folosite pentru interpolare';
leg{2} = 'global';
leg{3} = 'lpp';
leg{4} = 'spline';
legend(leg);
```

```
grid on;
```

b) Verificaţi că executându-l obţineţi rezultatul din fig.4.8.

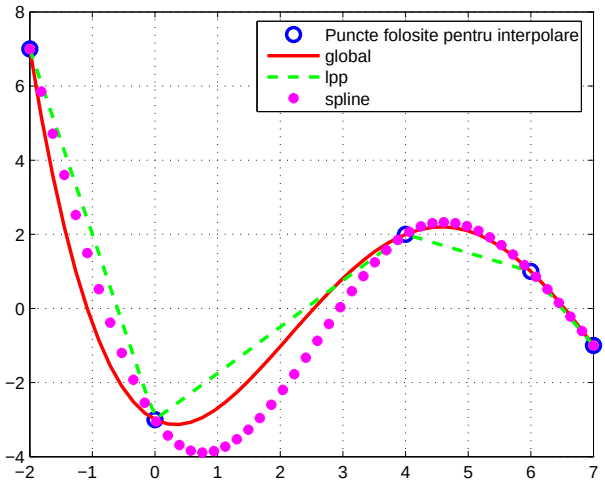

Figura 4.8: Pentru exerciţiul 4.15.

În Matlab există funcţii de interpolare. Scopul acestei teme a fost însă înţelegerea problemei interpolării precum şi modul de concepere al algoritmilor de interpolare.

Exerciţiul 4.16:

a) Citiţi documentaţia funcţiilor `interp1` şi `spline`.

b) Completaţi scriptul `main_spline.m` cu apelul funcţiilor Matlab şi comparaţi rezultatele.

4.2 Cazul 2D

Teoria generală a interpolării în două sau mai multe dimensiuni este mult mai dificilă decât cea unidimensională.

Iată cum poate fi extinsă *interpolarea Lagrange* în cazul bidimensional (2D).

Fie o funcţie $f : \Omega \to \mathbb{R}$, unde Ω este un domeniu bidimensional $[a, b] \times [c, d]$, cunoscută într-un număr de puncte din Ω. Presupunem că funcţia este cunoscută în nodurile unei grile date de diviziunea pe Ox:

$$a = x_0 < x_1 < \ldots < x_n = b$$

şi de diviziunea pe Oy:

$$c = y_0 < y_1 < y_2 < \ldots y_m = d.$$

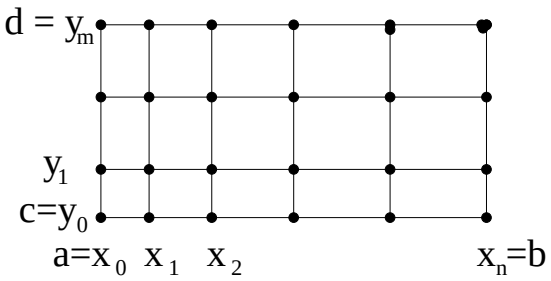

Figura 4.9: Grila de discretizare a lui Ω

Păstrând fixă una din variabilele independente, vom interpola unidimensional funcţia de-a lungul celeilalte variabile. De exemplu, pentru un y_j fixat al diviziunii pe Oy rezultă:

$$g(x, y_j) = \sum_{i=0}^{n} l_i(x) f(x_i, y_j). \tag{4.25}$$

Interpolând apoi după y, rezultă:

$$g(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{m} l_i(x) l_j(y) f(x_i, y_j). \tag{4.26}$$

În consecinţă, funcţiile Lagrange asociate unei grile bidimensionale sunt:

$$l_{ij}(x, y) = l_i(x) l_j(y) = \left(\prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{x - x_k}{x_i - x_k} \right) \left(\prod_{\substack{k=0 \\ k \neq j}}^{m} \frac{y - y_k}{y_j - y_k} \right). \tag{4.27}$$

Exerciţiul 4.17:

Implementarea în Matlab a funcţiei de calcul a polinomului Lagrange bidimensional este imediată. Scrieţi un fişier `lagrange2d.m` cu următorul conţinut.

```
function [zi] = lagrange2d(x,y,kx,ky,xi,yi)
%
[zi_x] = lagrange(x,kx,xi);
[zi_y] = lagrange(y,ky,yi);
zi = zi_x*zi_y';
```

Completaţi în acest fişier comentariile care descriu parametrii de intrare şi de ieşire ai funcţiei.

Exerciţiul 4.18:

Pentru testarea funcţiei implementate în exerciţiul anterior, scrieţi un script cu următorul conţinut:

```
clear all;
x = [0 1 3];
y = [5 7 10 11];
Nx = 50;
Ny = 40;
xi = linspace(0,3,Nx);
yi = linspace(5,11,Ny);
zi = lagrange2d(x,y,2,1,xi,yi);
figure(1);
surf(xi,yi,zi'); % Atentie, zi este transpus
xlabel('x');
ylabel('y');
zlabel('z');
colorbar
```

Prin rularea lui trebuie să obţineţi graficul din fig. 4.10. Folosind instrumentele oferite de Matlab, prezentaţi rezultatul privind perpendicular pe planul xOy, ca în fig. 4.11.

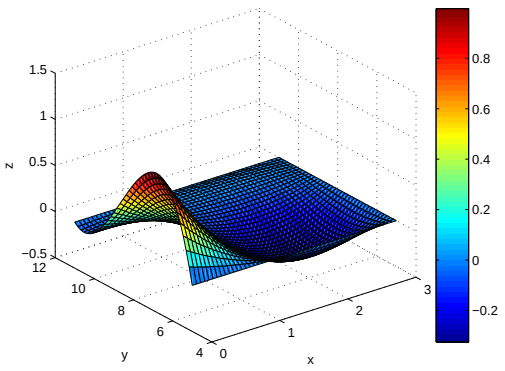

Figura 4.10: Funcţia Lagrange asociată gridului bidimensional [0 1 3] × [5 7 10 11] şi nodului (2,1).

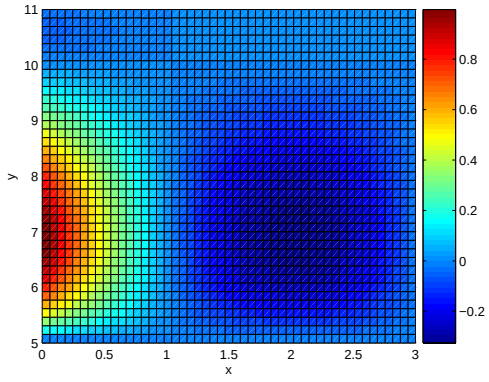

Figura 4.11: Funcţia Lagrange asociată nodului (2,1) - vedere perpendiculară pe xOy.

Exerciţiul 4.19:

a) Cărui nod îi este asociată funcţia Lagrange având reprezentarea din fig. 4.12?

b) O altă formă de vizualizare o puteţi obţine cu comanda `mesh`. Adăugaţi scriptului de mai sus liniile

```
figure(2);
mesh(xi,yi,zi');
```

```
xlabel('x');
ylabel('y');
zlabel('z');
colorbar
```

Ce reprezintă fig. 4.13?

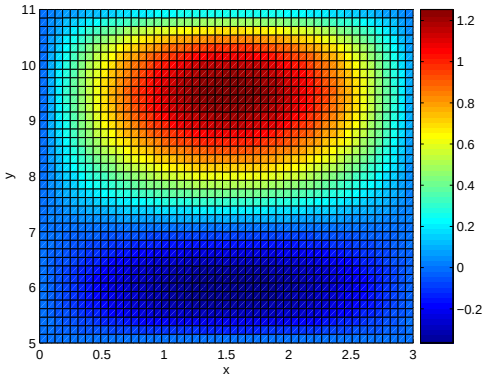

Figura 4.12: Pentru exerciţiul 4.19. Cărui nod îi este asociată această funcţie Lagrange?

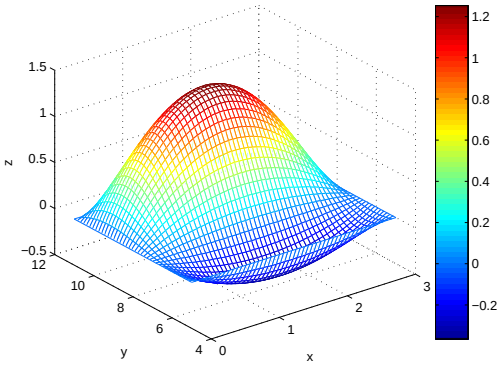

Figura 4.13: Pentru exerciţiul 4.19. Ce reprezintă această figură?

Pentru funcţiile scalare definite pe domenii bidimensionale este foarte sugestivă reprezentarea curbelor de nivel (adică curbe pe care valorile funcţiei reprezentate sunt constante, curbe numite şi curbe de echivalori, sau în mod particular, în funcţie de mărimea pe care o reprezintă: echipotenţiale, izoterme, etc).

Exerciţiul 4.20:

a) Adaugaţi scriptului de mai sus liniile

```
figure(3);
[C,h] = contour(xi,yi,zi');
set(h,'ShowText','on');
grid on;
xlabel('x');
ylabel('y');
zlabel('z');
```

Prin rularea lui trebuie să obţineţi fig. 4.14. Se vede acum cu uşurinţă care este nodul căruia îi este asociată funcţia Lagrange reprezentată. În acest nod, valoarea funcţiei este

1. Comparaţi rezultatul cu răspunsul pe care l-aţi dat la exerciţiul 4.19.
b) Care este diviziunea domeniului folosită pentru reprezentarea unei funcţii Lagrange în fig. 4.15. Cărui nod îi este asociată această funcţie?

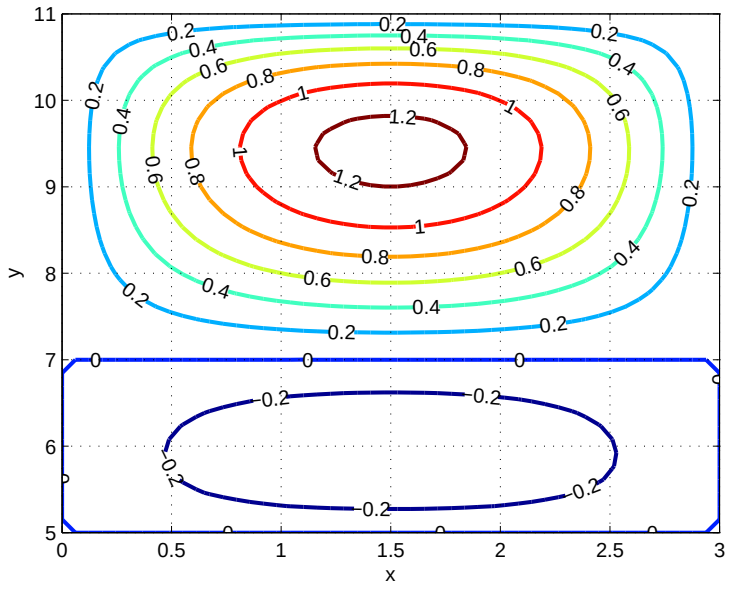

Figura 4.14: Pentru exerciţiul 4.20. Efectul comenzii `contour`.

Interpolarea globală în cazul 2d are aceleaşi dezavantaje ca interpolarea globală în cazul 1D. Mult mai utilă este interpolarea pe porţiuni.

Exerciţiul 4.21:
Citiţi documentaţia pentru funcţia Matlab `interp2`. Descrieţi succint metodele de interpolare disponibile.

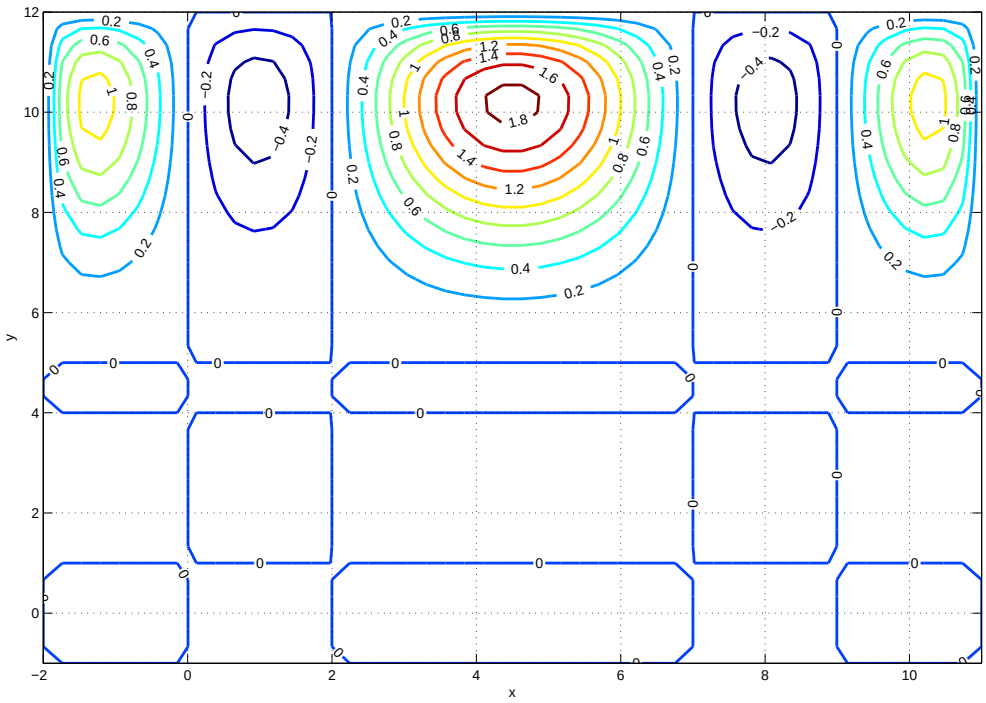

Figura 4.15: Pentru exerciţiul 4.20. Care este diviziunea domeniului? Cărui nod îi este asociată funcţia?

Capitolul 5

Derivarea numerică. Metoda diferenţelor finite.

Necesitatea evaluării derivatelor funcţiilor este importantă în problemele de inginerie. De exemplu, ele apar în evaluarea senzitivităţilor unor mărimi de interes în raport cu anumiţi parametri consideraţi variabili, senzitivităţi utile în proiectarea optimală a unor dispozitive. Un alt exemplu este cel în care modelul matematic al problemei conduce la ecuaţii sau sisteme de ecuaţii diferenţiale, foarte puţine dintre acestea putând fi rezolvate prin metode analitice.

Derivarea numerică se bazează pe interpolare. Formulele de derivare numerică provin din derivarea polinomului de interpolare şi ele se reduc la calcule aritmetice ce folosesc formule relativ simple. Prima parte a acestui capitol ilustrează deducerea acestor formule şi erorile care apar. Rezolvarea ecuaţiilor diferenţiale folosind formule de derivare numerică pentru derivatele ce intervin este cunoscută sub numele de *metoda diferenţelor finite*. Partea a doua a acestui capitol urmăreşte implementarea şi testarea acestei metode pentru câteva cazuri simple de test.

5.1 Formule de derivare în cazul unidimensional

Formulele de derivare numerică se pot deduce fie pornind de la polinomul de interpolare, fie prin intermediul seriilor Taylor. Această din urmă abordare are avantajul că furnizează şi informaţii legate de eroarea de trunchiere.

Vom reaminti câteva rezultate pentru cazul aproximării derivatelor unei funcţii unidimensionale $f : [a, b] \to \mathbb{R}$.

Considerând x_i punctele diviziunii intervalului $[a,b]$, vom folosi următoarele notaţii

$$f_i = f(x_i), \quad Df_i = \frac{\partial f}{\partial x}(x_i), \quad D^2 f_i = \frac{\partial^2 f}{\partial x^2}(x_i). \tag{5.1}$$

Cazul unei diviziuni uniforme: $x_{i+1} - x_i = h$

- Aproximarea derivatelor pornind de la polinomul de interpolare de gradul 1:

$$\text{(progresivă)} \quad Df_i = \frac{f_{i+1} - f_i}{h} + O(h) \tag{5.2}$$

$$\text{(regresivă)} \quad Df_i = \frac{f_i - f_{i-1}}{h} + O(h) \tag{5.3}$$

- Aproximarea derivatelor pornind de la polinomul de interpolare de gradul 2:

$$\text{(centrată)} \quad Df_i = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2) \tag{5.4}$$

$$\text{(progresivă)} \quad Df_i = \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2h} + O(h^2) \tag{5.5}$$

$$\text{(regresivă)} \quad Df_i = \frac{f_{i-2} - 4f_{i-1} + 3f_i}{2h} + O(h^2) \tag{5.6}$$

$$\text{(centrată)} \quad D^2 f_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h) \tag{5.7}$$

Cazul unei diviziuni neuniforme: $x_{i+1} - x_i = \alpha h$, $x_i - x_{i-1} = \beta h$

- Aproximarea derivatelor de ordinul 1 pornind de la polinomul de interpolare de gradul 2:

$$\text{(centrată)} \quad Df_i = \frac{\beta^2 f_{i+1} + (\alpha^2 - \beta^2) f_i - \alpha^2 f_{i-1}}{\alpha\beta(\alpha+\beta)h} + O(h^2) \tag{5.8}$$

$$\text{(progresivă)} \quad Df_i = \frac{-\alpha(\alpha+2\beta) f_i + (\alpha+\beta)^2 f_{i+1} - \beta^2 f_{i+2}}{\alpha\beta(\alpha+\beta)h} + O(h^2) \tag{5.9}$$

$$\text{(regresivă)} \quad Df_i = \frac{\alpha^2 f_{i-2} - (\alpha+\beta)^2 f_{i-1} + \beta(2\alpha+\beta) f_i}{\alpha\beta(\alpha+\beta)h} + O(h^2) \tag{5.10}$$

- Aproximarea derivatei de ordinul doi cu o eroare de ordinul[1] h:

$$D^2 f_i = \frac{\beta f_{i+1} - (\alpha+\beta) f_i - \alpha f_{i-1}}{\alpha\beta(\alpha+\beta)\frac{h^2}{2}} + O(h) \tag{5.11}$$

[1] Derivata de ordinul doi nu poate fi aproximată cu o eroare de ordinul h^2 dacă sunt folosite numai punctele x_{i-1}, x_i şi x_{i+1}.

Exerciţiul 5.1:
Pornind de la polinomul de interpolare de grad doi, ce trece prin punctele (x_{i-1}, f_{i-1}), (x_i, f_i), (x_{i+1}, f_{i+1}) deduceţi una din formulele de derivare de mai sus

Exerciţiul 5.2:
a) Scrieţi într-un fişier următorul script:

```
clear all;
% calculeaza derivata numerica a functiei sinus in pct x0
x0 = pi/4;
h = pi/10; % pasul de derivare

dp = (sin(x0+h) - sin(x0))/h; % derivata progresiva de ordinul 1
dr = (sin(x0) - sin(x0-h))/h; % derivata regresiva de ordinul 1
dc = (sin(x0+h) - sin(x0-h))/(2*h); % derivata centrata de ordinul 2
dexact = cos(x0);

erp = abs(dp - dexact);
err = abs(dr - dexact);
erc = abs(dc - dexact);
disp(sprintf(' ==== Der fct sin in x0 = %e, h = %e',x0,h));
disp(sprintf('valoarea exacta a der  = %e',dexact));
disp(sprintf('derivata nr progresiva = %e, eroare = %e',dp,erp));
disp(sprintf('            regresiva  = %e, eroare = %e',dr,err));
disp(sprintf('            centrata   = %e, eroare = %e',dc,erc));
```

b) Comentaţi valorile obţinute pentru cele trei formule. Schimbaţi pasul de derivare şi comentaţi rezultatele.

Este foarte probabil ca în acest moment să fiţi tentaţi să credeţi că, cu cât pasul de derivare este mai mic, cu atât eroarea e mai mică. Pentru a vedea dacă această afirmaţie este adevărată, este util să reprezentaţi grafic eroarea în funcţie de pasul de derivare.

Exerciţiul 5.3:
Scrieţi un script care să permită:
a) Reprezentarea grafică a erorii formulei de derivare progresivă de ordinul 1, în funcţie de pasul de derivare. Pentru un pas h cuprins între 1e-10 şi 1e-5 trebuie să obţineţi un grafic similar celui din partea stângă a fig. 5.1. Acest grafic a fost trasat folosind 1e5 valori. Repetaţi experienţa pentru derivarea regresivă de ordinul 1. Comentaţi rezultatul.
c) Reprezentarea grafică a erorii formulei de derivare centrată de ordinul 2, în funcţie de

pasul de derivare. Pentru un pas h cuprins între 1e-8 şi 1e-3 trebuie să obţineţi un grafic similar celui din partea dreaptă a fig. 5.1. Acest grafic a fost trasat folosind 1e5 valori.

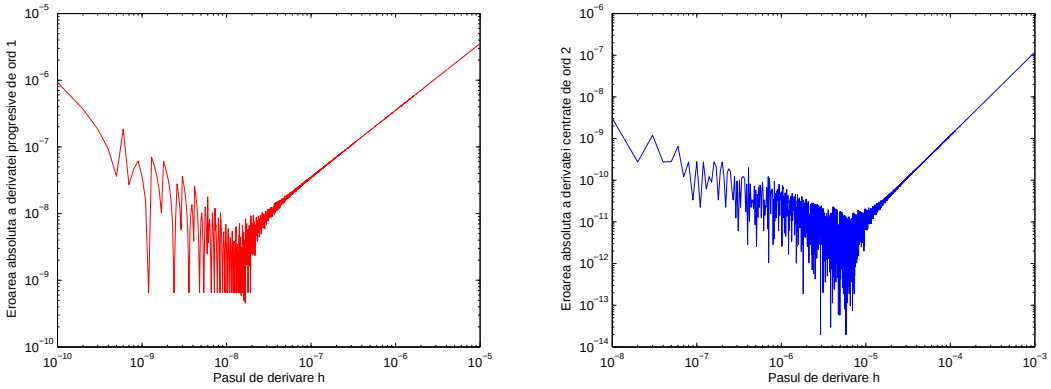

Figura 5.1: Eroarea în funcţie de pasul de derivare pentru formula de derivare progresivă de ordinul 1 (stânga) şi formula de derivare centrată de ordinul 2 (dreapta).

Zona de instabilităţi numerice, care apare pentru paşi mai mici decât o anumită valoare, se datorează erorilor de rotunjire care devin mai mari decât erorile de trunchiere. Pasul de la care încep să predomine erorile de trunchiere se numeşte *pas optim* de derivare numerică. Pasul optim nu este pasul pentru care eroarea este minimă. După cum se poate observa, erori mai mici se pot obţine pentru paşi aflaţi în zona de instabilitate. Totuşi lucrul în această zonă nu se recomandă. Este de preferat ca h să fie în zona în care predomină erorile de trunchiere, zonă în care erorile de trunchiere se pot estima.

Exerciţiul 5.4:

Inspectând graficele obţinute

a) comparaţi paşii optimi de derivare numerică pentru cele trei formule pe care le-aţi folosit până acum.

b) verificaţi că eroarea de trunchiere la formulele progresivă şi regresivă de ordinul 1 este $O(h)$ iar la formula centrată de ordinul 2 este $O(h^2)$.

Derivata centrată (5.4) oferă o acurateţe mai bună decât formulele progresivă, regresivă de ordinul 1 (formulele (5.2) şi (5.3)). O comparaţie numerică este sugerată în exerciţiul următor.

Exerciţiul 5.5:

Reprezentaţi pe acelaşi grafic eroarea derivatei numerice progresive de ordinul 1 şi centrate de ordinul 2 pentru paşi cuprinşi între 1e-8 şi 1e-3. Trebuie să obţineţi un grafic similar celui din figura 5.2.

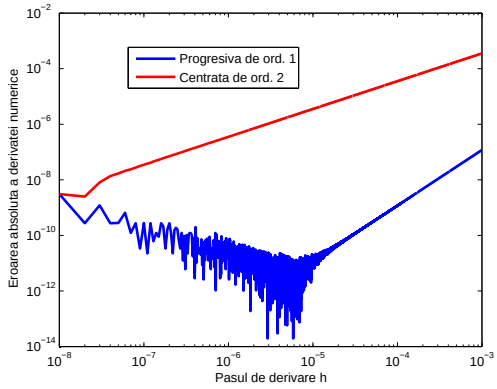

Figura 5.2: Acest grafic trebuie obţinut la exerciţiul 5.5.

5.2 Metoda diferenţelor finite

Aplicarea metodei diferenţelor finite pentru rezolvarea oricărei ecuaţii diferenţiale presupune realizarea următorilor paşi:

- **Pasul 1:** Se alege o schemă de diferenţe finite pentru aproximarea derivatelor din ecuaţii şi se rescrie ecuaţia ca ecuaţie cu diferenţe finite.
- **Pasul 2:** Se stabileşte grila de discretizare şi se scrie ecuaţia discretizată pentru fiecare nod.
- **Pasul 3:** Se rezolvă sistemul de ecuaţii pentru determinarea valorilor necunoscute în nodurile grilei.
- **Pasul 4:** Calculul altor valori, în afara celor din noduri, se face prin interpolare.

Se obişnuieşte să se spună că paşii 1 şi 2 reprezintă etapa de *preprocesare*, pasul 3 reprezintă *rezolvarea*, iar pasul 4 *postprocesarea*.

5.2.1 Studiul regimului tranzitoriu al unui circuit de ordinul 1

Cel mai simplu exemplu pe care ni-l putem imagina este cel care ne conduce matematic la o ecuaţie diferenţială ordinară de ordinul 1. Vom considera un condensator de capacitate $C = 4\mu$F, iniţial descărcat, care se încarcă de la o sursă de tensiune continuă $E = 20$ mV printr-un rezistor de rezistenţă $R = 10\ \Omega$.

Formularea corectă a problemei

Dacă notăm cu i curentul prin circuit şi cu u tensiunea la bornele condensatorului, atunci teorema Kirchhoff II scrisă pe bucla circuitului este

$$Ri(t) + u(t) = E. \tag{5.12}$$

Înlocuind în această relaţie dependenţa dintre tensiunea şi curentul prin condensator

$$i(t) = C\frac{\mathrm{d}u(t)}{\mathrm{d}t}, \tag{5.13}$$

rezultă ecuaţia diferenţială de ordinul unu, satisfăcută de u:

$$RC\frac{\mathrm{d}u(t)}{\mathrm{d}t} + u(t) = E. \tag{5.14}$$

Soluţia acestei ecuaţii este unică deoarece se specifică starea iniţială a condensatorului:

$$u(0) = u_0 = 0. \tag{5.15}$$

Avantajul acestui exemplu este acela că are o soluţie analitică, uşor de găsit. Vom reaminti modul de calcul al acesteia.

Conform teoriei cunoscute de la matematică, soluţia acestei ecuaţii se scrie

$$u(t) = u_o(t) + u_p, \tag{5.16}$$

unde $u_o(t)$ este soluţia ecuaţiei "omogene" (cum zic matematicienii) adică a ecuaţiei diferenţiale cu termen liber nul:

$$RC\frac{\mathrm{d}u_o(t)}{\mathrm{d}t} + u_o(t) = 0. \tag{5.17}$$

Soluţia ecuaţiei omogene este

$$u_o(t) = A\exp(xt), \tag{5.18}$$

unde x este soluţia ecuaţiei caracteristice

$$RCx + 1 = 0. \tag{5.19}$$

Dacă notăm produsul dintre rezistenţă şi capacitate cu

$$\tau = RC, \tag{5.20}$$

mărime care se numeşte *constanta de timp* a circuitului, rezultă soluţia ecuaţiei omogene

$$u_o(t) = A\exp(-t/\tau). \tag{5.21}$$

Mărimea u_p este o soluţie particulară, de forma termenului liber al ecuaţiei, deci o constantă. Să notăm această constantă cu B. Soluţia particulară (constantă) trebuie să satisfacă ecuaţia (5.14), deci

$$RC\frac{\mathrm{d}B}{\mathrm{d}t} + B = E, \tag{5.22}$$

de unde rezultă

$$B = E \tag{5.23}$$

deoarece derivata unei constante este nulă. Soluţia generală a ecuaţiei este în consecinţă

$$u(t) = A\exp(-t/\tau) + E. \tag{5.24}$$

Constanta A se determină din impunerea condiţiei iniţiale $u(0) = u_0$, de unde

$$A = u_0 - E. \tag{5.25}$$

Expresia finală a soluţiei analitice este

$$u(t) = (u_0 - E)\exp(-t/\tau) + E. \tag{5.26}$$

Exerciţiul 5.6:

a) Care este valorea spre care tinde tensiunea pe condensator?

b) Scrieţi un script Matlab `main_RC.m` cu următorul conţinut

```
clear all;
% rezolva cu MDF un circuit RC simplu
E = 20e-3;
R = 10;
C = 4e-6;
u_initial = 0;
tau = R*C;      % constanta de timp
tmax = 10*tau;  % timp maxim de simulare

% solutia analitica - pentru referinta
t_ref = linspace(0,tmax,1e6);
uc_ref = (u_initial - E)*exp(-t_ref/tau) + E;
figure(1);
plot(t_ref,uc_ref,'linewidth',2);
leg{1} = 'analitic';
```

Executaţi-l şi verificaţi că aţi răspuns corect la punctul a.

Rezolvarea cu diferenţe finite

Vom rezolva acum ecuaţia diferenţială cu diferite scheme de derivare numerică. Ecuaţia (5.14) o rescriem ca

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} + \frac{1}{\tau}u(t) = \frac{1}{\tau}E. \tag{5.27}$$

Vom urmări calculul numeric în intervalul de timp $[0, t_{max}]$ unde $t_{max} = 10\tau$ într-o reţea echidistantă de N puncte t_k, unde pasul de discretizare este

$$t_{k+1} - t_k = h, \quad \text{pentru} \quad k = 1, \ldots, N-1. \tag{5.28}$$

Vom nota valorile discrete obţinute prin rezolvare numerică cu u_k. Ele vor fi aproximaţii ale mărimii reale u.

$$u_k \approx u(t_k). \tag{5.29}$$

Varianta I - utilizarea formulei de diferenţe finite progresive de ordinul 1

Vom discretiza ecuaţia (5.27) prin scrierea ei în momentul de timp t_k şi folosind pentru derivată o formulă de diferenţe finite progresive de ordinul 1:

$$\frac{u_{k+1} - u_k}{h} + \frac{1}{\tau}u_k = \frac{1}{\tau}E, \tag{5.30}$$

de unde

$$u_{k+1} - u_k + \frac{h}{\tau}u_k = \frac{h}{\tau}E. \tag{5.31}$$

Se observă că mărimea u_{k+1} poate fi calculată explicit cu formula

$$u_{k+1} = u_k\left(1 - \frac{h}{\tau}\right) + \frac{h}{\tau}E. \tag{5.32}$$

Această formulă este extrem de uşor de implementat.

Exerciţiul 5.7:

a) Completaţi scriptul de mai sus cu

```
%%%%%%%%%%%%%%%% preprocesare %%%%%%%%%%%%%%%%%%%%%
h = tau/2;                % pasul gridului de discretizare pe axa timpului
NI = floor(tmax/h);       % nr de intervale de timp
N = NI + 1;               % numarul de puncte
t = linspace(0,NI*h,N); % vectorul timp

%%%%%%%%%%%%%%%% rezolvare %%%%%%%%%%%%%%%%%%%%%%%%
up = mdf_circuitRC(u_initial,N,h,tau,E,'progresiv1');
```

b) Scrieţi o funcţie `mdf_circuitRC.m` de următorul tip

```
function u = mdf_circuitRC(u_initial,N,h,tau,E,metoda)
% rezolva ecuatia diferentiala du/dt + 1/tau u = 1/tau E cu
% metoda diferentelof finite

u = zeros(N,1);
u(1) = u_initial;
switch metoda
    case 'progresiv1'
        % calcul explicit
        for k = 1:N-1
            ................. completati..........
        end
    otherwise
        error('Metoda de discretizare necunoscuta');
end
```

c) Completati scriptul `main_RC.m` cu

```
%%%%%%%%%%%%%%%% postprocesare %%%%%%%%%%%%%%%%%%%
figure(1);
hold on;
plot(t,up,'r-*','linewidth',2);
leg{2} = 'prog ord 1';
title(sprintf('h/tau = %f',h/tau));
```

Experimentaţi comportamentul metodei pentru $h = 2\tau, \tau, \tau/10$. Observaţi că metoda este instabilă pentru $h < \tau$. Această metodă este cunoscută şi sub numele de metoda Euler explicită pentru rezolvarea ecuaţiilor diferenţiale ordinare.

Varianta a II-a - utilizarea formulei de diferenţe finite regresive de ordinul 1

Vom discretiza ecuaţia (5.27) prin scrierea ei în momentul de timp t_k şi folosind pentru derivată o formulă de diferenţe finite regresive de ordinul 1:

$$\frac{u_k - u_{k-1}}{h} + \frac{1}{\tau} u_k = \frac{1}{\tau} E, \tag{5.33}$$

de unde

$$u_k - u_{k-1} + \frac{h}{\tau} u_k = \frac{h}{\tau} E. \tag{5.34}$$

Se observă că mărimea u_k poate fi calculată explicit cu formula

$$u_k = \left(u_{k-1} + \frac{h}{\tau}E\right) / \left(1 + \frac{h}{\tau}\right). \tag{5.35}$$

Şi această formulă este extrem de uşor de implementat.

Exerciţiul 5.8:

a) Completaţi funcţia `mdf_circuitRC.m` cu cazul "regresiv1":

```
case 'regresiv1'
    % calcul explicit
    for k = ..............completati
        u(k) = ...........completati
    end
```

b) Completaţi scriptul cu apelul funcţiei în cazul regresiv

```
ur = mdf_circuitRC(u_initial,N,h,tau,E,'regresiv1');
```

şi partea de postprocesare cu reprezentarea grafică a soluţiei obţinute, pe acelaşi grafic ca soluţia analitică şi soluţia obţinută în prima variantă.

Experimentaţi comportamentul metodei pentru $h = 2\tau, \tau, \tau/10$. Observaţi că metoda nu mai este instabilă pentru $h < \tau$. Această metodă este cunoscută şi sub numele de metoda Euler implicită[2] pentru rezolvarea ecuaţiilor diferenţiale ordinare.

Varianta a III-a - utilizarea formulei de diferenţe finite centrate de ordinul 2

Vom discretiza ecuaţia (5.27) prin scrierea ei în momentul de timp t_k şi folosind pentru derivată o formulă de diferenţe finite centrate de ordinul 2 dată de relaţia (5.4):

$$\frac{u_{k+1} - u_{k-1}}{2h} + \frac{1}{\tau}u_k = \frac{1}{\tau}E, \tag{5.36}$$

de unde

$$-u_{k-1} + \frac{2h}{\tau}u_k + u_{k+1} = \frac{2h}{\tau}E. \tag{5.37}$$

Se observă că mărimea u_k nu mai poate fi calculată explicit. Relaţia (5.37) poate fi scrisă pentru $k = 2, \ldots, N-1$ şi reprezintă $N-2$ ecuaţii cu N necunoscute. Pentru a obţine

[2]Ecuaţia generală este $dy/dt = f(t, y)$ unde f este o funcţie în general neliniară. Folosirea derivatei regresive pentru f conduce la o relaţie implicită pentru calculul lui u_k, relaţie ce reprezintă o ecuaţie neliniară. În cazul particular studiat, f este liniară şi de aceea soluţia se poate calcula explicit.

un sistem bine formulat, trebuie să adăugăm încă două relaţii. Acestea sunt relaţiile la capete. La $t = 0$ vom impune condiţia iniţială:

$$u(1) = u_0, \tag{5.38}$$

iar pentru ultimul punct $k = N$ vom scrie ecuaţia discretizată dar în care vom folosi pentru derivată o formulă de diferenţe regresive de ordinul 2 dată de relaţia (5.6):

$$\frac{u_{N-2} - 4u_{N-1} + 3u_N}{2h} + \frac{1}{\tau}u_N = \frac{1}{\tau}E, \tag{5.39}$$

de unde

$$u_{N-2} - 4u_{N-1} + \left(3 + \frac{2h}{\tau}\right)u_N = \frac{2h}{\tau}E. \tag{5.40}$$

Exerciţiul 5.9:

a) Completaţi funcţia `mdf_circuitRC.m` cu cazul "centrat2":

```
case 'centrat2'
       %% asamblare sistem
       A = sparse(N,N);
       tl = zeros(N,1);
       A(1,1) = 1;          % conditia initiala
       tl(1) = u_initial;
       for k = 2:N-1        % noduri interioare
           A(k,k-1) = -1;
           ................completetati..........
       end
       A(N,N-2) = 1;        % conditia la tmax
       ..................completetati............
       tl(N) = 2*h*E/tau;
       %% rezolvare
       u = A\tl;
```

b) Completaţi scriptul cu apelul funcţiei în cazul centrat şi partea de postprocesare cu reprezentarea grafică a soluţiei obţinute, pe acelaşi grafic ca soluţia analitică şi soluţia obţinută în prima variantă.

Experimentaţi comportamentul metodei pentru $h = 2\tau, \tau, \tau/10$. Observaţi că metoda nu este instabilă şi este mai precisă decât implementările anterioare. Acest lucru era de aşteptat deoarece formulele de derivare numerică de ordinul 2 sunt mai precise decât cele de ordinul 1. Creşterea acurateţii se face însă pe seama creşterii complexităţii algoritmului.

Pentru verificare, comparaţi graficele pe care le-aţi obţinut cu cele din figurile 5.3 ÷ 5.5.

Exerciţiul 5.10:

Completaţi scriptul `main_RC.m` cu comenzi astfel încât, într-o altă fereastră grafică să obţineţi graficele erorilor absolute în funcţie de pasul de timp.

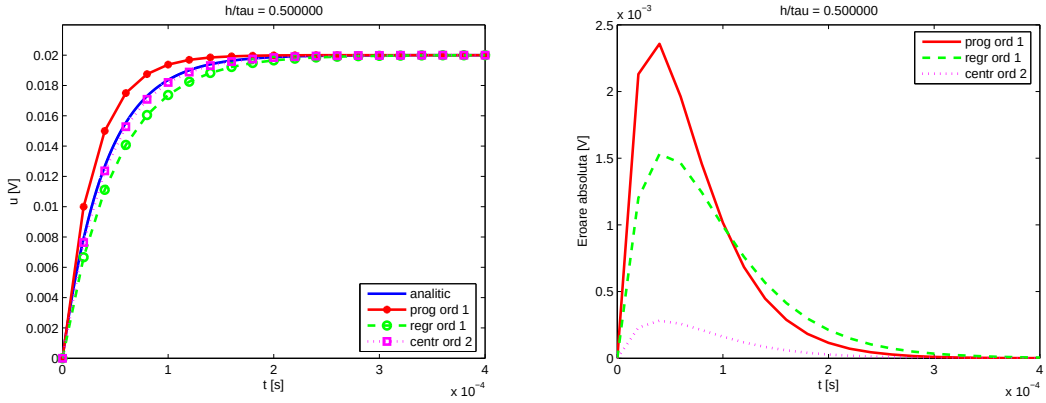

Figura 5.3: Cazul $h = \tau/2$.

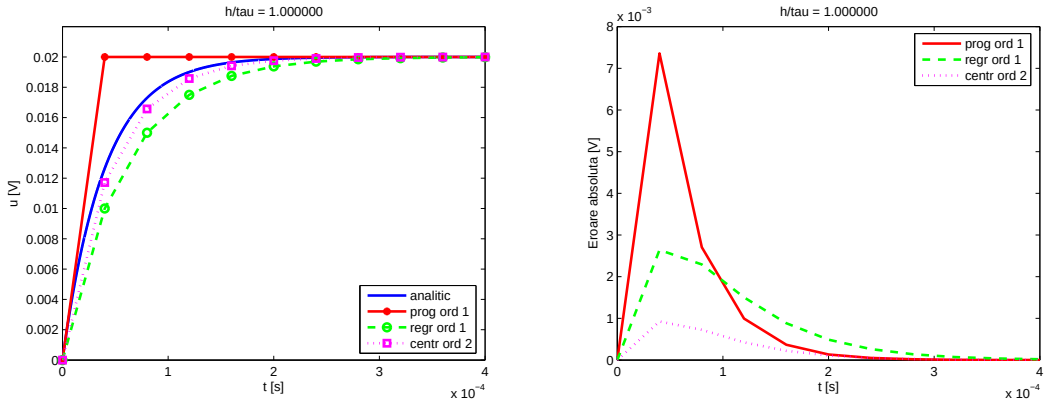

Figura 5.4: Cazul $h = \tau$.

În capitolul 3, aţi văzut că asamblarea matricelor în Matlab se poate face mult mai rapid dacă se construieşte formatul pe coordonate al matricei, după ce, în prealabil, a fost alocat un spaţiu de memorie exact cât este necesar pentru stocarea matricei.

Exerciţiul 5.11:

a) Completaţi funcţia `mdf_circuitRC.m` cu cazul "centrat2b":

```
case 'centrat2b'
      %% asamblare sistem
```

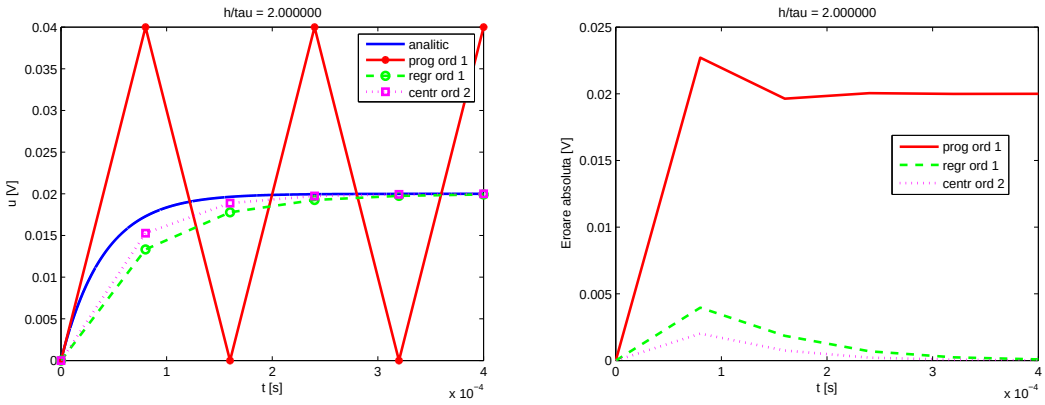

Figura 5.5: Cazul $h = 2\tau$.

```
tic;
no_nnz = 3*(N-1) + 1;
r_idx = zeros(no_nnz,1);
c_idx = zeros(no_nnz,1);
val = zeros(no_nnz,1);
tl = zeros(N,1);
m = 1;
% A(1,1) = 1          % conditia initiala
r_idx(m) = 1;
c_idx(m) = 1;
val(m) = 1;
tl(1) = u_initial;
for k = 2:N-1          % noduri interioare
    %A(k,k-1) = -1;
    m = m + 1;
    r_idx(m) = k;
    c_idx(m) = k-1;
    val(m) = -1;
    ..................completati...........
end
.....................completati.............
tl(N) = 2*h*E/tau;
A = sparse(r_idx,c_idx,val,N,N);
t2 = toc;
disp(sprintf('centrat2b - timp asamblare matrice = %e',t2));
%% rezolvare
```

```
u = A\tl;
```

b) Completaţi scriptul cu apelul funcţiei în acest caz. Pentru a verifica faptul că nu s-au strecurat greşeli, reprezentaţi grafic într-o altă fereastră soluţia obţinută în cazul "centrat2b" cu soluţia obţinută în cazul "centrat".
c) Introduceţi instrucţiuni pentru contorizarea timpului de asamblare al matricei în ambele variante de implementari. Comparaţi aceşti timpi pentru cazul $h = \tau/1000$.

5.2.2 Studiul regimului electrocinetic staţionar al unui conductor bidimensional

Exemplul din acest paragraf va conduce la o ecuaţie diferenţială cu derivate parţiale de ordinul 2, de tip eliptic.

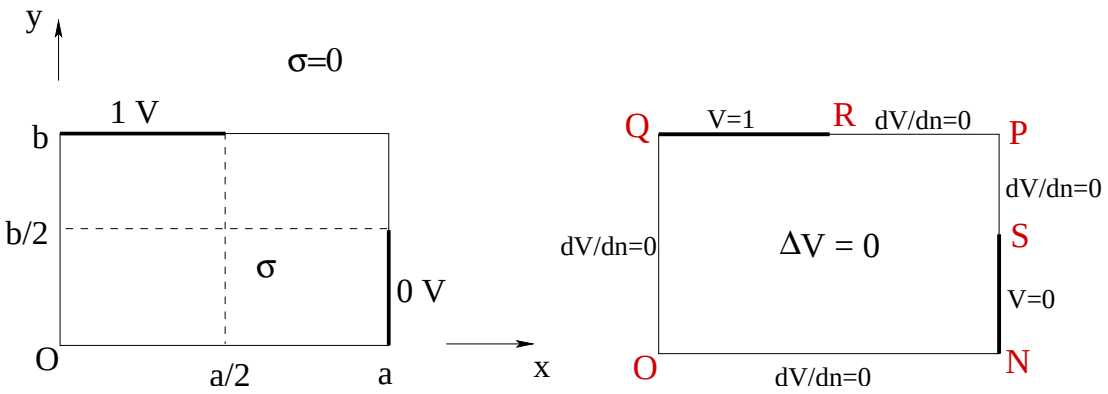

Figura 5.6: Problema 2D de regim electrocinetic: domeniul de calcul (stânga), condiţii de frontieră (dreapta).

Vom considera un conductor omogen, de conductivitate σ, situat într-un mediu perfect izolant. Conductorul are o dimensiune după axa Oz mult mai lungă decât celelalte două. Figura 5.6 reprezintă o secţiune perpendiculară pe direcţia dimensiunii foarte mari. Această secţiune este un dreptunghi, de dimensiuni a, b. Conductorul are două borne supraconductoare, una aflată la potenţial $V_0 = 1$V, iar cealaltă aflată la potenţial nul. Bornele sunt plasate ca în figură. Dorim să reprezentăm liniile echipotenţiale şi spectrul liniilor de curent. Ar fi interesant să calculăm şi mărimi globale cum ar fi energia câmpului acumulată în domeniu precum şi rezistenţa pe unitatea de lungime a acestui rezistor. La aceste aspecte ne vom întoarce însă după ce vom discuta problema integrării numerice.

Formularea corectă a problemei

Înainte de orice, o problemă de câmp electromagnetic trebuie corect formulată, în conformitate cu teorema de unicitate demonstrată pentru regimul studiat.

Problema fiind de regim electrocinetic staţionar, formularea se va face în V – potenţial electrocinetic, unde $\mathbf{E} = -\text{grad}\, V$, $\mathbf{E}$ fiind intensitatea câmpului electric.

Ecuaţia de ordinul doi satisfăcută de potenţialul electrocinetic este:

$$-\text{div}\,(\sigma\,\text{grad}\, V) = 0 \tag{5.41}$$

unde $V : \mathcal{D} \to \mathbb{R}$ este potenţialul definit pe domeniul bidimensional $\mathcal{D} = \text{ONPQ}$. Ecuaţia (5.41) este o ecuaţie eliptică, de tip Laplace generalizată.

Deoarece domeniul este omogen (conductivitatea σ are aceeaşi valoare în orice punct al domeniului), ecuaţia se simplifică la o ecuaţie de tip Laplace:

$$\Delta V = 0, \tag{5.42}$$

unde $\Delta = \text{div}\,\text{grad}$ este operatorul Laplace, care în coordonate carteziene (cazul 2D) are expresia

$$\Delta V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2}. \tag{5.43}$$

În consecinţă, ecuaţia diferenţială ce trebuie rezolvată este o ecuaţie cu derivate parţiale:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0, \tag{5.44}$$

unde $V = V(x, y) : \mathcal{D} \to \mathbb{R}$.

Pentru buna formulare a problemei, trebuie impuse pentru potenţial condiţiile de frontieră. Frontierele supraconductoare (SN şi QR) au potenţial constant, şi în consecinţă pe ele trebuie impuse codiţii Dirichlet în concordanţă cu valorile potenţialului. Pe frontierele de lângă izolant (NOQ şi RPS), trebuie impuse condiţii Neumann nule[3].

Condiţiile de frontieră impuse potenţialului sunt:

$$V = V_0 \quad \text{pe} \quad \text{QR} \quad \text{(Dirichlet)} \tag{5.45}$$

$$V = 0 \quad \text{pe} \quad \text{SN} \quad \text{(Dirichlet)} \tag{5.46}$$

$$\frac{\partial V}{\partial n} = 0 \quad \text{pe} \quad \text{NOQ} \cup \text{RPS} \quad \text{(Neumann)} \tag{5.47}$$

[3]În izolant nu există curent de conducţie, în consecinţă $\mathbf{J} \cdot \mathbf{n} = 0$ la frontiera cu izolantul, deci $-\sigma\text{grad}\, V \cdot \mathbf{n} = 0$, echivalent cu $-\sigma \frac{\partial V}{\partial n} = 0$ pe acele frontiere.

Şi pentru o astfel de problemă se pot găsi rezolvări analitice bazate de exemplu pe metoda separării variabilelor sau pe aproximarea liniilor de câmp. Deşi nu este dificilă, prezentarea acestor posibile soluţii nu este atât de simplă ca cea din exemplul anterior. Scopul nostru este însă de a prezenta modul în care se poate rezolva această problemă cu metoda diferenţelor finite aşa încât în cele ce urmează ne vom concentra exclusiv asupra acestui aspect.

Rezolvarea cu diferenţe finite

Faţă de cazul unidimensional discutat în exemplul anterior, aici gridul de discretizare este unul bidimensional, iar în loc de aproximarea unei derivate totale de ordinul 1, acum avem de aproximat derivate parţiale de ordinul 2.

Gridul de discretizare bidimensional poate fi privit ca fiind obţinut din produsul cartezian dintre un grid de discretizare pe axa Ox şi un grid de discretizare pe axa Oy:

$$0 = x_1 < x_2 < \ldots < x_{n_x} = a$$
$$0 = y_1 < y_2 < \ldots < y_{n_y} = b.$$

astfel încât, un nod al gridului este identificat de poziţia proiecţiilor sale pe cele două axe:

$$(x_i, y_j) \quad i = 1, \ldots, n_x, \quad j = 1, \ldots, n_y.$$

Metoda diferenţelor finite va determina valorile potenţialului în nodurile acestui grid:

$$V(x_i, y_j) \stackrel{not}{=} V_{i,j} \quad i = 1, n_x, \quad j = 1, n_y. \tag{5.48}$$

Forma discretizată (aproximativă) a ecuaţiei potenţialului este obţinută aproximând prin diferenţe finite ecuaţia (5.44). Forma specifică depinde de tipul nodului: interior sau pe frontiera Neumann. Nodurile de pe frontiera Dirichlet nu ridică probleme, ecuaţia asociată unui astfel de nod constând în simpla atribuire a valorii potenţialului nodului respectiv.

Ecuaţia asociată unui nod interior:

Ecuaţia aproximativă pentru un nod interior se deduce înlocuind derivatele parţiale după x şi după y cu formule de tipul (5.7).

Pentru a simplifica scrierea formulelor, vom presupune că cele două griduri pe Ox şi Oy sunt uniforme, cu paşi h_x şi respectiv h_y. Derivatele parţiale se vor înlocui cu:

$$\frac{\partial^2 V}{\partial x^2}(x_i, y_j) = \frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{h_x^2}, \tag{5.49}$$
$$\frac{\partial^2 V}{\partial y^2}(x_i, y_j) = \frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{h_y^2}. \tag{5.50}$$

Ecuaţia discretizată asociată unui nod interior devine

$$\frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{h_x^2} + \frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{h_y^2} = 0, \tag{5.51}$$

sau

$$2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) V_{i,j} - \frac{1}{h_x^2} V_{i+1,j} - \frac{1}{h_x^2} V_{i-1,j} - \frac{1}{h_y^2} V_{i,j+1} - \frac{1}{h_y^2} V_{i,j-1} = 0. \tag{5.52}$$

Relaţia (5.52) arată că potenţialul într-un nod interior este o combinaţie liniară a potenţialelor nodurilor învecinate. Dacă paşii de discretizare pe cele două axe ar fi egali ($h_x = h_y$), atunci potenţialul într-un nod interior este media aritmetică a potenţialelor celor patru noduri vecine.

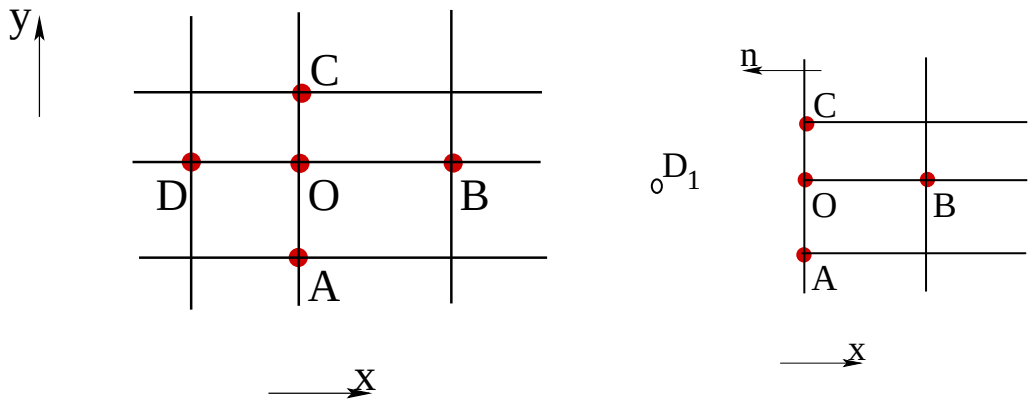

Figura 5.7: Nodurile marcate intervin în scrierea ecuaţiilor: stânga - cazul unui nod interior; dreapta - cazul unui nod pe frontieră Neumann dreaptă.

Exerciţiul 5.12:

În cazul unor griduri neuniforme, folosind notaţiile din fig. 5.7, arătaţi că ecuaţia discretizată asociată unui nod interior este

$$V_O\left(\frac{1}{h_B h_D} + \frac{1}{h_A h_C}\right) - V_A \frac{1}{h_A(h_A + h_C)} - V_B \frac{1}{h_B(h_B + h_D)} - \\ -V_C \frac{1}{h_C(h_A + h_C)} - V_D \frac{1}{h_D(h_B + h_D)} = 0, \tag{5.53}$$

unde $h_A = \|OA\|$, $h_B = \|OB\|$, $h_C = \|OC\|$, $h_D = \|OD\|$.

Ecuaţia asociată unui nod pe frontieră dreaptă:

În cazul unui punct pe frontieră O (fig. 5.7- dreapta), să notăm cu $g = -\sigma \frac{\partial V}{\partial n}$ condiţia de frontieră Neumann şi cu g_O valoarea ei medie în punctul O:

$$g_O = \frac{g_{OA} h_A + g_{OC} h_C}{h_A + h_C} \tag{5.54}$$

unde g_{OA} este condiţia de frontieră asociată segmenului OA iar g_{OC} este condiţia de frontieră asociată segmentului OC.

Pentru deducerea ecuaţiei aproximative pentru punctul O, vom considera un nod "fantomă" D_1, situat în exterior, la o distanţă $h_D = \|OD_1\|$ de punctul O. Pentru simplitate putem considera $h_D = h_B$.

Din condiţia de frontieră Neumann, deducem valorea potenţialului fantomă în funcţie de valoarea acestei condiţii. Pentru cazul din figură (normala exterioară în sens contrar axei x) rezultă:

$$\frac{\partial V}{\partial x} = \frac{g}{\sigma}. \tag{5.55}$$

Scriind relaţia aproximativă (5.4) pentru condiţia de frontieră Neumann, rezultă că:

$$V_{D_1} = V_B - 2h_B \frac{g_O}{\sigma}. \tag{5.56}$$

Pentru nodul O se scrie acum o ecuaţie de tipul (5.53), ca pentru un nod interior şi înlocuind expresia (5.56) rezultă ecuaţia finală (5.57):

$$V_O \left(\frac{1}{h_B^2} + \frac{1}{h_A h_C} \right) - V_A \frac{1}{h_A(h_A + h_C)} - V_B \frac{1}{h_B^2} - V_C \frac{1}{h_C(h_A + h_C)} = -\frac{g_O}{\sigma h_B}. \tag{5.57}$$

În cazul unei condiţii Neumann nule, relaţia devine:

$$V_O \left(\frac{1}{h_B^2} + \frac{1}{h_A h_C} \right) - V_A \frac{1}{h_A(h_A + h_C)} - V_B \frac{1}{h_B^2} - V_C \frac{1}{h_C(h_A + h_C)} = 0. \tag{5.58}$$

Ecuaţia asociată unui nod pe frontieră - colţ exterior:

Nodurile situate în colţuri reprezintă o problemă pentru metoda diferenţelor finite. Aceasta deoarece pentru un colţ nu poate fi definită direcţia normalei[4]. Dacă notăm $g = -\sigma \frac{\partial V}{\partial n}$, şi notăm cu g_{OA} valoarea lui g pe segmentul OA (fără capătul O) şi cu g_{OB} valoarea lui g pe segmentul OB (fără capătul O), atunci vom presupune că în nodul O (fig. 5.8) este cunoscută derivata după o direcţie care rezultă din prelungirea în O a valorilor funcţiei g.

Pentru deducerea ecuaţiei vom considera două noduri "fantomă" C_1 şi D_1, situate în exterior, la distanţele $h_D = \|OD_1\|$ şi $h_C = \|OC_1\|$ de punctul O. Pentru simplitate putem considera $h_D = h_B$ şi $h_C = h_A$.

[4]În realitate, colţul reprezintă o modelare brutală a realităţii. Trecerea de la o suprafaţă la alta se face prin racordări.

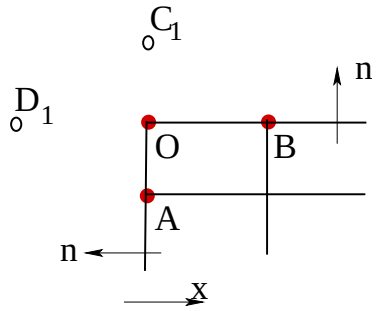

Figura 5.8: Nodurile marcate intervin în scrierea ecuaţiei unui nod pe frontieră colţ exterior.

Din condiţia de frontieră Neumann pe segmentul OB, scrisă în nodul O, putem deduce valoarea potenţialului fantomă C_1:

$$V_{C_1} = V_A - 2h_A \frac{g_{OB}}{\sigma}. \tag{5.59}$$

Din condiţia de frontieră Neumann pe segmenul OA, scrisă în nodul O, putem deduce valoarea potenţialului fantomă D_1:

$$V_{D_1} = V_B - 2h_B \frac{g_{OA}}{\sigma}. \tag{5.60}$$

Pentru nodul O se scrie acum o ecuaţie de tipul (5.53), ca pentru un nod interior, şi înlocuind expresiile (5.59) şi (5.60) rezultă ecuaţia finală (5.61):

$$V_O \left(\frac{1}{h_A^2} + \frac{1}{h_B^2} \right) - V_A \frac{1}{h_A^2} - V_B \frac{1}{h_B^2} = -\frac{g_{OB}}{\sigma h_A} - \frac{g_{OA}}{\sigma h_B} \tag{5.61}$$

În cazul unor condiţii Neumann nule, relaţia devine:

$$V_O \left(\frac{1}{h_A^2} + \frac{1}{h_B^2} \right) - V_A \frac{1}{h_A^2} - V_B \frac{1}{h_B^2} = 0 \tag{5.62}$$

În concluzie, discretizarea problemei conduce la un sistem de ecuaţii algebric liniar, prin a cărui rezolvare vom obţine valorile potenţialelor în nodurile gridului de discretizare. Pentru asamblarea acestui sistem cu ajutorul unui algoritm, este util ca nodurile să fie numerotate astfel încât necunoscutele să reprezinte un vector, nu o matrice cum sugerează notaţiile de până acum. Vom numerota nodurile de jos în sus şi de la stânga la dreapta, ca în figura 5.9. Se poate verifica uşor că relaţia între numărul nodului k şi poziţiile proiecţiilor sale pe cele două axe i şi j este

$$k = (i-1)n_y + j, \quad i = 1, \ldots, n_x, \quad j = 1, \ldots, n_y. \tag{5.63}$$

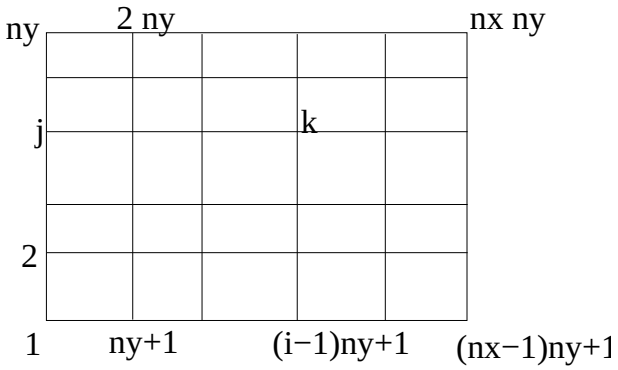

Figura 5.9: Numerotarea nodurilor.

unde n_x şi n_y reprezintă numărul de puncte de discretizare pe axele x, respectiv y.

Exerciţiul 5.13:

Scrieţi ecuaţiile discretizate pentru cazul $n_x = ny = 3$, $h_x = h_y = 1$. Completaţi matricea coeficienţilor A şi vectorul termenilor liberi t pentru sistemul de ecuaţii de rezolvat:

$$A = \begin{bmatrix} & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \qquad t = \begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

În acest moment putem trece la conceperea algoritmului. Vom scrie funcţii speciale pentru: citirea datelor problemei, generarea gridului, asamblarea sistemului de ecuaţii de rezolvat şi pentru postprocesare.

Să începem cu citirea datelor problemei. În principiu, am putea citi datele de la tastatură sau din fişiere, dar cum scopul nostru este de a construi şi testa partea cea mai complicata a algoritmului (preprocesarea), vom prefera ca datele problemei să le "citim" prin instanţiere, într-o funcţie numită `citire_date_elcin`. Datele problemei sunt grupate într-o structură numită `date`, ce are mai multe câmpuri.

Exerciţiul 5.14:

a) Scrieţi un script `mdf_elcin.m` cu următorul conţinut:

```
clear all;
```

```
% rezolva ecuatia Laplace V = 0
% intr-un domeniu dreptunghiular de dimensiuni a, b
% prin metoda diferentelor finite
% Conditii de frontiera
%      * pe latura de sus, in stanga exista un electrod la potential V1
%      * pe latura din dreapta, jos exista un electrod la potential 0
%      * in rest cond Neumann nule

%% citirea datelor problemei
date = citire_date_elcin();
```

b) Scrieţi o funcţie `citire_date_elcin.m` cu următorul conţinut

```
function date = citire_date_elcin()
date.a = 2;          % dim pe OX
date.b = 2;          % dim pe Oy
date.T1 = date.a/2;  % dim terminalului de sus
date.T2 = date.b/2;  % dim terminalului din dreapta
date.V1 = 1;         % potentialul lui T1
date.V2 = 0;         % potentialul lui T2
```

c) Rulaţi codul scris până acum pentru a observa dacă s-au strecurat greşeli.

Acum începe partea de preprocesare. Mai întâi trebuie pregătite informaţiile legate de gridul de discretizare. Vom presupune că acesta este uniform pe cele două axe.

Exerciţiul 5.15:

a) Completaţi scriptul cu comenzile

```
%% ====== PREPROCESARE =======
%% alegerea gridului
my_grid = generare_grid_elcin(date);
```

b) Scrieţi o funcţie `generare_grid_elcin.m` cu următorul conţinut

```
function grid = generare_grid_elcin(date)
a = date.a;
b = date.b;
nx = 3;        % nr pct de discretizare pe Ox
ny = 3;        % nr pct de discretizare pe Oy
```

```
% pasi de discretizare pe Ox si Oy
hx = ..................... completati.....
hy = ..................... completati.....
% numarul total de noduri
N = ..................... completati.....
grid.nx = nx;
grid.ny = ny;
grid.hx = hx;
grid.hy = hy;
grid.x = linspace(0,a,nx);
grid.y = linspace(0,b,ny);
grid.N = N;
```

c) Ce reprezintă `grid.x` şi `grid.y`?

Urmează acum partea cea mai complicată, asamblarea sistemului de ecuaţii.

Exerciţiul 5.16:

a) Completaţi scriptul cu comenzile

```
%% asamblare sistem
[A,tl] = asamblare_mdf_elcin(date,my_grid);
```

b) Scrieţi o funcţie `asamblare_mdf_elcin.m` cu următorul conţinut:

```
function [A,tl] = asamblare_mdf_elcin(date,my_grid)
N = my_grid.N;
nx = my_grid.nx;
ny = my_grid.ny;
hx = my_grid.hx;
hy = my_grid.hy;
a = date.a;
b = date.b;
T1 = date.T1;
T2 = date.T2;
V1 = date.V1;
V2 = date.V2;

%% asamblarea sistemului de ecuatii
```

```
A = sparse(N,N);
tl = sparse(N,1);
%% nodurile interioare
for i = 2:nx-1
    for j = 2:ny-1
        k = (i-1)*ny + j;
        ksus = k+1;
        kjos = k-1;
        kst = k - ny;
        kdr = k + ny;
        A(k,k) = 2*(1/hx^2+1/hy^2);
        A(k,ksus) = -1/hy^2;
        ................. completati .................
    end
end
%% cond. de frontiera Neumann - segm vertical stanga
i = 1;
for j = 2:ny-1
    k = (i-1)*ny + j;
    ksus = k+1;
    kjos = k-1;
    kdr = k + ny;
    A(k,k) = 2*(1/hx^2+1/hy^2);
    A(k,ksus) = -1/hy^2;
    A(k,kjos) = -1/hy^2;
    A(k,kdr) = -2/hx^2;
end
%% cond. de frontiera Neumann - segm orizontal jos
j = 1;
for i = 2:nx-1
    ..................... completati ......................
end
%% cond. de frontiera Neumann - segm orizontal sus (partial)
iT1dr = floor(T1/a*nx)+1; % idx pe i atins in dreapta de T1
j = ny;
for i = iT1dr+1:nx-1
    ..................... completati ......................
```

```
end
%% cond. de frontiera Neumann - segm vertical dreapta (partial)
jT2sus = floor(T2/b*ny)+1; % idx pe j atins sus de T2
i = nx;
for j = jT2sus+1:ny-1
    ..................... completati ......................
end
%% cond de frontiera Neumann - colt stanga jos
k = 1;
kdr = ny+1;
ksus = 2;
A(k,k) = 2*(1/hx^2+1/hy^2);
A(k,ksus) = -2/hy^2;
A(k,kdr) =  -2/hx^2;
%% cond de frontiera Neumann - colt dreapta sus
..................... completati ......................
%% cond de frontiera Dirichlet - T1
j = ny;
for i = 1:iT1dr
    k = (i-1)*ny + j;
    A(k,k) = 1;
    tl(k) = V1;
end
%% cond de frontiera Dirichlet - T2
..................... completati ......................
```

c) Verificaţi codul scris cu comanda `mlint`. Executaţi programul scris până acum şi verificaţi că matricea coeficienţilor şi vectorul termenilor liberi au valorile pe care le-aţi obţinut la exerciţiul 5.13

În Matlab, rezolvarea acestui sistem este extrem de uşoară din punct de vedere al utilizatorului:

Exerciţiul 5.17:
Completaţi scriptul cu comenzile

```
%% ====== REZOLVARE =======
V = A\tl;
```

Urmează etapa de postprocesare. Vom reprezenta liniile echipotenţiale şi vom desena gridul de discretizare.

Exerciţiul 5.18:

a) Completaţi scriptul cu comenzile

```
%% ====== POSTPROCESARE =======
echipotentiale_elcin(my_grid,V);
draw_my_grid(my_grid);
```

b) Scrieţi o funcţie `echipotentiale_elcin.m` cu următorul conţinut

```
function echipotentiale_elcin(my_grid,V)
nx = my_grid.nx;
ny = my_grid.ny;
x = my_grid.x;
y = my_grid.y;
v = zeros(nx,ny);
for i = 1:nx
    for j = 1:ny
        k = (i-1)*ny + j;
        v(i,j) = V(k);
    end
end
.................completetati.........(ceva similar ati facut la capitolul 1)
```

c) Scrieţi o funcţie `draw_my_grid.m` cu următorul conţinut

```
function draw_my_grid(my_grid)
nx = my_grid.nx;
ny = my_grid.ny;
x = my_grid.x;
y = my_grid.y;
for i = 1:nx
    xcrt = x(i);
    hold on;
    plot([xcrt xcrt],[y(1),y(ny)],'k:');
end
.................completetati.........
```

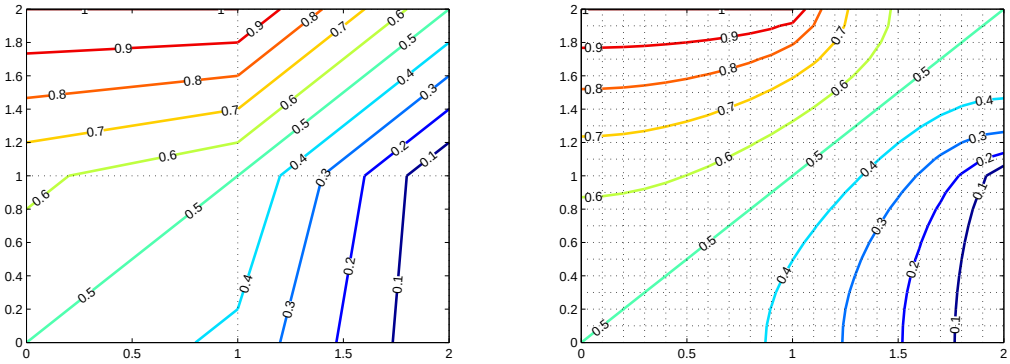

Figura 5.10: Linii echipotenţiale pentru un grid cu $n_x = n_y = 3$ (stânga) şi un grid cu $n_x = n_y = 21$ (dreapta).

d) Verificaţi că aţi lucrat corect, comparând rezultatele cu cele din fig. 5.10.

Exerciţiul 5.19:

a) Observaţi cu atenţie fig. 5.10 - stânga şi descrieţi ideea care stă la baza algoritmului de desenare al curbelor de echivalori.

b) Propuneţi îmbunătăţiri posibile pentru funcţia de asamblare.

c) Ştiind că $\mathbf{E} = -\text{grad}\, V$, scrieţi formulele cu care se poate calcula câmpul electric în nodurile gridului de discretizare.

5.2.3 Studiul propagării undei scalare

În acest paragraf vom rezolva cu metoda diferenţelor finite cel mai simplu exemplu de ecuaţie hiperbolică, şi anume *ecuaţia undei scalare*:

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2}, \tag{5.64}$$

unde mărimea scalară necunoscută $u(x,t) : [0,a] \times [0,T] \to \mathbb{R}$ depinde de o singură coordonată spaţială şi de timp, iar v este o constantă cunoscută. Soluţia acestei ecuaţii este o *undă* care se propagă cu viteză v.

Buna formulare a acestei probleme necesită impunerea

- *condiţiei iniţiale* $u(x,0) = h_0(x)$,
- *condiţiilor la capete* - relaţii în care intervin mărimile $u(0,t) = h_1(t)$ şi $u(a,t) = h_2(t)$.

Se poate demonstra că soluţia ecuaţiei undei scalare (5.64) este

$$u(x,t) = ud(x,t) + ui(x,t) + U_0, \tag{5.65}$$

unde $ud(x,t)$ se numeşte *undă directă* şi este o funcţie care se poate scrie sub forma

$$ud(x,t) = f(x - vt), \tag{5.66}$$

iar $ui(x,t)$ se numeşte *undă inversă* şi este o funcţie care se poate scrie sub forma

$$ud(x,t) = g(x + vt). \tag{5.67}$$

Funcţiile f şi g rezultă în mod univoc, din impunerea condiţiilor iniţiale şi condiţiilor la capete. Mărimea $f(x - vt)$ este o undă directă deoarece o anumită valoare a acestei funcţii, într-un anumit punct x şi într-un anumit moment de timp t se va regăsi după un interval de timp Δt în punctul $x + \Delta x$, unde $\Delta x = v\Delta t$. Unda directă se propagă deci în sensul pozitiv al axei Ox. Un raţionament similar se poate face pentru unda inversă.

Următorul exerciţiul vă permite să vizualizaţi conceptul de undă. El nu reprezintă rezolvarea numerică a vreunei ecuaţii diferenţiale.

Exerciţiul 5.20:

a) Scrieţi o funcţie cu următorul conţinut

```
function demo_unde()
a = 1; % domeniul spatial este [0,a]
T = 1; % domeniul temporal este [0, T]
N = 500; % nr pct pentru discretizarea spatiala
M = 500; % nr pct pentru discretizarea temporala

v = 1;   % viteza de propagare
dt = T/(M-1); % pas de discretizare temporal - uniform

x = linspace(0,a,N);
t = 0;
u = ud(x,t,v);
plot(x,u);
drawnow;
for j = 1:M
    t = t + dt;
    u = ud(x,t,v);
```

```
    plot(x,u);
    drawnow;
end

function rez = ud(x,t,v)
rez = sin(2*pi*t - 2*pi*x/v);
```

Observaţi rezultatul executării ei.
b) Modificăti funcţia astfel încât să vizualizaţi propagarea unei unde inverse.
c) Modificaţi funcţia astfel încât să vizualizaţi propagarea sumei dintre unda directă şi unda inversă. Veţi avea nevoie să fixaţi axele reprezentării grafice. Pentru aceasta, înainte de prima comanda `drawnow` adăugaţi

```
xmin = 0; xmax = a; ymin = -2.2; ymax = 2.2;
axis([xmin, xmax,ymin,ymax]);
```

şi înainte de a doua comandă `drawnow` adăugaţi doar comanda `axis` ca mai sus. Ceea ce obţineţi sunt *unde staţionare*.

Vom concepe acum un algoritm bazat pe metoda diferenţelor finite, care rezolvă ecuaţia (5.64). Ca exemplu numeric vom considera $a = 1$ m, $v = 1$ m/s.

Avem nevoie de o discretizare spaţială şi de una temporală. Pentru a simplifica prezentarea, vom presupune că ambele discretizări sunt uniforme şi vom nota cu Δz pasul discretizării spaţiale şi cu Δt pasul discretizării temporale. Vom nota cu N numărul de puncte de discretizare spaţiale şi cu M numărul de paşi de timp simulaţi. În consecintă timpul maxim de simulare este $T = M\Delta t$.

Într-un prim exemplu, vom considera condiţii iniţiale nule şi condiţii Dirichlet la ambele capete:

- *condiţia iniţială* nulă $u(x, 0) = h_0(x) = 0$,
- *condiţia la capătul din stânga* - excitaţia cu un impuls Gauss: $u(0, t) = h_1(t) = exp(-(t - T/10).^2/(2 * (T/50)^2))$
- *condiţia la capătul din dreapta* $u(a, t) = h_2(t) = 0$.

Mărimea $u(x, t)$ este discretizată atât în spaţiu cât şi în timp. Prin rezolvarea cu metoda diferenţelor finite, vom obţine aproximaţii pentru valorile reale $u(x_k, t_j)$.

$$u(x_k, t_j) \approx u_k^{(j)}, \quad k = 1, \ldots, N, \quad j = 1, \ldots, M. \tag{5.68}$$

Alegând pentru derivatele ce intervin în (5.64) formule de derivare ce provin din polinomul de interpolare de ordin doi, rezultă următoarea relaţie discretizată:

$$\frac{u_k^{(j-1)} - 2u_k^{(j)} + u_k^{(j+1)}}{(\Delta t)^2} = v^2 \frac{u_{k-1}^{(j)} - 2u_k^{(j)} + u_{k+1}^{(j)}}{(\Delta x)^2}, \tag{5.69}$$

de unde rezultă că mărimea u, într-un anumit punct k şi la un anumit moment de timp $j+1$ depinde explicit de valoarea în acel punct şi în cele învecinate la momentul de timp anterior j şi de valoarea în acel punct la momentul $j-1$:

$$u_k^{(j+1)} = \left(\frac{v\Delta t}{\Delta x}\right)^2 \left(u_{k-1}^{(j)} - 2u_k^{(j)} + u_{k+1}^{(j)}\right) + 2u_k^{(j)} - u_k^{j-1}, \quad k = 2, \ldots, N-1, \quad j = 0, \ldots, M-1. \tag{5.70}$$

Momentul -1 îl vom considera identic cu momentul iniţial 0.

Exerciţiul 5.21:

a) Scrieţi un script cu următorul conţinut

```
clear all;
a = 1; % domeniul spatial este [0,a]
T = 1; % intervalul de timp necesar propagarii pe distanta a
N = 500; % nr pct pentru discretizarea spatiala
M = 500; % nr pct pentru discretizarea temporala a lui T

v = 1; % viteza de propagare
dx = a/(N-1); % pas de discretizare spatial - uniform
dt = T/(M-1); % pas de discretizare temporal - uniform

r = v*dt/dx;
r2 = r^2;

% conditia initiala
x = linspace(0,1,N);
solv = zeros(1,N);
solvv = solv;
plot(x,solv);
xmin = 0; xmax = a; ymin = -2.2; ymax = 2.2;
axis([xmin, xmax,ymin,ymax]);

MM = 2*M;
```

```
for idx_t = 1:MM
    t = idx_t*dt;
    % conditie de frontiera la stanga
    sol(1) = exp(-(t - T/10).^2 / (2 * (T/50)^2));
    % conditie de frontiera la dreapta
    sol(N) = 0;
    for idx_x = 2:N-1
        ..............  completati  .................
    end
    solvv = solv;
    solv = sol;
    plot(x,solv);
    title(sprintf('tmax = %4.2e, no_t = %d, idx_t = %d',T,MM,idx_t))
    axis([xmin, xmax,ymin,ymax]);
    drawnow
end
```

b) Observaţi ce se întâmplă după reflexie (fig. 5.11). În momentul reflexiei, apare o undă inversă deoarece condiţia Dirichlet la capătul din dreapta impune ca mărimea să fie zero. În consecinţă, unda inversă la capătul din dreapta este exact opusul undei directe, şi ea se propagă în sens contrar axei Ox.
c) Cazul implementat mai sus corespunde situaţiei $\Delta x = v\Delta t$. Observaţi ce se întâmplă dacă $\Delta x > v\Delta t$. Alegeţi de exemplu $N = 100$, $M = 500$.
c) Observaţi ce se întâmplă dacă $\Delta x < v\Delta t$. Alegeţi de exemplu $N = 100$, $M = 500$.

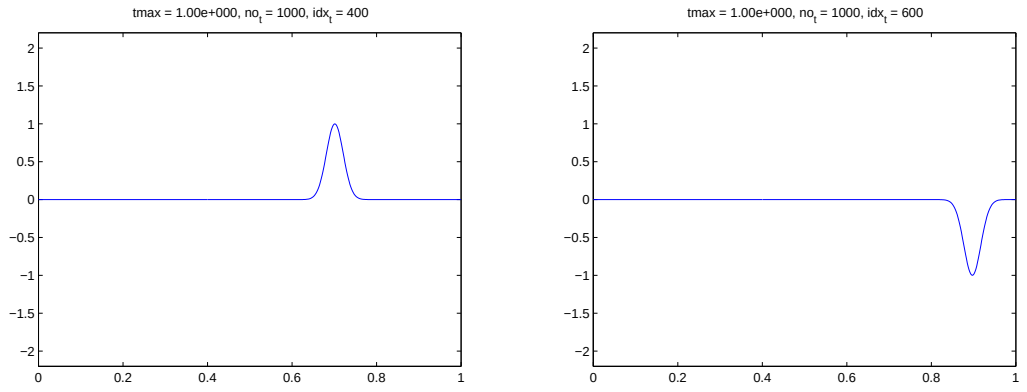

Figura 5.11: Propagarea unui impuls Gauss (stânga) şi rezultatul reflexiei după atingerea unei frontiere pe care s-a impus condiţie Dirichlet nulă (dreapta).

În concluzie, atunci când o problemă necesită atât o discretizare spaţială cât şi una temporală, pentru ca soluţia numerică să fie stabilă, este necesar să fie îndeplinită *condiţia lui Courant*

$$|v|\Delta t \leq \Delta x. \tag{5.71}$$

Exerciţiul 5.22:

a) Reveniţi la cazul $\Delta x = v\Delta t$ şi schimbaţi condiţia la capătul din stânga cu $u(0,t) = h_1(t) = sin(10\pi t)$.

b) Observaţi apariţia undelor staţionare, de data aceasta obţinute prin rezolvarea ecuaţiei undei scalare (fig. 5.12).

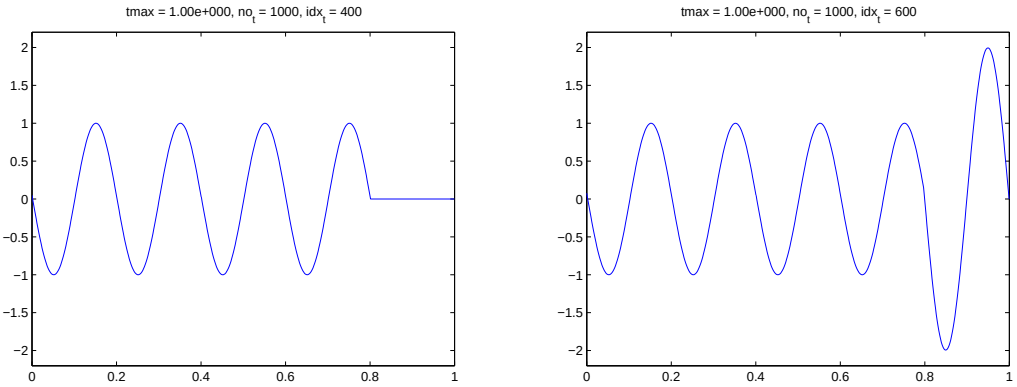

Figura 5.12: Propagarea unui unde sinusoidale (stânga) şi rezultatul reflexiei după atingerea unei frontiere pe care s-a impus condiţie Dirichlet nulă (dreapta).

În problemele în care apare propagare, este util uneori ca frontiera domeniului spaţial să fie modelată ca o frontieră absorbantă, "invizibilă" din punct de vedere al propagării mărimilor în domeniul spaţial modelat. O condiţie de frontieră absorbantă pentru capătul din dreapta al problemei studiate este

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial z} = 0. \tag{5.72}$$

Exerciţiul 5.23:

a) Implementaţi condiţia de frontieră absorbantă (5.72) folosind o discretizare cu diferenţe progresive de ordinul 1 pentru derivata temporală şi diferenţe regresive de ordinul 1 pentru derivata spaţială:

$$v\frac{u_N^{(j)} - u_{N-1}^{(j)}}{\Delta z} + \frac{u_N^{(j+1)} - u_N^{(j)}}{\Delta t} = 0. \tag{5.73}$$

b) Verificaţi codul implementat pentru cazul în care la capătul din stânga este impusă condiţia Dirichlet folosită la exerciţiul 5.21

Capitolul 6

Temă de stabilit

- Generarea sistemelor de stare pentru circuite liniare; analiza in frecventa folosind esantionarea adaptiva a frecventelor
- Analiza de circuite liniare in regim tranzitoriu
- Rezolvare de sisteme neliniare; aplicaţie - circuite neliniare in regim stationar
- Elemente finite; aplicaţie - problema de câmp
- Integrale finite; aplicaţie - problema de câmp
- Elemente de frontieră; aplicaţie - problema de câmp
- Algoritmi de optimizare