

G a b r i e l a
CIUPRINA

**Algoritmi numerici
prin exerciții și
implementări în Matlab**

Gabriela Ciuprina

Algoritmi numerici

prin exerciții și implementări în Matlab

**MATRIX ROM
București 2013**

MATRIX ROM
C.P. 16 – 162
062510 – BUCUREȘTI
tel. 021.4113617, 031.4012438, 0372743840
fax 021.4114280
email: office@matrixrom.ro
www.matrixrom.ro

**Editura MATRIX ROM este acreditată de
CONSILIUL NAȚIONAL AL CERCETĂRII ȘTIINȚIFICE DIN ÎNVĂȚĂMÂNTUL SUPERIOR**

Referenți științifici: Prof. univ. dr. ing. F.M.G. Tomescu
Prof. univ. dr. ing. Daniel Ioan
Universitatea « Politehnica » București

Descrierea CIP a Bibliotecii Naționale a României
CIUPRINA, GABRIELA
Algoritmi numerici prin exerciții și implementări în Matlab /
Gabriela Ciuprina. - București : Matrix Rom, 2013
Bibliogr.
ISBN 978-606-25-0009-2

519.6:621.3(075.8)
004.42 MATLAB

ISBN 978 – 606 – 25 – 0009 – 2

Gabriela Ciuprina

Algoritmi numerici

prin exerciții și implementări în Matlab

Referenți științifici:

Prof.dr.ing.F.M.G. Tomescu

Prof.dr.ing. Daniel Ioan

2013

Introducere

Această carte constituie o parte a suportului didactic pentru disciplina *Algoritmi numerici*, din programul *Inginerie electrică și informatică aplicată*, al Facultății de Inginerie Electrică din Universitatea Politehnica București. Lucrarea poate fi utilă tuturor celor care nu au experiență în metode numerice și care doresc să înțeleagă conceptele de bază și modul de gândire algoritmic, structurat, într-un mod practic, prin efectuarea de experimente numerice într-un mediu de programare.

Lucrarea este structurată în 5 capitole care tratează următoarele aspecte: familiarizarea cu Matlab (cap.1), evaluarea algoritmilor (cap.2) cu aplicație la conceperea, implementarea și evaluarea unor algoritmi numerici pentru analiza circuitelor rezistive liniare (cap.3), metode de interpolare a funcțiilor (cap.4) și metode de derivare numerică (cap.5) aplicate pentru rezolvarea unor ecuații diferențiale.

Prezentarea conceptelor alternează cu propunerea de exerciții de raționament sau implementare, pentru care se dau indicații și răspunsuri. În acest fel, cititorul poate înțelege și învăța mai ușor, prin experimente numerice. O parte din exercițiile propuse au fost preluate, îmbunătățite și adaptate din lucrarea *Modelarea numerică a câmpului electromagnetic prin programe Scilab*. Capitolele referitoare la evaluarea algoritmilor, analiza circuitelor și rezolvarea unor ecuații diferențiale de tip hiperbolic sunt complet noi.

Autoarea mulțumește referenților științifici Prof.dr.ing.F.M.G. Tomescu și Prof.dr.ing. Daniel Ioan pentru comentariile și sugestiile de îmbunătățire care au fost incluse în această carte. Autoarea mulțumește și trei Ioana Popescu, studentă la Facultatea de Inginerie Electrică, pentru efectuarea tuturor exercițiilor propuse și sugestiile de îmbunătățire ale lor.

Orice comentarii și sugestii în vederea îmbunătățirii acestui material sunt binevenite la adresa gabriela.ciuprina@upb.ro.

*Sunt lucruri pe care, câtă vreme nu le înveți, nu le poți face;
sunt altele pe care, câtă vreme nu le faci, nu le poți învăța.*

Proverb Armenesc

Cuprins

| | | |
|----------|---|-----------|
| 1 | Familiarizarea cu mediul de lucru | 1 |
| 1.1 | Variabile și constante | 1 |
| 1.2 | Atribuirii și expresii | 2 |
| 1.3 | Generarea vectorilor și matricelor | 6 |
| 1.4 | Instrucțiuni grafice | 9 |
| 1.5 | Programare în <i>Matlab</i> | 10 |
| 1.5.1 | Editarea programelor | 10 |
| 1.5.2 | Operații de intrare/ieșire | 10 |
| 1.5.3 | Structuri de control | 12 |
| 1.5.4 | Funcții | 14 |
| 1.6 | Implementări eficiente ale operațiilor cu matrice | 15 |
| 1.7 | Facilități de post-procesare | 17 |
| 1.8 | Referințe utile | 20 |
| 2 | Evaluarea algoritmilor | 21 |
| 2.1 | Timpul de calcul | 21 |
| 2.2 | Necesarul de memorie | 27 |
| 2.3 | Erori în calculele numerice | 34 |
| 2.3.1 | Erori de rotunjire | 36 |
| 2.3.2 | Erori inerente | 37 |
| 2.3.3 | Erori de trunchiere | 39 |

| | | |
|----------|---|------------|
| 3 | Analiza circuitelor electrice rezistive liniare | 41 |
| 3.1 | Metoda potențialelor nodurilor | 42 |
| 3.2 | Structuri de date | 46 |
| 3.3 | Etapa de preprocesare | 48 |
| 3.4 | Etapa de rezolvare | 52 |
| 3.5 | Etapa de postprocesare | 53 |
| 3.6 | Analiza complexității. Optimizarea algoritmului. | 53 |
| 4 | Interpolarea polinomială | 67 |
| 4.1 | Cazul 1D | 67 |
| 4.1.1 | Formularea problemei | 67 |
| 4.1.2 | Interpolarea globală | 68 |
| 4.1.3 | Interpolarea pe porțiuni | 75 |
| 4.2 | Cazul 2D | 82 |
| 5 | Derivarea numerică. Metoda diferențelor finite. | 89 |
| 5.1 | Formule de derivare în cazul unidimensional | 89 |
| 5.2 | Metoda diferențelor finite | 93 |
| 5.2.1 | Studiul regimului tranzitoriu al unui circuit de ordinul 1 | 93 |
| | Formularea corectă a problemei | 94 |
| | Rezolvarea cu diferențe finite | 96 |
| 5.2.2 | Studiul regimului electrocinetic staționar al unui conductor 2D . . . | 102 |
| | Formularea corectă a problemei | 103 |
| | Rezolvarea cu diferențe finite | 104 |
| 5.2.3 | Studiul propagării undei scalare | 114 |
| | Referințe | 121 |

Capitolul 1

Familiarizarea cu mediul de lucru

Aceast capitol are ca scop familiarizarea cu mediul de lucru, în vederea rezolvării exercițiilor din capitolele următoare, și prezintă doar câteva aspecte ale lucrului cu *Matlab*. Cei pe deplin familiarizați cu acest mediu pot sări peste exercițiile de început. Cei cu mai puțină experiență trebuie să citească cu atenție documentul *Matlab - getting started*.

1.1 Variabile și constante.

În rezolvarea exercițiilor din această carte veți 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.

Exercițiul 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]
```

Exercițiul 1.2:

Știind 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?

Exercițiul 1.3:

Ce reprezintă e în următoarea instrucțiune?

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

Observație: 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 și memoria disponibilă se folosește comanda `who`. Pentru a vizualiza câtă memorie ocupă variabilele existente la un moment dat, se folosește comanda `whos`. Dacă se dorește eliberarea memoriei de toate variabilele generate se folosește comanda `clear`.

Exercițiul 1.4:

Executați instrucțiunea `clear`. Acum nu mai există nici o variabilă în mediul de lucru (verificați cu `who`). Executați totuși, următoarele instrucțiuni. Comentați rezultatul lor.

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

1.2 Atribuirii și expresii

Instrucțiunea de atribuire are sintaxa:

```
variabila = expresie
```

sau simplu:

```
expresie
```

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

• *Operatorii aritmetici*¹ recunoscuți de **Matlab** sunt:

- + adunare;
- scădere;
- * înmulțire;
- / împărțire la dreapta;
- \ împărțire la stânga;
- ^ ridicare la putere.

Pentru transpunerea unei matrice se folosește 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 și conjugata dacă aceasta are elemente complexe.

Observații:

1. Adunarea și scăderea pot fi efectuate:
 - între două matrice cu aceleași dimensiuni;
 - între o matrice și un număr (caz în care numărul este adunat, respectiv scăzut din fiecare din elementele matricei).
2. Înmulțirea poate fi efectuată:
 - între două matrice dacă lungimea liniei primei matrice este egală cu lungimea coloanei celei de a doua matrice;
 - între un număr și o matrice (caz în care numărul este înmulțit cu fiecare din elementele matricei);
 - între două matrice cu aceleași dimensiuni (element cu element), caz în care operatorul * este precedat de un punct, ca în exemplul:

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

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

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

echivalent cu:

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

$$\rightarrow X = B * \text{inv}(A)$$

sau cu:

$$\rightarrow X = B * A^{-1}$$

- la stânga:

$$\rightarrow X = A \setminus B$$

echivalent cu:

$$\rightarrow X = \text{inv}(A) * B$$

sau cu:

$$\rightarrow X = A^{-1} * B$$

Dacă A este o matrice dreptunghiulară de dimensiuni $m \times n$, iar b este un vector coloană cu m elemente, atunci împărțirea la stânga $x = A \setminus b$ calculează soluția ecuației $Ax = b$ în sensul celor mai mici pătrate.

- împărțirea unei matrice la un număr (să îl notăm cu u):

$$\rightarrow Y = A / u$$

- împărțirea element cu element a matricelor de dimensiuni egale:

$$\rightarrow C = A ./ B$$

sau

$$\rightarrow C = A .\setminus B$$

4. Ridicarea la putere A^p se face astfel ²:

- dacă p este un întreg pozitiv: dacă matricea A este pătrată atunci A se înmulțește cu ea însăși 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 înmulțește cu ea însăși de $-p$ ori; dacă matricea A este dreptunghiulară atunci se

²Nu sunt descrise toate situațiile posibile.

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 și 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 și valorilor proprii ale inversei matricei; dacă matricea A este dreptunghiulară, atunci se ridică la puterea p fiecare element din matricea A .

• *Operatorii de relație*³ recunoscuți de **Matlab** sunt:

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

Aceștia permit testarea unor condiții, rezultatul având valoarea de adevăr fals (0) sau adevărat (1). Dacă operanzii sunt matrice de dimensiuni egale, atunci operațiile logice se fac între elementele de pe aceleași poziții, iar rezultatul este o matrice cu elementele 0 și 1.

• *Operatori logici*⁴ recunoscuți de **Matlab** sunt:

- & conjuncția logică;
- | disjuncția logică;
- ~ negația logică.

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

• *Funcții elementare*. Operanzii unor expresii pot fi și apeluri de funcții elementare (de exemplu trigonometrice) sau alte funcții cunoscute. Aceste funcții aplicate unei matrice acționează asupra fiecărui element în mod independent.

³Operatorii de relație se aplică unor operanzi aritmetici iar rezultatul este logic.

⁴Operatorii logici se aplică unor operanzi logici iar rezultatul este logic.

Exercițiul 1.5:

Fie $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $b = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$ și $v = \begin{bmatrix} 4 & 3 \end{bmatrix}$. Care sunt comenzile **Matlab** pentru rezolvarea ecuației $A(v^T + x) = b$.

1.3 Generarea vectorilor și matricelor

- Vectorii ai căror elemente formează o progresie aritmetică pot fi generați cu construcția:

valin : pas : valfin

Exercițiul 1.6:

Comentați rezultatele următoarelor instrucțiuni:

```
>> 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 generați cu construcția:

logspace(d1, d2, n)

Exercițiul 1.7:

a) Care este semnificația mărimilor $d1, d2, n$ din comanda `logspace` ?

b) Ce generează comanda `linspace` ?

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

$A = [X \ Y; \ U \ V];$

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

Exercițiul 1.8:

Care este rezultatul comenzii:

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

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

Exercițiul 1.9:

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

Notați rezultatele și comentați 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 funcțiilor:
`eye` matrice cu elementele unitare pe diagonală și nule în rest;
`zeros` matrice nulă;
`ones` matrice cu toate elementele unitare;
`rand` matrice cu elemente aleatoare în intervalul $(0,1)$;
`diag` construiește o matrice cu o anumită diagonală, sau extrage diagonală dintr-o matrice.

Exercițiul 1.10:

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

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

Exercițiul 1.11:

Comentați 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 execuției unui program. Pentru a obține dimensiunea unei matrice X se folosește instrucțiunea:

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

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

```
length(v)
```

care are semnificația `max(size(v))`.

Exercițiul 1.12:

Definiți o matrice oarecare B (de exemplu cu 3 linii și 4 coloane). Executați și comentați următoarele instrucțiuni:

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

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

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

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

1.4 Instrucțiuni grafice

Funcția 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. Instrucțiunea:

```
plot(A)
```

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

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

Exercițiul 1.13:

Realizați graficul din figura 1.1. El reprezintă funcția $y(t) = 10 * \sin(t)$ pentru $t \in [0, 4\pi]$. Puneți etichete axelor, rastrul și legenda ca în figură.

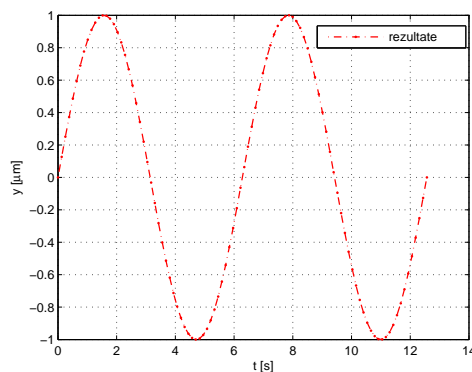


Figura 1.1: Acest grafic trebuie obținut la exercițiul 1.13.

1.5 Programare în *Matlab*

Matlab permite utilizarea unor structuri de control (decizii, cicluri, definiții de funcții) 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 ușurința cu care se pot postprocesa rezultatele, ținând seama de facilitățile grafice ale sale. Un alt avantaj deosebit de important este acela că se pot efectua operații cu vectori și matrice, operații care sunt deosebit de eficiente mai ales pentru vectori și matrice rare. În acest fel *Matlab* este un mediu foarte potrivit pentru testarea unor noi algoritmi.

1.5.1 Editarea programelor

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

Un script în *Matlab* are următoarea structură posibilă:

```
% comentarii
.....
instrucțiune; % fără afișarea rezultatului
instrucțiune % cu afișarea rezultatului
```

Exercițiul 1.14:

Scrieți comenzile cu care ați rezolvat exercițiul 13, într-un fișier numit *mytest.m*, din directorul *AlgoritmiNumerici/Tema1* și apoi executați-l fie cu comanda:

```
>> mytest
```

fie apăsând butonul *Run* din editor. Observați ce se întâmplă dacă la sfârșitul fiecărei instrucțiuni adăugați terminatorul “;”.

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

1.5.2 Operații de intrare/ieșire

- *Introducerea datelor.*

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

```
a = 5
```

În cazul unui program scris într-un fișier, este mai convenabil să se folosească funcția `input`. Funcția `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 șir de caractere ce va fi afișat, ajutând utilizatorul la identificarea informației ce trebuie introdusă. De exemplu:

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

- *Inspectarea și afișarea rezultatelor*

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

```
>> a
```

pentru ca interpretorul să afișeze valoarea lor.

Dacă se dorește eliminarea afișării numelui variabilelor din fața valorii sale, atunci se folosește funcția `disp`:

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

Această funcție poate fi folosită și pentru afișarea textelor, de exemplu:

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

Formatul în care sunt afișate valorile numerice poate fi modificat de utilizator cu ajutorul instrucțiunii:

```
format
```

Operația de ieșire se poate realiza și prin apelul funcției `fprintf` în instrucțiuni de forma:

```
fprintf('format',variabile)
```

în care “variabile” sunt variabilele care vor fi scrise în formatul corespunzător instrucțiunii, iar “format” este un șir de caractere ce descrie formatul de afișare. Sunt recunoscute următoarele construcții, 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 șirul “format” sunt afișate ca atare, de exemplu:

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

Afișarea rezultatelor se poate face și grafic (vezi paragraful 1.4).

Exercițiul 1.15:

Scrieți într-un fișier un program prietenos care va permite introducerea de la consola *Matlab* a două numere reale, va calcula suma lor, și va afișa rezultatul în formatul cu exponent.

1.5.3 Structuri de control

- *Decizii*

Decizia simplă:

| Pseudolimbaj | <i>Matlab</i> | Observații |
|--|------------------------------------|---|
| dacă condiție atunci instrucțiuni | if condiție instrucțiuni end | “condiție” este o expresie care este evaluată, iar dacă rezultatul este adevărat (T), atunci se execută “instrucțiuni”, altfel se execută prima instrucțiune ce urmează după end . |

Decizia cu alternativă:

| Pseudolimbaj | <i>Matlab</i> | Observații |
|---|--|---|
| dacă condiție atunci instrucțiuni1 altfel instrucțiuni2 | if condiție instrucțiuni1 else instrucțiuni2 end | “condiție” este o expresie care este evaluată, iar dacă rezultatul este adevărat (1), atunci se execută “instrucțiuni1”, iar dacă rezultatul este fals (0), se execută “instrucțiuni2”. |

Decizia de tip selecție:

| Pseudolimbaj | <i>Matlab</i> | Observații |
|---|--|---|
| dacă condiție1 atunci instrucțiuni1 altfel dacă condiție2 instrucțiuni2 altfel instrucțiuni3 | if condiție1 instrucțiuni1 elseif condiție2 instrucțiuni2 else instrucțiuni3 end | Pot exista oricâte alternative de selecție. |
| | switch expresie, case expresie1 instrucțiuni1, case expresie2 instrucțiuni2, otherwise instrucțiuni, end | Pot exista oricâte cazuri. “instrucțiuni1” sunt executate dacă expresie == expresie1, etc. |

- *Cicluri*

Ciclul cu test inițial:

| Pseudolimbaj | <i>Matlab</i> | Observații |
|--|---|---|
| cât timp condiție instrucțiuni | while condiție instrucțiuni end | Se repetă corpul ciclului, adică “instrucțiuni”, cât timp “condiție” este adevărată. S-ar putea ca “instrucțiuni” 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. “instrucțiuni” nu sunt executate niciodată dacă vectorul “valin:pas:valfin” este incorect definit (vid) sau dacă “expresie” este matricea vidă.

| Pseudolimbaj | <i>Matlab</i> | Observații |
|---|---|---------------------------------|
| pentru contor = valin, valfin, pas instrucțiuni | for contor = valin : pas : valfin, instrucțiuni end | Forma a doua este cea generală. |
| | for variabila = expresie, instrucțiuni end | |

Ieșirile forțate din cicluri se pot face cu instrucțiunea **break**.

Exercițiul 1.16:

Scrieți un program care să determine cel mai mare număr întreg n pentru care 10^n poate fi reprezentat în **Matlab**. Indicație: folosiți un ciclu cu test, în care condiția de intrare în ciclu testează egalitatea dintre 10^n și constanta **Inf**.

1.5.4 Funcții

Funcțiile sunt rutine **Matlab** care accepta parametri de intrare și întorc parametri de ieșire.

Este bine ca fiecare funcție să fie definită într-un fișier separat, care are același nume ca numele funcției și extensia *.m. Un fișier conținând o funcție trebuie să înceapă astfel:

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

unde y_i sunt variabilele de ieșire calculate în funcție de variabilele de intrare x_j și, eventual, de alte variabile existente în **Matlab** în momentul execuției funcției. Se recomandă ca acest fișier să se numească nume_functie.m.

Exercițiul 1.17:

Edițați un fișier numit “combin.m” cu următorul conținut:

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

și un fișier numit “main_combin.m” cu următorul conținut:

```
clear all;
a = input('Introduceți a. a = ');
b = input('Introduceți 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));
```

Executați în **Matlab** comenzile din “main_combin.m”:

```
>> main_combin;
```

- a) Explicați comanda `disp(sprintf(...))`;
- b) Comentați necesitatea instrucțiunii `clear all`.
- c) Rulați programul pas cu pas și urmăriți fereastra *Workspace*
- d) Adaugați în funcția `combin`, instrucțiunea inutilă $z = 7$. Rulați programul pas cu pas și urmăriți fereastra *Workspace*. Comentați.
- e) Pe exemplul de la punctul c), dați în consola Matlab comanda

```
>> mlint main_combin
```

Comentați rezultatul ei, după ce vă informați asupra comenzii `mlint`.

IMPORTANT: Folosirea comenzii `mlint` trebuie să fie o practică obișnuită în cazul folosirii limbajului Matlab. În versiunile noi de Matlab, rezultatul acestei comenzi este sugerat de un pătrățel de culoare verde dacă nu sunt erori și nici avertizări, galben dacă sunt doar avertizări și roșu dacă există erori.

1.6 Implementări eficiente ale operațiilor cu matrice

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

Exercițiul 1.18:

Scrieți o funcție `ps_v1` care să calculeze produsul scalar a doi vectori a și b de dimensiune $1 \times n$ prin implementarea formulei $\sum_{i=1}^n a_i b_i$ și o altă funcție `ps_v2` care să implementeze calculul produsului scalar folosind operațiile cu matrice $a * b'$. Verificați corectitudinea funcțiilor scrise cu ajutorul unui script în care să comparați rezultatele.

Exercițiul 1.19:

Comentați conținutul scriptului de mai jos. Executați-l și comparați rezultatul cu cel din figura 1.2.

```
clear all;  
nm = linspace(1e6,1e7,10);  
N = length(nm);  
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');

```

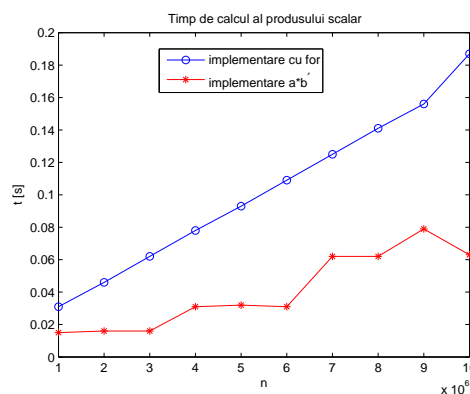


Figura 1.2: Timpul de calcul al produsului scalar în funcție de dimensiunea vectorilor. Comparație între cele două implementări propuse la exercițiul 1.19.

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

Exercițiul 1.20:

Scrieți o funcție `pmv_v1` care să calculeze produsul dintre o matrice pătrată a de dimensiune n și un vector coloană b prin implementarea formulei $c_i = \sum_{j=1}^n a_{ij}b_j$ și o altă funcție `pmv_v2` care să implementeze calculul aceluiași produs folosind operațiile cu matrice $c = a * b$. Comparați eficiența celor două implementări. Comparați rezultatele obținute cu graficele din figura 1.3.

Exercițiul 1.21:

Scrieți o funcție `pmm_v1` care să calculeze produsul dintre două matrice pătrate a și b de dimensiune n prin implementarea formulei $c_{i,j} = \sum_{k=1}^n a_{i,k}b_{k,j}$ și o altă funcție `pmm_v2` care să implementeze calculul aceluiași produs folosind operațiile cu matrice $c = a * b$. Comparați eficiența celor două implementări. Comparați rezultatele obținute cu graficele din figura 1.4.

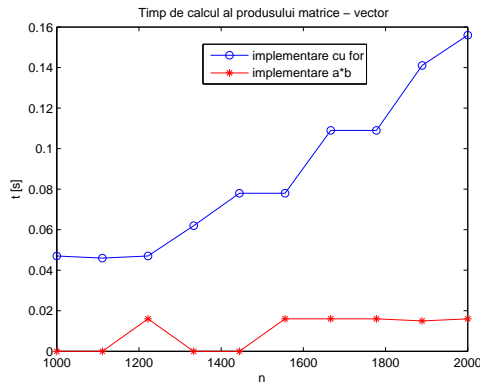


Figura 1.3: Timpul de calcul al produsului dintre o matrice pătrată și un vector în funcție de dimensiunea problemei. Comparație între cele două implementări propuse la exercițiul 1.20.

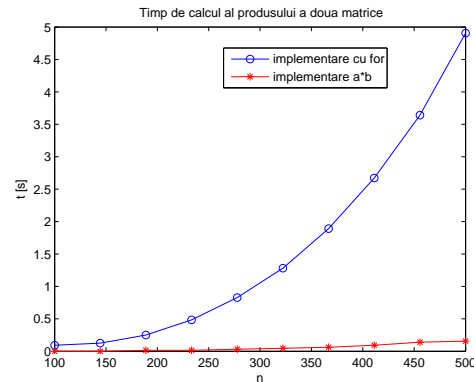


Figura 1.4: Timpul de calcul al produsului dintre două matrice pătrate în funcție de dimensiunea problemei. Comparație între cele două implementări propuse la exercițiul 1.21.

1.7 Facilități de post-procesare

Acest exercițiu urmărește exersarea facilităților de postprocesare ale programului, necesare temelor viitoare. Exercițiul propus se referă la reprezentarea grafică a câmpului electric produs de o sarcină punctuală plasată în vid.

Exercițiul 1.22:

Fie o sarcină punctuală $q = 10^{-10}$ C, situată în vid ($\epsilon_0 = 8.8 \cdot 10^{-12}$ F/m), în punctul de coordonate $(x_0, y_0, z_0) = (0, 0, 0)$.

- a) Să se reprezinte grafic liniile echipotențiale în planul $z = 0$;
- b) Să se reprezinte grafic potențialul $V(x, y, 0)$;
- c) Să se reprezinte grafic potențialul $V(x, 0, 0)$;
- 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 poziție

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

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 poziție în punctul în 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 unește punctul în care se afla sarcina cu punctul în care se calculează 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 și cu ajutorul potențialului 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 echipotențiale și vectorii se pot trasa cu ajutorul comenzilor `contour` și `quiver`. Pentru aceasta, se vor determina valorile potențialului și ale componentelor câmpului într-o mulțime discretă de puncte din spațiu, plasate în nodurile unei rețele bidimensionale generată de un vector de abscise și un vector de ordinate.

Pentru rezolvarea problemei se va scrie o funcție care calculează potențialul și câmpul unei sarcini punctuale situată într-un punct oarecare din spațiu. Un exemplu de astfel de funcție este următorul.

```
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 funcția 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);
```

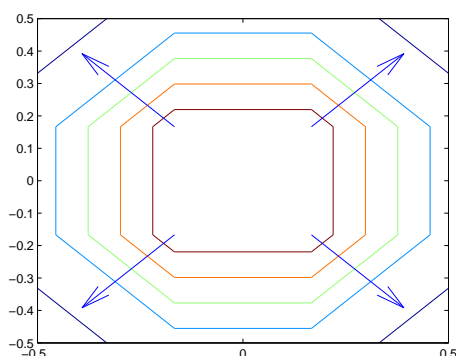


Figura 1.5: Efectul comenzilor `contour` și `quiver`. Curbele au această formă datorită gridului extrem de rar folosit.

Exercițiul 1.23:

a) Observați și comentați alura echipotențialelor pentru diferite grade de finețe ale rețelei folosite.

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

c) Propuneți un algoritm care să genereze o rețea adaptată problemei, astfel încât liniile echipotențiale să aibă racordări cât mai netede.

1.8 Referințe utile

- Documentația 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, și în consecință, vor exista mai mulți algoritmi diferiți 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 soluție corectă într-un timp scurt și 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 și atunci vom urmări să avem erori rezonabile pentru o anumită aplicație. 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 contradicție cu criteriul referitor la memoria necesară algoritmului.

În consecință, la alegerea unui algoritm potrivit pentru o aplicație dată, trebuie făcut un compromis între trei criterii: timpul de calcul necesar obținerii soluției, necesarul de memorie și acuratețea soluției. Exercițiile 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 performanțele intrinseci ale calculatorului și limbajului de programare folosit și evident, de algoritm. Este util însă să poată fi apreciată doar calitatea algoritmului și 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 relația dintre timpul de calcul exprimat în număr de operații elementare și 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 ecuații algebrice liniare, dimensiunea problemei este dimensiunea sistemului. Uneori, este potrivit să exprimăm dimensiunea problemei în funcție 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 și numărul de laturi.

Timpul de calcul este de fapt suma timpilor necesari pentru executarea tuturor instrucțiunilor algoritmului. Nu toate instrucțiunile durează la fel de mult, de aceea, estimarea complexității nu se poate face foarte precis, dar nici nu este necesar acest lucru. Se alege o *operație elementară*, considerată a fi cea care durează cel mai mult (de exemplu evaluarea unei anumite funcții) și se numără câte astfel de operații 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 operație elementară orice operație algebrică (adunare, scădere, înmulțire, împărțire) și neglijând timpul de calcul petrecut în declarații și atribuiri, rezultă că în algoritmul de mai sus se fac $2n$ operații elementare, câte două (o adunare și o înmulțire) pentru fiecare valoare a contorului i . Timpul de calcul este proporțional cu dimensiunea problemei, în acest caz n . Un astfel de algoritm se spune că este un *algoritm liniar*, sau de *ordinul 1* și se notează $T = O(n)$. În cazul produsului scalar a doi vectori de dimensiune n , putem scrie $T = O(2n) \approx O(n)$. Constanta 2 nu este atât de relevantă, ceea ce este important este dependența de dimensiunea problemei.

Fie acum cazul înmulțirii 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 $2n$ operații (judecând exact ca la produsul scalar), deci pentru toate cele n valori ale lui i se vor face $2n^2$ operații elementare. Un algoritm pentru care timpul de calcul T este proporțional cu pătratul dimensiunii problemei n se spune că este un *algoritm pătratic, sau de ordinul 2* și se notează $T = O(n^2)$. Înmulțirea dintre o matrice și un vector are deci complexitatea $T = O(2n^2) \approx O(n^2)$.

Exercițiul 2.1:

Scrieți pseudocodul pentru înmulțirea a două matrice pătrate de dimensiune n și evaluați ordinul de complexitate din punct de vedere al timpului de calcul.

Exercițiul 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 și  $n$  coloane
tablou real  $x[n]$ 
tablou real  $y[m]$ 
; rezultat
tablou real  $b[m]$ 
; alte declarații
...
pentru  $i = 1, m$ 
     $b_i = y_i$ 
    pentru  $j = 1, n$ 
         $b_i = b_i + a_{ij}x_j$ 
    •
•
retur

```

- Estimați ordinul de complexitate din punct de vedere al timpului de calcul. Particularizați rezultatul pentru cazul $m = n$.
- Implementați procedura `gaxpy` în Matlab.
- Cum veți apela această funcție în Matlab pentru ca rezultatul să se scrie tot în variabila y ?
- Scrieți un script care să contorizeze timpul de calcul necesar acestei proceduri, pentru cazul $n = m$. Observați că rulări distincte ale acestuia nu conduc la exact aceleași valori

numerice pentru timpul de calcul. Comparați timpul obținut cu cel necesar instrucțiunii Matlab scrise cu vectori. Pentru aceasta, realizați un grafic de tipul celui din figura 2.1. Comentați rezultatele obținute.

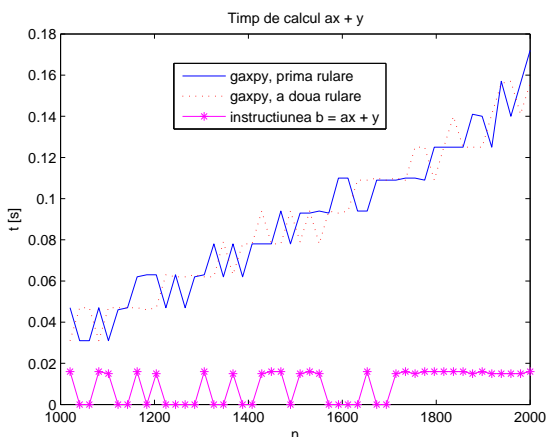


Figura 2.1: Un grafic de acest tip trebuie obținut la exercițiul 2.

În mod riguros, notația $T = O(\dots)$ folosită mai sus, definește o margine asimptotică superioară pentru dependența 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 și numai dacă există o constantă pozitivă $C > 0$ și un număr n_0 astfel încât $T \leq Cg(n)$ pentru orice $n \geq n_0$. Se poate defini și o margine inferioară pentru această dependență. Un algoritm are *ordinul de complexitate* $\Omega(g(n))$ din punct de vedere al timpului de calcul dacă și numai dacă există o constantă pozitivă $C > 0$ și 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 reală $xcrt$ în interiorul unui șir de n valori x ordonate crescător. Dacă valoarea $xcrt$ se află în afara intervalului $[x_1, x_n]$ atunci funcția va întoarce valoarea -1, alfel ea va întoarce indexul din stânga al sub-intervalului în care se află valoarea. Fie pseudocodul următor:

```

funcție caută_v1( $n, x, xcrt$ )
; caută valoarea  $xcrt$  în șirul ordonat  $x$  cu  $n$  valori
întreg  $n$ 
tablou real  $x[n]$  ; șirul 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) și (flag = 0))
        dacă xcrt ≤ xk+1
            rez = k
            flag = 1
        altfel
            k = k + 1
    •
    •
    •
    întoarce rez

```

Exercițiul 2.3:

- Explicați cum se face căutarea în funcția `caută_v1`.
- Arătați că ordinul de complexitate în cazul cel mai defavorabil este $T = O(n)$ și în cazul cel mai favorabil $T = \Omega(1)$. Se va considera ca operație de referință compararea a două numere reale (instrucțiunea $k \leq n - 1$ este o comparație între două numere întregi, iar $flag = 0$ este o comparație între două variabile logice, timpul necesar acestor instrucțiuni se va neglija).
- Implementați funcția în Matlab și scrieți un script pentru testarea corectitudinii ei.

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

```

funcție caută_v2(n, x, xcrt)
...
dacă (xcrt < x1) sau (xcrt > xn)
    rez = -1
altfel
    k1 = 1
    k2 = n
    cât timp k2 - k1 ≠ 1
        km = [(k1 + k2)/2] ; [...] este partea întreagă
        dacă xcrt < xkm
            k2 = km
        altfel

```

```

        k1 = km
    •
    •
    rez = k1
•
întoarce rez

```

Exercițiul 2.4:

- Explicați cum se face căutarea în funcția `caută_v2`.
- Arătați că ordinul de complexitate este $T = O(\log(n))$, unde \log este logaritmul în baza 2.
- Implementați funcția în Matlab și completați script-ul scris la exercițiul anterior pentru testarea corectitudinii ei.

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

```

funcție 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

```

```

funcție 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 complexității algoritmilor recursivi se face prin scrierea unei relații de recurență. De exemplu, pentru algoritmul recursiv de mai sus, la fiecare iterație 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

$$T(n) = T(n/2) + O(1).$$

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

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

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

Exercițiul 2.5:

Implementați funcția `caută_v3` în Matlab și completați script-ul scris la exercițiul anterior pentru testarea corectitudinii ei.

2.2 Necesarul de memorie

Complexitatea unui algoritm din punct de vedere al necesarului de memorie este dependența dintre necesarul de memorie exprimat în număr de locații elementare de memorie și dimensiunea problemei. De obicei, o locație elementară de memorie este cea corespunzătoare unui număr real.

Exercițiul 2.6:

Executați următoarele comenzi în consola Matlab:

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

Cați bytes ocupă un număr real?

Exercițiul 2.7:

Executați următoarele comenzi în consola Matlab:

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

Comentați rezultatul. Pentru înțelegerea detaliilor, căutați în documentația 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 și a unui scalar real pentru rezultat. Ordinul de complexitate este deci $M = O(2n + 1) = O(2n) \approx O(n)$.

Exercițiul 2.8:

Care este ordinul de complexitate din punct de vedere al necesarului de memorie pentru algoritmul implementat la exercițiul 2.1? Dar la exercițiul 2.2?

Exercițiul 2.9:

Ce spațiu de memorie este necesar pentru stocarea unei matrice pătrate de dimensiune 100000?

În rezolvarea numerică a problemelor reale din inginerie, un sistem de ecuații cu 100000 ecuații 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, algoritmi de calcul se vor adapta acestei scheme de memorare, rezultând avantaje atât din punct de vedere al necesarului de memorie cât și al timpului de calcul. *O matrice care conține un număr foarte mare de elemente nule se numește matrice rară.* Despre o matrice care nu este rară se spune că este *matrice densă* sau *plină*.

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

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

Cea mai naturală metodă ar fi cea în care se memorează doar valorile nenule și indicii 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 spațiu de $M_{plin} = 8mn$ B, sunt necesare doar n_{nz} locații de memorie pentru numere reale și $2n_{nz}$ locații 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, și doi vectori de numere întregi, de dimensiune 6, astfel:

$$\mathbf{M} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 \\ 2 & 3 & 0 & 7 \end{bmatrix} \Rightarrow \begin{cases} val & = [4 \ 2 \ 3 \ 5 \ 1 \ 7] \\ r_idx & = [1 \ 3 \ 3 \ 2 \ 2 \ 3] \\ c_idx & = [1 \ 1 \ 2 \ 3 \ 4 \ 4] \end{cases}$$

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

Memorarea poate fi și mai eficientă decât atât. Informația 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 $m \times n$ cu un număr n_{nz} de elemente nenule, va fi stocată cu ajutorul a trei tablouri unidimensionale astfel:

- un tablou unidimensional val , de dimensiune nnz , care conține toate valorile nenule, de sus în jos și de la stânga la dreapta;
- un talou unidimensional c_ptr , de dimensiune $n + 1$ (câte un element pentru fiecare coloană, plus un element adițional care marchează sfârșitul tabloului), care conține indicii ce indică în tabloul val locurile în care încep coloanele. Mai precis, coloana j a matricei inițiale are valorile în val de la poziția $c_ptr(j)$ la poziția $c_ptr(j + 1) - 1$;
- un talou unidimensional r_idx , de dimensiune n_{nz} , care conține, pentru fiecare element din val , indicele 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.

Același exemplu, memorat în format CCS este:

$$\mathbf{M} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 \\ 2 & 3 & 0 & 7 \end{bmatrix} \Rightarrow \begin{cases} val & = [4 \ 2 \ 3 \ 5 \ 1 \ 7] \\ c_ptr & = [1 \ 3 \ 4 \ 5 \ 7] \\ r_idx & = [1 \ 3 \ 3 \ 2 \ 2 \ 3] \end{cases}$$

Formatul CCS este și modul de stocare a matricilor rare în Matlab. Puteți încerca în Matlab următorul exercițiu demonstrativ (matricea folosită este aceeași 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 $8 \times 12 = 96$ B, Ms este o matrice rară de dimensiune 3×4 , cu 6 elemente nenule, în format CCS, deci are nevoie de $8 \times 6 + 4 \times (4 + 1) + 4 \times 6 = 92$ B. Câștigul nu este aici foarte mare pentru că matricea aceasta are o densitate $d = 6 / (3 \times 4) = 50\%$, ea nefiind în realitate o matrice rară.

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

Exercițiul 2.10:

Executați în Matlab următoarele comenzi și comentați rezultatul.

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

```
>> 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 ieșire:

```
>> 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 indici, există intrări multiple, cu linii și coloane identice? Pentru a răspunde, analizați 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)
```

Exercițiul 2.11:

Executați 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?

Exercițiul 2.12:

- Ce spațiu de memorie este necesar pentru stocarea în format CCS a unei matrice pătrate de dimensiune 100000, știind că pe fiecare coloană sunt exact patru elemente nenule? Comparați rezultatul cu cel obținut la exercițiul 2.8
- Calculați densitatea acestei matrice.
- Scrieți un script matlab care să rezolve punctele a) și b).

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

Exercițiul 2.13:

- Scrieți în Matlab o funcție care să adune elementele unei matrice pătrate de dimensiune n , parcurgând elementele pe coloane:

funcție 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) Scrieți în Matlab o funcție care să adune elementele unei matrice pătrate de dimensiune n , parcurgând elementele pe linii:

funcție suma_acces_linii(n, a)

...

; completați acest pseudocod

c) Apelați următorul script și comentați 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]');

```

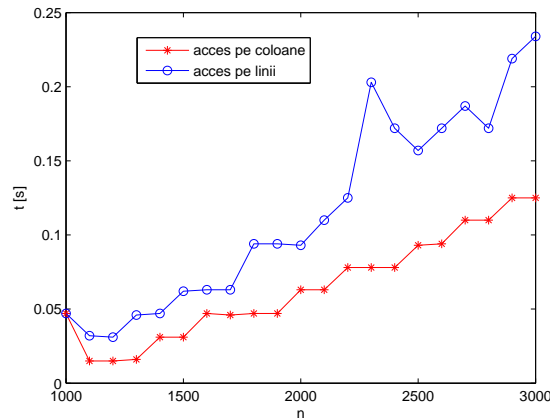


Figura 2.2: Timpul de calcul depinde și de modul în care se accesează elementele unei matrice. Un astfel de grafic trebuie obținut la exercițiul 2.13.

În cazul în care matricele sunt memorate ca structuri rare, algoritmi ce operează cu ele sunt adaptați 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
; declarații
...
pentru  $i = 1, n$ 
     $b_i = y_i$ 
    pentru  $p = col\_ptr(i), col\_ptr(i + 1) - 1$ 
         $b_{row\_idx(p)} = b_{row\_idx(p)} + val_px_i$ 
    •
•
retur

```

Exercițiul 2.14:

- a) Arătați că ordinul de complexitate al procedurii `sparse_gaxpy` este $O(2*nnz)$.
 b)¹ Implementați procedura în Matlab și testați corectitudinea ei.

2.3 Erori în calculele numerice

Unul din criteriile de evaluare a unui algoritm îl reprezintă eroarea cu care se obține 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 diferența dintre valoarea aproximativă \bar{x} și valoarea exactă x a mărimii

$$e_x = \bar{x} - x. \quad (2.1)$$

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

$$\|e_x\| \leq a_x. \quad (2.2)$$

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

$$\bar{x} - a_x \leq x \leq \bar{x} + a_x. \quad (2.3)$$

Altfel spus, cunoașterea marginii erorii absolute permite definirea intervalului în care este plasată soluția exactă. Relația (2.3) se mai scrie și sub forma

$$x = \bar{x} \pm a_x. \quad (2.4)$$

Dezavantajul folosirii erorii absolute este acela că valoarea numerică depinde de sistemul de unități de măsură folosit, făcând dificilă aprecierea gradului de acuratețe a

¹Facultativ

soluției. De aceea, se preferă folosirea unei mărimi relative, invariantă la sistemul de unități de măsură.

Eroarea relativă ε_x a unei mărimi se definește ca fiind raportul dintre eroarea absolută și norma mărimii

$$\varepsilon_x = \frac{e_x}{\|x\|}. \quad (2.5)$$

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

$$\|\varepsilon_x\| \leq r_x. \quad (2.6)$$

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

$$x = \bar{x} \pm r_x\%. \quad (2.7)$$

Erorile dintr-un algoritm se pot clasifica în funcție de tipul cauzelor care le generează în: erori inerente, erori de rotunjire și 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, și de aceea ele sunt afectate de erori. Aceste erori nu pot fi eliminate și 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 funcție 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 raționale ale numerelor reale. Numerele reale sunt reprezentate cu un număr finit de cifre semnificative. Nici aceste erori nu pot fi eliminate și, ca și 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 soluție 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 pași. În acest caz este importantă evaluarea erorii care se face datorită trunchierii procesului infinit.

Exercițiile 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 raționale ale acestora. În consecință, 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 părți fracționare f și a unui exponent n .

$$\bar{x} = f \cdot 10^n. \quad (2.8)$$

Mai mult, pentru orice număr în afara lui 0, prin alegerea convenabilă a lui n , partea fracționară satisface $0.1 \leq |f| < 1$. De exemplu $3.14 = 0.314 \cdot 10^1$, $-0.007856 = -0.7856 \cdot 10^{-2}$. În termenii convenției formulare, cifrele părții fracționare se numesc cifre semnificative.

Exercițiul 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 părțile fracționare 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 înțelegerea acestei afirmații 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, înseamnă că de fapt cifra 4 este pierdută. La adunare x_2 este "văzut" a fi zero și 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 configurația hard, de limbajul de programare folosit și de declarația făcută pentru variabile, dar ideea este exact cea descrisă de acest exemplu simplu.

Se definește zeroul (acuratețea, precizia, "epsilon-ul") mașinii ca fiind cel mai mic număr real care adunat la unitate îi modifică valoarea. Am putea spune că zeroul mașinii eps este cel mai mic număr pentru care $1 + \text{eps} > 1$. Pentru orice număr a mai mic

decât zeroul mașinii $1 + a = 1$, unde această din urmă relație o considerăm efectuată în calculator și nu în matematica exactă.

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

```

funcție zeroul_mașinii ()
real epsilon
epsilon = 1
cât timp (1 + epsilon > 1)
    epsilon = epsilon/2
•
epsilon = epsilon * 2
întoarce epsilon

```

Ca o consecință 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 obține un rezultat afectat cât mai puțin de erori de rotunjire, trebuie ca numerele să fie adunate în ordinea crescătoare a valorii lor.

Exercițiul 2.16:

- Implementați în Matlab funcția `zeroul_mașinii`.
- Comparați rezultatul obținut cu cel al funcției Matlab `eps`.

2.3.2 Erori inerente

Se poate demonstra că, în cazul operațiilor 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 $r_y =$ | $\frac{x_1}{x_1+x_2} r_{x_1} + \frac{x_2}{x_1+x_2} r_{x_2}$ | $\frac{x_1}{x_1-x_2} r_{x_1} + \frac{x_2}{x_1-x_2} r_{x_2}$ |

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

| Erori | Înmulțire $y = x_1x_2$ | Împărțire $y = \frac{x_1}{x_2}$ |
|-----------------------------------|---|---|
| Eroare absolută $e_y =$ | $x_2e_{x_1} + x_1e_{x_2}$ | $\frac{1}{x_2}e_{x_1} - \frac{x_1}{x_2^2}e_{x_2}$ |
| 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 înmulțirii și împărțirii a două numere reale în funcție de erorile datelor de intrare.

Precizăm că operația $x_1 + x_2$ este considerată adunare dacă ambii operanzi au același semn. În caz contrar, operația este de fapt o scădere. Similar, $x_1 - x_2$ este o scădere dacă ambii operanzi au același 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 $|x_1/(x_1 + x_2)|$ și $|x_2/(x_1 + x_2)|$, ambele ponderi fiind subunitare. Aceasta înseamnă că eroarea rezultatului adunării a două numere reale nu este amplificată, ea rămâne de același ordin de mărime cu erorile datelor de intrare. Se spune că *adunarea este o operație stabilă numeric*. Nu același lucru se poate spune despre scădere. Ponderile în acest caz sunt $|x_1/(x_1 - x_2)|$ și $|x_2/(x_1 - x_2)|$, care pot fi oricât de mari pentru că diferența $x_1 - x_2$ poate fi oricât de mică. În consecință, rezultatul unei scăderi poate fi afectat de erori relative mult mai mari decât erorile relative ale datelor de intrare. Se spune că *scăderea este o operație prost condiționată*. Proasta condiționare la scădere apare mai ales atunci când numerele sunt foarte apropiate, efect cunoscut și sub numele de *efect de anulare prin scădere*.

Efectuarea unor calcule de acest tip, în care se urmărește nu numai valoarea rezultatului ci și modul în care acesta este afectat de erorile datelor de intrare se numește *calcul cu intervale*. Este acum mai evident faptul că adunarea numerelor reale în calculator nu este o operație asociativă, afirmație făcută cu ocazia discuției referitoare la erorile de rotunjire.

Un ultim exemplu interesant este cel al funcției $y = \sqrt{x}$ pentru care rezultă o eroare absolută $e_y = 1/(2\sqrt{x})e_x$ și 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 proprietățile erorilor inerente de efectul rotunjirii. De aceea, vom accepta un fel de superpoziție 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, rezultatelor care estimează marginea erorii relative trebuie să li se adauge o eroare de rotunjire, de exemplu

$$\varepsilon_{\sqrt{x}} = \frac{\varepsilon_x}{2} + eps. \quad (2.9)$$

Exercițiul 2.17:

Scrieți un program Matlab care să calculeze soluțiile ecuației $ax^2 + bx + c = 0$, 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 caracterul finit al algoritmilor. Există metode matematice a căror soluție 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 funcții:

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

Exercițiul 2.18:

Scrieți seria alternată obținută din dezvoltarea în serie Taylor a funcției $\sin(x)$ în jurul punctului $x_0 = 0$.

Conform teoriei seriilor alternate, se știe că eroarea absolută făcută prin această trunchiere este mai mică decât ultimul termen considerat. Cunoașterea acestui rezultat teoretic ne permite să elaborăm un algoritm de evaluare a funcției sinus care să stabilească singur când se va opri. Algoritmul va aduna termeni la suma parțială până când termenul curent sumat este mai mic decât o anumită valoare. Această valoare nu trebuie să fie prea mare căci atunci soluția va fi nesatisfăcătoare. De asemenea, o valoare mai mică decât zero roul mașinii este lipsită de sens deoarece un termen curent cu o astfel de valoare, adunat la suma parțială acumulată până în acel moment, nu mai are nici o influență asupra valorii sumei parțiale. Într-un astfel de moment, continuarea sumării seriei este inutilă, ea nu mai îmbunătățește rezultatul, fenomen total diferit de teoria matematică în care rezultatul este cu atât mai precis cu cât suma conține mai mulți termeni. Pseudocodul următor implementează calculul funcției sinus într-un punct x , cu o eroare impusă err .

```

funcție sinus( $x, err$ )
; întoarce valoarea funcției sinus în punctul  $x$ 
; prin trunchierea seriei Taylor dezvoltată în 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  $s$ 

```

Exercițiul 2.19:

- Implementați în Matlab pseudocodul de mai sus.
- Scrieți un program care să reprezinte grafic variația modului termenului curent (ca în figura 2.3) și modulul diferenței dintre sumele parțiale consecutive (figura 2.4) în funcție de iterație. Comentați rezultatul.

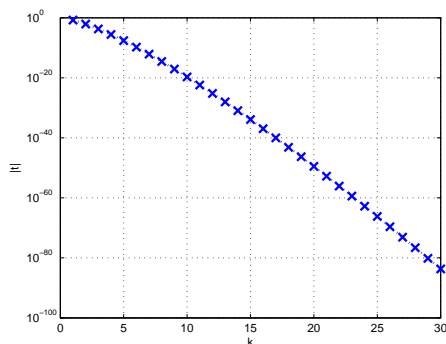


Figura 2.3: Modulul termenului curent al dezvoltării în serie Taylor a funcției sinus. Un astfel de grafic trebuie obținut la exercițiul 2.18.

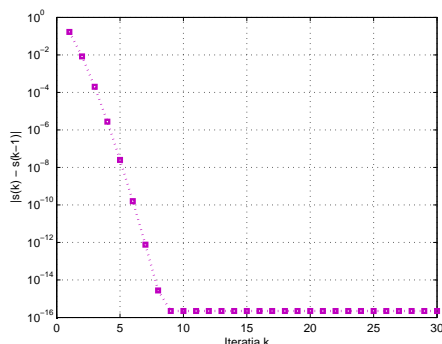


Figura 2.4: Modulul diferenței dintre sume parțiale consecutive la dezvoltarea în serie Taylor a funcției sinus. Un astfel de grafic trebuie obținut la ex. 2.18.

Capitolul 3

Analiza circuitelor electrice rezistive liniare

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

Un circuit rezistiv liniar este un circuit ce conține rezistoare, surse ideale de tensiune și curent și surse comandate liniar. Problema fundamentală a analizei acestor circuite are ca date topologia circuitului și valorile parametrilor (rezistențele, valorile surselor), și urmărește calculul curenților și tensiunilor din fiecare latură.

Există mai multe metode de rezolvare a unor astfel de circuite: metoda ecuațiilor Kirchhoff, metoda potențialelor nodurilor, metoda curenților ciclici. Atât metoda potențialelor nodurilor cât și metoda curenților ciclici generează un sistem de ecuații mai mic decât metoda Kirchhoff, având o matrice a coeficienților cu proprietăți mai bune (de exemplu simetrie, diagonal dominantă). Metoda curenților ciclici necesită și alegerea unui sistem convenabil de bucle independente, fiind mai dificil de transpus într-un algoritm decât metoda potențialelor nodurilor (numită mai scurt *metoda* sau *tehnica nodală*).

Exercițiile din acest capitol vă vor ajuta să înțelegeți modul în care se concepe și se testează un algoritm folosind tehnica nodală pentru asamblarea sistemului de ecuații, metode directe de rezolvare a sistemului rezultat și tehnici de matrice rare pentru a obține un algoritm cu o complexitate cât mai bună.

3.1 Metoda potențialelor nodurilor

Primul pas către elaborarea unui algoritm îl constituie înțelegerea teoriei metodei de rezolvare și 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.

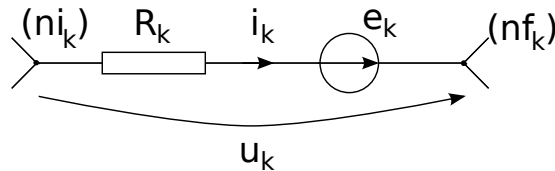


Figura 3.1: Latura standard.

Datele problemei sunt

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

Se cere să se calculeze

- toate tensiunile u_k la bornele laturilor,
- toți curenții i_k ce străbat laturile,
- puterea consumată și puterea generată în circuit.

Deși alegerea sensurilor de referință poate fi arbitrară, în vederea transformării metodei nodale într-un algoritm, este mult mai ușor dacă se aleg sensuri de referință în mod sistematic, ca de exemplu: sensul de referință al curentului este dat de sensul sursei de tensiune (definit de săgeata interioară) și sensul de referință al tensiunii se alege astfel încât să se respecte convenția de la receptoare (așa cum sunt reprezentate în fig. 3.1). Sensul de referință al curentului ce străbate o latură k stabilește și o relație de ordonare între cele două noduri ale laturii respective, el fiind sensul de la nodul inițial notat (ni_k) la nodul final notat (nf_k) .

Conform teoriei circuitelor, fenomenele sunt complet descrise de un număr de $N - 1$ ecuații¹ Kirchhoff I

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

un număr de $L - N + 1$ ecuații Kirchhoff II scrie pentru un sistem de bucle independente

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

și L relații Joubert, scrise pentru toate laturile

$$u_k = R_k i_k - e_k, \quad k = 1, \dots, L, \quad (3.3)$$

în total un număr de $2L$ ecuații cu $2L$ necunoscute (toți curenții și toate tensiunile).

În tehnica nodală necunoscutele principale ale problemei (adică necunoscutele care se calculează mai întâi) sunt potențialele nodurilor v_k , $k = 1, \dots, N$, unde unul din noduri are, în mod convențional, potențialul nul. Vom presupune că numerotarea nodurilor este făcută astfel încât nodul de indice maxim este cel de referință, $v_N = 0$. Prin exprimarea tensiunilor ca diferențe de potențial, teorema Kirchhoff II este identic satisfăcută. Altfel spus, relația (3.2) este satisfăcută dacă se scriu toate relațiile

$$u_k = v_{ni_k} - v_{nf_k}, \quad k = 1, \dots, L. \quad (3.4)$$

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

$$\begin{aligned} \mathbf{u} &= [u_1 \ u_2 \ \dots \ u_L]^T \in \mathbb{R}^{L \times 1} && \text{vectorul tensiunilor laturilor,} \\ \mathbf{i} &= [i_1 \ i_2 \ \dots \ i_L]^T \in \mathbb{R}^{L \times 1} && \text{vectorul curenților prin laturi,} \\ \mathbf{v} &= [v_1 \ v_2 \ \dots \ v_{N-1}]^T \in \mathbb{R}^{(N-1) \times 1} && \text{vectorul potențialelor nodurilor,} \\ \mathbf{e} &= [e_1 \ e_2 \ \dots \ e_L]^T \in \mathbb{R}^{L \times 1} && \text{vectorul tensiunilor electromotoare,} \\ \mathbf{R} &= \text{diag}([R_1 \ R_2 \ \dots \ R_L]) \in \mathbb{R}^{L \times L} && \text{matricea diagonală a rezistențelor laturilor.} \end{aligned}$$

Cu aceste notații, relațiile Kirchhoff I (3.1) se scriu în mod compact

$$\mathbf{A}\mathbf{i} = \mathbf{0}, \quad (3.5)$$

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

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

¹Notația \sum^A reprezintă o sumă algebrică, adică $\sum^A x_k = \sum \varepsilon_k x_k$, unde $\varepsilon_k = \pm 1$.

Exercițiul 3.1:

Fie circuitul din fig.3.2.

- Orientați laturile în conformitate cu convenția descrisă mai sus;
- Numerotați laturile;
- Numerotați nodurile;
- Scrieți matricea incidentelor laturi noduri $\mathbf{A} \in \mathbf{Z}^{4 \times 9}$.

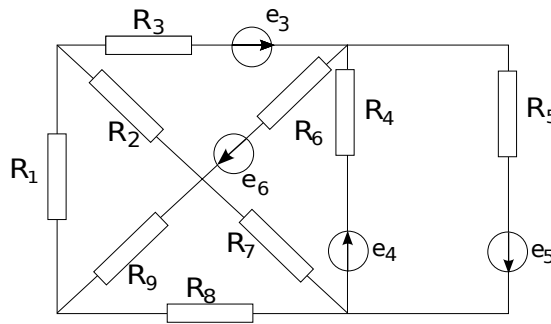


Figura 3.2: Circuit simplu de test.

Relația Kirchhoff II în forma (3.4) se scrie

$$\mathbf{u} = \mathbf{A}^T \mathbf{v}, \quad (3.6)$$

iar relațiile lui Joubert (3.3) se scriu compact

$$\mathbf{u} = \mathbf{R}\mathbf{i} - \mathbf{e}. \quad (3.7)$$

Exercițiul 3.2:

Verificați relațiile (3.6) și (3.7) pentru circuitul de test din fig. 3.2.

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

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

Înlocuind (3.8) și (3.6) în (3.5), rezultă ecuația matriceală satisfăcută de vectorul potențialelor

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

Matricea coeficienților sistemului algebric liniar de rezolvat

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

este numită *matricea conductanțelor nodale*.

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

Figura 3.3: Contribuția unei laturi k la matricea conductanțelor nodale.

Figura 3.4: Contribuția unei laturi k la vectorul injecțiilor de curent.

Exercițiul 3.3:

Calculați matricea \mathbf{G} dată de relația (3.10) pentru circuitul simplu de test.

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

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

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

Termenul liber al sistemului de ecuații

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

este numit *vectorul injecțiilor de curent*.

Exercițiul 3.4:

Calculați vectorul \mathbf{t} dat de relația (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 , și cu semnul minus în caz contrar.

Este important să remarcăm că matricea \mathbf{G} este simetrică și pozitiv definită. În relația

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

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

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

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

Atunci

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

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

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

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

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} , și în consecință vectorul \mathbf{x} este nul, ceea ce contrazice ipoteza făcută. În concluzie, matricea conductanțelor 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 și se assemblează sistemul de ecuații de rezolvat, *etapa de rezolvare* în care se apelează o procedură propriu-zisă de rezolvare a sistemului de ecuații rezultat (procedură numită foarte adesea "solver") și *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 concordanță cu teoria prezentată în paragraful anterior. Reamintim că trebuie să știm informațiile legate de topologie:

numărul de noduri N , numărul de laturi L și, pentru fiecare latură, care este nodul inițial ni_k și care este nodul final nf_k . De asemenea, trebuie să știm parametrii fiecărei laturi: rezistența R_k și tensiunea electromotoare e_k .

Declarații 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 inițiale ale laturilor
tablou întreg $nf[L]$; noduri finale ale laturilor
tablou real $R[L]$; rezistențe
tablou real $e[L]$; tensiuni electromotoare

Exercițiul 3.5:

Pentru circuitul simplu de test, considerați 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 importanță practică, el va fi folosit exclusiv pentru prima validare a programului ce va fi implementat. Completați 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 funcții, recomandăm însă agregarea datelor, ca de exemplu

; declarații date - varianta B

înregistrare circuit

întreg N ; număr de noduri
întreg L ; număr de laturi
tablou întreg $ni[L]$; noduri inițiale ale laturilor
tablou întreg $nf[L]$; noduri finale ale laturilor
tablou real $R[L]$; rezistențe
tablou real $e[L]$; tensiuni electromotoare

De asemenea, un alt aspect important este acela că matricea conductanțelor nodale și vectorul injecțiilor de curent sunt rare. Totuși, pentru a păstra simplă descrierea algoritmului, în cele ce urmează vom păstra același tip de declarații și pentru matricile rare și vectorii rari, urmând a discuta exploatarea rarității ulterior. În consecință, variabile mai importante care vor apărea în algoritm sunt:

; declarații variabile utile

tablou real $G[N, N]$; matricea conductanțelor nodale (în final va fi stocată rar)

tablou real $t[N]$; vectorul injecțiilor de curent (în final va fi stocat rar)

tablou real $v[N]$; vectorul potențialelor

3.3 Etapa de preprocesare

Etapa de preprocesare constă în citirea datelor (de exemplu de la tastatură sau din fișiere) și în asamblarea sistemului de ecuații de rezolvat.

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

procedură citire_date_A (N, L, ni, nf, R, e)

; declarații

...

citește N, L

pentru $k = 1, L$

citește ni_k, nf_k, R_k, e_k

•

retur

În cazul variantei B de declarații, rutina de citire a datelor poate fi o funcție, de tipul

funcție citire_date_B ()

; declarații

...

citește circuit.N, circuit.L

pentru $k = 1, \text{circuit.L}$

citește circuit.ni_k, circuit.nf_k, circuit.R_k, circuit.e_k

•

întoarce circuit

Exercițiul 3.6:

Pentru primele teste vom prefera chiar o procedură de "citire" rapidă a circuitului simplu de test. Pentru aceasta, scrieți un fișier `citire_circuitRE_simplu.m` ca mai jos. Completați datele circuitului în conformitate cu tabelul de la exercițiul 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 și de a asambla sistemul de ecuații. O variantă ar fi cea în care se parcurg nodurile și se scriu ecuațiile una câte una. Așa ar face și 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 și adunarea contribuțiilor acestora la sistem. Această variantă este mult mai simplă de conceput deoarece fiecare latură are exact două noduri. Din același motiv și descrierea circuitului a fost orientată pe laturi și nu pe noduri.

Algoritmul metodei nodale este următorul

```
procedură nodalRE_v1 (circuit,  $G$ ,  $t$ )
; asamblează sistemul de ecuații 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 ieșire:
;  $G$  - matricea conductanțelor nodale și
;  $t$  - vectorul injecțiilor de curent
; declarații
....
 $L$  =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 și ale vectorului termenilor liberi t
G = 0
t = 0
; assemblează sistem
pentru k = 1, L ; parcurge laturi
    i = ni_k ; nodul inițial 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

```

•

Exercițiul 3.7:

- a) Implementați procedura `nodalRE_v1` ca o funcție Matlab.
b) Scrieți un script `main_nodal` cu următorul conținut:

```

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

```

Verificați că rezultatul este cel așteptat.

- c) Arătați că această procedură de asamblare are complexitatea $T = O(12L)$.

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

procedură `nodalRE_v2` (circuit, G , t)

....

```

pentru k = 1, L ; parcurge laturi
    i = ni_k ; nodul inițial al laturii k
    j = nf_k ; nodul final al laturii k
    Glat = 1/R_k
    Isc = e_k*Glat

```

$$\begin{aligned}
 G_{ii} &= G_{ii} + \text{Glat} \\
 G_{jj} &= G_{jj} + \text{Glat} \\
 G_{ij} &= G_{ij} - \text{Glat} \\
 G_{ji} &= G_{ji} - \text{Glat} \\
 t_i &= t_i - \text{Isc} \\
 t_j &= t_j + \text{Isc}
 \end{aligned}$$

•

Exercițiul 3.8:

- Implementați procedura `nodalRE_v2` ca o funcție Matlab.
- Adăugați scriptului `main_nodal` apelul acestei proceduri. Verificați corectitudinea rezultatului.
- Care este complexitatea procedurii `nodalRE_v2`?
- Contorizați timpul de calcul pentru cele două proceduri și comparați.

O altă idee de îmbunătățire este aceea care se bazează pe faptul că matricea este simetrică și putem asambla doar jumătate din ea. Procedura următoare assemblează triunghiul inferior.

procedură `L.nodalRE_v3` (circuit, G , t)

....

pentru $k = 1, L$

; parcurge laturi

$$i = \text{ni}_k$$

; nodul inițial al laturii k

$$j = \text{nf}_k$$

; nodul final al laturii k

$$\text{Glat} = 1/R_k$$

$$\text{Isc} = e_k * \text{Glat}$$

$$G_{ii} = G_{ii} + \text{Glat}$$

$$G_{jj} = G_{jj} + \text{Glat}$$

dacă $i > j$

$$G_{ij} = G_{ij} - \text{Glat}$$

altfel

; acesta este cazul $i < j$,

$$G_{ji} = G_{ji} - \text{Glat}$$

; cazul $i = j$ nu este posibil pentru date de intrare corecte

•

$$t_i = t_i - \text{Isc}$$

$$t_j = t_j + \text{Isc}$$

•

Exercițiul 3.9:

- Implementați procedura `L.nodalRE_v3` ca o funcție Matlab.

- b) Adăugați scriptului `main_nodal` apelul acestei proceduri. Verificați 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 referință nefiind tratat special. Acest lucru a făcut extrem de ușoară scrierea pseudocodului pentru asamblarea matricei. Pentru rezolvare însă, sistemul trebuie să fie de dimensiune $N - 1$. Presupunând că nodul N este nodul de referință ($v_N = 0$), atunci matricea coeficienților se obține eliminând ultima linie și ultima coloană din matricea asamblată, iar vectorul termenilor liberi se obține eliminând ultima componentă.

Exercițiul 3.10:

Adăugați scriptului `main_nodal` următoarele comenzi matlab.

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

Comentați aceste comenzi.

Operatorul *backslash* în Matlab implementează într-un mod eficient metoda Gauss pentru rezolvarea sistemelor de ecuații algebrice liniare. Există și alte posibilități de rezolvare a sistemului de ecuații, cum ar fi metode bazate pe factorizare sau metode iterative.

Vectorul soluție întors reprezintă potențialele a $N - 1$ noduri. Potențialul ultimului nod a fost considerat zero și de aceea, pentru completitudine, este adăugată 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, curenți prin laturi, puteri. Următorul pseudocod ilustrează acest calcul

```

procedură postprocesare_crl (circuit, v)
; declarații
...
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 = vnik - vnfk ; tensiunea laturii
    c = (u + ek)/Rk ; curentul prin latură
    scrie "Latura" k "are tensiunea" u "și curentul" c
    Pc = Pc + Rkc2 ; adaugă contribuția laturii la puterea consumată
    Pg = Pg + ekc ; adaugă contribuția laturii la puterea generată
•
scrie Pc, Pg
retur

```

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

Exercițiul 3.11:

- Implementați în Matlab procedura de postprocesare;
- Completați scriptul principal cu apelul ei;
- Rulați programul și verificați că bilanțul de puteri este verificat.

3.6 Analiza complexității. Optimizarea algoritmului.

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

Ne propunem acum să analizăm complexitatea algoritmului și să vedem ce îmbunătățiri 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 și ne propunem să determinăm conductanța echivalentă între nodul marcat cu A și masă. Dat fiind modul de asamblare a matricei conductanțelor nodale, am preferat să marcăm pe rezistoare valorile conductanțelor. Se poate verifica cu ușurință faptul că valoarea conductanței echivalente între nodul A și masă este $G_{ech} = 2G$.

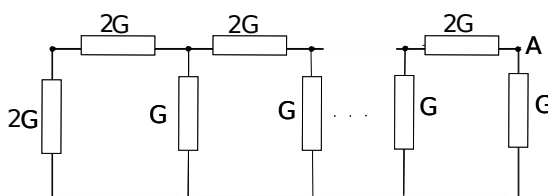


Figura 3.5: Circuit de test pentru analiza complexității. Pe rezistoare sunt marcate valorile conductanțelor.

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

$$J = G_{ech}U, \quad (3.18)$$

unde U este tensiunea între nodul A și masă, adică potențialul nodului A: $U = v_A$. Dacă se alege $J = 1A$, atunci

$$G_{ech} = 1/v_A. \quad (3.19)$$

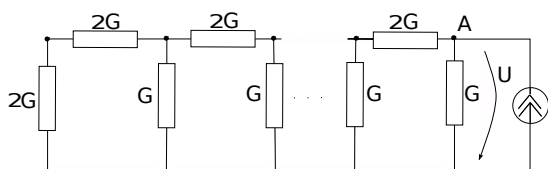


Figura 3.6: Excitarea nodului A în vederea determinării conductanței echivalente.

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

1. Preprocesare:

- Asamblează matricea conductanțelor nodale;
- Asamblează vectorul injecțiilor de curent. Acesta va avea toate componentele 0, cu excepția poziției corespunzătoare nodului A, unde valoarea va fi 1.

2. Rezolvare: se va rezolva sistemul de ecuații.
3. Postprocesare: conductanța echivalentă va fi inversul potențialului nodului A .

În vederea implementării etapei de preprocesare, trebuie să pregătim circuitul, numerotându-i nodurile și laturile. Pentru un circuit cu o structură regulată, este util dacă numerotarea se face ordonat, de exemplu așa 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 indice dublul nodului din stânga, iar laturile verticale sunt numerotate cu numere impare, având drept index dublul nodului de sus minus 1.

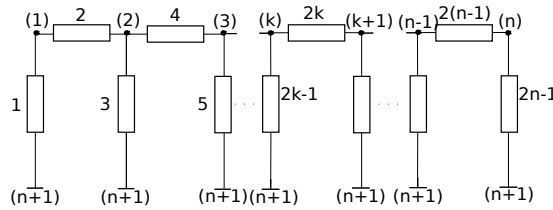


Figura 3.7: Numerotarea nodurilor și laturilor.

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 rezistențe vom lucra cu conductanțe. Astfel, structura de date ce descrie circuitul poate fi asamblată astfel:

```

funcție citire_circuit_1d (n, valG)
; declarații
...
L = 2n - 1 ; numărul total de laturi
ground = n + 1 ; indicele nodului de masă
doivalG = 2*valG
pentru k = 1, n - 1
    ; latura impară, de indice 2k - 1
    idx = 2k - 1
    ni_idx = k
    nf_idx = ground
    G_idx = valG

    ; latura pară, de indice 2k
    idx = 2k

```

```

ni_idx = k
nf_idx = k + 1
G_idx = doivalG
•
; corecție conductanța latura 1
G1 = doivalG
; ultima latură
ni_L = n
nf_L = ground
GL = valG
; asamblare structură circuit
circuit.N = n + 1
circuit.L = L
circuit.ni = ni
circuit.nf = nf
circuit.G = G
întoarce circuit

```

Exercițiul 3.12:

- Implementați în Matlab funcția `citire_circuit_1d`;
- Verificați funcția scrisă cu comanda `mlint`;
- Observați 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 $M_{\text{circuit}} = O(?)$. Comparați estimarea teoretică cu cea experimentală.

Vom implementa acum asamblarea matricei conductanțelor nodale, modificând extrem de puțin funcția `nodalRE_v2` implementată pentru exemplul simplu discutat anterior.

```

procedură nodal_circuit_1d_v1 (circuit,G,t)
; parametri de intrare - circuit
; parametri de ieșire - 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 și
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

```

Exercițiul 3.13:

- a) Implementați procedura de mai sus ca o funcție Matlab
`[G,t] = function nodal_circuit_1d_v1 (circuit)`. Verificați-o cu `mlint`.
b) Scrieți 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)));
```

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

Exercițiul 3.14:

Observați structura matricei conductanțelor nodale după asamblare și înainte de rezolvarea propriu-zisă. Pentru aceasta adăugați

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

imediat după asamblare și

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

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

Exercițiul 3.15:

a) Estimați ordinul de complexitate din punct de vedere al necesarului de memorie. Puteți experimenta numeric și 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) Estimați teoretic (nu încercați experimental!) necesarul de memorie pentru stocarea matricei conductanțelor nodale în cazul $n = 10000$.

Matricea conductanțelor 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

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

Exercițiul 3.16:

- a) Cât este densitatea matricei coeficienților sistemului în cazul $n = 10$? Estimați teoretic cu formula de mai sus și numeric folosind funcția Matlab `nnz`. Explicați diferența.
- b) Estimați teoretic densitatea matricei pentru $n = 10000$.

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

Exercițiul 3.17:

Observați diferența dintre funcțiile `zeros` și `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.

Exercițiul 3.18:

- a) Copiați fișierul `nodal_circuit_1d_v1.m` în `nodal_circuit_1d_v2.m`. Pentru aceasta, puteți da comanda Matlab:

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

- b) Editați fișierul `nodal_circuit_1d_v2.m`.

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

Corectați numele funcției și înlocuiți funcția `zeros` cu `sparse`.

Exercițiul 3.19:

- a) Comparați necesarul de memorie pentru stocarea matricei conductanțelor 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
```

Exercițiul 3.20:

Estimați necesarul de memorie pentru stocarea matricei conductanțelor nodale în cazul folosirii structurilor rare. Puteți 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 și "dizolva" informațiile legate de topologia circuitului în procedura nodal.

procedură nodal_circuit_1d_v3(n, valG, G, t)

; parametri de intrare - n, valG

; parametri de ieșire - G, t

$N = n + 1$; numărul total de noduri

ground = N ; indexul nodului de masă

$G = \text{sparse}(N, N)$; se alocă memorie pentru o matrice rară de dimensiune N

$t = \text{sparse}(N, 1)$

doivalG = $2 * \text{valG}$

pentru $k = 1 : n - 1$

 ; latura impară, de index $2k - 1$

$i = k$

$j = \text{ground}$

$G_{ii} = G_{ii} + \text{valG}$

$G_{jj} = G_{jj} + \text{valG}$

$G_{ij} = G_{ij} - \text{valG}$

$G_{ji} = G_{ji} - \text{valG}$

 ; latura pară, de index $2k$

$i = k$

$j = k + 1$

$G_{ii} = G_{ii} + \text{doivalG}$

$G_{jj} = G_{jj} + \text{doivalG}$

$G_{ij} = G_{ij} - \text{doivalG}$

$G_{ji} = G_{ji} - \text{doivalG}$

•

; corecție conductanța latura 1

$G_{11} = G_{11} + \text{valG}$

```

; ultima latură
i = n
j =ground
Gii = Gii+valG
Gjj = Gjj+valG
Gij = Gij-valG
Gji = Gji-valG
tN-1 = 1
retur

```

Exercițiul 3.21:

- Observați diferența între modul de asamblare al sistemului în cele două variante.
- Implementați această variantă în Matlab și testați-i corectitudinea.
- Comparați necesarul de memorie și timpul de calcul necesar etapei de preprocesare în variantele 2 și 3 pentru $n = 10000$. Pentru aceasta puteți 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 incidentelor laturi-noduri și calculul matricei conductanțelor nodale în acord cu formula (3.10). Pseudocodul acestei proceduri este următorul.


```

procedură nodal_circuit_1d_v4( $n, \text{valG}, G, t$ )
; parametri de intrare -  $n, \text{valG}$ 
; parametri de ieșire -  $G, t$ 
 $N = n + 1$  ; numărul total de noduri
 $L = 2n - 1$  ; numărul total de laturi
ground =  $N$  ; indexul nodului de masă
; alocare spație de memorie pentru:
 $A = \text{sparse}(N, L)$  ; - matricea incidențelor laturi-noduri,
Glat =  $\text{sparse}(L, 1)$  ; - vectorul conductanțelor laturilor,
 $t = \text{sparse}(N, 1)$  ; - vectorul injecțiilor de curent.
doivalG =  $2 * \text{valG}$ ;

```

```

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

```

```

; latura pară, de index  $2k$ 
idx =  $2k$ 
 $i = k$ 
 $j = k + 1$ 
 $A_{i, \text{idx}} = 1$ 
 $A_{j, \text{idx}} = -1$ 
Glatidx = doivalG

```

•

```

; corecție conductanță latura 1
Glat1 = doivalG
; ultima latură
 $i = n$ 
 $j = \text{ground}$ 
 $A_{iL} = 1$ 
 $A_{jL} = -1$ 
GlatL = valG

```

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

$t_{N-1} = 1$

retur

Important: în algoritmul de mai sus instrucțiunea în care se calculează matricea conductanțelor nodale folosește operații cu matrice rare.

Exercițiul 3.22:

a) Observați rezultatul comenzii `diag` în Matlab. Comentăți următoarele instrucțiuni

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

Ce efect (intern) în Matlab ar avea comanda `sparse(diag(v))` ?

b) Implementați variantă v4 a procedurii nodal în Matlab și testați-i corectitudinea.

c) Comparați necesarul de memorie necesar etapei de preprocesare în variantele 3 și 4 pentru diferite valori ale lui n . Pentru aceasta puteți 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) Modificați acest script pentru a obține grafic această dependență (fig.3.8).

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

Exercițiul 3.23:

- Copiați funcția `nodal_circuit_1d_v4` în `nodal_circuit_1d_v5` și deschideți fișierul în editorul Matlab. Corectați numele funcției;
- Înlocuiți alocarea de memorie pentru matricea de incidențe 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;
```

- Înlocuiți toate atribuirile pentru componentele matricei incidențelor 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 faceți șase astfel de modificări.

- Asamblați matricea astfel

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

- Testați corectitudinea noii funcții.
- Comparați timpul necesar preprocesării, completând scripturile pe care deja le-ați folosit în exercițiile anterioare. Trebuie să obțineți un grafic de tipul celui din fig.3.9.

În concluzie, această temă ilustrează următoarele:

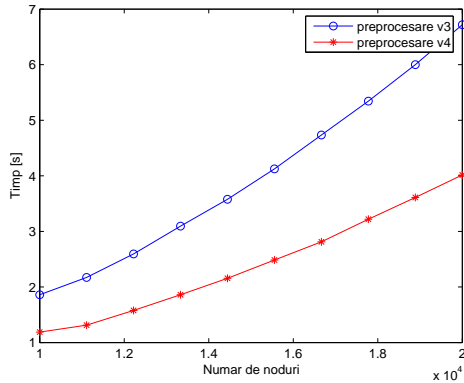


Figura 3.8: Comparație între două implementări ale etapei de preprocesare. Un astfel de grafic trebuie să obțineți la exercițiul 3.22.

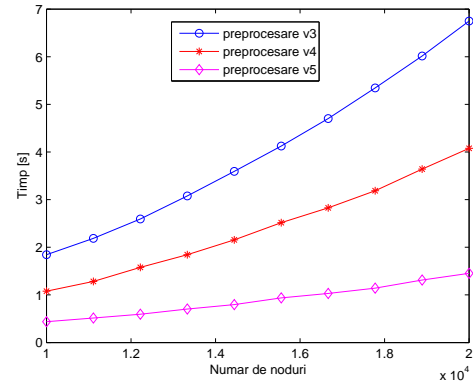


Figura 3.9: Comparație între trei implementări ale etapei de preprocesare. Un astfel de grafic trebuie să obțineți la exercițiul 3.23.

1. Etapele conceperii unui algoritm dedicat rezolvării unei probleme sunt:
 - înțelegerea perfectă a bazei teoretice a algoritmului;
 - alegerea structurilor de date pentru datele de intrare și de ieșire;
 - implementarea algoritmului într-un program;
 - testarea și depanarea programului pentru o problemă simplă, cu rezultat cunoscut;
 - rularea programului pe probleme de dimensiuni crescute, în vederea stabilirii complexității sale și măsurarea performanței;
 - optimizarea codului.
2. Dacă implementați algoritmi în Matlab, este mai bine să profitați, acolo unde este posibil, de funcțiile Matlab care folosesc operații cu vectori și matrice. Chiar și în cazul structurilor pline, funcțiile sunt optimizate (așa cum ați văzut într-un capitol anterior). În cazul în care structurile sunt rare, folosirea unor proceduri adaptate acestor structuri este foarte ușoară. Practic, nu este nevoie să scrieți 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 înțelegerea algoritmilor numerici pentru interpolarea funcțiilor, derivarea funcțiilor și integrarea lor. Exercițiile din această temă ilustrează interpolarea funcțiilor.

Problema interpolării are ca date un tabel de valori cunoscute care reprezintă date și rezultate ale unei probleme și are ca scop estimarea rezultatelor pentru un set de alte date, care nu se găsesc în tabelul inițial de valori. Ideea metodei este aceea de a găsi o funcție aproximativă care poate fi evaluată cu ușurință și care satisface condițiile impuse de valorile din tabel. Estimarea rezultatelor corespunzătoare unor date noi se face prin evaluarea acestei funcții 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 și 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:

| | | | | |
|--------------|-------|-------|----------|-------|
| \mathbf{x} | x_0 | x_1 | \cdots | x_n |
| \mathbf{y} | y_0 | y_1 | \cdots | y_n |

- Se cere: să se estimeze valoarea rezultatelor \mathbf{y}_i pentru alte date \mathbf{x}_i 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 \mathbf{x} este ordonat crescător.

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

- valorile vectorului \mathbf{x} sunt distincte;
- vom căuta o funcție aproximativă g care să satisfacă *condițiile de interpolare*:

$$g(x_k) = y_k, \quad k = 0, \dots, n, \quad (4.1)$$

- funcția g se va căuta în spațiul polinoamelor generalizate, adică g se caută de forma

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

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

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

4.1.2 Interpolarea globală

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

- metoda clasică:

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

- metoda Lagrange:

$$\varphi_k(x) = l_k(x), \quad k = 0, \dots, n, \quad (4.4)$$

unde cu l sunt notate polinoamele Lagrange,

- metoda Newton:

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

Din punct de vedere matematic, toate aceste trei metode sunt echivalente, ele conduc la același 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 condiționări, iar metoda Newton cea mai bună datorită bunei condiționări, posibilității de a estima eroarea de interpolare și faptului că, la adăugarea 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 condițiilor de interpolare, rezultă că valorile coeficienților 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

$$x_0 < x_1 < \dots < x_n$$

și unui punct k :

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

Expresia *polinomului de interpolare* este deci

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

Exercițiul 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) Calculați polinomul de interpolare globală cu formula (4.7).
 c) Verificați că polinomul obținut la punctul anterior satisface condițiile de interpolare.

Exercițiul 4.2:

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

Ne propunem să scriem o funcție Matlab care să implementeze evaluarea polinomului Lagrange asociat unei diviziuni. În Matlab, vectorii sunt indexați 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 funcții Lagrange de grad $n - 1$

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

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

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

Exercițiul 4.3:

- a) Să se scrie o funcție 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 - indicele 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) Verificați codul scris cu comanda `mlint`.

Exercițiul 4.4:

a) Scrieți un script Matlab `main_lagrange.m` cu următorul conținut.

```

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)');

```

Verificați că rularea lui furnizează figura 4.1.

b) Copiați acest script într-un alt fișier. Modificați-l astfel încât să obțineți figura 4.2.

Care este acum diviziunea folosită pe x ?

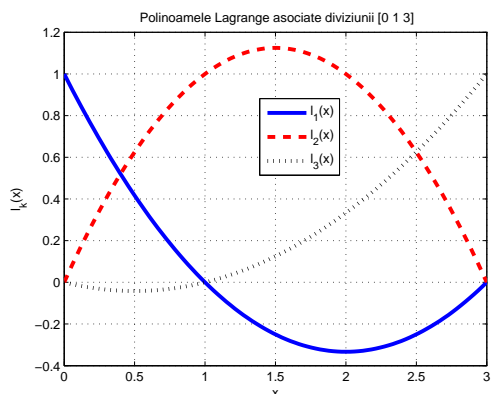


Figura 4.1: Polinoamele Lagrange asociate diviziunii $[0 \ 1 \ 3]$.

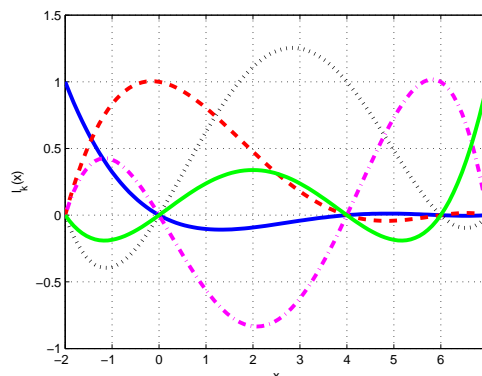


Figura 4.2: Pentru exercițiul 4.4: Ce reprezintă această figură?

Până acum ați văzut cum arată funcțiile de bază ale interpolării Lagrange. Să construim acum polinomul de interpolare.

Exercițiul 4.5:

a) Scrieți o funcție 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) Verificați codul scris cu comanda `mlint`.

Exercițiul 4.6:

a) Verificați corectitudinea funcției 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 obțineți rezultatul din fig.4.3.

b) Copiați acest script în alt fișier. Modificați-l astfel încât să obțineți rezultatul din fig. 4.4. Ce grad are polinomul de interpolare?

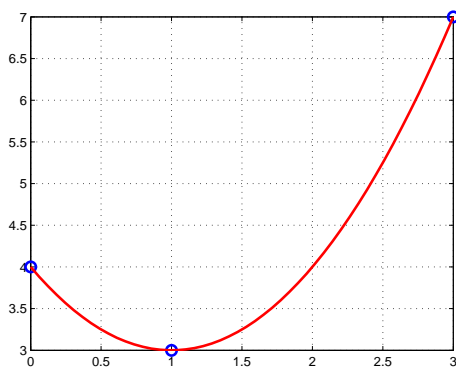


Figura 4.3: Polinomul de interpolare pentru tabelul $\mathbf{x} = [0 \ 1 \ 3]$, $\mathbf{y} = [4 \ 3 \ 7]$.

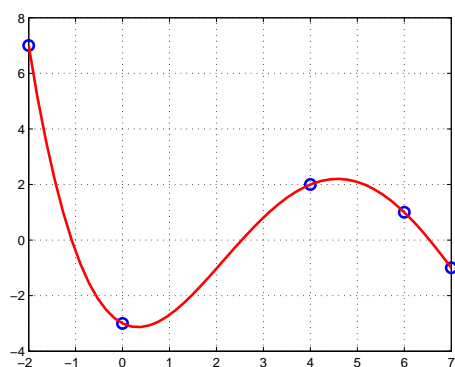


Figura 4.4: Pentru exercițiul 4.6: Ce grad are polinomul de interpolare?

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

Exercițiul 4.7:

a) Scrieți un script Matlab cu următorul conținut

```
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);
```

Comentați rezultatul obținut (fig. 4.5). Experimentați pentru alte grade ale polinomului de interpolare.

b) Încercați să dați o explicație acestui fenomen.

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

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

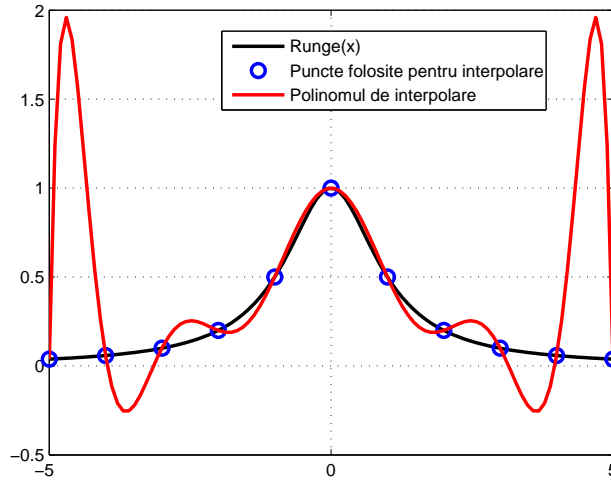


Figura 4.5: Fenomenul Runge.

4.1.3 Interpolarea pe porțiuni

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

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

De exemplu, polinoamele Lagrange definite pe porțiuni satisfac de asemenea condițiile $l_k(x_k) = 1$ și $l_k(x_j) = 0$ pentru $j \neq k$, dar variația între nodurile rețelei de discretizare este liniară:

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

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

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

Exercițiul 4.8:

a) Scrieți o funcție Matlab care să evalueze polinoamele Lagrange pe porțiuni, 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); % 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);

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));
        end
    end
end
```

```

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
end
end

```

b) Verificați codul scris cu comanda `mlint`.

c) Modificați scriptul `main_lagrange.m`, apelând funcția `lagrangep` în loc de `lagrange`. Prin rularea lui trebuie să obțineți graficul din fig. 4.6.

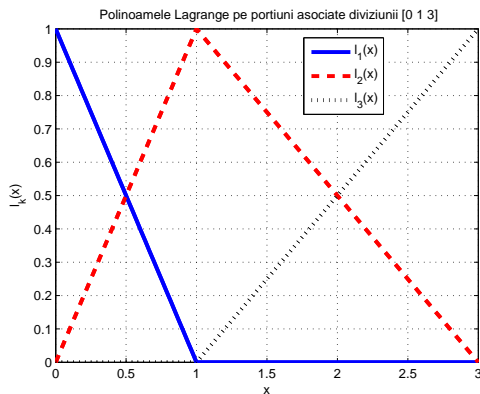


Figura 4.6: Funcții Lagrange pe porțiuni.

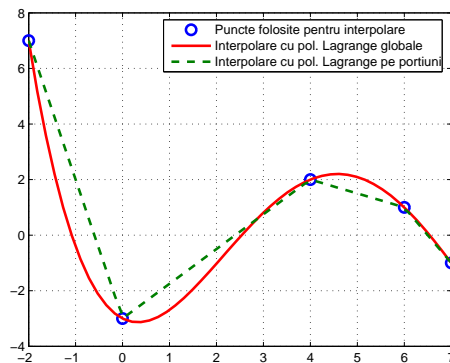


Figura 4.7: Interpolare globală și interpolare pe porțiuni.

Exercițiul 4.9:

a) Completați funcția `my_interp1` cu cazul "lagrangep".

b) Adaugați la scriptul `main_interp1` cazul interpolării pe porțiuni. Rularea lui trebuie să vă conducă la un rezultat similar cu cel din fig. 4.7.

Interpolarea obținută cu ajutorul polinoamelor Lagrange pe porțiuni este cunoscută sub numele de *interpolare liniară pe porțiuni*. Evaluarea polinomului de interpolare liniară pe porțiuni poate fi implementată cu un algoritm mai simplu, bazat pe căutarea intervalului în care se află punctul în care se dorește 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));

```

Exercițiul 4.10:

a) Copiați o funcție de căutare binară implementată anterior într-un fișier `cauta.m`.

Modificați numele funcției și modificați sintaxa acesteia astfel încât să poată fi realizată extrapolarea funcției. În afara domeniului de definiție funcția se va aproxima prin prelungirea primului și, respectiv, ultimului segment al interpolării liniare pe porțiuni.

b) Adaugați la funcția `my_interp1` cazul `lpp` al interpolării pe porțiuni implementată prin căutare binară. Verificați corectitudinea codului implementat pe un exemplu simplu.

Interpolarea liniară pe porțiuni, este o metodă de interpolare în care nu apare efectul Runge și poate fi aplicată și în cazul în care funcția de interpolat nu este cunoscută prin cod. Ea are dezavantajul că polinomul de interpolare generat nu este neted, are "colțuri" în nodurile rețelei de interpolare. Din punct de vedere matematic, funcția obținută nu este derivabilă în nodurile rețelei de discretizare.

Ne dorim însă uneori ca polinomul de interpolare să fie nu numai continuu, ci și derivabil. Soluția constă în folosirea *interpolării Hermite*. Problema interpolării Hermite are ca date nu numai valorile funcției într-o mulțime discretă de puncte, ci și 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 $[x_k, x_{k+1}]$ condițiile de interpolare impun acum atât valorile funcției cât și valorile derivatei. Impunând patru condiții de interpolare pentru fiecare interval, rezultă că pentru fiecare astfel de interval vor exista patru coeficienți ai polinomului local de interpolare, care va avea, în consecință, gradul trei. Este util dacă acest polinom se scrie sub forma

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

Impunerea celor patru *condiții de interpolare Hermite*:

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

permite calculul coeficienților de interpolare pentru fiecare polinom de interpolare pe porțiuni:

$$c_{0k} = y_k, \quad (4.15)$$

$$c_{1k} = y'_k, \quad (4.16)$$

$$c_{2k} = \frac{1}{(x_{k+1} - x_k)^2} [3(y_{k+1} - y_k) - (x_{k+1} - x_k)(2y'_k + y'_{k+1})], \quad (4.17)$$

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

Exercițiul 4.11:

Demonstrați relațiile (4.15) ÷ (4.18).

De regulă însă derivatele funcției nu sunt cunoscute și atunci ele trebuie estimate. O variantă este aceea de a folosi pentru estimarea lor formule de derivare numerică (caz în care interpolarea se numește Bessel). O metodă mai avantajoasă este cea în care derivatele necesare sunt determinate din condiții 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 funcția de interpolare obținută minimizează curbura pătratică medie a funcției. Este util de precizat că *spline* este un cuvânt preluat din limba engleză, care înseamnă printre altele, o bucată de lemn flexibil, ce era folosit pentru desenarea curbelor. Pentru o astfel de aplicație, acest obiect avea tendința de a se orienta astfel încât curbura să fie minimă.

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

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

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

$$g''(x_k + 0) = g''(x_k - 0), \quad k = 2, \dots, n-1. \quad (4.20)$$

Înlocuind expresia (4.19) precum și expresiile coeficienților (4.15) ÷ (4.18) în (4.20) rezultă următoarele relații satisfăcute de derivatele funcției:

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

Exercițiul 4.12:

Demonstrați relația (4.21).

Relațiile (4.21) sunt în număr de $n - 2$ și au n necunoscute. Cele două relații lipsă se obțin din impunerea condițiilor la capete. Fie se impun valorile derivatelor y'_1 și y'_n ca la interpolarea Bessel, caz în care se spune că interpolarea spline are condiții la capete *forțate*, fie se impune ca valorile derivatei de ordinul doi să fie nule la capete, caz în care se spune că interpolarea spline are condiții *naturale* la capete. Vom considera în cele ce urmează condițiile naturale:

$$g''(x_1) = 0, \quad (4.22)$$

$$g''(x_n) = 0. \quad (4.23)$$

Din impunerea condiției (4.22) rezultă următoarea relație ce trebuie să existe între derivatele funcției:

$$2y'_1 + y'_2 = 3 \frac{y_2 - y_1}{x_2 - x_1}. \quad (4.24)$$

Exercițiul 4.13:

- a) Demonstrați relația (4.24).
- b) Deduceți relația ce trebuie impusă între derivate pentru a fi satisfăcută condiția naturală (4.23) la capătul din dreapta.

Exercițiul 4.14:

- a) Scrieți o funcție Matlab ca mai jos, completând acolo unde este cazul. Calculul derivatelor y'_k în nodurile de interpolare se face prin rezolvarea unui sistem de ecuații algebrice liniare, având matricea coeficienților A și vectorul termenilor liberi b. Valorile componentelor din A și b corespund coeficienților din relațiile (4.21), (4.24) și relația dedusă la exercițiul anterior.

```
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) Verificați funcția scrisă cu comanda `mlint`.

Exercițiul 4.15:

a) Scrieți un script `main_spline.m` cu următorul conținut:

```

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) Verificați că executându-l obțineți rezultatul din fig.4.8.

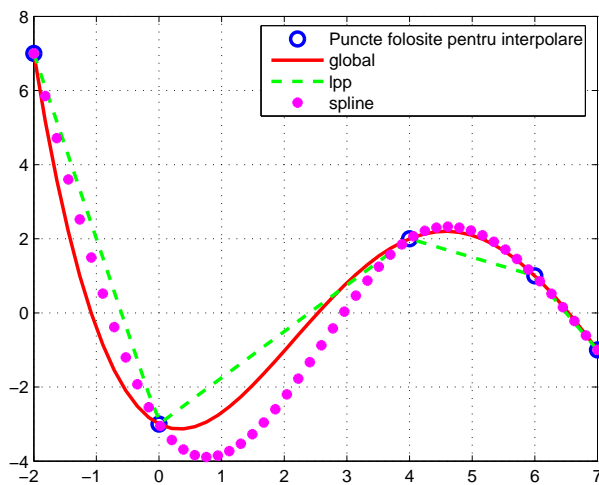


Figura 4.8: Pentru exercițiul 4.15.

În Matlab există funcții de interpolare. Scopul acestei teme a fost însă înțelegerea problemei interpolării precum și modul de concepere al algoritmilor de interpolare.

Exercițiul 4.16:

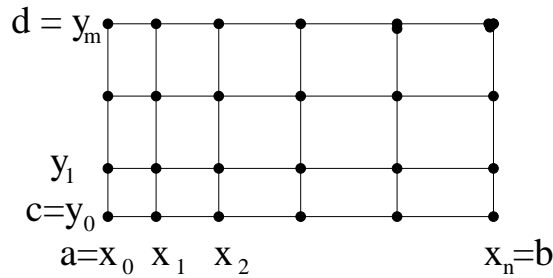
- Citiți documentația funcțiilor `interp1` și `spline`.
- Completați scriptul `main_spline.m` cu apelul funcțiilor Matlab și comparați rezultatele.

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 funcție $f : \Omega \rightarrow \mathbb{R}$, unde Ω este un domeniu bidimensional $[a, b] \times [c, d]$, cunoscută într-un număr de puncte din Ω . Presupunem că funcția este cunoscută în nodurile unei

Figura 4.9: Rețeaua bidimensională de discretizare a lui Ω

rețele date de diviziunea pe Ox:

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

și de diviziunea pe Oy:

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

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

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

Interpolând apoi după y , rezultă:

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

În consecință, funcțiile Lagrange asociate unei rețele bidimensionale sunt:

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

Exercițiul 4.17:

Implementarea în Matlab a funcției de calcul a polinomului Lagrange bidimensional este imediată. Scrieți un fișier `lagrange2d.m` cu următorul conținut.

```
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';
```

Completați în acest fișier comentariile care descriu parametrii de intrare și de ieșire ai funcției.

Exercițiul 4.18:

Pentru testarea funcției implementate în exercițiul anterior, scrieți un script cu următorul conținut:

```
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,1,2,xi,yi);
figure(1);
surf(xi,yi,zi'); % Atentie, zi este transpus
xlabel('x');
ylabel('y');
zlabel('z');
colorbar
```

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

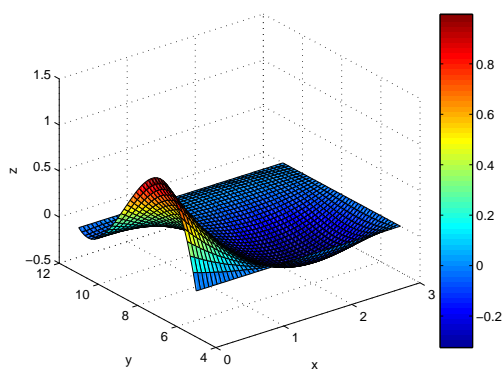


Figura 4.10: Funcția Lagrange asociată gridului bidimensional $[0 \ 1 \ 3] \times [5 \ 7 \ 10 \ 11]$ și nodului (1,2).

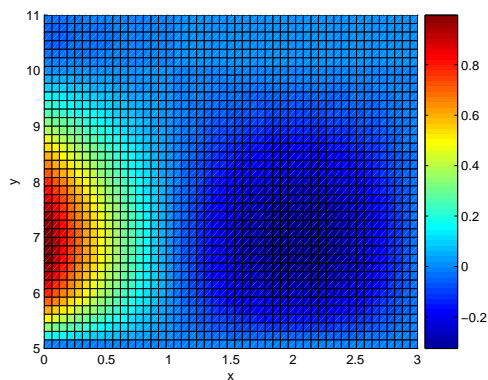


Figura 4.11: Funcția Lagrange asociată nodului (1,2) - vedere perpendiculară pe xOy.

Exercițiul 4.19:

- a) Cărui nod îi este asociată funcția Lagrange având reprezentarea din fig. 4.12?
 b) O altă formă de vizualizare o puteți obține cu comanda `mesh`. Adăugați scriptului de mai sus liniile

```
figure(2);
mesh(xi,yi,zi');
xlabel('x');
ylabel('y');
zlabel('z');
colorbar
```

Ce reprezintă fig. 4.13?

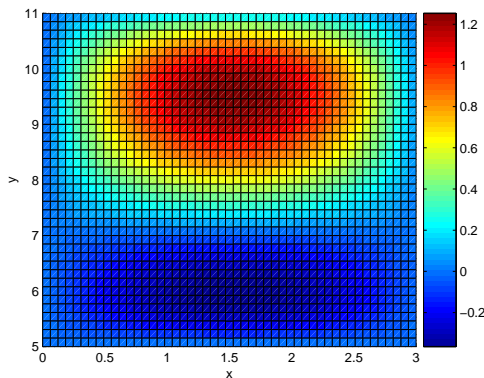


Figura 4.12: Pentru exercițiul 4.19. Cărui nod îi este asociată această funcție Lagrange?

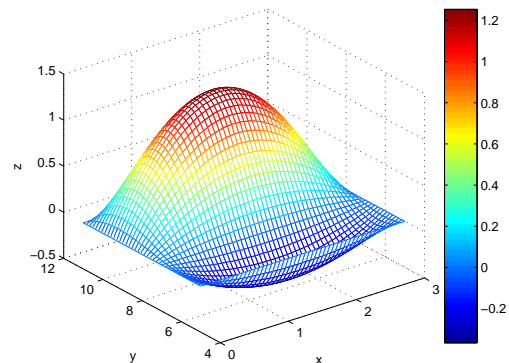


Figura 4.13: Pentru exercițiul 4.19. Ce reprezintă această figură?

Pentru funcțiile scalare definite pe domenii bidimensionale este foarte sugestivă reprezentarea curbilor de nivel (adică curbe pe care valorile funcției reprezentate sunt constante, curbe numite și curbe de echivalori, sau în mod particular, în funcție de mărimea pe care o reprezintă: echipotențiale, izoterme, etc).

Exercițiul 4.20:

- a) Adăugați 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 obțineți fig. 4.14. Se vede acum cu ușurință care este nodul căruia îi este asociată funcția Lagrange reprezentată. În acest nod, valoarea funcției este 1. Comparați rezultatul cu răspunsul pe care l-ați dat la exercițiul 4.19.

b) Care este diviziunea domeniului folosită pentru reprezentarea unei funcții Lagrange în fig. 4.15. Cărui nod îi este asociată această funcție?

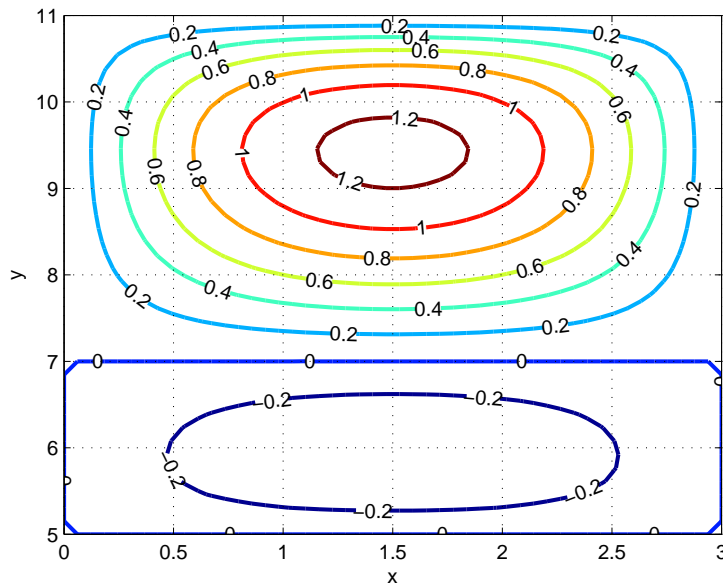


Figura 4.14: Pentru exercițiul 4.20. Efectul comenzii `contour`.

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

Exercițiul 4.21:

Citiți documentația pentru funcția Matlab `interp2`. Descrieți succint metodele de interpolare disponibile.

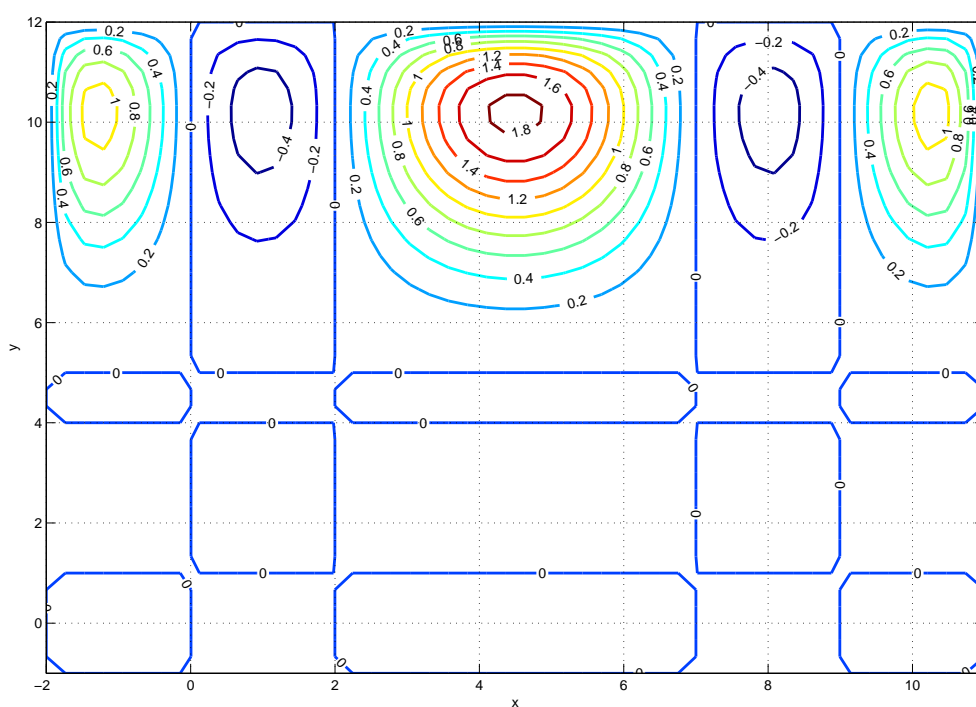


Figura 4.15: Pentru exercițiul 4.20. Care este diviziunea domeniului? Cărui nod îi este asociată funcția?

Capitolul 5

Derivarea numerică.

Metoda diferențelor finite.

Necesitatea evaluării derivatelor funcțiilor este importantă în problemele de inginerie. De exemplu, ele apar în evaluarea sensibilităților unor mărimi de interes în raport cu anumiți parametri considerați variabili, sensibilități utile în proiectarea optimală a unor dispozitive. Un alt exemplu este cel în care modelul matematic al problemei conduce la ecuații sau sisteme de ecuații diferențiale, foarte puține dintre acestea putând fi rezolvate prin metode analitice.

Derivarea numerică se bazează pe interpolare. Formulele de derivare numerică provin din derivarea polinomului de interpolare și ele se reduc la calcule aritmetice ce folosesc formule relativ simple. Prima parte a acestui capitol ilustrează deducerea acestor formule și erorile care apar. Rezolvarea ecuațiilor diferențiale folosind formule de derivare numerică pentru derivatele ce intervin este cunoscută sub numele de *metoda diferențelor finite*. Partea a doua a acestui capitol urmărește implementarea și 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 și informații legate de eroarea de trunchiere.

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

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

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

- Aproximarea derivatei de ordinul doi cu o eroare de ordinul¹ h :

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

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 și x_{i+1} .

Exercițiul 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}) deduceți una din formulele de derivare de mai sus

Exercițiul 5.2:

a) Scrieți într-un fișier 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) Comentați valorile obținute pentru cele trei formule. Schimbați pasul de derivare și comentați rezultatele.

Este foarte probabil ca în acest moment să fiți tentați să credeți că, cu cât pasul de derivare este mai mic, cu atât eroarea e mai mică. Pentru a vedea dacă această afirmație este adevărată, este util să reprezentați grafic eroarea în funcție de pasul de derivare.

Exercițiul 5.3:

Scrieți un script care să permită:

- a) Reprezentarea grafică a erorii formulei de derivare progresivă de ordinul 1, în funcție de pasul de derivare. Pentru un pas h cuprins între $1e-10$ și $1e-5$ trebuie să obțineți un grafic similar celui din partea stângă a fig. 5.1. Acest grafic a fost trasat folosind $1e5$ valori. Repetați experiența pentru derivarea regresivă de ordinul 1. Comentați rezultatul.
- c) Reprezentarea grafică a erorii formulei de derivare centrată de ordinul 2, în funcție de

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

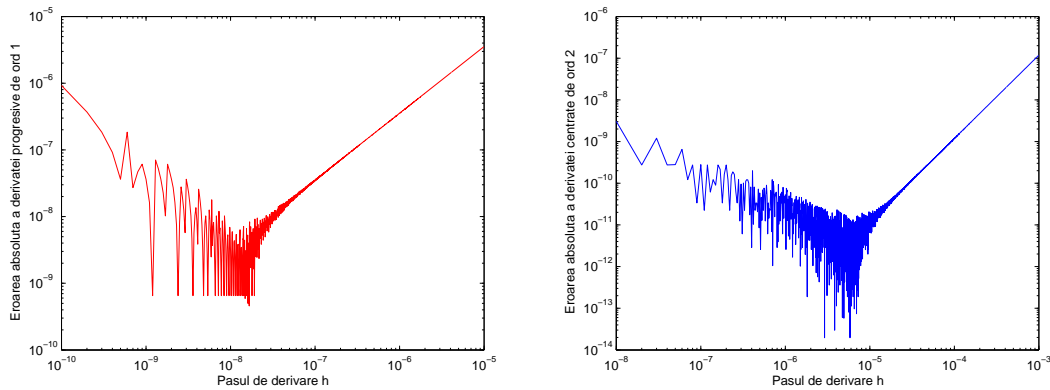


Figura 5.1: Eroarea în funcție de pasul de derivare pentru formula de derivare progresivă de ordinul 1 (stânga) și formula de derivare centrată de ordinul 2 (dreapta).

Zona de instabilități numerice, care apare pentru pași 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 predominere erorile de trunchiere se numește *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 obține pentru pași aflați în zona de instabilitate. Totuși 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.

Exercițiul 5.4:

Inspectând graficele obținute

- comparați pașii optimi de derivare numerică pentru cele trei formule pe care le-ați folosit până acum.
- verificați că eroarea de trunchiere la formulele progresivă și regresivă de ordinul 1 este $O(h)$ iar la formula centrată de ordinul 2 este $O(h^2)$.

Derivata centrată (5.4) oferă o acuratețe mai bună decât formulele progresivă, regresivă de ordinul 1 (formulele (5.2) și (5.3)). O comparație numerică este sugerată în exercițiul următor.

Exercițiul 5.5:

Reprezentați pe același grafic eroarea derivatei numerice progresive de ordinul 1 și centrate de ordinul 2 pentru pași cuprinși între $1e-8$ și $1e-3$. Trebuie să obțineți un grafic similar celui din figura 5.2.

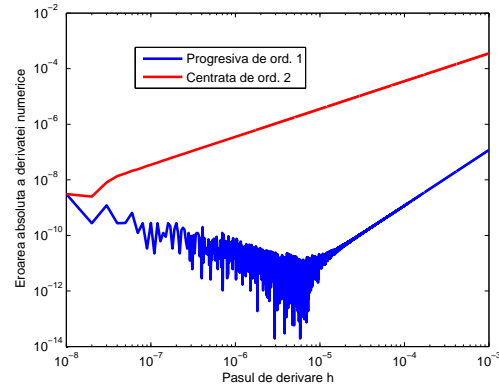


Figura 5.2: Acest grafic trebuie obținut la exercițiul 5.5.

5.2 Metoda diferențelor finite

Aplicarea metodei diferențelor finite pentru rezolvarea oricărei ecuații diferențiale presupune realizarea următorilor pași:

- **Pasul 1:** Se alege o schemă de diferențe finite pentru aproximarea derivatelor din ecuații și se rescrie ecuația ca ecuație cu diferențe finite.
- **Pasul 2:** Se stabilește rețeaua de discretizare și se scrie ecuația discretizată pentru fiecare nod.
- **Pasul 3:** Se rezolvă sistemul de ecuații pentru determinarea valorilor necunoscute în nodurile rețelei.
- **Pasul 4:** Calculul altor valori, în afara celor din noduri, se face prin interpolare.

Se obișnuiește să se spună că pașii 1 și 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 ecuație diferențială ordinară de ordinul 1. Vom considera un condensator de capacitate $C = 4\mu\text{F}$, inițial descărcat, care se încarcă de la o sursă de tensiune continuă $E = 20\text{ mV}$ printr-un rezistor de rezistență $R = 10\ \Omega$.

Formularea corectă a problemei

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

$$Ri(t) + u(t) = E. \quad (5.12)$$

Înlocuind în această relație dependența dintre tensiunea și curentul prin condensator

$$i(t) = C \frac{du(t)}{dt}, \quad (5.13)$$

rezultă ecuația diferențială de ordinul unu, satisfăcută de u :

$$RC \frac{du(t)}{dt} + u(t) = E. \quad (5.14)$$

Soluția acestei ecuații este unică deoarece se specifică starea inițială a condensatorului:

$$u(0) = u_0 = 0. \quad (5.15)$$

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

Conform teoriei cunoscute de la matematică, soluția acestei ecuații se scrie

$$u(t) = u_o(t) + u_p, \quad (5.16)$$

unde $u_o(t)$ este soluția ecuației "omogene" (cum zic matematicienii) adică a ecuației diferențiale cu termen liber nul:

$$RC \frac{du_o(t)}{dt} + u_o(t) = 0. \quad (5.17)$$

Soluția ecuației omogene este

$$u_o(t) = A \exp(xt), \quad (5.18)$$

unde x este soluția ecuației caracteristice

$$RCx + 1 = 0. \quad (5.19)$$

Dacă notăm produsul dintre rezistență și capacitate cu

$$\tau = RC, \quad (5.20)$$

mărime care se numește *constantă de timp* a circuitului, rezultă soluția ecuației omogene

$$u_o(t) = A \exp(-t/\tau). \quad (5.21)$$

Mărimea u_p este o soluție particulară, de forma termenului liber al ecuației, deci o constantă. Să notăm această constantă cu B . Soluția particulară (constantă) trebuie să satisfacă ecuația (5.14), deci

$$RC \frac{dB}{dt} + B = E, \quad (5.22)$$

de unde rezultă

$$B = E \quad (5.23)$$

deoarece derivata unei constante este nulă. Soluția generală a ecuației este în consecință

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

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

$$A = u_0 - E. \quad (5.25)$$

Expresia finală a soluției analitice este

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

Exercițiul 5.6:

- a) Care este valoarea spre care tinde tensiunea pe condensator?
- b) Scrieți un script Matlab `main_RC.m` cu următorul conținut

```
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';
```

Executați-l și verificați că ați răspuns corect la punctul a.

Rezolvarea cu diferențe finite

Vom rezolva acum ecuația diferențială cu diferite scheme de derivare numerică. Ecuația (5.14) o rescriem ca

$$\frac{du(t)}{dt} + \frac{1}{\tau}u(t) = \frac{1}{\tau}E. \quad (5.27)$$

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

$$t_{k+1} - t_k = h, \quad \text{pentru } k = 1, \dots, N - 1. \quad (5.28)$$

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

$$u_k \approx u(t_k). \quad (5.29)$$

Varianta I - Utilizarea diferențelor finite progresive de ordinul 1

Vom discretiza ecuația (5.27) prin scrierea ei în momentul de timp t_k și folosind pentru derivată o formulă de diferențe finite progresive de ordinul 1:

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

de unde

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

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

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

Această formulă este extrem de ușor de implementat.

Exercițiul 5.7:

a) Completați 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) Scrieți o funcție `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
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));
```

Experimentați comportamentul metodei pentru $h = 2\tau, \tau, \tau/10$. Observați că metoda este instabilă pentru $h < \tau$. Această metodă este cunoscută și sub numele de metoda Euler explicită pentru rezolvarea ecuațiilor diferențiale ordinare.

Varianta a II-a - Utilizarea diferențelor finite regresive de ordinul 1

Vom discretiza ecuația (5.27) prin scrierea ei în momentul de timp t_k și folosind pentru derivată o formulă de diferențe finite regresive de ordinul 1:

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

de unde

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

Se observă că mărimea u_k poate fi calculată explicit cu formula

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

Și această formulă este extrem de ușor de implementat.

Exercițiul 5.8:

a) Completați funcția `mdf_circuitRC.m` cu cazul "regresiv1":

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

b) Completați scriptul cu apelul funcției în cazul regresiv

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

și partea de postprocesare cu reprezentarea grafică a soluției obținute, pe același grafic ca soluția analitică și soluția obținută în prima variantă.

Experimentați comportamentul metodei pentru $h = 2\tau, \tau, \tau/10$. Observați că metoda nu mai este instabilă pentru $h < \tau$. Această metodă este cunoscută și sub numele de metoda Euler implicită² pentru rezolvarea ecuațiilor diferențiale ordinare.

Varianta a III-a - Utilizarea diferențelor finite centrate de ordinul 2

Vom discretiza ecuația (5.27) prin scrierea ei în momentul de timp t_k și folosind pentru derivată o formulă de diferențe finite centrate de ordinul 2 dată de relația (5.4):

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

de unde

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

Se observă că mărimea u_k nu mai poate fi calculată explicit. Relația (5.37) poate fi scrisă pentru $k = 2, \dots, N - 1$ și reprezintă $N - 2$ ecuații cu N necunoscute. Pentru a obține

²Ecuația generală este $dy/dt = f(t, y)$ unde f este o funcție în general neliniară. Folosirea derivatei regresive pentru f conduce la o relație implicită pentru calculul lui u_k , relație ce reprezintă o ecuație neliniară. În cazul particular studiat, f este liniară și de aceea soluția se poate calcula explicit.

un sistem bine formulat, trebuie să adăugăm încă două relații. Acestea sunt relațiile la capete. La $t = 0$ vom impune condiția inițială:

$$u(1) = u_0, \quad (5.38)$$

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

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

de unde

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

Exercițiul 5.9:

a) Completați funcția `mdf_circuitRC.m` cu cazul "centrat2":

```
case 'centrat2'
    %% asamblare sistem
    A = sparse(N,N);
    t1 = zeros(N,1);
    A(1,1) = 1;           % conditia initiala
    t1(1) = u_initial;
    for k = 2:N-1        % noduri interioare
        A(k,k-1) = -1;
        .....completati.....
    end
    A(N,N-2) = 1;       % conditia la tmax
    .....completati.....
    t1(N) = 2*h*E/tau;
    %% rezolvare
    u = A\t1;
```

b) Completați scriptul cu apelul funcției în cazul centrat și partea de postprocesare cu reprezentarea grafică a soluției obținute, pe același grafic ca soluția analitică și soluția obținută în prima variantă.

Experimentați comportamentul metodei pentru $h = 2\tau, \tau, \tau/10$. Observați că metoda nu este instabilă și este mai precisă decât implementările anterioare. Acest lucru era de așteptat deoarece formulele de derivare numerică de ordinul 2 sunt mai precise decât cele de ordinul 1. Creșterea acurateții se face însă pe seama creșterii complexității algoritmului.

Pentru verificare, comparați graficele pe care le-ați obținut cu cele din figurile 5.3 ÷ 5.5.

Exercițiul 5.10:

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

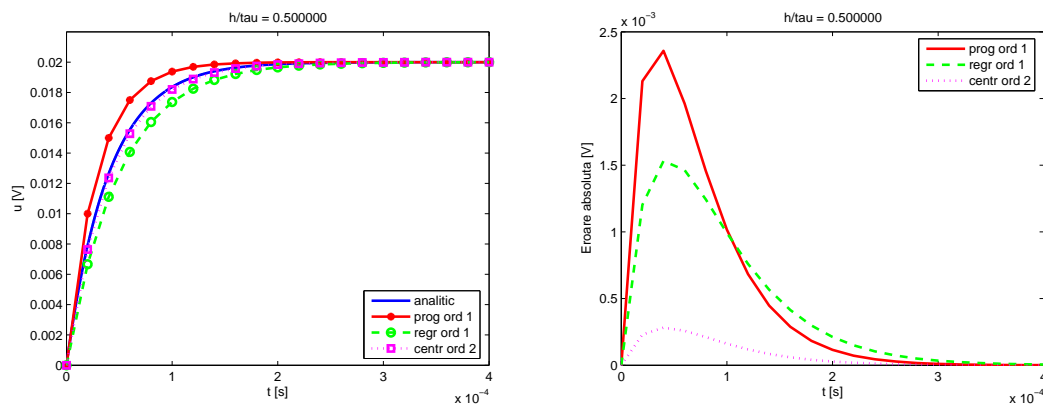


Figura 5.3: Cazul $h = \tau/2$.

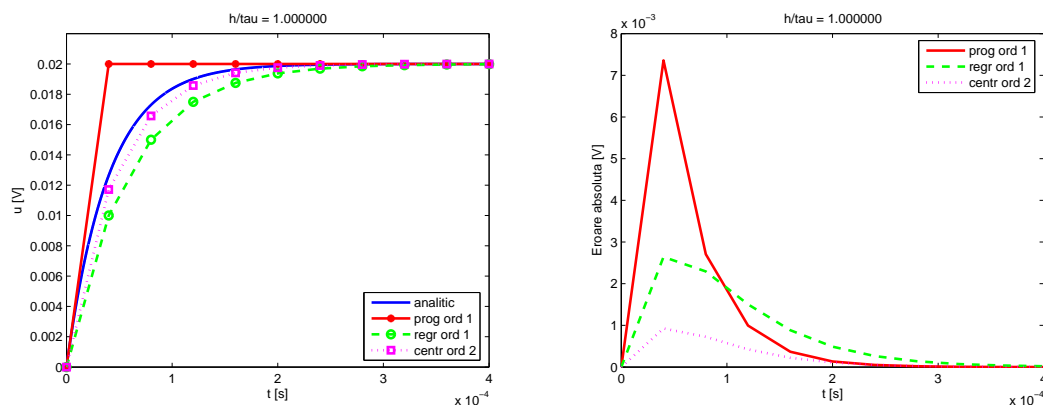


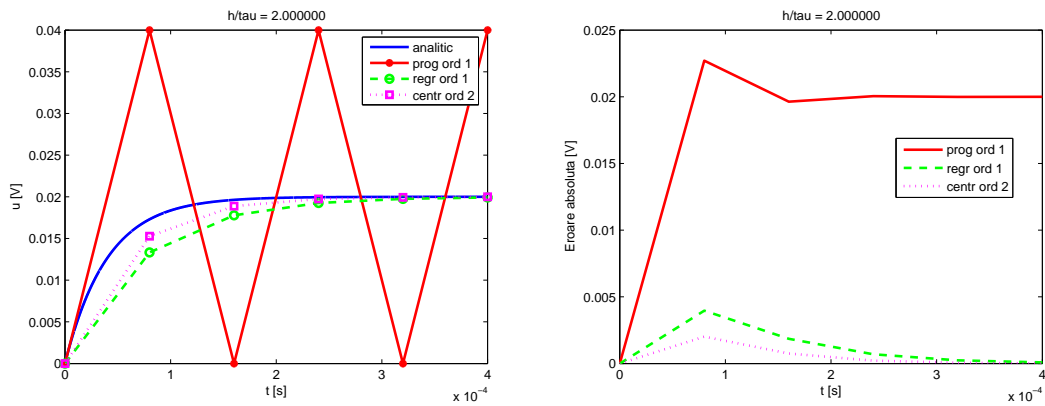
Figura 5.4: Cazul $h = \tau$.

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

Exercițiul 5.11:

a) Completați funcția `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

```


$$u = A \setminus t1;$$

b) Completați scriptul cu apelul funcției în acest caz. Pentru a verifica faptul că nu s-au strecurat greșeli, reprezentați grafic într-o altă fereastră soluția obținută în cazul "centrat2b" cu soluția obținută în cazul "centrat".

c) Introduceți instrucțiuni pentru contorizarea timpului de asamblare al matricei în ambele variante de implementari. Comparați acești timpi pentru cazul $h = \tau/1000$.

5.2.2 Studiul regimului electrocinetic staționar al unui conductor bidimensional

Exemplul din acest paragraf va conduce la o ecuație diferențială cu derivate parțiale de ordinul 2, de tip eliptic.

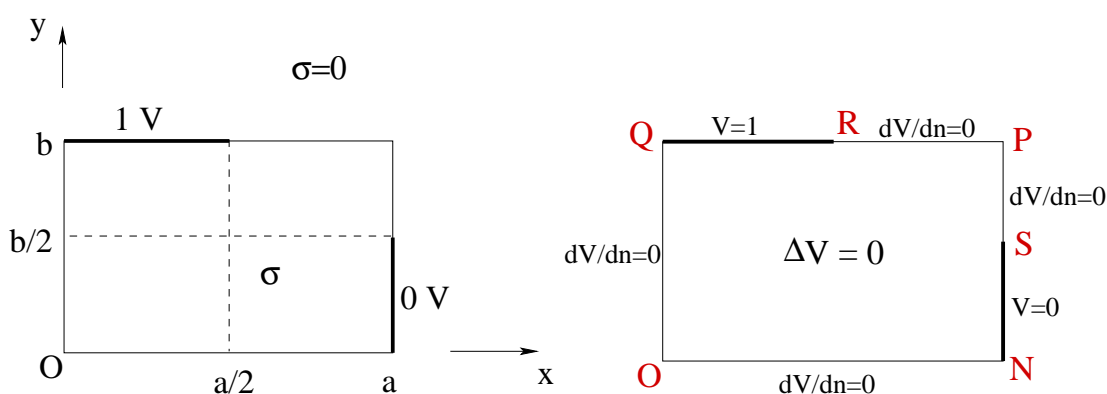


Figura 5.6: Problema 2D de regim electrocinetic: domeniul de calcul (stânga), condiții de frontieră (dreapta).

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, astfel încât problema poate fi considerată bidimensională. Figura 5.6 reprezintă o secțiune perpendiculară pe direcția dimensiunii foarte mari. Această secțiune este un dreptunghi, de dimensiuni a , b . Conductorul are două borne supraconductoare, una aflată la potențial $V_0 = 1\text{ V}$, iar cealaltă aflată la potențialul nul. Bornele sunt plasate ca în figură. Dorim să reprezentăm liniile echipotențiale și spectrul liniilor de curent. Ar fi interesant să calculăm și mărimi globale cum ar fi puterea disipată în domeniu precum și rezistența 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 staționar, formularea se va face în V – potențial electrocinetic, unde $\mathbf{E} = -\text{grad } V$, \mathbf{E} fiind intensitatea câmpului electric.

Ecuția de ordinul doi satisfăcută de potențialul electrocinetic este:

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

unde $V : \mathcal{D} \rightarrow \mathbb{R}$ este potențialul definit pe domeniul bidimensional $\mathcal{D} = \text{ONPQ}$. Ecuția (5.41) este o ecuație eliptică, de tip Laplace generalizată.

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

$$\Delta V = 0, \quad (5.42)$$

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

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

În consecință, ecuația diferențială ce trebuie rezolvată este o ecuație cu derivate parțiale:

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

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

Pentru buna formulare a problemei, trebuie impuse pentru potențial condițiile de frontieră. Frontierele supraconductoare (SN și QR) au potențial constant, și în consecință pe ele trebuie impuse condiții Dirichlet în concordanță cu valorile potențialului. Pe frontierele de lângă izolanț (NOQ și RPS), trebuie impuse condiții Neumann nule³.

Condițiile de frontieră impuse potențialului sunt:

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

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

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

³În izolanț nu există curent de conducție, în consecință $\mathbf{J} \cdot \mathbf{n} = 0$ la frontiera cu izolanțul, deci $-\sigma \text{grad } V \cdot \mathbf{n} = 0$, echivalent cu $-\sigma \frac{\partial V}{\partial n} = 0$ pe acele frontiere.

Și pentru o astfel de problemă se pot găsi rezolvări analitice bazate de exemplu pe metoda separării variabilelor sau pe aproximarea liniilor de câmp. Deși nu este dificilă, prezentarea acestor posibile soluții 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 diferențelor finite așa încât în cele ce urmează ne vom concentra exclusiv asupra acestui aspect.

Rezolvarea cu diferențe finite

Față de cazul unidimensional discutat în exemplul anterior, aici rețeaua de discretizare este una bidimensională, iar în loc de aproximarea unei derivate totale de ordinul 1, acum avem de aproximat derivate parțiale de ordinul 2.

Rețeaua de discretizare bidimensională poate fi privită ca fiind obținută din produsul cartezian dintre o rețea de discretizare pe axa Ox și o rețea de discretizare pe axa Oy:

$$\begin{aligned} 0 &= x_1 < x_2 < \dots < x_{n_x} = a \\ 0 &= y_1 < y_2 < \dots < y_{n_y} = b. \end{aligned}$$

astfel încât, un nod al rețelei este identificat de poziția proiecțiilor sale pe cele două axe:

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

Metoda diferențelor finite va determina valorile aproximative ale potențialului în nodurile acestei rețele:

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

Forma discretizată (aproximativă) a ecuației potențialului este obținută aproximând prin diferențe finite ecuația (5.44). Forma specifică depinde de tipul nodului: interior sau pe frontiera Neumann. Nodurile de pe frontiera Dirichlet nu ridică probleme, ecuația asociată unui astfel de nod constând în simpla atribuire a valorii potențialului nodului respectiv.

Ecuația asociată unui nod interior:

Ecuația aproximativă pentru un nod interior se deduce înlocuind derivatele parțiale după x și după y cu formule de tipul (5.7).

Pentru a simplifica scrierea formulelor, vom presupune că cele două griduri pe Ox și, respectiv, Oy sunt uniforme, cu pași h_x și respectiv h_y . Derivatele parțiale se vor înlocui

cu:

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

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

Ecuția discretizată asociată unui nod interior devine

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

sau

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

Relația (5.52) arată că potențialul într-un nod interior este o combinație liniară a potențialelor nodurilor învecinate. Dacă pașii de discretizare pe cele două axe ar fi egali ($h_x = h_y$), atunci potențialul într-un nod interior este media aritmetică a potențialelor celor patru noduri vecine.

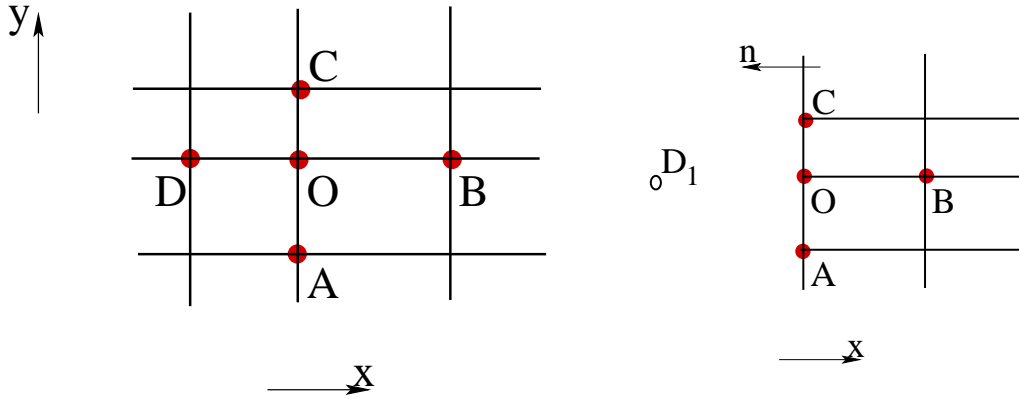


Figura 5.7: Nodurile marcate intervin în scrierea ecuațiilor: stânga - cazul unui nod interior; dreapta - cazul unui nod pe frontieră Neumann dreaptă.

Exercițiul 5.12:

În cazul unor rețele neuniforme, folosind notațiile din fig. 5.7-stânga, arătați că ecuația discretizată asociată unui nod interior este

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

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

Ecuția asociată unui nod pe frontieră dreaptă:

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

$$g_O = \frac{g_{OA}h_A + g_{OC}h_C}{h_A + h_C} \quad (5.54)$$

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

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

Din condiția de frontieră Neumann, deducem valoarea potențialului fantomă în funcție de valoarea acestei condiții. Pentru cazul din figură (normala exterioară în sens contrar axei Ox) rezultă:

$$\frac{\partial V}{\partial x} = \frac{g}{\sigma}. \quad (5.55)$$

Scriind relația aproximativă (5.4) pentru condiția de frontieră Neumann, rezultă că:

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

Pentru nodul O se scrie acum o ecuație de tipul (5.53), ca pentru un nod interior și înlocuind expresia (5.56), rezultă ecuația finală:

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

În cazul unei condiții Neumann nule, relația devine:

$$V_O \left(\frac{1}{h_B^2} + \frac{1}{h_A h_C} \right) - V_A \frac{1}{h_A(h_A + h_C)} - V_B \frac{1}{h_B^2} - V_C \frac{1}{h_C(h_A + h_C)} = 0. \quad (5.58)$$

Ecuția asociată unui nod pe frontieră - colț exterior:

Nodurile situate în colțuri reprezintă o problemă pentru metoda diferențelor finite. Aceasta deoarece pentru un colț nu poate fi definită direcția normalei⁴. Dacă notăm $g = -\sigma \frac{\partial V}{\partial n}$, și notăm cu g_{OA} valoarea lui g pe segmentul OA (fără capătul O) și 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 direcție care rezultă din prelungirea în O a valorilor funcției g .

⁴În realitate, colțul reprezintă o modelare brutală a realității. Trecerea de la o suprafață la alta se face prin racordări.

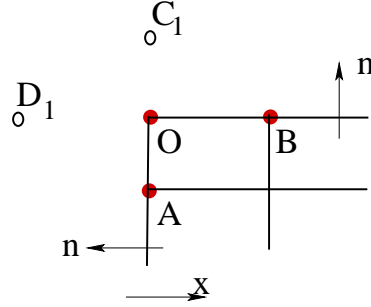


Figura 5.8: Nodurile marcate intervin în scrierea ecuației unui nod pe frontieră colț exterior.

Pentru deducerea ecuației vom considera două noduri “fantomă” C_1 și D_1 , situate în exterior, la distanțele $h_D = \|OD_1\|$ și $h_C = \|OC_1\|$ de punctul O . Pentru simplitate putem considera $h_D = h_B$ și $h_C = h_A$.

Din condiția de frontieră Neumann pe segmentul OB , scrisă în nodul O , putem deduce valoarea potențialului fantomă C_1 :

$$V_{C_1} = V_A - 2h_A \frac{g_{OB}}{\sigma}. \quad (5.59)$$

Din condiția de frontieră Neumann pe segmenul OA , scrisă în nodul O , putem deduce valoarea potențialului fantomă D_1 :

$$V_{D_1} = V_B - 2h_B \frac{g_{OA}}{\sigma}. \quad (5.60)$$

Pentru nodul O se scrie acum o ecuație de tipul (5.53), ca pentru un nod interior, și înlocuind expresiile (5.59) și (5.60), rezultă ecuația finală:

$$V_O \left(\frac{1}{h_A^2} + \frac{1}{h_B^2} \right) - V_A \frac{1}{h_A^2} - V_B \frac{1}{h_B^2} = -\frac{g_{OB}}{\sigma h_A} - \frac{g_{OA}}{\sigma h_B} \quad (5.61)$$

În cazul unor condiții Neumann nule, relația devine:

$$V_O \left(\frac{1}{h_A^2} + \frac{1}{h_B^2} \right) - V_A \frac{1}{h_A^2} - V_B \frac{1}{h_B^2} = 0 \quad (5.62)$$

În concluzie, discretizarea problemei conduce la un sistem de ecuații algebrice liniare, prin a cărui rezolvare vom obține valorile potențialelor în nodurile rețelei 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 notațiile de până acum. Vom numerota nodurile de jos în sus și de la stânga la dreapta,

Exercițiul 5.14:

a) Scrieți un script `mdf_elcin.m` cu următorul conținut:

```
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) Scrieți o funcție `citire_date_elcin.m` cu următorul conținut

```
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) Rulați codul scris până acum pentru a observa dacă s-au strecurat greșeli.

Acum începe partea de preprocesare. Mai întâi trebuie pregătite informațiile legate de rețeaua de discretizare. Vom presupune că acesta este uniform pe cele două axe.

Exercițiul 5.15:

a) Completați scriptul cu comenzile

```
%% ===== PREPROCESARE =====
%% alegerea re'telei
my_grid = generare_grid_elcin(date);
```

b) Scrieți o funcție `generare_grid_elcin.m` cu următorul conținut


```
function grid = generare_grid_elcin(date)
a = date.a;
b = date.b;
nx = 3;          % nr pct de discretizare pe 0x
ny = 3;          % nr pct de discretizare pe 0y
% pasi de discretizare pe 0x si 0y
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` și `grid.y`?

Urmează acum partea cea mai complicată, asamblarea sistemului de ecuații.

Exercițiul 5.16:

a) Completați scriptul cu comenzile

```
%% asamblare sistem
[A,t1] = asamblare_mdf_elcin(date,my_grid);
```

b) Scrieți o funcție `asamblare_mdf_elcin.m` cu următorul conținut:

```
function [A,t1] = 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);
t1 = 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;
    t1(k) = V1;
end
%% cond de frontiera Dirichlet - T2
..... completati .....

```

c) Verificați codul scris cu comanda `mlint`. Executați programul scris până acum și verificați că matricea coeficienților și vectorul termenilor liberi au valorile pe care le-ați obținut la exercițiul 5.13

În Matlab, rezolvarea acestui sistem este extrem de ușoară din punct de vedere al utilizatorului:

Exercițiul 5.17:

Completați scriptul cu comenzile

```
%% ===== REZOLVARE =====
V = A\tl;
```

Urmează etapa de postprocesare. Vom reprezenta liniile echipotențiale și vom desena rețeaua de discretizare.

Exercițiul 5.18:

a) Completați scriptul cu comenzile

```
%% ===== POSTPROCESARE =====
echipotentiale_elcin(my_grid,V);
draw_my_grid(my_grid);
```

b) Scrieți o funcție `echipotentiale_elcin.m` cu următorul conținut

```
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
.....completati.....(ceva similar ati facut la capitolul 1)
```

c) Scrieți o funcție `draw_my_grid.m` cu următorul conținut

```
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
.....completati.....

```

d) Verificați că ați lucrat corect, comparând rezultatele cu cele din fig. 5.10.

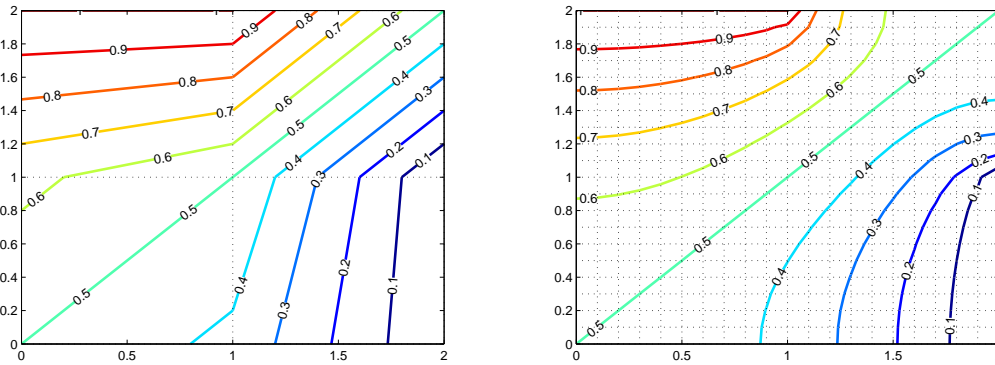


Figura 5.10: Linii echipotențiale pentru o rețea cu $n_x = n_y = 3$ (stânga) și o rețea cu $n_x = n_y = 21$ (dreapta).

Exercițiul 5.19:

- Observați cu atenție fig. 5.10 - stânga și descrieți ideea care stă la baza algoritmului de desenare al curbelor de echivalori.
- Propuneți îmbunătățiri posibile pentru funcția de asamblare.
- Știind că $\mathbf{E} = -\text{grad } V$, scrieți formulele cu care se poate calcula câmpul electric în nodurile rețelei de discretizare.

5.2.3 Studiul propagării undei scalare

În acest paragraf vom rezolva cu metoda diferențelor finite cel mai simplu exemplu de ecuație hiperbolică, și anume *ecuația undei scalare*:

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2}, \quad (5.64)$$

unde mărimea scalară necunoscută $u(x, t) : [0, a] \times [0, T] \rightarrow \mathbb{R}$ depinde de o singură coordonată spațială și de timp, iar v este o constantă cunoscută. Soluția acestei ecuații este o *undă* care se propagă cu viteză v .

Buna formulare a acestei probleme necesită impunerea

- *condiției inițiale* $u(x, 0) = h_0(x)$,
- *condițiilor la capete* - relații în care intervin mărimile $u(0, t) = h_1(t)$ și $u(a, t) = h_2(t)$.

Se poate demonstra că soluția ecuației unei scalare (5.64) este

$$u(x, t) = ud(x, t) + ui(x, t) + U_0, \quad (5.65)$$

unde $ud(x, t)$ se numește *undă directă* și este o funcție care se poate scrie sub forma

$$ud(x, t) = f(x - vt), \quad (5.66)$$

iar $ui(x, t)$ se numește *undă inversă* și este o funcție care se poate scrie sub forma

$$ui(x, t) = g(x + vt). \quad (5.67)$$

Funcțiile f și g rezultă în mod univoc, din impunerea condițiilor inițiale și condițiilor la capete. Mărimea $f(x - vt)$ este o undă directă deoarece o anumită valoare a acestei funcții, într-un anumit punct x și 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 raționament similar se poate face pentru unda inversă.

Următorul exercițiu vă permite să vizualizați conceptul de undă. El nu reprezintă rezolvarea numerică a vreunei ecuații diferențiale.

Exercițiul 5.20:

a) Scrieți o funcție cu următorul conținut

```
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 temporală

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);

```

Observați rezultatul executării ei.

- b) Modificați funcția astfel încât să vizualizați propagarea unei unde inverse.
c) Modificați funcția astfel încât să vizualizați propagarea sumei dintre unda directă și unda inversă. Veți avea nevoie să fixați axele reprezentării grafice. Pentru aceasta, înainte de prima comandă `drawnow` adăugați

```

xmin = 0; xmax = a; ymin = -2.2; ymax = 2.2;
axis([xmin, xmax,ymin,ymax]);

```

și înainte de a doua comandă `drawnow` adăugați doar comanda `axis` ca mai sus. Ceea ce obțineți sunt *unde staționare*.

Vom concepe acum un algoritm bazat pe metoda diferențelor finite, care rezolvă ecuația (5.64). Ca exemplu numeric vom considera $a = 1$ m, $v = 1$ m/s.

Avem nevoie de o discretizare spațială și de una temporală. Pentru a simplifica prezentarea, vom presupune că ambele discretizări sunt uniforme și vom nota cu Δx pasul discretizării spațiale și cu Δt pasul discretizării temporale. Vom nota cu N numărul de puncte de discretizare spațiale și cu M numărul de pași de timp simulați. În consecință, timpul maxim de simulare este $T = M\Delta t$.

Într-un prim exemplu, vom considera condiții inițiale nule și condiții Dirichlet la ambele capete:

- condiția inițială nulă $u(x, 0) = h_0(x) = 0$,
- condiția la capătul din stânga - excitația cu un impuls Gauss: $u(0, t) = h_1(t) = \exp(-(t - T/10)^2 / (2 * (T/50)^2))$
- condiția la capătul din dreapta $u(a, t) = h_2(t) = 0$.

Mărimea $u(x, t)$ este discretizată atât în spațiu cât și în timp. Prin rezolvarea cu metoda diferențelor finite, vom obține aproximații pentru valorile reale $u(x_k, t_j)$.

$$u(x_k, t_j) \approx u_k^{(j)}, \quad k = 1, \dots, N, \quad j = 1, \dots, M. \quad (5.68)$$

Alegând pentru derivatele ce intervin în (5.64) formule de derivare ce provin din polinomul de interpolare de ordin doi, rezultă următoarea relație discretizată:

$$\frac{u_k^{(j-1)} - 2u_k^{(j)} + u_k^{(j+1)}}{(\Delta t)^2} = v^2 \frac{u_{k-1}^{(j)} - 2u_k^{(j)} + u_{k+1}^{(j)}}{(\Delta x)^2}, \quad (5.69)$$

de unde rezultă că mărimea u , într-un anumit punct k și într-un anumit moment de timp $j + 1$, depinde explicit de valoarea în acel punct și în cele învecinate în momentul de timp anterior j și de valoarea în acel punct în momentul $j - 1$:

$$u_k^{(j+1)} = \left(\frac{v\Delta t}{\Delta x} \right)^2 \left(u_{k-1}^{(j)} - 2u_k^{(j)} + u_{k+1}^{(j)} \right) + 2u_k^{(j)} - u_k^{(j-1)}, \quad k = 2, \dots, N-1, \quad j = 0, \dots, M-1. \quad (5.70)$$

Momentul -1 îl vom considera identic cu momentul inițial 0.

Exercițiul 5.21:

a) Scrieți un script cu următorul conținut

```
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 temporală 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) Observați ce se întâmplă după reflexie (fig. 5.11). În momentul reflexiei, apare o undă inversă deoarece condiția Dirichlet la capătul din dreapta impune ca mărimea să fie zero. În consecință, unda inversă la capătul din dreapta este exact opusul undei directe, și ea se propagă în sens contrar axei Ox.

c) Cazul implementat mai sus corespunde situației $\Delta x = v\Delta t$. Observați ce se întâmplă dacă $\Delta x > v\Delta t$. Alegeți de exemplu $N = 100$, $M = 500$.

c) Observați ce se întâmplă dacă $\Delta x < v\Delta t$. Alegeți de exemplu $N = 100$, $M = 500$.

În concluzie, atunci când o problemă necesită atât o discretizare spațială cât și una temporală, pentru ca soluția numerică să fie stabilă, este necesar să fie îndeplinită *condiția*

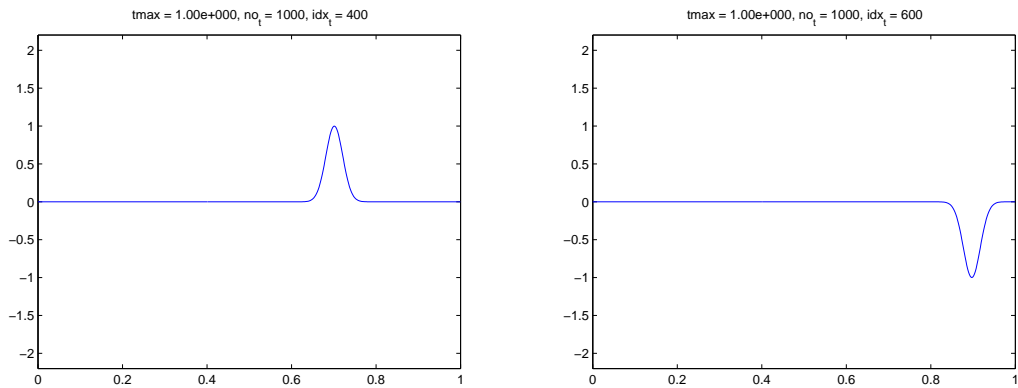


Figura 5.11: Propagarea unui impuls Gauss (stânga) și rezultatul reflexiei după atingerea unei frontiere pe care s-a impus condiție Dirichlet nulă (dreapta).

lui Courant

$$|v|\Delta t \leq \Delta x. \quad (5.71)$$

Exercițiul 5.22:

- Reveniți la cazul $\Delta x = v\Delta t$ și schimbați condiția la capătul din stânga cu $u(0, t) = h_1(t) = \sin(10\pi t)$.
- Observați apariția undelor staționare, de data aceasta obținute prin rezolvarea ecuației unei scalare (fig. 5.12).

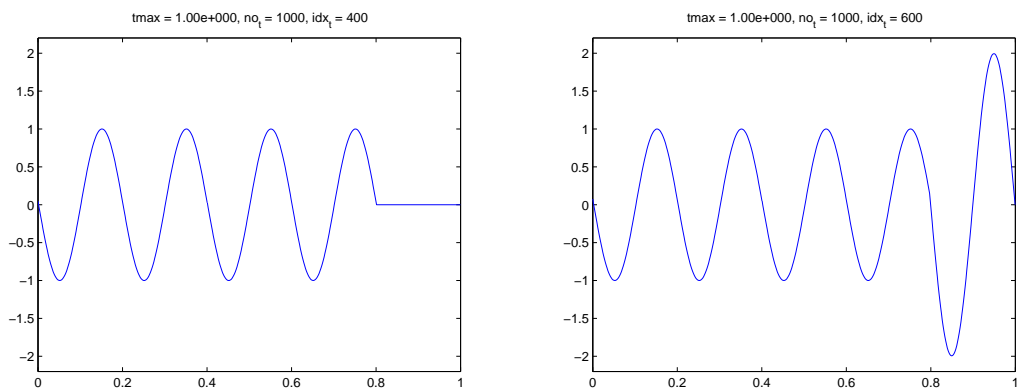


Figura 5.12: Propagarea unui unde sinusoidale (stânga) și rezultatul reflexiei după atingerea unei frontiere pe care s-a impus condiție Dirichlet nulă (dreapta).

În problemele în care apare propagare, este util uneori ca frontiera domeniului spațial să fie modelată ca o frontieră absorbantă, "invizibilă" din punct de vedere al propagării mărimilor în domeniul spațial modelat. O condiție de frontieră absorbantă pentru capătul

din dreapta al problemei studiate este

$$\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} = 0. \quad (5.72)$$

Exercițiul 5.23:

a) Implementați condiția de frontieră absorbantă (5.72) folosind o discretizare cu diferențe progresive de ordinul 1 pentru derivata temporală și diferențe regresive de ordinul 1 pentru derivata spațială:

$$v \frac{u_N^{(j)} - u_{N-1}^{(j)}}{\Delta x} + \frac{u_N^{(j+1)} - u_N^{(j)}}{\Delta t} = 0. \quad (5.73)$$

b) Verificați codul implementat pentru cazul în care la capătul din stânga este impusă condiția Dirichlet folosită la exercițiul 5.21

Referințe

- [1] Gene Golub și Charles van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996.
- [2] D. Ioan, I. Munteanu, B. Ionescu, M. Popescu, R. Popa, M. Lăzărescu și G. Ciuprina. *Metode numerice în ingineria electrică*. MatrixROM, București, 1998.
- [3] Cleve Moler. *Numerical Computing with MATLAB*. SIAM, 2004. <http://www.mathworks.com/moler/>.
- [4] Irina Munteanu, Gabriela Ciuprina și F.M.G. Tomescu. *Modelarea numerică a câmpului electromagnetic prin programe Scilab*. Editura Printech, 2000. http://www.lmn.pub.ro/gabriela/studenti/an4/carte_MNCE.pdf.
- [5] T.H. Cormen C.E. Leiserson R.R. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill, 1990. <http://www.cs.dartmouth.edu/thc/CLRS3e/>.
- [6] F.M.G. Tomescu. *Fundamentals of Electrical Engineering, Electric Circuits*. MatrixRom, 2011. București.
- [7] Lloyd Trefethen și David Bau. *Numerical Linear Algebra*. SIAM - Society for Industrial and Applied Philadelphia, 1997.

Lucrarea de față este o introducere interactivă în studiul și aplicarea metodelor numerice de rezolvare a problemelor științifice în inginerie, în special în ingineria electrică. Ea este gândită pentru a fi folosită atât independent, cât și drept ghid practic însoțitor al lucrării **ALGORITMI NUMERICI PENTRU CALCULE ȘTIINȚIFICE ÎN INGINERIA ELECTRICĂ**, elaborată de același autor. Ariditatea aparentă a subiectului este spulberată de alegerea inspirată a unei mulțimi de exemple clarificatoare și exerciții antrenante. Astfel, lucrarea se constituie într-o construcție completă și riguroasă, accesibilă și antrenantă, cu ilustrații pertinente, acompaniate de o prezentare grafică excelentă. În această abordare nouă, deosebită, de o reală valoare pedagogică, a unei materii de mare utilitate în activitatea inginerului modern, lucrarea de față va fi bine-venită pentru cei interesați în aplicarea metodelor de calcul numeric la rezolvarea problemelor științifice și ingineresti.

București, 22 septembrie 2013

Prof. dr. ing. F.M.G. TOMESCU,
Facultatea de Inginerie Electrică,
Universitatea Politehnica – București



9786062500092