Gabriela
CIUPRINA

Algoritmi numerici pentru calcule științifice în ingineria electrică

MATRIX ROM
BUCUREȘTI

Algoritmi numerici

pentru calcule științifice în ingineria electrică

Gabriela Ciuprina

Partea I

MATRIX ROM
București 2013

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

Editura MATRIX ROM este acreditată de
CONSILIUL NAŢIONAL AL CERCETĂRII ŞTIINŢIFICE DIN ÎNVĂŢĂMÂNTUL SUPERIOR

Referenţi ştiinţifici: **Prof. univ. dr. ing. F.M.G. Tomescu**

Prof. univ. dr. ing. Daniel Ioan
Universitatea « Politehnica » **Bucureşti**

Descrierea CIP a Bibliotecii Naţionale a României
CIUPRINA, GABRIELA
Algoritmi numerici pentru calcule ştiinţifice în ingineria electrică /
Gabriela Ciuprina. - Bucureşti : Matrix Rom, 2013
Bibliogr.
ISBN 978-606-25-0008-5

519.6:621.3(075.8)

ISBN 978 – 606 – 25 – 0008 – 5

Algoritmi numerici

pentru calcule ştiinţifice în ingineria electrică

Gabriela Ciuprina

Partea I

Referenţi ştiinţifici:
Prof.dr.ing.F.M.G. Tomescu
Prof.dr.ing. Daniel Ioan

2013

Introducere

Cartea de faţă reprezintă o extindere a primei părţi a suportului pentru cursul *Metode numerice în ingineria electrică*, predat studenţilor din anul II de licenţă ai Facultăţii de Inginerie Electrică din Universitatea Politehnica Bucureşti. Lucrarea se adresează tuturor celor care nu au experienţă în metode numerice şi care doresc să se familiarizeze cu conceptele de bază şi cu modul de gândire algoritmic, structurat.

Această carte este structurată în 5 capitole care tratează următoarele aspecte: descrierea şi evaluarea algoritmilor (cap.1), rezolvarea numerică a sistemelor de ecuaţii algebrice liniare (cap.2) cu aplicaţie la conceperea unor algoritmi numerici pentru analiza circuitelor rezistive liniare (cap.3), metode de interpolare a funcţiilor (cap.4) şi metode de derivare numerică (cap.5) aplicate pentru rezolvarea unor ecuaţii diferenţiale.

Partea a II-a, ce va face obiectul unei alte cărţi, va trata problema integrării numerice, a rezolvării numerice a sistemelor de ecuaţii algebrice neliniare, cu aplicaţie la algoritmii numerici pentru analiza circuitelor rezistive neliniare şi rezolvarea numerică a sistemelor de ecuaţii cu derivate ordinare, cu aplicaţii la algoritmii numerici pentru analiza circuitelor în regim tranzitoriu.

Autoarea mulţumeşte referenţilor ştiinţifici Prof.dr.ing.F.M.G. Tomescu şi Prof.dr.ing. Daniel Ioan pentru comentariile constructive şi sugestiile de îmbunătăţire, care au fost incluse în această carte.

Orice comentarii şi sugestii în vederea îmbunătăţirii acestui suport de curs sunt binevenite la adresa gabriela.ciuprina@upb.ro.

Studenţilor mei, cu speranţa că studiul acestei discipline le va face plăcere.

Copiilor mei, Andreea şi Ştefan, cu toată dragostea.

Cuprins

Capitolul 1

Descrierea şi evaluarea algoritmilor

Un algoritm este o metodă de rezolvare a unei probleme corect formulate, constând într-un număr finit de etape simple, elementare, susceptibile de a fi implementate pe un calculator.

În consecinţă, pentru a putea descrie un algoritm trebuie cunoscute etapele elementare ce pot fi implementate pe un calculator. Implementarea se realizează cu ajutorul unui limbaj de programare (C, FORTRAN sau Matlab, Scilab, etc). Deşi limbajele de programare au o sintaxă proprie ce trebuie respectată cu rigurozitate, acţiunile descrise sunt însă aceleaşi, în oricare dintre acestea. De aceea, este mult mai util şi mai eficient ca atunci când se concepe un algoritm, acesta să fie descris într-un mod cât mai natural, neîncorsetat de rigorile vreunui limbaj de programare, eventual cu notaţii matematice compacte, astfel încât să poată fi uşor de citit şi înţeles. Ca urmare, în ceea ce urmează, vom descrie algoritmii într-un aşa numit *pseudolimbaj* sau *pseudocod*. Nu există niciun standard pentru sintaxa pseudocodului. Un pseudocod este un fals limbaj, nu poate fi compilat. El însă reprezintă punctul de plecare pentru scrierea programului. Există şi aşa numitele scheme logice care pot fi văzute ca reprezentări grafice ale pseudocodului. În această carte se va promova însă folosirea pseudocodului şi nu a schemelor logice, pentru că trecerea de la pseudocod la limbajul de programare este mult mai naturală. În prima parte a acestui capitol este descris pseudocodul folosit pe parcursul acestei cărţi.

Deoarece în general o problemă admite mai multe metode de rezolvare, înseamnă că pentru rezolvarea ei cu ajutorul calculatorului pot fi concepuţi algoritmi diferiţi. De aceea, în partea a doua a acestui capitol sunt prezentate criteriile după care pot fi evaluaţi algoritmii, criterii ce sunt independente de calculator, sistemul de operare şi limbajul de programare folosit.

1.1 Pseudocodul

1.1.1 Descrierea pseudocodului

Pseudocodul este o metodă de descriere convenţională, simplă, a algoritmilor. El nu are o sintaxă strictă, şi poate chiar folosi cuvinte cheie din limba maternă a celui care îl concepe. Este important însă ca el să fie clar şi neambiguu. *Cuvintele cheie* sunt acele cuvinte care au corespondenţe în fiecare limbaj de programare. Pe parcursul acestei cărţi cuvintele cheie sunt scrise cu un alt font şi sunt subliniate.

Pseudocolul este alcătuit din linii care fie descriu acţiuni, fie descriu date. Liniile ce descriu *acţiuni* se vor traduce în limbajul de programare cu ajutorul unor *instrucţiuni*. Liniile ce descriu *date* se vor traduce în limbajul de programare cu ajutorul unor *declaraţii*.

Există limbaje de programare (de tip Matlab, Scilab, Octave) care nu necesită declararea datelor şi limbaje de programare (C, FORTRAN) la care declaraţiile sunt obligatorii. Folosirea unui limbaj din prima categorie facilitează implementarea prototipurilor de programe. Scrierea declaraţiilor în pseudocod are avantajul că, pe de o parte, permite analiza tipurilor unor expresii ce pot apărea în pseudocod şi, pe de altă parte, permite estimarea mai uşoară a necesarului de memorie, mărime utilă în evaluarea algoritmilor.

Un concept deosebit de important în descrierea algoritmilor îl reprezintă *variabila*.

Variabila este o zonă din memorie cu trei caracteristici: nume, valoare şi tip.

Numele unei variabile este un şir de caractere alfa-numerice care permit identificarea zonei (adresei) de memorie în care se află variabila. Pseudocodul utilizat în această carte permite ca numele să conţină atât litere cât şi cifre, dar cu următoarele două restricţii, menite a evita confuziile: să nu existe spaţii goale în interiorul numelui şi numele să înceapă obligatoriu cu o literă.

Valoarea unei variabile reprezintă conţinutul zonei de memorie. Acest conţinut este un număr, fără să aibă asociată o unitate de măsură. În pseudolimbaj nu vom face deosebire între nume şi valoare.

Tipul este un atribut foarte important al variabilei care permite interpretarea conţinutului zonei de memorie.

Un algoritm are nevoie de mai multe variable, care vor ocupa în calculator o anumită *zonă de memorie*.

Declaraţiile sunt linii de pseudocod ce descriu datele. Sintaxa declaraţiilor diferă în funcţie de tipul variabilelor. Pseudocodul utilizat admite două tipuri principale de variabile: fundamentale (simple) şi agregate.

Tipurile fundamentale de date sunt: logic, întreg, real sau caracter. Cuvintele cheie folosite pentru declararea acestor tipuri în pseudocod sunt, respectiv, `logic`, `întreg`,

`real`, `caracter`.

Tipurile agregate pot fi tablouri sau înregistrări, pentru declararea cărora se vor folosi cuvintele cheie `tablou` şi, respectiv, `înregistrare`.

Declararea unei variabile se face specificând cuvântul cheie asociat tipului, urmat de numele variabilei respective sau de lista numelor variabilelor de tipul respectiv.

Iată exemple de declaraţii ale unor variabile simple:

```
logic T, F          ; adevărat (true - în engl.), fals
întreg N            ; numărul de noduri
real Pc, Pg         ; putere consumată, putere generată
caracter c, C
```

În pseudolimbaj se face deosebirea între literele mari şi cele mici. De asemenea, este permis să se declare mai multe variabile de acelaşi tip în aceeaşi declaraţie. De asemenea, tot ce este după semnul ";" va fi considerat comentariu. Este bine ca variabilele alese să aibă nume cât mai sugestive.

Deşi în anumite limbaje de programare nu este permis, în pseudocod nu este grav dacă sunt omise declaraţiile unor variabile. Este totuşi recomandat să se declare toate variabilele folosite pe de o parte pentru analiza corectă a operaţiilor de evaluare a unor expresii, şi, pe de altă parte pentru analiza corectă a complexităţii algoritmului (a se vedea şi paragraful 1.2).

Tabloul este o mulţime de elemente de acelaşi tip, cu nume comun. Numărul de elemente al tabloului este cunoscut anterior, chiar din declaraţie.

Declararea unui tablou se face astfel:

```
tablou tip nume[dimensiune]
```

unde "tip" este un cuvânt cheie ce desemnează unul din tipurile fundamentale, "nume" este numele variabilei de tip tablou, iar "dimensiune" reprezintă numărul de elemente al tabloului.

În declaraţia următoare, vă puteţi imagina că, în calculator se rezervă spaţiu de memorie pentru un număr de 10 variabile de tip real, ce poartă numele V:

```
tablou real V[10]
```

În algoritmul propriu-zis pot fi folosite cel mult zece variabile, componente ale tabloului V.

O variantă mai elegantă, care conduce şi la economie de memorie, este de a declara exact atâtea variabile cât este nevoie. De aceea, se va admite în pseudocod o declaraţie de tipul

`întreg` N
...
`tablou real` $V[N]$

Detaliile despre cum se implementează acest tip de declaraţie în limbajul C sunt descrise în paragraful 1.1.2.

Adresarea elementelor unui tablou se poate face cu paranteze sau cu indici. De exemplu, adresarea elementelor din tabloul declarat în exemplu anterior se poate face astfel $V(1), V(5)$ sau V_1, V_5, etc.

Unele metode de rezolvare sunt descrise în mod natural cu ajutorul unor tablouri care au indici ce pornesc din 0. De aceea, se va accepta ca element posibil şi $B(0)$ sau B_0. La fiecare algoritm se va preciza dacă indicii sunt de la 1 la N sau de la 0 la $N-1$.

Înregistrarea este o mulţime de elemente de tipuri diferite. Fiecare element al unei înregistrări poartă numele de *câmp* şi este identificat printr-un nume.

Declararea unei înregistrări se face astfel:

```
înregistrare nume
    tip1 nume1
    tip2 nume2
```

unde "nume" este numele înregistrării, "tip1", "tip2", etc. sunt cuvinte cheie asociate tipurilor fundamentale, iar "nume1", "nume2", etc. sunt nume de câmpuri.

Iată un exemplu de înregistrare

```
înregistrare punct
    logic polar
    real coord1
    real coord2
```

unde, dacă "polar" este fals atunci "coord1" reprezintă abscisa şi "coord2" reprezintă ordonata unui punct în plan, iar dacă "polar" are valoarea adevărat atunci "coord1" reprezintă raza, şi "coord2" reprezintă unghiul în sistemul de coordonate polar.

Adresarea elementelor unei înregistrări se face invocând numele înregistrării, urmat de separatorul ".", urmat de numele câmpului. Adresarea elementelor înregistrării declarate în exemplul anterior se face cu

```
punct.polar
punct.coord1
punct.coord2
```

Instrucţiunile sunt linii de pseudocod ce reprezintă acţiunile ce se execută asupra datelor. Există două categorii de instrucţiuni: simple şi structurate. *Instrucţiunile simple* pot fi la rândul lor: de intrare, de ieşire şi de atribuire. *Instrucţiunile structurate* sunt: secvenţa, decizia fără sau cu alternativă, ciclul cu test iniţial/final sau cu contor, rutinele (proceduri sau funcţii). Descrierea fiecărui tip de instrucţiune este prezentată în cele ce urmează.

Cele mai simple instrucţiuni sunt cele de transfer de la dispozitivul de intrare la unitatea centrală de prelucrare (CPU) şi cele de la CPU la dispozitivul de ieşire.

Instrucţiunea de intrare are sintaxa

`citeşte` nume_var

iar *instrucţiunea de ieşire* are sintaxa

`scrie` nume_var.

Iată un exemplu de utilizare a instrucţiunilor de intrare şi ieşire.

```
citeşte N
citeşte q, p        ; în pseudocod se admite citirea unei liste de variabile
scrie N
scrie q, p          ; în pseudocod se admite scrierea unei liste de variabile
```

Instrucţiunea de atribuire are sintaxa

nume_variabilă = expresie

în care "nume_variabilă" este numele unei variabile, iar "expresie" este un şir de operatori şi operanzi care respectă anumite reguli sintactice.

Este important de reamintit faptul că nu orice tip de operand poate fi folosit cu orice tip de operator şi rezultatul evaluării expresiei poate fi de tip logic sau de tip aritmetic (real sau întreg) în funcţie de tipul de operanzi şi operatori folosiţi.

O *expresie logică* (al cărei rezultat se atribuie unei variabile de tip logic) poate avea fie *operatori logici* `şi`, `sau`, `nu`, caz în care operanzii trebuie să fie logici, fie *operatori de relaţie* $<$, $>$, $\leq$, $\geq$, $=$, $\neq$, caz în care operanzii trebuie să fie aritmetici (reali sau întregi), sau de tip caracter (caz în care se compară poziţia lor în tabelul ASCII[1]).

Să considerăm următorul exemplu:

[1]ASCII - *American Standard Code for Information Interchange* - codificare de tip text a 128 de caractere, din care 26 reprezintă litere ale alfabetului latin.

`logic` a	; rezultatul evaluării expresiei logice
`logic` b, c	; operanzi logici
`real` x, y	; operanzi aritmetici
$a = (b$ `sau` (`nu` c))	; expresie logică cu operatori logici
$a = (x \leq y)$	; expresie logică cu operatori de relaţie
$a = (x = y)$	; expresie logică cu operatori de relaţie

Ultima linie a acestui exemplu conţine de două ori semnul "=". Primul reprezintă simbolul de atribuire, iar al doilea este un operator de relaţie. În pseudocod nu se va face deosebirea între modul de scriere al lor, dar în limbajele de programare nu se întâmplă la fel. De exemplu, în C sau în Matlab, această linie se scrie $a = (x == y)$, iar în Pascal $a := (x = y)$. În unele cărţi se evită această confuzie folosind simbolul $\leftarrow$ pentru a reprezenta operaţia de atribuire, caz în care linia de mai sus s-ar fi scris $a \leftarrow (x = y)$.

O *expresie aritmetică* (al cărei rezultat se atribuie unei variabile de tip real sau întreg) are operanzi aritmetici, iar operatorii pot fi: $+$, $-$, $\cdot$, $/$, dar şi $\sqrt{\ }$, funcţii standard (de exemplu funcţiile trigonometrice).

Iată un exemplu de pseudocod în care apar expresii aritmetice.

`întreg` i
`real` d, x, y
$i = i + 1$
$d = xy + \sin(y)$
$d = \sqrt{d}$
$d = \frac{\frac{d^2}{2} + \frac{d}{3}}{\frac{1}{d}}$

În pseudocod se admite scrierea simbolului radical, scrierea unui exponent şi chiar a fracţiilor etajate. Se admite chiar să nu se pună simbolul explicit pentru înmulţire dacă acest lucru nu produce confuzii. Astfel, sintaxa nu este rigidă, ca cea a unui limbaj de programare. Totul este ca expresia să fie clară, astfel încât implementarea să fie imediată.

Secvenţa este un şir de instrucţiuni simple, scrise una sub alta, dar aliniate la stânga şi indentate. Indentarea este deosebit de importantă pentru a obţine un cod clar şi uşor de înţeles[2]. În exemplele următoare se foloseşte scrierea indentată.

Decizia fără alternativă (simplă) are sintaxa:

`dacă` condiţie [`atunci`]
    secvenţă

[2]Mediile de programare oferă facilităţi pentru indentare. De exemplu, în Linux există comanda `indent`, iar în editorul din Matlab există comanda `Smart indent`.

Cuvântul cheie atunci poate lipsi. Pentru claritate, "secvenţa" este scrisă indentat (deplasat la dreapta) faţă de cuvântul cheie dacă. Ea se va executa doar dacă rezultatul evaluării expresiei "condiţie" este adevărat.

Iată un exemplu de folosire a scrierii indentate.

```
real x
x = 2
dacă x < 0
        x = x + 3
        x = x/2
        x = x^2
x = 2x
```

În acest caz, la sfârşitul execuţiei acestui cod, x va avea valoarea 4. O modificare "minoră" a indentării, ca de exemplu

```
real x
x = 2
dacă x < 0
        x = x + 3
        x = x/2
x = x^2
x = 2x
```

schimbă rezultatul final, care va fi în acest caz 8. Pentru mărirea claritaţii pseudocodului, uneori se va marca suplimentar sfârşitul instrucţiunii de decizie cu •, ca de exemplu

```
real x
x = 2
dacă x < 0
        x = x + 3
        x = x/2
        x = x^2
•
x = 2x
```

Este recomandat ca fiecare instrucţiune să fie scrisă separat pe o linie, utilizând scrierea indentată. În fond, codul de mai sus se putea scrie pe o singură linie astfel

```
real x  x = 2  dacă x < 0 atunci x = x + 3  x = x/2  x = x^2  • x = 2x
```

Cu siguranţa că s-ar fi făcut economie de spaţiu, dar claritatea s-ar fi pierdut. Iar acesta nu a fost decât un exemplu banal. Algoritmii problemelor reale sunt cu mult mai complicaţi.

Decizia cu alternativă are sintaxa

```
dacă condiţie [atunci]
    secvenţa1
altfel
    secvenţa2
```

În acest caz, "condiţie" este o expresie care este evaluată, iar dacă rezultatul este adevărat, atunci se execută "secvenţa1", în caz contrar executându-se "secvenţa2".

Iată un exemplu de pseudocod în care se citeşte un număr real şi se afişează modulul lui. În acest pseudocod se foloseşte o decizie cu alternativă.

```
real x                ; un număr real - dată de intrare
real modul            ; dată de ieşire - modulul numărului real
citeşte x
dacă x ≥ 0
    modul = x
altfel
    modul = −x
•
scrie modul
```

Ciclul cu test iniţial are sintaxa

```
cât timp condiţie [repetă]
    secvenţă
```

În acest caz se repetă corpul ciclului, adică "secvenţă" cât timp rezultatul evaluării expresiei logice "condiţie" este adevărată. S-ar putea ca "secvenţă" să nu fie executată niciodată. Acest lucru se întâmplă dacă în momentul execuţiei ciclului rezultatul evaluării expresiei "condiţie" este fals.

Iată un exemplu de folosire a unui ciclu cu test iniţial.

```
întreg N
întreg i
tablou real x[N]
N = 3
x_1 = 1
```

$x_2 = -1$
$x_3 = 2$
$i = 1$
cât timp $(i \geq 1)$ **şi** $(i \leq N)$
    dacă $x_i \geq 0$
        scrie "Elementul ", i ," este pozitiv"
    •
    i = i + 1
•

Ciclul cu test final are sintaxa

repetă
    secvenţă
cât timp condiţie

În acest caz se repetă corpul ciclului, adică "secvenţă" cât timp "condiţie" este adevărată. Spre deosebire de ciclul cu test iniţial, ciclul cu test final este executat cel puţin o dată.

Este posibil ca sintaxa ciclului cu test final să o scriem şi astfel

repetă
    secvenţă
până când condiţie

În acest caz corpul ciclului se repetă cât timp "condiţie" este falsă. Când "condiţie" devine adevărată, programul iese din ciclu. Logica este deci inversă primei variante.

Oricare din aceste două variante pot fi folosite în pseudocod. Ele doar trebuie traduse cu grijă în limbajul de programare ales[3].

Iată un exemplu de folosire al unui ciclu cu test final:

întreg N
tablou **real** $x[N]$
real s, ε
întreg k
...
$k = 0$
$s = 0$

[3]În C ciclul cu test final foloseşte cuvântul cheie `while` (care în limba engleză înseamnă cât timp), iar în Pascal ciclul cu test final foloseşte cuvântul cheie `until` (până când).

repetă
    $k = k + 1$
    $s = s + x_k$
cât timp $|x_k| \geq \varepsilon$

Un astfel de pseudocod calculează suma unei serii, adunând termenii până când ultimul termen adunat este mai mic în modul decât o anumită valoare, suficient de mică.

Ciclul cu contor are sintaxa

pentru contor = val_in, val_fin[, pas] [repetă]
    secvenţă

Într-un astfel de ciclu, variabila "contor" ia, pe rând, valorile întregi "val_in", "val_in" + "pas", ..., "val_fin". Pentru fiecare din aceste valori se execută "secvenţa". Valoarea "pas" poate lipsi, caz în care este considerată, în mod implicit, 1. Această valoare poate fi pozitivă sau negativă.

Ciclul cu contor se foloseşte atunci când se cunoaşte de la început numărul de repetiţii ale secvenţei, pe când ciclurile cu test se folosesc atunci când acest număr nu este cunoscut.

Un exemplu simplu de folosire a unui ciclu cu contor îl reprezintă citirea elementelor unei matrice.

întreg N
citeşte N
tablou real $a[N, N]$
întreg i, j
pentru $i = 1, N$
    pentru $j = 1, N$
        citeşte $a_{i,j}$

Rutinele sunt folosite atunci când aceeaşi secvenţă apare de mai multe ori în acelaşi algoritm. În acest fel, secvenţa respectivă va fi memorată într-un singur loc în memoria calculatorului şi va fi apelată ori de câte ori este nevoie. În general se preferă utilizarea rutinelor chiar dacă ele sunt apelate o singură dată. În acest fel, algoritmul se modularizează şi, chiar dacă el face lucruri complicate, nu este lung ci face apel la rutine. Modularizarea are avantajul că permite şi împărţirea problemei de rezolvat în sub-probleme, astfel încât conceperea unui algoritm pentru rezolvarea unei probleme complicate poate fi făcut în echipă. Se recomandă ca programul principal să fie scris cu cât mai puţine linii de cod, el făcând apel la rutine. O dată ce o rutină este apelată, programul îi transferă ei execuţia. După ce rutina a fost executată, întoarcerea se face imediat după punctul de apel.

După numărul de parametri de ieşire, rutinele se împart în proceduri şi funcţii.

Procedura este o rutină care poate avea oricâţi parametri de ieşire.

Definiţia unei proceduri se face astfel

```
procedură nume_proc (lista argumentelor formale de intrare şi ieşire)
; comentarii ce descriu ce face procedura şi parametrii acesteia
...
; declaraţii pentru argumente
...
; instrucţiuni
...
retur                    ; cuvânt cheie ce comandă întoarcerea în punctul de apel
```

Apelul unei proceduri se face simplu, prin invocarea numelui ei, urmat de lista argumentelor actuale:

```
nume_proc (lista argumentelor actuale de intrare şi ieşire)
```

Funcţia este o rutină care are exact un singur parametru de ieşire.

Definiţia unei funcţii se face astfel

```
funcţie nume_fct (lista argumentelor formale de intrare)
; comentarii ce descriu ce face funcţia şi parametrii acesteia
...
; declaraţii pentru argumente
...
; instrucţiuni
...
întoarce valoare      ; cuvântul cheie comandă întoarcerea în punctul de apel
```

Apelul unei funcţii se face în instrucţiuni de atribuire, funcţia putând fi înţeleasă ca un operator aplicat unor operanzi ce apar ca o listă de argumente actuale de intrare ale funcţiei:

```
val = nume_fct (lista argumentelor actuale de intrare)
```

Funcţia poate fi privită ca un caz particular de procedură. Este foarte important ca în cazul rutinelor, să nu se confunde argumentele formale, folosite la definirea rutinei,

cu argumentele actuale, cele pentru care se face calculul propriu-zis. Există o restricţie valabilă în toate limbajele de programare şi anume ca argumentele formale să concorde cu cele actuale ca număr, ordine şi tip. Nerespectarea acestui lucru generează neclarităti în pseudocod şi greşeli în implementarea propriu-zisă.

Ca exemplu, vom modulariza, cu ajutorul rutinelor, următorul algoritm ce calculează produsul scalar a doi vectori.

```
; program principal
întreg N                          ; dimensiunea vectorilor
citeşte N
tablou real a[N], b[N]            ; date de intrare: doi vectori de numere reale, indici de la 1
real p                            ; rezultatul: produsul scalar al vectorilor a şi b
întreg i                          ; alte variabile utile
pentru i = 1, N
     citeşte a_i
•
pentru i = 1, N
     citeşte b_i
•
p = 0
pentru i = 1, N
     p = p + a_i b_i
•
scrie p
```

Vom rescrie acest pseudocod folosind o procedură pentru citirea vectorilor şi o funcţie pentru calculul produsului scalar. Codul rezultat este următorul:

```
; program principal
întreg N                          ; dimensiunea vectorilor
citeşte N
tablou real a[N], b[N]            ; date de intrare: doi vectori de numere reale, indici de la 1
real p                            ; rezultatul: produsul scalar al vectorilor a şi b
citeşte_vector(N,a)
citeşte_vector(N,b)
p = produs_scalar(N,a,b)
scrie p

procedură citeşte_vector(N,x)
```

```
întreg N                          ; dimensiunea vectorului - argument formal de intrare
tablou real x[N]                  ; vectorul propriu-zis - argument formal de ieşire
întreg i
pentru i = 1, N
    citeşte x_i
•
retur

funcţie produs_scalar(N,v,w)
întreg N
tablou real v[N], w[N]
întreg i
real r
r = 0
pentru i = 1, N
    r = r + v_i w_i
•
întoarce r
```

În cazul în care algoritmii sunt complicaţi, este permis ca primele versiuni ale pseudocodului să fie scrise într-un pseudocod simplificat, ce permite o scriere de tip matriceal. Într-un astfel de pseudocod un tablou unidimensional va fi notat cu literă mică scrisă cu font aldin, iar un tablou bidimensional va fi notat cu literă mare scrisă cu font aldin. Tabloul unidimensional poate fi primit ca un vector coloană, iar cel bidimensional ca o matrice. Astfel, funcţia ce calculează produsul scalar de mai sus ar putea fi scrisă simplificat astfel:

```
funcţie produs_scalar(v, w)
întoarce v^T · w
```

În acest caz, nici nu mai este nevoie să se complice pseudocodul programului principal cu instrucţiunea

p = produs_scalar($\mathbf{a}, \mathbf{b}$)

ci se poate scrie simplu

$p = \mathbf{a}^T \cdot \mathbf{b}$

Pe parcursul acestei cărţi se vor folosi ambele variante de pseudocod, de fiecare dată fiind folosită varianta ce permite explicarea mai clară a noţiunilor.

În momentul în care un algoritm este definitivat şi se doreşte implementarea lui într-un limbaj de programare, atunci stilul de lucru va depinde şi de limbajul ales. Următoarele două paragrafe oferă sugestii de lucru utilizatorilor de C şi, respectiv, Matlab.

1.1.2 Sugestii pentru utilizatorii de C

Metodele care vor fi studiate în această carte conduc în exclusivitate la algoritmi numerici, care efectuează de cele mai multe ori calcule cu vectori şi matrice. În acest paragraf se vor face câteva consideraţii asupra declarării şi alocării vectorilor şi matricelor în C.

În C există o strânsă corespondenţă între adrese (pointeri) şi tablouri. De exemplu, în cazul unui tablou unidimensional declarat ca

```
float a[4]
```

valoarea reprezentată de `a[j]` este acelaşi lucru cu `*(a+j)` adică "conţinutul adresei obţinute incrementând pointer-ul `a` cu `j`". O consecinţă a acestei definiţii este aceea că dacă `a` este adresa unei locaţii valide, atunci `a[0]` este întotdeauna definit. Tablourile unidimensionale sunt numerotate în C, în mod natural, începând de la 0. Şirul definit mai sus are referinţele valide `a[0], a[1], a[2]` şi `a[3]`, dar nu şi `a[4]`.

Mulţi algoritmi sunt însă descrişi în mod natural cu indici care încep de la 1. Cu siguranţă că aceşti algoritmi pot fi modificaţi, dar aceasta presupune o aritmetică suplimentară în operarea cu indici, lucru care nu este prea plăcut. Se poate folosi însă puterea limbajului C pentru ca această problemă să dispară. Ideea este simplă:

```
float a[4], *aa;
aa = a - 1;
```

Pointer-ul `aa` indică acum o locaţie înaintea lui `a`. În consecinţă, elementele `aa[1], aa[2], aa[3]` şi `aa[4]` există şi vectorul `aa` are indici ce pornesc de la 1.

Uneori este convenabil ca vectorii să aibă indici care pornesc de la 0, iar alteori este convenabil ca indicii să înceapă cu 1. De exemplu, coeficienţii unui polinom $a_0 + a_1 x + \ldots a_n x^n$ sunt descrişi în mod natural de un vector cu indici ce încep cu 0, pe când termenul liber al unui sistem de ecuaţii $b_i, i = 1, \ldots, n$ este descris în mod natural cu un vector cu indici ce încep de la 1.

Pentru a evita rescrierea algoritmilor ce sunt deduşi în mod natural cu indici ce pornesc de la 1, se poate folosi o funcţie cu următoarea definiţie.

```
typedef float *VECTOR;
VECTOR vector (int nl, int nh)
{
  VECTOR v;
  v = (float *) malloc ((unsigned) (nh - nl + 1) * sizeof (float));
  if (!v)
    nrerror ("Eroare de alocare in vector()");
  return v - nl;
}
```

Această funcţie alocă un vector de variable de tip `float`, care vor fi accesate cu indici cuprinşi între `nl` şi `nh`. O utilizare tipică a acestei funcţii este

```
VECTOR a;
a = vector(1,4);
```

Această funcţie precum şi funcţii similare ce alocă vectori de întregi se găsesc în fişierul `nrutil.c` pe care îl găsiţi[4] în anexa A. Acest fişier conţine şi rutinele corespunzătoare de dealocare. De exemplu, pentru dealocarea memoriei ocupate de vectorul definit mai sus, instrucţiunea este

```
free_vector(a,1,4);
```

Problema indicilor ce pornesc de la 0 sau de la 1 apare şi în cazul matricelor. În sintaxa C, lucrul cu tabele bidimensionale este puţin mai complicat. Fie o valoare reală `a[i][j]` unde `i` şi `j` sunt întregi. Un compilator de C va genera coduri maşină diferite pentru această referinţă, aceasta depinzând de declaraţia pentru variabila `a`. Dacă `a` a fost declarată de dimensiune fixă, de exemplu `float a[2][4]` atunci codul maşină ar putea fi descris astfel: "la adresa `a` adună de 4 ori `i`, apoi adună `j` şi întoarce valoarea astfel adresată. Observaţi că valoarea constantă 4 trebuie cunoscută pentru a efectua în mod corect calculele.

Dacă `a` a fost declarat ca `float **a`, atunci codul maşină `a[i][j]` este "la adresa `a` adună `i`, valoarea astfel adresată consider-o o nouă adresă, la care adună `j` şi întoarce valoarea astfel adresată.

Se observă că în al doilea caz nu este necesară cunoaşterea dimensiunii matricei şi nu este nevoie de nici o înmulţire. O indirectare suplimentară înlocuieşte aceste informaţii.

În concluzie, tablourile de dimensiune fixă trebuie evitate. Ele nu sunt structuri de date potrivite pentru reprezentarea vectorilor şi matricelor în calculele ştiinţifice.

[4]Acesta este stilul de lucru al cărtii *Numerical Recipies in C* [9], http://www.nr.com/.

Ca exemplificare a celor spuse mai sus, iată traducerea în C a pseudocodului algoritmului ce calculeaza produsul scalar a doi vectori, descris la sfărşitul paragrafului anterior.

```
#include "nrutil.h"
void citeste_vector (int N, VECTOR x);
float produs_scalar (int N, VECTOR v, VECTOR w);

void main (void)
{
/* program principal */
  int N; /* dimensiunea vectorilor */
  VECTOR a, b; /* vectorii de intrare */
  float p; /* produsul scalar dintre a si b */
  int i;
  printf ("\n Introduceti dimensiunea vectorilor ");
  scanf ("%d", &N);
  a = vector (1, N); /* aloca spatiu de memorie pentru vectori */
  b = vector (1, N);

  printf ("\n Introduceti componentele vectorului a ");
  citeste_vector (N, a);
  printf ("\n Introduceti componentele vectorului b ");
  citeste_vector (N, b);
  p = produs_scalar (N, a, b);
  printf ("\n Rezultatul este %f \n", p);

  free_vector (a, 1, N); /* elibereaza spatiul de memorie */
  free_vector (b, 1, N);
}

void citeste_vector (int N, VECTOR x)
{
/* citeste un vector x de dimensiune N, indicii incep de la 1 */
  int i;
  for (i = 1; i <= N; i++)
    {
      printf ("\n componenta %d = ", i);
      scanf ("%f", &x[i]);
```

```
    }
}

float produs_scalar (int N, VECTOR v, VECTOR w)
{
/* calculeaza produsul scalar a doi vectori v si w,
   ai caror indici incep de la 1 */
  int i;
  float r;
  r = 0;
  for (i = 1; i <= N; i++)
    r = r + v[i] * w[i];
  return r;
}
```

1.1.3 Sugestii pentru utilizatorii de Matlab

În limbaje de programare de tip Matlab, Scilab, Octave, implementările pot arăta chiar mai simplu decât algoritmul prezentat în sintaxa pseudocodului descrisă anterior. Acest lucru se datorează pe de o parte faptului că în aceste limbaje nu sunt necesare declaraţiile variabilelor, iar pe de altă parte în aceste limbaje se poate opera direct cu vectori şi matrice.

Vom ilustra aceste afirmaţii folosind exemplul produsului scalar. O traducere exactă în Matlab a algoritmului prezentat este următoarea:

```
% program principal, fisier main_ps.m
clear all;
N = input('Introduceti N = ');
a = zeros(N,1);  % alocare de memorie pentru vectorul a
b = zeros(N,1);  % alocare de memorie pentru vectorul b
a = citeste_vector(N,a);
b = citeste_vector(N,b);
p = produs_scalar(N,a,b);
disp(sprintf('p = %g',p));

% fisier citeste_vector.m
function a = citeste_vector(N,a)
for i = 1:N
```

```
    a(i) = input(sprintf('Introduceti componenta %d = ',i));
end

% fisier produs_scalar.m
function p = produs_scalar(N,a,b)
p = 0;
for i = 1:N
    p = p + a(i)*b(i);
end
```

Funcţia de citire se poate simplifica având în vedere că sintaxa comenzii `input` permite chiar introducerea vectorului, nu numai a unei componente:

```
% fisier citeste_vector.m - varianta a doua
function a = citeste_vector(N,a)
a = input(sprintf('Introduceti un vector coloana de dimensiune %d :',N));
if or(size(a,1)~=N,size(a,2)~=1)
    error('Eroare');
end
```

De asemenea, calculul produsului scalar se poate face mult mai simplu, ştiind că se pot folosi operaţii cu vectori şi matrice.

```
% fisier citeste_vector.m - varianta a doua
function p = produs_scalar(N,a,b)
p = a'*b;
```

În Matlab de altfel, codurile ce folosesc operaţii cu matrice sunt mai eficiente din punct de vedere al timpului de calcul decât folosirea unui cod echivalent ce detaliază toate operaţiile, aşa cum s-ar fi făcut în C. Operarea cu matrice ascunde însă operaţiile elementare şi, în consecinţă, va face mai dificilă estimarea complexităţii codului.

Exemplul de mai sus ilustrează doar câteva aspecte ale implementării în Matlab. De fapt, funcţia `produs_scalar` nici nu mai are nevoie de N, primul argument de intrare[5]. Mai mult, este lipsit de sens şi chiar ineficient să se scrie funcţii care au o singură linie de cod deoarece se pierde timp şi la transferul execuţiei de la programul principal la funcţie şi înapoi.

[5]Pentru a curăţa programul de astfel de lucruri inutile, este deosebit de folositor să se folosească comanda Matlab `mlint` care permite identificarea variabilelor inutile şi chiar a liniilor de cod cu probleme.

1.2 Complexitatea algoritmilor

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

Nu există un singur criteriu de evaluare a algoritmilor. Algoritmul ideal trebuie să fie simplu, să dea o soluţie corectă într-un timp scurt şi să ocupe o zonă mică de memorie. Nu există însă nici un algoritm care să rezolve perfect o problemă, fără nici un fel de eroare şi atunci se va urmări ca erorile pentru o anumită aplicaţie să fie rezonabile. O analiză a erorilor posibile dintr-un algoritm este prezentată în paragraful 1.3. De asemenea, criteriul referitor la timpul de calcul intră de multe ori în contradicţie cu criteriul referitor la memoria necesară algoritmului.

În consecinţă, la alegerea unui algoritm potrivit pentru o aplicaţie dată, trebuie făcut un compromis între trei criterii: timpul de calcul necesar obţinerii soluţiei, necesarul de memorie şi acurateţea soluţiei. Analiza primelor două criterii furnizează informaţii despre complexitatea algoritmului.

1.2.1 Timp de calcul

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

Complexitatea unui algoritm din punct de vedere al timpului de calcul este relaţia dintre timpul de calcul exprimat în număr de operaţii elementare şi dimensiunea problemei.

Dimensiunea problemei depinde de problema studiată. De exemplu, pentru calculul produsului scalar a doi vectori, dimensiunea problemei este dimensiunea vectorilor. Pentru rezolvarea unui sistem de ecuaţii algebrice liniare, dimensiunea problemei este dimensiunea sistemului. Uneori, este potrivit să exprimăm dimensiunea problemei în funcţie de două numere în loc de unul. De exemplu, dacă datele de intrare reprezintă graful unui circuit, dimensiunea problemei este reprezentată de perechea alcătuită din numărul de noduri şi numărul de laturi.

Timpul de calcul este de fapt suma timpilor necesari pentru executarea tuturor instrucţiunilor algoritmului. Nu toate instrucţiunile durează la fel de mult, de aceea, estimarea complexităţii nu se poate face foarte precis, dar nici nu este necesar acest lucru. Se alege o

operaţie elementară, considerată a fi cea care durează cel mai mult (de exemplu evaluarea unei anumite funcţii) şi se numără câte astfel de operaţii elementare sunt executate.

Algoritmi polinomiali

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

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

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

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

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

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

Să considerăm acum cazul înmulţirii a două matrice pătrate de dimensiune n. Rezultatul este o matrice pătrată ale cărei componente sunt $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

```
pentru i = 1, n
    pentru j = 1, n
        c_ij = 0
        pentru k = 1, n
            c_ij = c_ij + a_ik b_kj
        •
    •
•
```

În acest caz se fac $2n^3$ operaţii elementare. Un algoritm pentru care timpul de calcul T este proporţional cu cubul dimensiunii problemei n se spune că este un *algoritm cubic, sau de ordinul 3* şi se notează $T = \mathrm{O}(n^3)$. Înmulţirea a două matrice pătrate are complexitatea $T = \mathrm{O}(2n^3) \approx \mathrm{O}(n^3)$.

Se spune că un algoritm este *polinomial de ordin k* din punct de vedere al timpului de calcul, şi se notează $T = \mathrm{O}(n^k)$ dacă şi numai dacă există o constantă $C > 0$ şi un număr n_0 astfel încât $T \leq Cn^k$ pentru orice $n \geq n_0$.

Un algoritm pentru care timpul de calcul nu depinde de dimensiunea problemei se numeşte *algoritm de ordin zero* şi pentru el se scrie $T = \mathrm{O}(1)$.

Ordonarea algoritmilor s-ar putea face pe o scară astfel: $\mathrm{O}(1) < \mathrm{O}(\log n) < \mathrm{O}(n) < \mathrm{O}(n \log n) < \mathrm{O}(n^2) < \ldots \mathrm{O}(e^n) < \mathrm{O}(n!)$. Între algoritmii de ordin zero şi algoritmii liniari se afla algoritmii *logaritmici*, al căror timp de calcul este proporţional cu logaritmul dimensiunii problemei. Între algoritmii liniari şi cei pătratici se afla algoritmii *liniaritmici* având timpul de calcul majorat de la un moment dat de o funcţie proporţională cu $n \log n$. Cei mai costisitori, şi de alfel lipsiţi de importanţă practică, sunt algoritmii nonpolinomiali. Rezolvarea unui sistem de ecuaţii algebrice liniare cu regula lui Cramer conduce de exemplu la un algoritm cu complexitate factorială. La fel ar fi şi algoritmul unui jucător de şah perfect, care la orice mutare analizează toate combinaţiile posibile de pe tabla de şah. În practică sunt utili cel mult algoritmii având o complexitate cubică din punct de vedere a timpului de calcul.

Dacă analizăm cele trei exemple anterioare, observăm că gradul de indentare este în strânsă corelare cu ordinul algoritmului. Cu cât gradul de indentare este mai mare, cu atât este de aşteptat ca ordinul algoritmului să fie mai mare. Acest lucru este adevărat dacă operaţia elementară apare în instrucţiunea indentată cel mai mult.

Vom exemplifica acum evaluarea complexităţii timpului de calcul în cazul a trei algoritmi diferiţi pentru evaluarea unui polinom de grad n, cu coeficienţi reali

$$P(x) = a_0 + a_1 x + \ldots + a_n x^n. \tag{1.1}$$

Relaţia (1.1), scrisă compact

$$P(x) = a_0 + \sum_{i=1}^{n} a_i x^i \tag{1.2}$$

se poate implementa "natural" astfel (declaraţiile au fost omise):

```
; Varianta 1
P = a0;
pentru i = 1, n
        t = ai
        pentru j = 1, i
                t = t * x
        •
        P = P + t;
•
```

Considerând ca operaţie elementară orice operaţie algebrică, rezultă că pentru fiecare valoare i se fac $i + 1$ operaţii elementare (i înmulţiri şi o adunare). Cum i ia valori de la 1 la n rezultă că numărul total de operaţii elementare este $\sum_{i=1}^{n}(i+1) \approx n^2/2$. Ordinul de complexitate al acestui algoritm va fi deci $T_1 = \mathrm{O}(n^2/2) \approx \mathrm{O}(n^2)$, aşteptat pentru că există două nivele de indentare.

O altă variantă de implementare poate calcula fiecare termen al sumei mai simplu, în funcţie de termenul adunat anterior. Ea se bazează pe scrierea, echivalentă din punct de vedere matematic

$$P(x) = a_0 + a_1 t_1 + a_2 t_2 + \ldots + a_n t_n = a_0 + \sum_{i=1}^{n} a_i t_i, \tag{1.3}$$

unde

$$t_1 = x, \quad t_i = t_{i-1} x, \quad i = 2, \ldots, n. \tag{1.4}$$

Pentru această scriere matematică, implementarea naturală este

```
; Varianta 2
P = a0;
t = 1;
pentru i = 1, n
        t = t * x
        P = P + ai * t
•
```

Deoarece există un singur nivel de indentare, este de aşteptat ca algoritmul să fie liniar. Într-adevăr, pentru fiecare i se execută 3 operaţii elementare (două înmulţiri şi o adunare), deci complexitatea algoritmului este $T_2 = \mathrm{O}(3n) \approx \mathrm{O}(n)$.

O altă idee de implementare este sugerată de efectuarea calculelor conform relaţiei

$$P(x) = a_0 + x(a_1 + x(a_2 + \ldots x(a_{n-1} + a_n x)\ldots)). \tag{1.5}$$

Implementarea corespunzătoare acestei expresii este

```
; Varianta 3
P = a_n;
pentru i = n - 1, 0, -1
        P = a_i + P * x
•
```

Această variantă este cea mai eficientă, ordinul de complexitate fiind $T_3 = \mathrm{O}(2n) \approx \mathrm{O}(n)$. Din punct de vedere matematic, cele trei relaţii (1.1), (1.3) şi (1.5) sunt echivalente, dar algoritmii corespunzători sunt diferiţi din punct de vedere al complexităţii lor.

Un algoritm se poate îmbunătăţi din punct de vedere al timpului de calcul dacă operaţia elementară este scoasă din instrucţiunea indentată cel mai mult. Un alt aspect important este acela de a nu repeta operaţiile efectuate o dată. Pentru a clarifica aceste afirmaţii, să considerăm următorul exemplu.

```
; Varianta A
întreg i, j, n
real a, b
...
tablou real c[n][n]
pentru i = 1, n
        pentru j = 1, n
                c_ij = f(i * a) + f(j * b)      ; f este o funcţie definită în altă parte
        •
•
```

Considerând ca operaţie elementară evaluarea funcţiei f, rezultă o complexitate a algoritmului $T_A = \mathrm{O}(2n^2)$.

Acest cod se poate simplifica observând că termenul $f(i * a)$ este evaluat de mai multe ori, chiar pentru aceeaşi valoare a lui i. De aceea, se poate scoate în afara ciclului în

j evaluarea funcţiei f pentru argumentul $i * a$ rezultând următoarea variantă de implementare. Pentru aceasta este însă nevoie de o variabilă suplimentară, notată mai jos cu p.

```
; Varianta B
întreg i, j, n
real a, b, p
...
tablou real c[n][n]
pentru i = 1, n
        p = f(i * a)
        pentru j = 1, n
                c_ij = p + f(j * b)
        •
•
```

Pentru această implementare, numărul de apeluri ale funcţiei f este $T_B = \mathrm{O}(n(n+1)) = \mathrm{O}(n^2 + n) \approx \mathrm{O}(n^2)$. Pentru valori mari ale lui n această variantă de implementare este de două ori mai rapidă decât prima variantă, dar ambii algoritmi sunt pătratici.

Implementarea poate fi simplificată şi mai mult, dacă se observă că, pentru fiecare valoare a contorului i, funcţia f este evaluată din nou în b, $2b$, ..., nb. Se repetă astfel calcule inutile, consumatoare de timp. Este evident că este nevoie ca funcţia f să fie evaluată în doar $2n$ puncte: a, $2a$, ..., na, b, $2b$, ..., nb. Următoarea variantă de implementare execută un număr minim de evaluări, cu preţul memorării lor anterioare asamblării rezultatului.

```
; Varianta C
întreg i, j, n
real a, b
...
tablou real c[n][n]
tablou real p[n], q[n]
pentru i = 1, n
        p_i = f(i * a)
        q_i = f(i * b)
•
pentru i = 1, n
        pentru j = 1, n
```

$c_{ij} = p_i + q_j$

-
-

Într-adevăr, în acest caz complexitatea algoritmului este $T_C = \mathrm{O}(2n)$. Algoritmul este acum liniar. Reducerea timpului de calcul s-a facut pe baza alocării suplimentare de memorie pentru două tablouri unidimensionale.

Notaţii asimptotice

În cele de mai sus, am discutat şi exemplificat cazul algoritmilor polinomiali pentru care timpul de calcul este mărginit superior de o funcţie de forma Cn^k, cel puţin de la o anumită valoare a lui n în sus. Această definiţie se poate generaliza astfel[10].

Un algoritm are *ordinul de complexitate* $\mathrm{O}(g(n))$ din punct de vedere al timpului de calcul T (exprimat în unităţi de timp, de exemplu secunde) dacă şi numai dacă **există** o constantă pozitivă $C > 0$ şi un număr n_0 astfel încât $T \leq Cg(n)$ pentru orice $n \geq n_0$. *Notaţia* O descrie deci o margine superioară pentru timpul de execuţie al unui program, aspect ilustrat în figura 1.1.

Notaţia Ω defineşte o delimitare asimptotică inferioară. Un algoritm are *ordinul de complexitate* $\Omega(g(n))$ din punct de vedere al timpului de calcul dacă şi numai dacă **există** o constantă pozitivă $C > 0$ şi un număr n_0 astfel încât $Cg(n) \leq T$ pentru orice $n \geq n_0$. Această definiţie este ilustrată în figura 1.2.

Notaţia Θ delimitează asimptotic atât inferior cât şi superior, timpul de execuţie al unui algoritm. Se mai spune că notaţia Θ furnizează o margine asimptotică *strânsă*. Un algoritm are *ordinul de complexitate* $\Theta(g(n))$ din punct de vedere al timpului de calcul

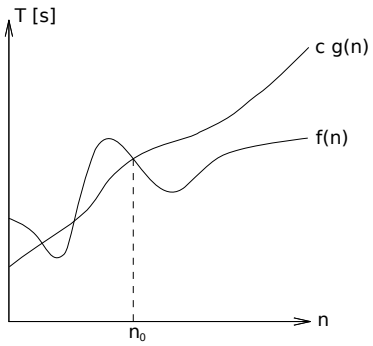


Figura 1.1: Notaţia O: $T = f(n) = \mathrm{O}(g(n))$.

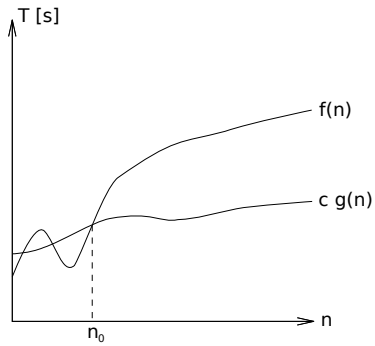


Figura 1.2: Notaţia Ω: $T = f(n) = \Omega(g(n))$.

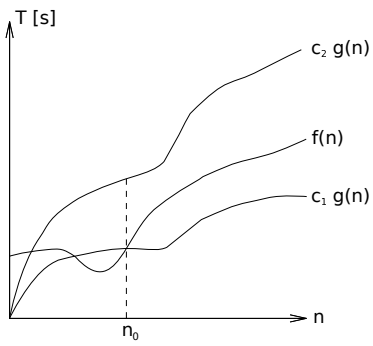


Figura 1.3: Notaţia Θ: $T = f(n) = \Theta(g(n))$.

dacă şi numai dacă **există** două constante pozive C_1 şi C_2 şi un număr n_0 astfel încât $C_1 g(n) \leq T \leq C_2 g(n)$ pentru orice $n \geq n_0$. Acestă definiţie este ilustrată în figura 1.3.

Se poate demonstra următoarea echivalenţă

$$T = \Theta(g(n)) \quad \Leftrightarrow \qquad T = O(g(n)) \quad \text{şi} \quad T = \Omega(g(n)).$$

De asemenea, se poate arăta că un algoritm pentru care timpul de calcul se exprimă printr-un polinom $T = \sum_{i=0}^{k} a_i n^i$ unde $a_k > 0$ este un algoritm de complexitate $T = \Theta(n^k)$, deci, conform echivalenţei anterioare $T = O(n^k)$ şi $T = \Omega(n^k)$.

O margine asimptotică superioară, dată de notaţia O poate să fie sau nu o margine asimptotic strânsă. Pentru a desemna o margine asimptotic superioară care nu este strânsă se foloseşte *notaţia* o.

Un algoritm are *ordinul de complexitate* $o(g(n))$ din punct de vedere al timpului de calcul dacă şi numai dacă pentru **orice** constantă pozitivă $C > 0$ există un număr n_0 astfel încât $T \leq Cg(n)$ pentru orice $n \geq n_0$. Definiţiile pentru notaţia O şi o sunt similare. Principala diferenţă este aceea că la notaţia O relaţia are loc pentru o anumită constantă C, în timp ce la notaţia o, relaţia are loc pentru toate constantele C. Intuitiv, în cazul notaţiei o, timpul de calcul $T = T(n)$ devine neglijabil faţă de $g(n)$ atunci când n tinde la infinit: $\lim_{n\to\infty} T(n)/g(n) = 0$.

Similar, pentru marginea inferioară se foloseşte *notaţia* ω. Un algoritm are *ordinul de complexitate* $\omega(g(n))$ din punct de vedere al timpului de calcul dacă şi numai dacă pentru **orice** constantă pozitivă $C > 0$ există un număr n_0 astfel încât $Cg(n) \leq T$ pentru orice $n \geq n_0$. Aceasta implică faptul că T devine oricât de mare faţă de $g(n)$ atunci când n tinde la infinit: $\lim_{n\to\infty} T(n)/g(n) = \infty$.

Această analiză rafinată a ordinului de complexitate este importantă mai ales atunci când performanţa algoritmului depinde şi de modul în care sunt furnizate datele de intrare. De exemplu, pentru un algoritm de sortare a unui şir de numere reale, analiza cazului cel mai favorabil (în care vectorul de intrare este gata sortat) va furniza informaţii despre complexitatea notată cu Ω în timp ce analiza cazului cel mai defavorabil (în care vectorul este sortat în ordine inversă) va furniza informaţii despre complexitatea notată cu O.

De exemplu, fie un şir de n valori reale, ordonate crescător $x_1 < x_2 < \ldots < x_n$ şi un alt număr real $xcrt$. Dorim să ştim în care subinterval $[x_k, x_{k+1}]$ se află valoarea $xcrt$.

Cu siguranţă ideea de căutare naturală este de a parcurge sub-intervalele ca în pseudocodul următor care întoarce, în cazul în care $xcrt$ este între x_1 şi x_n, indicele k pentru care $xcrt \in [x_k, x_{k+1}]$.

`funcţie` caută_v0$(n, x, xcrt)$
; caută plasarea valorii $xcrt$ în şirul ordonat x cu n valori

```
întreg n
tablou real x[n]     ; şirul este presupus ordonat crescător
real xcrt
...
dacă (xcrt < x_1)
      rez = 1
altfel dacă (xcrt > x_n)
      rez = n − 1
altfel
      flag = 0
      k = 1
      cât timp ((k <= n − 1) şi (flag = 0))
            dacă xcrt ≤ x_{k+1}
                  rez = k
                  flag = 1
            altfel
                  k = k + 1
            •
      •
•
întoarce rez
```

Considerând ca operaţie de referinţă compararea a două numere reale, este evident că ordinul de complexitate în cazul cel mai defavorabil este $T = \mathrm{O}(n)$ şi în cazul cel mai favorabil $T = \Omega(1)$. Căutarea se face mai eficient dacă se adoptă însă o strategie bazată pe înjumătăţirea numărului de subintervale în care are loc căutarea. Astfel, la fiecare iteraţie se elimină jumătate din numărul de sub-intervale posibile, întorcându-se indexul capătului din stânga al subintervalului. O astfel de procedură este utilă de exemplu în cazul în care se doreşte interpolarea pe porţiuni a unei funcţii descrise cu ajutorul unui tabel de valori. În cazul în care valoarea $xcrt$ nu se află în $[x_1, x_n]$ atunci rezultatul returnat este 1 daca punctul se află în stânga domeniului şi $n-1$ dacă punctul se află în dreapta domeniului.

```
funcţie caută(n, x, xcrt)
...
dacă (xcrt < x_1)
      rez = 1
altfel dacă (xcrt > x_n)
```

$rez = n - 1$
altfel
$k1 = 1$
$k2 = n$
cât timp $k2 - k1 \neq 1$
$km = [(k1 + k2)/2]$; $[\cdots]$ este partea întreagă
dacă $xcrt < x_{km}$
$k2 = km$
altfel
$k1 = km$
•
•
•
$rez = k1$
întoarce rez

În acest caz, la fiecare iteraţie se elimină jumătate din intervale. Numărul de intervale eliminate după k iteraţii este $n/2+n/4+n/8+\cdots n/2^k$. Căutarea durează cel mult până se elimină $n-1$ subintervale. Din condiţia $n/2 + n/4 + n/8 + \cdots n/2^k = n - 1$ rezultă $2^k = n$ de unde $k = \log_2(n)$. Algoritmul are în consecinţă o complexitate logaritmică.

Analiza complexităţii algoritmilor recursivi

Un algoritm recursiv este un algorim care se apelează pe el însuşi. Analiza complexităţii unui astfel de algoritm este mai dificilă decât în cazul algoritmilor nerecursivi. Ideea principală este aceea de a împărţi problema în subprobleme de acelaşi tip. Pentru analiza complexităţii algoritmilor recursivi se scrie mai întâi o relaţie de recurenţă pentru timpul de calcul. Această relaţie leagă complexitatea problemei iniţiale $T(n)$ de complexitatea unei subprobleme $T(n/b)$ unde b este numărul de subprobleme în care se descompune problema. Pentru aceasta trebuie evaluate numărul de apeluri recursive (fie acest număr a) precum şi complexitatea instrucţiunilor de trecere la problemă la subproblemă, iar

$$T(n) = aT(n/b) + C(n).$$

Există mai multe tipuri de metode pentru rezolvarea evaluării acestei recurenţe [10]. În cele ce urmează se va exemplifica *metoda iteraţiei* pentru estimarea complexităţii algoritmului de căutare binară în varianta recursivă.

funcţie caută_v2$(n, x, xcrt)$

```
...
dacă (xcrt < x_1)
        rez = 1
altfel dacă (xcrt > x_n)
        rez = n − 1
altfel
rez = binary_search(x, xcrt, 1, n);
•
întoarce rez
;
funcţie binary_search(x, xcrt, k1, k2)
; presupuneri:
; x - ordonate crescător, k1 < k2
; xcrt se afla in interiorul vectorului x
...
km = [(k1 + k2)/2]
dacă k2 = k1 + 1
        rez = km
altfel dacă x_km > xcrt
        rez = binary_search(x, xcrt, k1, km)
altfel
        rez = binary_search(x, xcrt, km, k2)
•
întoarce rez
```

Pentru acest algoritm recursiv, la fiecare iteraţie dimensiunea problemei se reduce la jumătate, iar timpul necesar pentru a decide care jumătate să fie eliminată este constant, nu depinde de dimensiunea problemei. De aceea putem scrie

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

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

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

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

1.2.2 Necesar de memorie

Complexitatea unui algoritm din punct de vedere al necesarului de memorie este dependenţa dintre necesarul de memorie exprimat în număr de locaţii elementare de memorie şi dimensiunea problemei.

De obicei, o locaţie elementară de memorie este cea corespunzătoare unui număr real.

Se spune că un algoritm este *polinomial de ordin k* din punct de vedere al necesarului de memorie M, şi se notează $M = \mathrm{O}(n^k)$ dacă şi numai dacă există o constantă C astfel încât $M \leq Cn^k$, unde n este dimensiunea problemei.

Necesarul de memorie se stabileşte cu uşurinţă inspectând declaraţiile făcute în pseudocod. Pentru cele trei variante (A, B, C) prezentate în paragraful 1.2.1 rezultă $M_A = \mathrm{O}(n^2+2) \approx \mathrm{O}(n^2)$ necesară pentru memorarea tabloului bidimensional c şi a variabilelor reale a şi b, $M_B = \mathrm{O}(n^2+3) \approx \mathrm{O}(n^2)$ care diferă de varianta A doar prin memorarea unei variabile suplimentare reale p şi $M_C = \mathrm{O}(n^2+2n+2) \approx \mathrm{O}(n^2)$ care diferă de varianta A prin memorarea a două tablouri unidimensionale de dimensiune n. Toate cele trei variante sunt, din punct de vedere al necesarului de memorie, de ordin 2. Câştigul variantei C este de aceea remarcabil, căci timpul de calcul a scăzut cu un ordin de mărime, pe seama unui necesar de memorie suplimentar, nesemnificativ la valori n mari ale dimensiunii problemei.

În concluzie, elaborarea unui algoritm înseamnă întotdeauna stabilirea unui compromis între timp de calcul şi necesar de memorie.

1.3 Erori în calculele numerice

Unul din criteriile de evaluare a unui algoritm îl reprezintă eroarea cu care se obţine rezultatul. Erorile nu pot fi înlăturate total dintr-un calcul deoarece, de exemplu, chiar numerele reale nu pot fi reprezentate în calculator cu o precizie infinită. Numărul real π are o infinitate de cifre semnificative, iar orice sistem de calcul lucrează cu un număr finit de cifre semnificative. Pe de altă parte, erorile se propagă în calcule, astfel încât, chiar dacă datele de intrare ale unui program sunt foarte precise, rezultatul poate avea doar câteva cifre semnificative corecte. De aceea este foarte important ca pentru orice algoritm să se estimeze eroarea rezultatului. În acest paragraf sunt prezentate tipurile de erori care pot afecta un algoritm şi modul în care ele pot fi evaluate. În final sunt discutate două aspecte importante legate de erori: condiţionarea unei probleme şi stabilitatea unui algoritm.

1.3.1 Tipuri de erori

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

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

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

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

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

Eroarea absolută $\mathbf{e_x}$ a unei mărimi din $\mathbb{R}^n$ este diferenţa dintre valoarea aproximativă $\bar{\mathbf{x}}$ şi valoarea exactă $\mathbf{x}$ a mărimii

$$\mathbf{e_x} = \bar{\mathbf{x}} - \mathbf{x}. \tag{1.6}$$

Este evident că o astfel de mărime nu se poate calcula deoarece valoarea exactă nu este cunoscută. De aceea, este mai utilă aflarea unei *margini a erorii absolute* a_x, adică a unei mărimi $a_x \in \mathbb{R}$ care majorează norma erorii absolute

$$\|\mathbf{e_x}\| \leq a_x. \tag{1.7}$$

Dacă presupunem că mărimea este scalară ($n = 1$), atunci din (1.6) şi (1.7) rezultă că

$$\bar{x} - a_x \leq x \leq \bar{x} + a_x. \tag{1.8}$$

Altfel spus, cunoaşterea marginii erorii absolute permite definirea intervalului în care este plasată soluţia exactă. Relaţia (1.8), echivalentă cu $x \in [\bar{x} - a_x, \bar{x} + a_x]$, se scrie pe scurt sub forma

$$"x = \bar{x} \pm a_x". \tag{1.9}$$

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

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

$$\varepsilon_{\mathbf{x}} = \frac{\mathbf{e_x}}{\|\mathbf{x}\|}. \tag{1.10}$$

Desigur, această mărime nu este definită pentru o valoare exactă nulă. Mai mult, ea nu se poate calcula deoarece nici eroarea absolută, nici mărimea exactă nu sunt cunoscute. De aceea se preferă folosirea unei *margini a erorii relative* r_x, mărime care satisface

$$\|\varepsilon_{\mathbf{x}}\| \leq r_x. \tag{1.11}$$

Cel mai adesea, marginea erorii relative se exprimă în procente, iar relaţia (1.11) se scrie sub forma

$$"\mathbf{x} = \bar{\mathbf{x}} \pm r_x\%". \tag{1.12}$$

Ca exemplu, valoarea exactă a numărului π este $x = 3.1415\ldots$, iar valoarea aproximativă cea mai uzuală este $\bar{x} = 3.14$. Rezultă o eroare absolută $e_x = -0.0015\ldots$ şi deci o margine a ei $a_x = 0.0016$. Eroarea relativă $\varepsilon_x = -0.0015\ldots/3.1415\ldots$ este majorată de $r_x = 0.0016/3 \leq 0.0006 = 0.06\%$. Putem scrie deci

$$\pi = 3.14 \pm 0.0016 \quad \text{sau} \quad \pi = 3.14 \pm 0.06\%.$$

În concluzie, relaţia "$\mathbf{x} = \bar{\mathbf{x}} \pm a_x$", unde $\mathbf{x}, \bar{\mathbf{x}} \in \mathbb{R}^n$ şi $a_x \in \mathbb{R}$ se interpretează astfel:

$$(\exists)\mathbf{e_x} \in \mathbb{R}^n, \|\mathbf{e_x}\| \leq a_x, \quad \textbf{astfel încât} \quad \bar{\mathbf{x}} = \mathbf{x} + \mathbf{e_x}, \tag{1.13}$$

iar relaţia "$\mathbf{x} = \bar{\mathbf{x}} \pm r_x\%$", unde $\mathbf{x}, \bar{\mathbf{x}} \in \mathbb{R}^n$, $r_x\% = 100 r_x$ şi $r_x \in \mathbb{R}$ se interpretează astfel:

$$(\exists)\varepsilon_{\mathbf{x}} \in \mathbb{R}^n, \|\varepsilon_{\mathbf{x}}\| \leq r_x, \quad \textbf{astfel încât} \quad \bar{\mathbf{x}} = \mathbf{x} + \|\mathbf{x}\|\varepsilon_{\mathbf{x}}. \tag{1.14}$$

În cazul unei mărimi scalare, relaţia (1.14) se scrie

$$\bar{x} = x(1 \pm \varepsilon_x), \tag{1.15}$$

semnul plus corespunzând unei valori x pozitive, iar semnul minus uneia negative.

1.3.2 Analiza erorilor de rotunjire

Erorile de rotunjire se datorează reprezentării finite a numerelor reale în calculator. Numerele reale nu pot fi reprezentate în calculator decât cu un număr finit de cifre semnificative. În consecinţă, sistemele de calcul nu pot lucra cu numere reale exacte, ci doar cu aproximări raţionale ale acestora.

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

$$\bar{x} = f \cdot 10^n. \tag{1.16}$$

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

În termenii convenţiei formulate, cifrele părţii fracţionare se numesc cifre semnificative. Astfel, numărul 3.14 are 3 cifre semnificative, iar numărul -0.007856 are patru cifre semnificative.

În calculator se pot memora un număr finit k de cifre semnificative. În consecinţă, un număr real oarecare x poate fi reprezentat doar ca

$$\bar{x} = \overbrace{0.\underbrace{***\cdots*}_{k \text{ cifre}}}^{f} \cdot 10^n, \tag{1.17}$$

unde fiecare simbol $*$ reprezintă o cifră, iar prima cifră nu este 0. Numărul exact ar fi avut o infinitate de cifre

$$x = 0.\underbrace{***\cdots*}_{k \text{ cifre}}\#\#\#\cdots \cdot 10^n, \tag{1.18}$$

cifrele marcate cu # fiind pierdute. Eroarea absolută datorată acestei reprezentări "rotunjite" este

$$e_x = \bar{x} - x = -0.\underbrace{000\cdots0}_{k \text{ cifre}}\#\#\#\cdots \cdot 10^n = -0.\#\#\#\cdots \cdot 10^{n-k}, \tag{1.19}$$

ceea ce conduce la o eroare relativă

$$\varepsilon_x = \frac{e_x}{x} = \frac{-0.\#\#\#\cdots \cdot 10^{n-k}}{0.\underbrace{***\cdots*}_{k \text{ cifre}}\#\#\#\cdots \cdot 10^n} = -\frac{0.\#\#\#\cdots}{0.***\cdots}10^{-k} \tag{1.20}$$

al cărei modul este majorat de

$$|\varepsilon_x| \leq \frac{1}{0.1}10^{-k} = 10^{-k+1}. \tag{1.21}$$

Marginea erorii relative de rotunjire a unui sistem de calcul depinde doar de numărul de cifre semnificative ce pot fi memorate. Pentru un sistem de calcul ce lucrează cu k cifre semnificative, marginea erorii relative de rotunjire este 10^{-k+1}.

Eroarea relativă de rotunjire nu depinde deci de cât de mare sau cât de mic este numărul (valoare care se reflectă în exponentul n), ci doar de numărul de cifre semnificative memorate.

Evident că dacă cifrele pierdute, notate cu # mai sus, sunt toate zero, atunci numărul poate fi reprezentat exact. Calculele în care intervin numere reale sunt afectate însă la rândul lor de procesul de rotunjire. De exemplu, adunarea a două numere reale se face adunând părţile fracţionare după ce, în prealabil, numărul mai mic a fost rescris astfel încât să aibă exponentul numărului mai mare. La rescrierea numărului mai mic se pot pierde cifre semnificative. Pentru înţelegerea acestei afirmaţii să ne imaginăm că lucrăm pe un calculator ipotetic în care numărul de cifre semnificative este 3. Într-un astfel de calculator, să adunăm $x_1 = 3.73 = 0.373 \cdot 10^1$ cu $x_2 = 0.006 = 6 \cdot 10^{-3}$. Numărul x_2, de modul mai mic, se rescrie astfel încât să aibă exponentul 1, ca al numărului mai mare. $x_2 = 6 \cdot 10^{-4} \cdot 10^1 = 0.0006 \cdot 10^1$. Deoarece calculatorul permite memorarea doar a 3 cifre după virgulă, iar cifra 6 ar fi a patra, însemnă că de fapt cifra 6 este pierdută. La adunare x_2 este "văzut" a fi zero şi rezultatul adunării este de fapt x_1. Evident că un astfel de calculator nu există, în mod uzual se lucrează cu mai mult de 12 cifre semnificative, acest lucru depinzând de configuraţia hard, de limbajul de programare folosit şi de declaraţia făcută pentru variabile, dar ideea este exact cea descrisă de acest exemplu simplu.

Se defineşte zeroul (acurateţea, precizia,"epsilon-ul") maşinii ca fiind cel mai mic număr real care adunat la unitate îi modifică valoarea. Am putea spune că zeroul maşinii `eps` este cel mai mic număr pentru care `1 + eps > 1`. Pentru orice număr a mai mic decât zeroul maşinii $1 + a = 1$, unde această din urmă relaţie o considerăm efectuată în calculator şi nu în matematica exactă. În mod uzual[6], zeroul maşinii are valoarea $2.22 \cdot 10^{-16}$. Există limbaje de programare în care această caracteristică a mediului de programare este predefinită: în Matlab există funcţia `eps`, iar în Scilab variabila `%eps`. **Zeroul maşinii nu trebuie confundat cu cel mai mic număr reprezentabil în calculator** şi care, în mod uzual are valoarea $2.23 \cdot 10^{-308}$.

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

```
funcţie zeroul_maşinii ()
real eps
```

[6] calculatoare ce folosesc standardul IEEE şi variabile de tip `double`

eps = 1
`cât timp` (1 + eps > 1)
    eps = eps/2
•
eps = eps*2
`întoarce` eps

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

1.3.3 Standardul IEEE pentru reprezentarea în virgulă mobilă

În paragraful anterior, pentru analiza erorilor de rotunjire, s-a folosit notaţia (1.16) care a permis explicarea intuitivă a modului în care se fac adunările într-un calculator. Această notaţie, care foloseşte un exponent al lui 10 se numeşte *notaţie ştiinţifică*. Se mai întâlneşte *notaţia ştiinţifică normalizată*, $x = a \cdot 10^b$ unde exponentul întreg b se alege astfel încât $1 \leq |a| < 10$, notaţie care pune în evidenţă ordinul de mărime al unui număr real. Mai există şi *notaţia inginerească* în care b se alege numai dintre multiplii lui 3, şi, în consecinţă modului lui a poate avea valori între $1 \leq a < 1000$. Aceasta are avantajul că poate fi citită uşor cu ajutorul unor prefixe ca: mega, kilo, mili, micro etc.

Reprezentarea numerelor reale în calculator este un fel de echivalent hardware al notaţiei ştiinţifice [7]. Calculatoarele folosesc un număr finit de biţi pentru a reprezenta un număr real. Din acest motiv, în calculator poate fi reprezentată numai o mulţime finită de numere reale. Aceasta conduce la două probleme: pe de o parte numerele reale reprezentate nu pot fi oricât de mari sau oricât de mici, iar pe de altă parte, există spaţii între ele. Primul aspect pune rare probleme în practică (numite *overflow* în cazul în care s-ar dori reprezentarea unui număr prea mare sau *underflow* în cazul reprezentării unui număr prea mic). Problema a doua este mai delicată şi a a condus la elaborarea în 1985 de către *Institute of Electrical and Electronics Engineers (IEEE)* şi *American National Standards Institute (ANSI)* a standardului 754 care reglementează reprezentarea în virgulă mobilă. Acest standard a reprezentat rezultatul muncii desfăşurate pe parcursul unui deceniu de o echipă formată din aproape o sută de cercetători matematicieni, informaticieni şi ingineri din universităţi si industrie (fabricanţi de microprocesoare şi calculatoare).

Toate calculatoarele construite după 1985 folosesc acest standard IEEE pentru reprezentarea în virgulă mobilă. Aceasta nu înseamnă că rezultatele obţinute sunt identice pentru toate calculatoarele, deoarece există o oarecare flexibilitate în cadrul standardului.

Dar aceasta însemnă că există un model independent de calculator pentru modul în care se fac calculele aritmetice cu numere reale.

În cadrul acestui standard, scrierea normalizată a unui număr real x este

$$\bar{x} = \pm(1+f)\cdot 2^e, \tag{1.22}$$

unde f se numeşte *mantisă* şi e *exponent*. Mai mult, mantisa f satisface $0 \le f < 1$ iar exponentul e este un număr întreg cuprins în intervalul $-1022 \le e \le 1023$. Ambele se reprezintă în calculator ca numere binare (în baza 2). În calculator, orice număr real se aproximează cu un număr reprezentat ca în (1.22). De exemplu, numărul 0.1 nu poate fi reprezentat exact în acest standard deoarece reprezentarea sa în binar necesită un număr infinit de biţi:

$$\frac{1}{10} = \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^{10}} + \frac{0}{2^{11}} + \frac{1}{2^{12}} + \cdots$$

după primul termen secvenţa $1, 0, 0, 1$ repetându-se la infinit. În consecinţă, reprezentarea numărului 0.1 este un pic mai mare decât valoarea exactă. **Iată deci că nu numai numerele iraţionale sunt afectate de erori de rotunjire.**

Numerele în dublă precizie sunt reprezentate ca un cuvânt de 64 de biţi din care 52 de biţi pentru f, 11 biţi pentru e şi 1 bit pentru semnul numărului. În acest standard, intervalul [1,2] este reprezentat de mulţimea discretă $\{1, 1+2^{-52}, 1+2\cdot 2^{-52}, 1+3\cdot 2^{-52}, \ldots, 2\}$. Spaţiile dintre numerele acestui interval sunt egale cu $2^{-52} \approx 2.22\cdot 10^{-16}$. Acesta este zeroul maşinii şi corespunde la aproximativ 16 cifre semnificative în baza 10. Un interval $[2^j, 2^{j+1}]$ este reprezentat de mulţimea de mai sus înmulţită cu 2^j. Spaţiile dintre numerele vecine sunt deci scalate în funcţie de dimensiunea numerelor.

Zeroul maşinii reflectă rezoluţia mulţimii reale discrete $\mathcal{R}$ ce poate fi reprezentată în calculator. El are următoarea proprietate

$$(\forall) x \in \mathbb{R}, (\exists)\bar{x} \in \mathcal{R} \quad \textbf{astfel încât} \quad |x-\bar{x}| \le \texttt{eps}\,|x|. \tag{1.23}$$

Pe scurt, eroarea relativă dintre numărul real x şi reprezentarea lui discretă $\bar{x}$ este mărginită superior de zeroul maşinii. Sau, conform relaţiei (1.15) există ε, unde $|\varepsilon| \le \texttt{eps}$ astfel încât

$$\bar{x} = x(1+\varepsilon). \tag{1.24}$$

Restricţiile impuse pentru mantisă şi exponent conduc şi la o restricţie pentru cel mai mare/cel mai mic număr reprezentabil în calculator. Numărul cel mai mare care se poate reprezenta este $\texttt{realmax} = (2-\texttt{eps})\cdot 2^{1023}$ a cărui valoare zecimală este aproximativ $1.797\cdot 10^{308}$. Dacă un calcul încearcă să producă o valoare decât aceasta, se spune că

apare o *depăşire* în virgulă mobilă (*overflow*). Cel mai mic număr strict pozitiv care poate fi reprezentat în calculator este `realmin` $= 2 \cdot 2^{-1022}$ a cărui valoare zecimală este aproximativ $2.2251 \cdot 10^{-308}$. Un calcul care încearcă să producă o valoare pozitivă mai mică decât aceasta reprezintă tot o depăsire în virgulă mobilă (*underflow*).

Dacă un calcul încearcă să producă o valoare care este nedefinită şi în matematica numerelor reale (cum ar fi $0/0$), atunci rezultatul este o valoare excepţională numită NaN (*not a number*).

1.3.4 Analiza erorilor de trunchiere

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

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

$$f(x) = f(x_0) + \frac{x - x_0}{1!} f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \cdots \tag{1.25}$$

aplicată pentru funcţia sinus şi punctul de dezvoltare $x_0 = 0$:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}. \tag{1.26}$$

Estimarea acestei sume cu ajutorul calculatorului se va face trunchiind-o şi aproximând rezultatul cu

$$\bar{s} = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!}. \tag{1.27}$$

Conform teoriei seriilor alternate, se ştie că eroarea absolută făcută prin această trunchiere este mai mică decât ultimul termen considerat:

$$|e_s| = |\bar{s} - s| \leq \frac{x^{2n+1}}{(2n+1)!}. \tag{1.28}$$

Cunoaşterea acestui rezultat teoretic permite elaborarea unui algoritm de evaluare a funcţiei sinus care să stabilească singur când se va opri. Algoritmul va aduna termeni la suma parţială până când termenul curent sumat devine mai mic decât o anumită valoare. Această valoare nu trebuie să fie prea mare căci atunci soluţia va fi nesatisfăcătoare. De asemenea, o valoare mai mică decât zeroul maşinii este lipsită de sens deoarece un termen curent cu o astfel de valoare, adunat la suma parţială acumulată până în acel moment, nu mai are nici o influenţă asupra valorii sumei parţiale. Într-un astfel de moment, continuarea sumării seriei este inutilă, ea nu mai îmbunătăţeşte rezultatul, fenomen total diferit

de teoria matematică în care rezultatul este cu atât mai precis cu cât suma conţine mai mulţi termeni. Pseudocodul următor implementează calculul funcţiei sinus într-un punct x, cu o eroare impusă e.

`funcţie` sinus(x, e)
; întoarce valoarea funcţiei sinus in punctul x
; prin trunchierea seriei Taylor dezvoltată in 0
`real` x ; punctul în care se va evalua funcţia sin
`real` e ; eroarea de trunchiere impusă
`real` t, s
`întreg` k
$t = x$
$s = t$
$k = 0$
`cât timp` $(|t| > e)$
$k = k + 1$
$t = (-1) * t * \frac{x^2}{(2k)(2k+1)}$
$s = s + t$
•
`intoarce` s

În figura 1.4 este reprezentat modulul termenului curent în cazul evaluării funcţiei sinus în punctul $x = 1$, iar în figura 1.5 modulul diferenţei dintre sumele parţiale consecutive. Se observă că încă de la $k = 8$ adăugarea unui nou termen curent nu mai poate îmbunătăţi rezultatul. Dacă, în matematică, adunarea a 30 de termeni ar fi însemnat obţinerea unei erori de trunchiere mai mică decât 10^{80}, numeric, eroarea s-a saturat după adunarea a 8 termeni, la o valoare de ordinul zeroului maşinii.

Acest paragraf a exemplificat un algoritm în care se realizează controlul erorii de trunchiere. În general, algoritmii se concep astfel încât ei să ofere şi informaţii despre eroare. Algoritmii care nu fac aşa ceva sunt algoritmi lipsiţi de valoare practică.

1.3.5 Analiza erorilor inerente

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

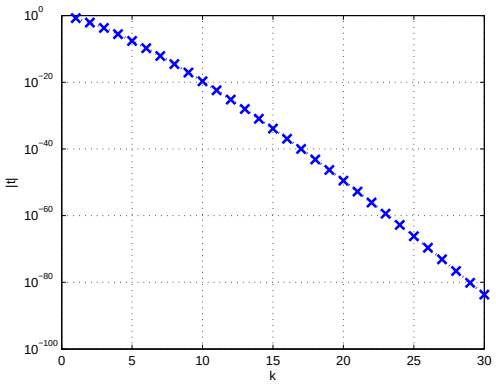


Figura 1.4: Modulul termenului curent al dezvoltării în serie Taylor a funcţiei sinus.

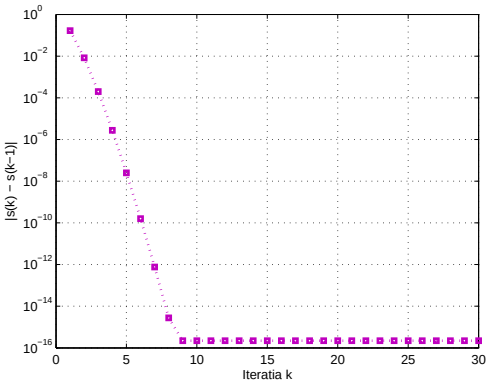


Figura 1.5: Modulul diferenţei dintre sume parţiale consecutive la dezvoltarea în serie Taylor a funcţiei sinus.

Pentru a putea evalua care este efectul micilor perturbaţii ale datelor de intrare independente notate $x_1,\ x_2,\ \ldots x_n$, considerate a fi mărimi reale, asupra rezultatului y considerat şi el un număr real, vom nota cu f dependenţa dintre date şi rezultat

$$y = f(x_1, x_2, \ldots, x_n). \tag{1.29}$$

Diferenţiala rezultatului va fi în consecinţă

$$\mathrm{d}y = \frac{\partial f}{\partial x_1}\mathrm{d}x_1 + \frac{\partial f}{\partial x_2}\mathrm{d}x_2 + \ldots \frac{\partial f}{\partial x_n}\mathrm{d}x_n. \tag{1.30}$$

Relaţia (1.30) poate fi scrisă aproximativ ca

$$\Delta y \approx \frac{\partial f}{\partial x_1}\Delta x_1 + \frac{\partial f}{\partial x_2}\Delta x_2 + \ldots \frac{\partial f}{\partial x_n}\Delta x_n. \tag{1.31}$$

Dacă mărimile Δx_k reprezintă mici perturbaţii ale datelor de intrare:

$$\Delta x_k = \bar{x_k} - x_k = e_{x_k}, \tag{1.32}$$

unde cu e_{x_k} a fost notată eroarea absolută a mărimii x_k, atunci rezultă că *eroarea absolută a rezultatului* $e_y = \bar{y} - y = \Delta y$ se poate calcula ca o combinaţie liniară a erorilor absolute a datelor de intrare:

$$e_y = \sum_{k=1}^{n} \frac{\partial f}{\partial x_k} e_{x_k}. \tag{1.33}$$

Pentru a estima o margine a erorii absolute a rezultatului se majorează modulul expresiei din membrul drept al relaţiei (1.33):

$$\left|\sum_{k=1}^{n} \frac{\partial f}{\partial x_k} e_{x_k}\right| \leq \sum_{k=1}^{n} \left|\frac{\partial f}{\partial x_k} e_{x_k}\right| = \sum_{k=1}^{n} \left|\frac{\partial f}{\partial x_k}\right| |e_{x_k}| \leq \sum_{k=1}^{n} \left|\frac{\partial f}{\partial x_k}\right| a_{x_k}, \tag{1.34}$$

unde a_{x_k} este marginea erorii absolute a datei de intrare: $|e_{x_k}| \leq a_{x_k}$. Din (1.34) rezultă că *marginea erorii absolute a rezultatului* se poate calcula ca

$$a_y = \sum_{k=1}^{n} \left| \frac{\partial f}{\partial x_k} \right| a_{x_k}. \tag{1.35}$$

Folosind (1.33) şi (1.29) se deduce că *eroarea relativă a rezultatului* $\varepsilon_y = e_y/|y|$ poate fi exprimată în funcţie de erorile relative ale datelor de intrare $\varepsilon_{x_k} = e_{x_k}/|x_k|$ astfel:

$$\varepsilon_y = \frac{\sum_{k=1}^{n} \frac{\partial f}{\partial x_k} e_{x_k}}{|y|} = \sum_{k=1}^{n} \frac{\partial f}{\partial x_k} \frac{e_{x_k}}{|y|} = \sum_{k=1}^{n} \frac{\partial f}{\partial x_k} \frac{|x_k|}{|y|} \varepsilon_{x_k}. \tag{1.36}$$

Majorând expresia din membrul drept, rezultă că *marginea erorii relative a rezultatului* r_y, unde $|\varepsilon_y| \leq r_y$ poate fi calculată în funcţie de marginile erorilor relative ale datelor de intrare r_{x_k}, unde $|\varepsilon_{x_k}| \leq r_{x_k}$:

$$r_y = \sum_{k=1}^{n} \left| \frac{\partial (\ln f)}{\partial x_k} \right| |x_k| r_{x_k}. \tag{1.37}$$

Este util să particularizăm aceste formule în cazul operaţiilor algebrice elementare: adunare, scădere, înmulţire şi împărţire. Rezultatele obţinute în cazul adunării şi scăderii a două numere reale sunt prezentate în tabelul 1.1.

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

Dacă analizăm marginea erorii relative de la adunare, se observă că erorile relative ale datelor de intrare sunt ponderate cu $|x_1/(x_1 + x_2)|$ şi $|x_2/(x_1 + x_2)|$, ambele ponderi fiind subunitare. Aceasta înseamnă că eroarea rezultatului adunării a două numere reale nu este amplificată, ea rămâne de acelaşi ordin de mărime cu erorile datelor de intrare. Se spune că **adunarea este o operaţie bine condiţionată**. Nu acelaşi lucru se poate spune despre

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

scădere. Ponderile în acest caz sunt $|x_1/(x_1-x_2)|$ şi $|x_2/(x_1-x_2)|$, care pot fi oricât de mari pentru că diferenţa $x_1 - x_2$ poate fi oricât de mică. În consecinţă, rezultatul unei scăderi poate fi afectat de erori relative mult mai mari decât erorile relative ale datelor de intrare. Se spune că **scăderea este o operaţie prost condiţionată**. Proasta condiţionare la scădere apare mai ales atunci când numerele sunt foarte apropiate, efect cunoscut şi sub numele de *efect de anulare prin scădere*.

De exemplu, dacă $x_1 = 1.23 \pm 1\%$ şi $x_2 = 1.22 \pm 1\%$, rezultă atunci că marginea erorii relative la scădere este $r = |1.23/0.01 \cdot 1/100 + 1.22/0.01 \cdot 1/100 = 1.23 + 1.22 = 2.45 = 245\%$, deci $x_1 - x_2 = 0.01 \pm 245\%$. Eroarea relativă la scădere este de peste 200 de ori mai mare decât erorile relative ale datelor de intrare. La adunare în schimb eroarea relativă este $r = |1.23/2.45 \cdot 1/100 + 1.22/2.45 \cdot 1/100 \approx 0.5 \cdot 1/100 + 0.5 \cdot 1/100 = 1/100 = 1\%$. Deci $x_1 + x_2 = 2.45 \pm 1\%$.

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

Tabelul 1.2 conţine particularizarea formulelor de eroare în cazul operaţiilor de înmulţire şi împărţire. Se observă în acest caz faptul că **înmulţirea şi împărţirea sunt operaţii bine condiţionate**, eroarea rezultatului fiind de ordinul de mărime al datelor de intrare. Pentru exemplul numeric de mai sus, erorile relative vor fi majorate de 2%.

În concluzie, din cele patru operaţii algebrice elementare, numai scăderea este prost condiţionată şi ea trebuie evitată pe cât posibil în calculele numerice, mai ales atunci când numerele care se scad sunt apropiate. Un exemplu tipic în acest sens îl reprezintă algoritmul de calcul al soluţiilor unei ecuaţii de gradul doi $ax^2 + bx + c = 0$, $x_{1,2} = (-b \pm \sqrt{b^2 - 4ac})/(2a)$. Presupunând că $b > 0$ şi că $b^2 \gg 4ac$, rezultă că rezultatul radicalului va fi foarte apropiat în modul de b, astfel încât, soluţia cu plus reprezintă de fapt scăderea a două numere foarte apropiate şi va fi afectată în consecintă de erori

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

Tabelul 1.2: Erorile rezultatului înmulţirii şi împărţirii a două numere reale în funcţie de erorile datelor de intrare.

foarte mari. În acest caz, este mult mai eficient să se calculeze doar soluţia obţinută prin adunare, $x_1 = (-b-\sqrt{b^2-4ac})/(2a)$, urmând ca soluţia a doua să se obţină din relaţia lui Viète: $x_2 = c/(ax_1)$, relaţie care presupune doar folosirea înmulţirii şi împărţirii, operaţii bine condiţionate. Pe scurt, în pseudocod, aceasta se scrie astfel

dacă $b > 0$
    $x1 = (-b - \sqrt{b^2-4ac})/(2a)$; soluţia cu minus este o adunare dacă $b > 0$
altfel
    $x1 = (-b + \sqrt{b^2-4ac})/(2a)$; soluţia cu plus este o adunare dacă $b < 0$
•
$x2 = c/(a * x1)$

Un alt exemplu interesant este cel al funcţiei $y = \sqrt{x}$ pentru care rezultă o eroare absolută

$$e_y = \frac{df}{dx}e_x = \frac{1}{2\sqrt{x}}e_x, \tag{1.38}$$

şi o eroare relativă egală cu jumătate din eroarea relativă a datei x:

$$\varepsilon_y = \frac{e_y}{y} = \frac{1}{2\sqrt{x}\sqrt{x}}e_x = \frac{e_x}{2x} = \frac{\varepsilon_x}{2}. \tag{1.39}$$

Am putea deduce de aici, în mod eronat, că în acest caz, dacă aplicăm radicalul în mod repetat rezultatului, eroarea tinde către zero. Toate calculele prezentate în acest paragraf (inclusiv acesta din urmă) au presupus un calcul exact, adică un calcul fără erori de rotunjire. Rotunjirea nu poate fi însă ignorată. Nu putem separa proprietăţile erorilor inerente de efectul rotunjirii. De aceea, vom accepta un fel de superpoziţie a erorilor în sensul că eroarea relativă într-un calcul aproximativ este egală cu eroarea relativă produsă de calculul aproximativ cu numere exacte (adică eroarea de rotunjire) plus eroarea relativă produsă de calculul exact cu numere aproximative (afectate deci de erori inerente). Pentru a justifica această afirmaţie, să notăm cu $\bar{y}$ valoarea rotunjită a rezultatului y_i (presupus pozitiv), în care s-au acumulat erori inerente. În consecinţă, folosind (1.15) rezultă

$$\bar{y} = y_i(1 + \texttt{eps}) = y(1 + \varepsilon_y)(1 + \texttt{eps}) \approx y(1 + \varepsilon_y + \texttt{eps}),$$

de unde $(\bar{y} - y)/y = \varepsilon_y + \texttt{eps}$. Cu această precizare, rezultatului dat de relaţia (1.39) trebuie să i se adauge o eroare de rotunjire

$$\varepsilon_{\sqrt{x}} = \frac{\varepsilon_x}{2} + \texttt{eps}. \tag{1.40}$$

În concluzie, eroarea relativă a oricărui rezultat numeric este cel puţin egală cu zeroul maşinii.

1.3.6 Condiţionare şi stabilitate

Condiţionarea şi stabilitatea sunt două aspecte fundamentale în rezolvarea numerică a problemelor. Despre condiţionare s-a discutat şi în paragraful anterior, unde s-a demonstrat că, dintre operaţiile algebrice, scăderea este prost condiţionată.

Condiţionarea se referă la comportarea problemei matematice la perturbaţii ale datelor.

Fie o problemă matematică, notată f, pentru care se doreşte aflarea unei soluţii $\mathbf{x}$ pentru datele $\mathbf{d}$. Problema f, privită ca o aplicaţie definită pe mulţimea datelor $\mathcal{D}$ cu valori în mulţimea soluţiilor $\mathcal{X}$, poate fi formulată astfel:

$$\begin{aligned} &\text{Fie} \quad f : \mathcal{D} \to \mathcal{X} \quad \text{şi} \quad \mathbf{d} \in \mathcal{D}. \\ &\text{Să se găsească} \quad \mathbf{x} \in \mathcal{X} \quad \text{astfel încât} \quad f(\mathbf{d}) = \mathbf{x}. \end{aligned} \tag{1.41}$$

O problemă este bine condiţionată dacă perturbaţii mici ale datelor conduc la perturbaţii mici ale rezultatului. Reprezentări intuitive ale unei probleme bine condiţionate sunt cele din figura 1.6. O problemă prost condiţionată este cea pentru care perturbaţii mici ale datelor conduc la perturbaţii mari ale rezultatului (figura 1.7).

Dacă datele şi rezultatele sunt numere reale, atunci o problemă f este prost condiţionată dacă ea are alura curbei din figura 1.8 - stânga.

De multe ori, problema matematică nu este formulată explicit, ci implicit:

$$\begin{aligned} &\text{Fie} \quad g : \mathcal{X} \to \mathcal{D} \quad \text{şi} \quad \mathbf{d} \in \mathcal{D}. \\ &\text{Să se găsească} \quad \mathbf{x} \in \mathcal{X} \quad \text{astfel încât} \quad g(\mathbf{x}) = \mathbf{d}. \end{aligned} \tag{1.42}$$

În acest caz, problema g este prost condiţionată dacă ea are alura curbei din figura 1.8, dreapta.

Pentru aprecierea cantitativă a bunei sau a proastei condiţionări a unei probleme se foloseşte *numărul de condiţionare*. El reprezintă o măsură a perturbaţiilor rezultatului la perturbaţii mici ale datelor şi poate fi definit atât în mod absolut cât şi relativ.

Se defineşte *numărul de condiţionare absolut* ca fiind

$$\hat{\kappa} = \lim_{\mu \to 0} \sup_{\|\delta \mathbf{d}\| < \mu} \frac{\|\delta f\|}{\|\delta \mathbf{d}\|}, \tag{1.43}$$

sau, într-o scriere simplificată

$$\hat{\kappa} = \sup_{\|\delta \mathbf{d}\|} \frac{\|\delta f\|}{\|\delta \mathbf{d}\|}, \tag{1.44}$$

unde $\delta f = f(\mathbf{d} + \delta \mathbf{d}) - f(\mathbf{d})$ şi $\delta \mathbf{d}$ sunt mărimi infinitezimale. Dacă f este derivabilă, atunci perturbaţia δf se poate aproxima în condiţia $\|\delta \mathbf{d}\| \to 0$ ca

$$\delta f = \mathbf{J}(\mathbf{d}) \delta \mathbf{d}, \tag{1.45}$$

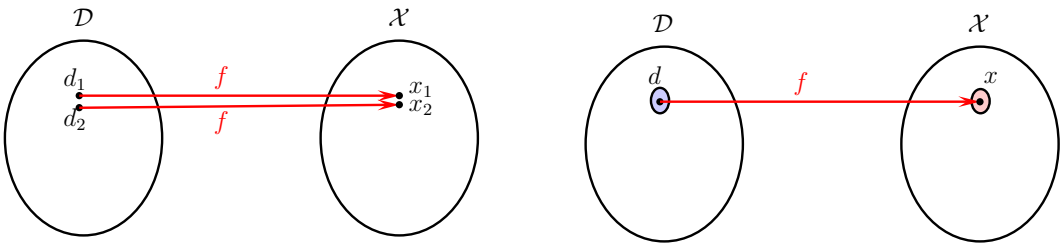


Figura 1.6: Reprezentări intuitive ale unei probleme bine condiţionate.

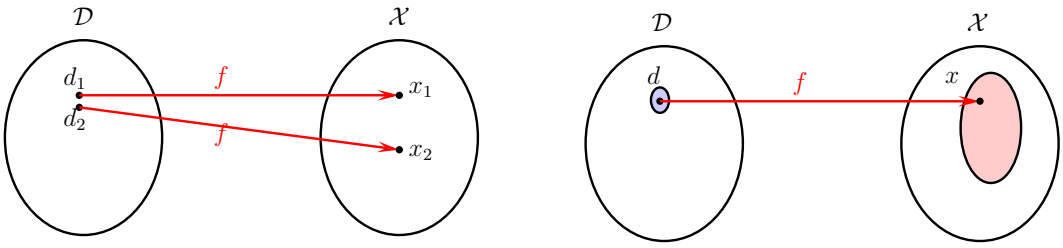


Figura 1.7: Reprezentări intuitive ale unei probleme prost condiţionate.

unde $\mathbf{J}(\mathbf{d})$ este matricea Jacobian[7] asociată funcţiei f. În consecinţă, numărul de condiţionare absolut este norma matricea Jacobian:

$$\hat{\kappa} = \|\mathbf{J}(\mathbf{d})\|. \tag{1.46}$$

În cazul în care mulţimea de date şi mulţimea de rezultate sunt incluse în $\mathbb{R}$, numărul de condiţionare absolut este modulul derivatei funcţiei f (corespunzătoare problemei definite explicit). O problemă prost condiţionată este o problema cu modulul derivatei mare, ceea ce corespunde unei pante mari a graficului funcţiei (figura 1.8 - stânga).

Atunci când interesează perturbaţiile relative, se utilizează *numărul de condiţionare relativ*

$$\kappa = \lim_{\mu \to 0} \sup_{\|\delta \mathbf{d}\| < \mu} \frac{\|\delta f\| / \|f(\mathbf{d})\|}{\|\delta \mathbf{d}\| / \|\mathbf{d}\|}, \tag{1.47}$$

sau, scris mai simplu în ipoteza unor variaţii infinitezimale

$$\kappa = \sup_{\|\delta \mathbf{d}\|} \frac{\|\delta f\| / \|f(\mathbf{d})\|}{\|\delta \mathbf{d}\| / \| \mathbf{d}\|}. \tag{1.48}$$

Dacă f este derivabilă, atunci

$$k = \frac{\|\mathbf{J}(\mathbf{d})\|}{\|f(\mathbf{d})\| / \|\mathbf{d}\|}. \tag{1.49}$$

[7]Daca f este o funcţie cu valori scalare, ce depinde de n date, atunci Jacobianul este un vector cu n componente, fiecare componentă fiind derivata parţială a funcţiei în raport cu o dată $\partial f / \partial d_j$, $j = 1, n$. Dacă f este o funcţie cu valori vectoriale (fie de dimensiune m), fiecare componentă depinzând de n date, atunci matricea Jacobian are dimensiunea $m \times n$, un element al ei fiind $\partial f_i / \partial d_j$, $i = 1, m$, $j = 1, n$.

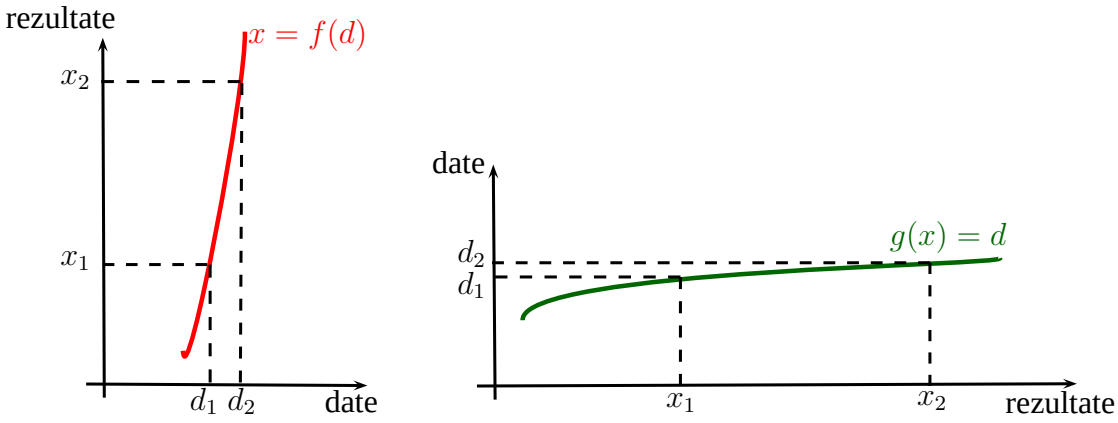


Figura 1.8: Problema prost condiţionată: stânga - formulată explicit: $x = f(d)$; dreapta - formulată implicit: $g(x) = d$.

O problemă este bine condiţionată dacă valoarea lui κ este mică şi prost condiţionată dacă valorea lui κ este mare. Ce însemnă mic sau mare, depinde de problemă.

Ca exemplu, să calculăm numărul de condiţionare al scăderii a două numere reale $f(\mathbf{d}) = d_1 - d_2$, unde $\mathbf{d} = [d_1, d_2]^T$. Atunci, Jacobianul este $\mathbf{J} = [\partial f/\partial d_1, \partial f/\partial d_2]^T = [1, -1]^T$, iar numărul de condiţionare calculat cu norme inf[8] este

$$\kappa = \frac{\|\mathbf{J}(\mathbf{d})\|}{\|f(\mathbf{d})\|/\|\mathbf{d}\|} = \frac{1}{|d_1 - d_2|/\max\{|d_1|, |d_2|\}}. \tag{1.50}$$

Este evident că scăderea este prost condiţionată dacă $d_1 \approx d_2$, rezultat justificat şi în paragraful anterior.

Un alt exemplu este numărul de condiţionare pentru înmulţirea dintre o matrice $\mathbf{A}$ şi un vector $\mathbf{d}$ unde se presupun perturbate valorile lui $\mathbf{d}$. În acest caz $f(\mathbf{d}) = \mathbf{A}\mathbf{d}$. Se poate demonstra [16] că numărul de condiţionare al acestei probleme este mărginit superior de

$$\kappa \leq \|\mathbf{A}\|\|\mathbf{A}^{-1}\|. \tag{1.51}$$

De asemenea, pentru problema înmulţirii inversei unei matrice $\mathbf{A}$ cu un vector, numărul de condiţionare satisface relaţia (1.51).

Produsul $\|\mathbf{A}\|\|\mathbf{A}^{-1}\|$ apare atât de des încât el a primit numele de *numărul de condiţionare al unei matrice*:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|. \tag{1.52}$$

În acest caz, el este ataşat unei matrice şi nu unei probleme. Dacă are valoare mică se spune că matricea e bine condiţionată.

[8]Norma inf a unui vector este maximum dintre modulele componentelor sale: $\|\mathbf{v}\|_\infty = \max_{1 \leq i \leq n} |v_i|$.

Un alt exemplu este numărul de condiţionare al rezolvării unui sistem de ecuaţii. El este discutat în detaliu în paragraful 2.1. Şi acest număr este legat de numărul de condiţionare al unei matrice.

Din (1.48) rezultă că, pentru o perturbaţie oarecare a datelor $\delta\mathbf{d}$, căreia îi corespunde o perturbaţie a rezultatului $\|\delta f\|$ rezultă

$$\frac{\|\delta f\|/\|f\|}{\|\delta\mathbf{d}\|/\|\mathbf{d}\|} \leq \kappa. \tag{1.53}$$

Perturbaţia rezultatului este o distanţa în spaţiul soluţiilor $\mathcal{X}$, deci reprezintă o eroare absolută $\mathbf{e_x} = \delta f$ sau relativă $\varepsilon_\mathbf{x} = \delta f/\|f\|$. Perturbaţia datelor, numită şi *reziduu* este o distanţă în spaţiul $\mathcal{D}$. Reziduul poate fi absolut $\mathbf{e_d} = \delta\mathbf{d}$, unde $\delta\mathbf{d} = \bar{\mathbf{d}} - \mathbf{d}$, sau relativ $\varepsilon_\mathbf{d} = \delta\mathbf{d}/\|\mathbf{d}\|$. Cu aceste notaţii, relaţia (1.53) devine

$$\|\mathbf{e_x}\| \leq \kappa\|\varepsilon_\mathbf{d}\|. \tag{1.54}$$

Eroarea şi reziduul sunt legate prin numărul de condiţionare. Din inegalitatea (1.54) rezultă că pentru o problemă cu număr de condiţionare mic, o perturbaţie mică în date va duce la o perturbaţie mică a rezultatului. Problemele matematice care au κ mare sunt prost condiţionate şi ele nu pot fi rezolvate cu ajutorul calculatorului. Pentru astfel de probleme, trebuie găsită o formulare matematică echivalentă din punct de vedere al rezultatului, dar bine condiţionată.

În cele ce urmează vom presupune că problema f este bine condiţionată şi pentru rezolvarea ei a fost conceput un algoritm $\bar{f}$.

Acurateţea unui algoritm se referă la eroarea soluţiei numerice. Interesează ca algoritmul să fie precis, adică eroarea rezultatului să fie mică (figura 1.9).

Un algoritm este precis dacă norma erorii relative

$$\|\varepsilon_\mathbf{x}\| = \frac{\|\bar{f}(\mathbf{d}) - f(\mathbf{d})\|}{\|f(\mathbf{d})\|} \tag{1.55}$$

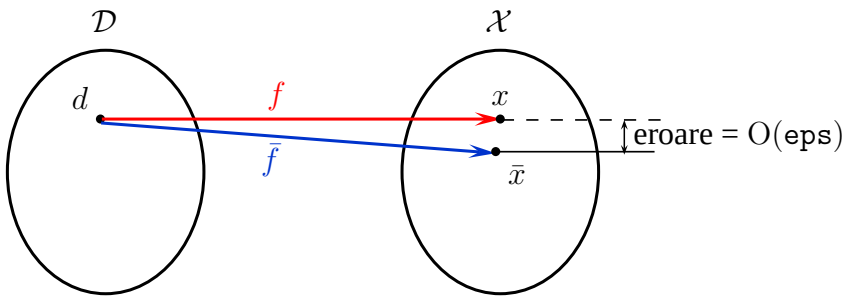


Figura 1.9: Reprezentarea intuitivă a unui algoritm a cărui precizie este ideală.

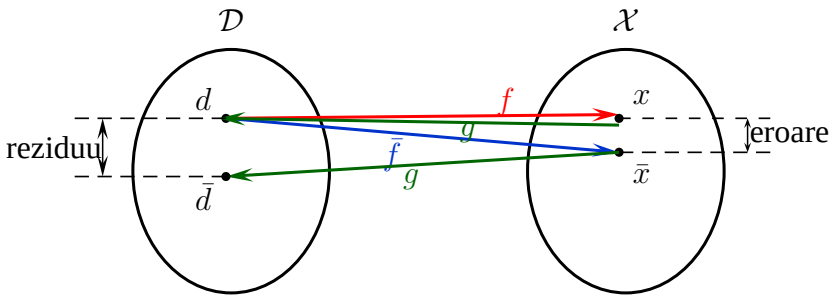


Figura 1.10: Reziduul este o distanţă în spaţiul datelor. Eroarea este o distanţă în spaţiul soluţiilor.

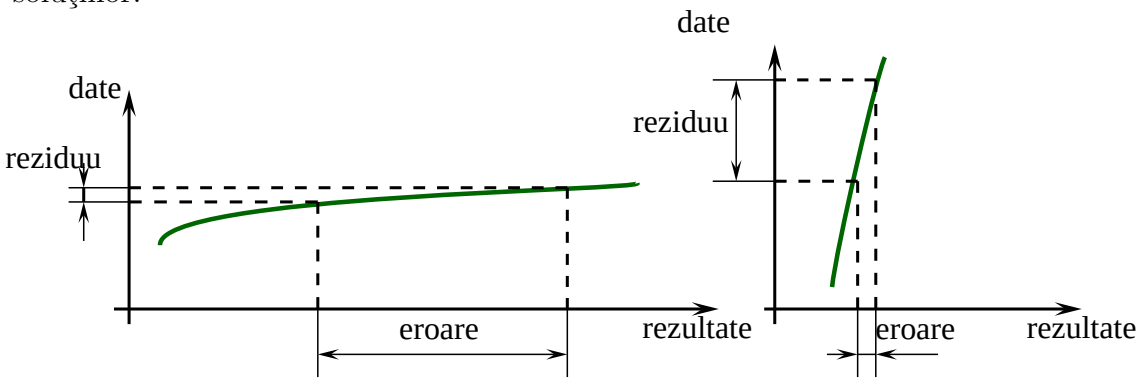


Figura 1.11: Reziduul nu oferă informaţii despre eroare: stânga - reziduul mic, eroare mare (problemă prost condiţionată), dreapta - reziduul mare, eroare mică.

este mică. În mod ideal, un algoritm este precis dacă modulul erorii relative este de ordinul zeroului maşinii

$$\frac{\|\bar{f}(\mathbf{d}) - f(\mathbf{d})\|}{\|f(\mathbf{d})\|} = \mathrm{O}(\mathtt{eps}). \tag{1.56}$$

În cazul în care problema este formulată implicit ca (1.42) este uşor a se calcula cât de bine este satisfăcută relaţia "$g(\mathbf{x}) = \mathbf{d}$ prin înlocuirea lui $\mathbf{x}$ cu $\bar{\mathbf{x}}$. Mărimea

$$g(\bar{\mathbf{x}}) - \mathbf{d} = \bar{\mathbf{d}} - \mathbf{d} = \mathbf{e_d} \tag{1.57}$$

este o distanţă în spaţiul datelor, deci un reziduu (figura 1.10). Reziduul dă informaţii asupra problemei care s-a rezolvat de fapt. În general însă, el nu dă informaţii despre eroare. În cazul unei probleme prost condiţionate reziduul poate fi mic deşi eroarea este mare, iar în cazul unei probleme bine condiţionate reziduul poate fi mare deşi eroarea este mică (figura 1.11).

Pentru a ilustra această afirmaţie, să considerăm sistemul de două ecuaţii cu două necunoscute

$$\mathbf{Ax} = \mathbf{b}, \quad \text{unde} \quad \mathbf{A} = \begin{bmatrix} 0.78 & 0.563 \\ 0.913 & 0.659 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0.217 \\ 0.254 \end{bmatrix}.$$

Matricea coeficienţilor acestui sistem are determinantul egal cu 1e-6, diferit de zero, ceea ce înseamnă că din punct de vedere matematic problema are o soluţie unică. Se verifică uşor că soluţia exactă a problemei este $\mathbf{x} = [1, -1]^T$. Dacă $\bar{\mathbf{x}}$ este o soluţie aproximativă, atunci reziduul este

$$\varepsilon_{\mathbf{b}} = \bar{\mathbf{b}} - \mathbf{b} = \mathbf{A}\bar{\mathbf{x}} - \mathbf{b}.$$

Dacă analizăm posibilele soluţii aproximative $\bar{\mathbf{x}}_1 = [0.999, -1.001]^T$ şi $\bar{\mathbf{x}}_2 = [0.341, -0.087]^T$, se observă că reziduul primei mărimi $\bar{\mathbf{d}}_1 = [-0.0013, -0.0016]^T$ este mai mare în orice normă decât reziduul celei de a doua $\bar{\mathbf{d}}_2 = [-10^{-6}, 0]^T$, deşi este evident că prima soluţie are eroarea mai mică. Acest exemplu este o problemă slab condiţionată. Intuitiv, cele două drepte în planul (x_1, x_2) corespunzătoare celor două ecuaţii, deşi nu sunt identice, sunt aproape suprapuse. O discuţie detaliată despre cum se poate estima cantitativ buna sau slaba condiţionare a rezolvării unui sistem algebric de ecuaţii liniare este prezentată în paragraful 2.1. **Pentru probleme prost condiţionate, reziduul poate furniza informaţii eronate despre eroare.**

În relaţia (1.55), mărimea $\bar{f}(\mathbf{d})$ semnifică "rezultatul algoritmului f aplicat datelor $\mathbf{d}$". În realitate, rotunjirea datelor de intrare este inevitabilă, erorile se acumulează şi perturbă rezultatul. În loc de a ţinti o astfel de acurateţe, este util să se ţintească *stabilitatea algoritmului*.

Stabilitatea unui algoritm se referă la comportarea algoritmului atunci când datele de intrare sunt perturbate. Un algoritm $\bar{f}$ folosit pentru rezolvarea unei probleme f este stabil dacă

$$\frac{\|\bar{f}(\bar{\mathbf{d}}) - f(\mathbf{d})\|}{\|f(\mathbf{d})\|} = \mathrm{O}(\mathtt{eps}), \tag{1.58}$$

pentru $(\forall)\bar{\mathbf{d}}, \mathbf{d}$ care satisfac $\|\bar{\mathbf{d}} - \mathbf{d}\|/\|\mathbf{d}\| = \mathrm{O}(\mathtt{eps})$. Pe scurt, **un algoritm stabil dă răspunsul aproape corect pentru date reprezentate aproape precis**.

Pentru a ilustra noţiunea de stabilitate a unui algoritm, să considerăm sistemul de două ecuaţii cu două necunoscute

$$\mathbf{Ax} = \mathbf{b}, \quad \text{unde} \quad \mathbf{A} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Sistemul, scris explicit ca

$$\begin{aligned} x_2 &= 1 \\ x_1 + x_2 &= 0 \end{aligned} \tag{1.59}$$

are soluţia $x_1 = -1$, $x_2 = 1$. Cu notaţiile anterioare $\mathbf{x} = f(\mathbf{d}) = [-1, 1]^T$.

Să considerăm acum că datele (numai matricea coeficienţilor) au fost perturbate:

$$\bar{\mathbf{A}} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix},$$

ceea ce corespunde scrierii explicite

$$\begin{aligned} 10^{-20}x_1 &+ x_2 = 1 \\ x_1 &+ x_2 = 0 \end{aligned} \tag{1.60}$$

Soluţia exactă a acestei probleme perturbate este $x_1' = -x_2' = 1/(10^{-20} - 1) \approx -1$, extrem de puţin perturbată faţa de soluţia exactă a problemei neperturbate. Într-adevăr, se poate demonstra că această problemă este bine condiţionată. Intuitiv, soluţia se găseşte la intersecţia celei de a doua bisectoare cu o dreaptă verticală.

Un posibil algoritm de rezolvare este algoritmul $\bar{f}_1$, descris pe scurt astfel:

- **Pasul 1:** se înmulţeşte prima ecuaţie a sistemului (1.60) cu (-10^{20}) şi se adună cu a doua, rezultând x_2;
- **Pasul 2:** se calculează x_1 din prima ecuaţie.

La pasul 1 se ajunge la ecuaţia $(1 - 10^{20})x_2 = -10^{20}$ care, în calculator devine datorită rotunjirilor $-10^{20}x_2 = -10^{20}$, de unde va rezulta $x_2 = 1$, ceea ce este corect. La pasul 2 ecuaţia de rezolvat devine $10^{-20}x_1 + 1 = 1$, de unde va rezulta $x_1 = 0$, ceea ce este greşit, foarte departe de valoarea adevărată. Acest algoritm este instabil.

Un alt algoritm posibil de rezolvare este algoritmul $\bar{f}_2$:

- **Pasul 1:** se înmulţeşte a doua ecuaţie a sistemului (1.60) cu (-10^{-20}) şi se adună cu prima, rezultând x_2;
- **Pasul 2:** se calculează x_1 din a doua ecuaţie.

La pasul 1 se ajunge la ecuaţia $(1 - 10^{-20})x_2 = 1$, care în calculator devine $x_2 = 1$. La pasul 2 ecuaţia de rezolvat este $x_1 + 1 = 0$, de unde $x_1 = -1$, ceea ce este corect.

Algoritmul $\bar{f}_1$ este instabil, pe când algoritmul $\bar{f}_2$ este stabil. Stabilitatea lui este foarte puternică, el a dat răspunsul exact pentru date de intrare aproape precise.

În concluzie, pentru estimarea acurateţii unei soluţii numerice trebuie parcurşi următorii paşi:

1. **Se estimează numărul de condiţionare al problemei. Se continuă numai dacă problema matematică este bine condiţionată.**
2. **Se investighează stabilitatea algoritmului. Cel mai simplu este ca acest lucru să se realizeze experimental, rulându-se algoritmul pentru date perturbate. Dacă dispersia rezultatelor este mare atunci algoritmul este instabil şi trebuie schimbat.**

3. **Dacă algoritmul este stabil, atunci acurateţea finală (modulul erorii relative) este majorată de produsul dintre numărul de condiţionare şi modulul reziduului relativ.**

Despre un algoritm stabil care generează erori mici pentru probleme bine condiţionate se spune că este robust.

Capitolul 2

Sisteme de ecuaţii algebrice liniare

Rezolvarea cu ajutorul calculatorului a sistemelor de ecuaţii algebrice liniare este cea mai frecvent întâlnită problemă matematică ce apare pe parcursul rezolvării aplicaţiilor de inginerie electrică. Un astfel de exemplu este analiza circuitelor rezistive liniare. Analiza circuitele rezistive neliniare conduce la sisteme de ecuaţii algebrice neliniare care se rezolvă iterativ, fiecare iteraţie necesitând rezolvarea unui sistem algebric liniar. Circuitele liniare în regim tranzitoriu necesită şi ele rezolvarea unui sistem de ecuaţii algebrice liniare la fiecare pas de timp. De asemenea, analiza oricărei probleme de câmp electromagnetic necesită rezolvarea unuia sau mai multor sisteme de ecuaţii algebrice liniare.

2.1 Formularea problemei

Problema matematică abordată în acest capitol este rezolvarea unui sistem de n ecuaţii algebrice liniare cu n necunoscute:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{cases} \tag{2.1}$$

Altfel spus, dată fiind matricea coeficienţilor

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n} \tag{2.2}$$

şi vectorul termenilor liberi

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}^T \in \mathbb{R}^n, \tag{2.3}$$

se cere să se rezolve sistemul

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.4}$$

unde $\mathbf{x}$ este soluţia

$$\mathbf{x} = \left[\begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} \right]^T \in \mathbb{R}^n. \tag{2.5}$$

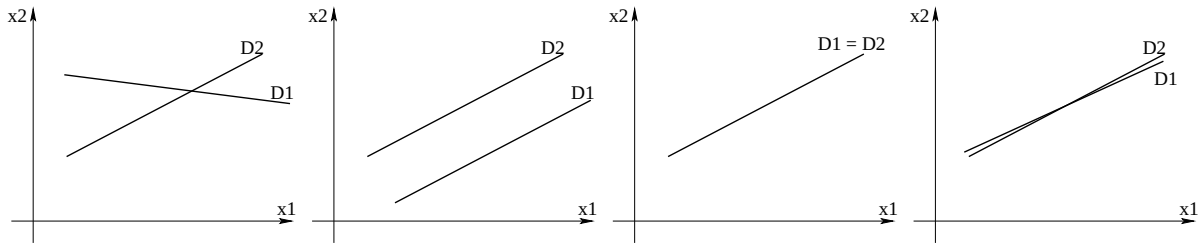
O astfel de problemă este bine formulată din punct de vedere matematic (soluţia există şi este unică) dacă şi numai dacă matricea $\mathbf{A}$ este nesingulară (are determinantul nenul). În aceste condiţii, se poate scrie că $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, unde $\mathbf{A}^{-1}$ este inversa matricei $\mathbf{A}$. Vom vedea însă că este ineficient ca soluţia $\mathbf{x}$ să se determine numeric folosind inversa matricei calculată explicit. În contextul metodelor numerice, scrierea "$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$" este pur formală, ea trebuie citită ca "$\mathbf{x}$ este soluţia sistemului algebric liniar $\mathbf{A}\mathbf{x} = \mathbf{b}$" şi nu "se calculează inversa matricei $\mathbf{A}$ care se înmulţeşte cu vectorul $\mathbf{b}$".

2.2 Condiţionarea problemei

În paragraful 1.3.6 s-a discutat problema condiţionării în general, definindu-se numărul de condiţionare ca (1.48) sau (1.49) în cazul în care funcţia matematică ce descrie problema de rezolvat este derivabilă.

Nu orice problemă de rezolvare a unui sistem de ecuaţii algebrice liniare care este bine formulată matematic este şi bine condiţionată. Această afirmaţie poate fi înţeleasă intuitiv pentru cazul $n = 2$. Găsirea soluţiei unui sistem algebric liniar de două ecuaţii cu două necunoscute are ca semnificaţie geometrică găsirea punctului de intersecţie dintre cele două drepte care reprezintă fiecare ecuaţie din sistem. În cazul unei bune formulări matematice, cele două drepte sunt concurente (fig. 2.1 a). Cazul unei matrice singulare are drept corespondent geometric fie două drepte paralele (caz în care sistemul nu are nici o soluţie - fig. 2.1 b), fie două drepte confundate (caz în care sistemul are o infinitate de soluţii - fig. 2.1 c).

În vecinătatea cazului de proastă formulare matematică există o mulţime de situaţii în care dreptele respective au pante foarte apropiate, fără să fie confundate (fig. 2.1 d). Ştiind că, de fapt, nici coeficienţii matricei, nici termenii liberi nu sunt cunoscuţi cu precizie infinită, aceasta însemnă că, la o mică perturbaţie a datelor, soluţia poate varia extrem de mult. Este ca şi cum dreptele nu sunt desenate cu un creion bine ascuţit ci cu un creion cu un vârf mai gros, astfel încât zona în care se află soluţia este mare. Dacă această zonă cuprinde mai multe ordine de mărime, atunci este dificil, dacă nu chiar imposibil de a se determina soluţia. Mici perturbaţii ale datelor conduc la perturbaţii mari ale soluţiei şi problema, deşi bine formulată matematic, este slab condiţionată.



a) b) c) d)

Figura 2.1: a) Problemă matematică bine formulată şi bine condiţionată. b) Problemă matematică prost formulată (nu există soluţie). c) Problemă matematică prost formulată (are o infinitate de soluţii). d) Problemă matematică bine formulată şi slab condiţionată.

În cele ce urmează se urmăreşte aflarea unui expresii cantitative pentru numărul de condiţionare al acestei probleme.

Fie de rezolvat sistemul

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.6}$$

unde $\mathbf{x}$ este soluţia exactă şi presupunem o perturbaţie $\mathbf{x} + \mathbf{e_x}$ a soluţiei, corespunzătoare unei perturbaţii $\mathbf{b} + \mathbf{e_b}$ a datelor:

$$\mathbf{A}(\mathbf{x} + \mathbf{e_x}) = \mathbf{b} + \mathbf{e_b}. \tag{2.7}$$

Din (2.6) şi (2.7) rezultă că şi perturbaţiile satisfac un sistem de ecuaţii algebrice liniare, cu aceeaşi matrice a coeficienţilor:

$$\mathbf{A}\mathbf{e_x} = \mathbf{e_b}. \tag{2.8}$$

Vom nota eroarea relativă a soluţiei cu ε_x şi eroarea relativă a datelor cu ε_b:

$$\varepsilon_x = \frac{\|\mathbf{e_x}\|}{\|\mathbf{x}\|}, \quad \varepsilon_b = \frac{\|\mathbf{e_b}\|}{\|\mathbf{b}\|}. \tag{2.9}$$

Din (2.8) rezultă

$$\mathbf{e_x} = \mathbf{A}^{-1}\mathbf{e_b}, \tag{2.10}$$

şi, în consecinţă, norma perturbaţiei soluţiei poate fi majorată de

$$\|\mathbf{e_x}\| = \|\mathbf{A}^{-1}\mathbf{e_b}\| \leq \|\mathbf{A}^{-1}\|\|\mathbf{e_b}\|. \tag{2.11}$$

Din (2.6) rezultă un minorant pentru norma soluţiei

$$\|\mathbf{b}\| = \|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\|\|\mathbf{x}\| \quad \Rightarrow \quad \|\mathbf{x}\| \geq \frac{\|\mathbf{b}\|}{\|\mathbf{A}\|}. \tag{2.12}$$

Din (2.11) şi (2.12) rezultă un majorant pentru eroarea asupra soluţiei

$$\varepsilon_x = \frac{\|\mathbf{e_x}\|}{\|\mathbf{x}\|} \leq \frac{\|\mathbf{A}^{-1}\|\|\mathbf{e_b}\|}{\frac{\|\mathbf{b}\|}{\|\mathbf{A}\|}} = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|\varepsilon_b. \tag{2.13}$$

Mărimea

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\| \tag{2.14}$$

se numeşte număr de condiţionare la inversare al matricei $\mathbf{A}$.

Relaţia (2.13) devine

$$\varepsilon_x \leq \kappa(\mathbf{A})\varepsilon_b, \tag{2.15}$$

ceea ce reprezintă de fapt relaţia (1.54) scrisă pentru problema rezolvării sistemelor algebrice liniare. Demonstrarea expresiei numărului de condiţionare se poate face şi pornind de la relaţia (1.49) astfel:

$$\begin{aligned}\kappa &= \frac{\|\mathbf{J}(\mathbf{b})\|}{\|\mathbf{f}(\mathbf{b})\|/\|\mathbf{b}\|} = \frac{\|\mathbf{A}^{-1}\|}{\|\mathbf{A}^{-1}\mathbf{b}\|/\|\mathbf{b}\|} = \frac{\|\mathbf{A}^{-1}\|\|\mathbf{b}\|}{\|\mathbf{A}^{-1}\mathbf{b}\|} = \frac{\|\mathbf{A}^{-1}\|\|\mathbf{A}\mathbf{A}^{-1}\mathbf{b}\|}{\|\mathbf{A}^{-1}\mathbf{b}\|} \leq \\ &\leq \frac{\|\mathbf{A}^{-1}\|\|\mathbf{A}\|\|\mathbf{A}^{-1}\mathbf{b}\|}{\|\mathbf{A}^{-1}\mathbf{b}\|} = \|\mathbf{A}^{-1}\|\|\mathbf{A}\| = \kappa(\mathbf{A}),\end{aligned} \tag{2.16}$$

unde s-au presupus doar datele $\mathbf{b}$ perturbate, iar soluţia problemei a fost scrisă formal ca $\mathbf{x} = \mathbf{f}(\mathbf{b}) = \mathbf{A}^{-1}\mathbf{b}$, având matricea Jacobian $\mathbf{J}(\mathbf{b}) = \mathbf{A}^{-1}$.

Un raţionament similar permite găsirea unei margini inferioare pentru eroarea asupra soluţiei. Din (2.8) rezultă un majorant pentru eroarea în rezultat

$$\|\mathbf{e_b}\| = \|\mathbf{A}\mathbf{e_x}\| \leq \|\mathbf{A}\|\|\mathbf{e_x}\| \quad \Rightarrow \quad \|\mathbf{e_x}\| \geq \frac{\|\mathbf{e_b}\|}{\|\mathbf{A}\|}. \tag{2.17}$$

Din (2.6) rezultă un minorant pentru norma soluţiei

$$\|\mathbf{x}\| = \|\mathbf{A}^{-1}\mathbf{b}\| \leq \|\mathbf{A}^{-1}\|\|\mathbf{b}\|. \tag{2.18}$$

Din (2.17) şi (2.18) rezultă un minorant pentru eroarea asupra soluţiei

$$\varepsilon_x = \frac{\|\mathbf{e_x}\|}{\|\mathbf{x}\|} \geq \frac{\|\mathbf{e_b}\|}{\|\mathbf{A}\|\|\mathbf{A}^{-1}\|\|\mathbf{b}\|} = \frac{\varepsilon_b}{\kappa(\mathbf{A})}. \tag{2.19}$$

În concluzie, relaţia între eroarea relativă asupra soluţiei şi eroarea relativă a termenilor liberi este dată de

$$\frac{\varepsilon_b}{\kappa(\mathbf{A})} \leq \varepsilon_x \leq \kappa(\mathbf{A})\varepsilon_b. \tag{2.20}$$

Numărul de condiţionare este întotdeauna supraunitar $\kappa(\mathbf{A}) \geq 1$, deoarece

$$1 = \|\mathbf{I}\| = \|\mathbf{A}\mathbf{A}^{-1}\| \leq \|\mathbf{A}\|\|\mathbf{A}^{-1}\| = \kappa(\mathbf{A}). \tag{2.21}$$

Dacă numărul de condiţionare $\kappa(\mathbf{A})$ are valori mici, inegalitatea (2.20) garantează că o mică perturbare a termenilor liberi nu este amplificată. În cazul cel mai favorabil,

$\kappa(\mathbf{A}) = 1$ şi $\varepsilon_x = \varepsilon_b$. Acest caz corespunde unei matrice ortogonale (în cazul $n = 2$ cele două drepte ce reprezintă ecuaţiile sistemului sunt perpendiculare).

Dacă numărul de condiţionare este mare, atunci chiar şi termeni liberi foarte puţin perturbaţi pot conduce la perturbaţii mari ale soluţiei numerice, problema fiind slab condiţionată.

Numărul de condiţionare este o proprietate a matricei şi nu are legătură nici cu metoda de rezolvare propriu-zisă, nici cu erorile de rotunjire care apar în mediul de calcul. Cu toate acestea, în practică se consideră că o problemă este slab condiţionată dacă numărul de condiţionare este mai mare decât inversul zeroului maşinii.

Dacă $\kappa(\mathbf{A}) > 1/\texttt{eps}$ problema se consideră slab condiţionată.

Demonstraţia de mai sus a fost făcută considerând o perturbaţie în vectorul termenilor liberi, matricea coeficienţilor fiind considerată neperturbată. Acelaşi număr de condiţionare este util şi dacă se consideră o perturbaţie în matricea coeficienţilor, vectorul termenilor liberi fiind neafectat de erori. Soluţia perturbată va satisface atunci relaţia

$$(\mathbf{A} + \mathbf{e_A})(\mathbf{x} + \mathbf{e_x}) = \mathbf{b}. \tag{2.22}$$

Din (2.6) şi (2.22) rezultă că perturbaţiile satisfac

$$\mathbf{A}\mathbf{e_x} = -\mathbf{e_A}(\mathbf{x} + \mathbf{e_x}), \tag{2.23}$$

de unde rezultă că perturbaţia soluţiei este mărginită superior de

$$\|\mathbf{e_x}\| = \| - \mathbf{A}^{-1}\mathbf{e_A}(\mathbf{x} + \mathbf{e_x})\| \leq \|\mathbf{A}^{-1}\|\|\mathbf{e_A}\|\|\mathbf{x} + \mathbf{e_x}\|. \tag{2.24}$$

Eroarea relativă, calculată aproximativ faţă de soluţia numerică este majorată de

$$\varepsilon_x \approx \frac{\|\mathbf{e_x}\|}{\|\mathbf{x} + \mathbf{e_x}\|} \leq \|\mathbf{A}^{-1}\|\|\mathbf{e_A}\| = \|\mathbf{A}^{-1}\|\|\mathbf{A}\|\frac{\|\mathbf{e_A}\|}{\|\mathbf{A}\|} = \kappa(\mathbf{A})\varepsilon_A. \tag{2.25}$$

Deoarece $\|\mathbf{x} + \mathbf{e_x}\| \leq \|\mathbf{x}\| + \|\mathbf{e_x}\|$, rezultă că $\|\mathbf{x}\| \geq \|\mathbf{x} + \mathbf{e_x}\| - \|\mathbf{e_x}\|$. Dacă presupunem că

$$\|\mathbf{x} + \mathbf{e_x}\| - \|\mathbf{e_x}\| > 0, \tag{2.26}$$

rezultă o majorare pentru eroarea relativă a soluţiei

$$\varepsilon_x = \frac{\|\mathbf{e_x}\|}{\|\mathbf{x}\|} \leq \frac{\|\mathbf{e_x}\|}{\|\mathbf{x} + \mathbf{e_x}\| - \|\mathbf{e_x}\|} = \frac{\|\mathbf{e_x}\|/\|\mathbf{x} + \mathbf{e_x}\|}{1 - \|\mathbf{e_x}\|/\|\mathbf{x} + \mathbf{e_x}\|} \leq \frac{\kappa(\mathbf{A})\varepsilon_A}{1 - \kappa(\mathbf{A})\varepsilon_A}, \tag{2.27}$$

relaţie valabilă în ipoteza $\kappa(\mathbf{A})\varepsilon_A < 1$.

2.3 Clasificarea metodelor

Există mai multe metode numerice posibile pentru rezolvarea cu calculatorul a sistemelor de ecuaţii algebrice liniare. Ele se pot clasifica în metode *directe* si *iterative*. La rândul lor, metodele iterative pot fi *staţionare* şi *nestaţionare* (numite şi *semiiterative*).

Metodele directe sunt metodele care găsesc soluţia teoretică a problemei într-un număr *finit* de paşi. Astfel de metode sunt metoda Gauss şi metoda factorizării LU. În realitate, calculele sunt afectate de erori de rotunjire şi la terminarea algoritmului nu se obţine soluţia exactă, ci o aproximare a ei. Ambele metode se bazează pe transformarea sistemului de ecuaţii într-unul echivalent din punct de vedere al soluţiei, mult mai uşor de rezolvat. Metodele directe sunt:

- Gauss (paragraful 2.4) - În această metodă sistemul este adus la o formă triunghiular superioară, uşor de rezolvat prin substituţie regresivă;
- Factorizare LU (paragraful 2.5) - În această metodă sistemul este adus la o formulare echivalentă, constând în două sisteme de ecuaţii uşor de rezolvat, unul triunghiular inferior, rezolvat prin substituţie progresivă, şi unul triunghiular superior, rezolvat prin substituţie regresivă.

Dezavantajul acestor metode este acela că, în anumite situaţii, efortul de generare a problemei echivalente (factorizarea) este mare sau necesarul de memorie poate deveni extrem de mare.

Metodele iterative generează un şir de aproximaţii ale soluţiei care se doreşte a fi convergent către soluţia exactă. În consecinţă, din punct de vedere teoretic, soluţia poate fi obţinută cu o astfel de metodă numai într-un număr infinit de paşi.

Metodele iterative se grupează în două categorii principale: staţionare şi nestaţionare. Metodele staţionare au apărut mai întâi, ele sunt simplu de înţeles şi de implementat dar nu sunt întotdeauna eficiente. Metodele nestaţionare au apărut mai târziu, analiza lor este mai dificilă, dar ele pot fi foarte eficiente.

Metodele staţionare se bazează pe partiţionarea matricei coeficienţilor în două şi construirea unei probleme echivalente de punct fix. Metodele staţionare sunt:

- Jacobi (paragraful 2.8) - În această metodă fiecare variabilă se calculează local, în funcţie de celelalte variabile. Metoda este uşor de înţeles şi implementat dar convergenţa ei este lentă;
- Gauss-Seidel (paragraful 2.9) - Metoda seamănă cu metoda Jacobi, dar valorile calculate sunt imediat folosite în calculele următoare. În general, dacă pentru

o problemă metoda Jacobi converge, metoda Gauss-Seidel converge mai rapid. Convergenţă este însă tot lentă.

- Metoda suprarelaxării succesive (SOR - *Succesice OverRelaxation*, paragraful 2.10)) - este dedusă din metoda Gauss-Seidel prin introducerea unui parametru de extrapolare. Pentru o valoare optimă a acestuia, SOR poate converge într-un număr de iteraţii cu un ordin de mărime mai mic decât numărul de iteraţii al metodei Gauss-Seidel.

- Metoda SOR simetrică (SSOR) - este SOR pentru matrice simetrice, utilă ca metodă de precondiţionare pentru metodele nestaţionare.

Metodele nestaţionare se bazează în general pe idea construirii unui şir de vectori ortogonali. Singura metodă care nu foloseşte această idee este metoda iteraţiilor Chebyshev. Metodele iterative nestaţionare se mai numesc si metode semiiterative deoarece, chiar dacă ele sunt metode iterative, într-o aritmetică exactă şi pentru un anumit tip de probleme, garantează obţinerea soluţiei exacte după un anumit număr de paşi. Metodele nestaţionare sunt

- Metoda gradienţilor conjugaţi (CG, paragraful 2.12) - generează un şir de vectori conjugaţi (sau ortogonali). Aceşti vectori sunt reziduurile corespunzătoare iteraţiilor. Ei sunt de asemenea vectorii gradient ai unei funcţionale pătratice asociate problemei, a cărei minimizare este echivalentă cu rezolvarea sistemului de ecuaţii. CG este o metodă foarte eficientă dacă matricea coeficienţilor este simetrică şi pozitiv definită, în acest caz fiind necesară doar memorarea unui număr limitat de vectori generaţi.

- Metoda reziduului minimin (MINRES) şi metoda LQ simetrică (SYMMLQ) - sunt alternative pentru CG atunci când matricea este simetrică dar posibil indefinită.

- CG pentru ecuaţia normală (CGNE sau CGNR). Dacă sistemul de rezolvat $\mathbf{Ax} = \mathbf{b}$ are matricea coeficienţilor $\mathbf{A}$ nesimetrică şi nesingulară, există două variante prin care putem folosi CG pentru rezolvare, ambele bazate pe faptul că $\mathbf{A}^T\mathbf{A}$ este simetrică şi pozitiv definită. Varianta CGNE este CG aplicată sistemului $(\mathbf{AA}^T)\mathbf{y} = \mathbf{b}$ urmând ca apoi $\mathbf{x} = \mathbf{A}^T\mathbf{y}$. Varianta CGNR este CG aplicată sistemului $(\mathbf{A}^T\mathbf{A})\mathbf{x} = \tilde{\mathbf{b}}$ unde $\tilde{\mathbf{b}} = \mathbf{A}^T\mathbf{b}$. Convergenţa acestor metode poate fi înceată, sistemul normal fiind mai prost condiţionat decât sistemul iniţial.

- Metoda reziduului minim generalizat (GMRES) - poate fi aplicată matricelor generale, nesimetrice. Ea calculează o secvenţă de vectori ortogonali ca şi MINRES, memorând întregul şir, necesitând astfel mai multă memorie. Din acest motiv se

practică restartarea metodei, la fiecare rulare numărul de iteraţii fiind limitat de un număr fixat de vectori care trebuie generaţi.

- Metoda gradienţilor biconjugaţi (BiCG) şi variante ale ei (CGS şi Bi-CGSTAB) - pot fi folosite pentru matrice nesingulare nesimetrice. Metoda generează două şiruri de vectori ca CG, unul pentru matricea originală a coeficienţilor $\mathbf{A}$ şi unul pentru matricea transpusă $\mathbf{A}^T$. În loc să ortogonalizeze fiecare şir, vectorii sunt construiţi mutual ("bi-") ortogonali. O îmbunătăţire a metodei, utilă atunci când BiCG nu converge, este metoda reziduului cvasiminimal (QMR).

- Metoda iteraţiilor Chebyshev - determină în mod recursiv polinoame cu coeficienţi aleşi astfel încât norma reziduului să fie minimizată în sens minimax. Se poate aplica pentru sisteme cu matricea coeficienţilor pozitiv definită şi necesită cunoaşterea valorilor proprii extreme.

Nu există o "cea mai bună metodă" dintre cele enumerate mai sus. Pentru fiecare clasă de probleme, utilizatorul trebuie să decidă ce metodă de rezolvare va alege. Ideile principale ale metodelor iterative şi criteriile după care pot fi alese aceste metode sunt prezentate în [17]. Aspecte algoritmice detaliate se găsesc în [1] iar aspecte legate de precondiţionare se găsesc de exemplu în [11].

În cele ce urmează vor fi descrise câteva metode de rezolvare din fiecare categorie, precum şi algoritmii corespunzători.

2.4 Metoda Gauss

Ideea metodei Gauss este de a rezolva sistemul de ecuaţii în două etape. Prima etapă, numită *eliminare gaussiană (triangularizare)* urmăreşte transformarea sistemului de ecuaţii astfel încât soluţia să rămână neschimbată, dar matricea coeficienţilor să aibă în final o structură specială, şi anume triunghiular superioară. A doua etapă este *retrosubstituţia (substituţia regresivă)* în care se calculează pe rând componentele vectorului soluţie, începând cu cel de indice cel mai mare. Pe scurt:

$$\mathbf{Ax} = \mathbf{b} \quad \underbrace{\Leftrightarrow}_{\text{eliminare}} \quad \mathbf{Ux} = \mathbf{b} \quad \underbrace{\Rightarrow}_{\text{subst.regresivă}} \quad \mathbf{x} = \mathbf{U}^{-1}\mathbf{b}. \tag{2.28}$$

2.4.1 Un exemplu simplu

Ideea metodei poate fi uşor înţeleasă pe un exemplu simplu. Fie sistemul de ecuaţii

$$\begin{cases} x_1 + 2x_2 - x_3 = -1 \\ -2x_1 + 3x_2 + x_3 = 0 \\ 4x_1 - x_2 - 3x_3 = -2. \end{cases} \tag{2.29}$$

Matricea coeficienţilor se va triangulariza în două sub-etape de eliminare. Mai întâi se vor anula coeficienţii de pe prima coloană, aflaţi sub prima linie, după cum urmează. Pentru a anula primul coeficient din a doua ecuaţie se va înmulţi prima ecuaţie cu 2 şi se va aduna la a doua. Apoi se va anula primul coeficient din a treia ecuaţie înmulţind prima ecuaţie cu -4 şi se va aduna la a treia. Valorile 2 şi -4 se numesc *multiplicatori*. Sistemul rezultat, echivalent din punct de vedere al soluţiei cu cel iniţial este

$$\begin{cases} x_1 + 2x_2 - x_3 = -1 \\ 7x_2 - x_3 = -2 \\ -9x_2 + x_3 = 2. \end{cases} \tag{2.30}$$

În a doua etapă de eliminare nu trebuie să se anuleze decât coeficientul -9 din ecuaţia a treia, şi pentru aceasta se înmulţeşte ecuaţia a doua cu multiplicatorul $9/7$ şi se adună la ecuaţia a treia. Sistemul rezultat după etapa de eliminare este unul superior triunghiular:

$$\begin{cases} x_1 + 2x_2 - x_3 = -1 \\ 7x_2 - x_3 = -2 \\ -2/7x_3 = -4/7. \end{cases} \tag{2.31}$$

Soluţia se calculează uşor, pornind de la ultima ecuaţie, către prima (retrosubstituţie), astfel:

$$\begin{aligned} x_3 &= (-4/7)/(-2/7) = 2, \\ x_2 &= (-2 + x_3)/7 = 0, \\ x_1 &= -1 - 2x_2 + x_3 = 1. \end{aligned} \tag{2.32}$$

2.4.2 Algoritm

Etapa de eliminare urmăreşte aducerea matricei coeficienţilor la forma triunghiular superioară. Mai precis, coeficienţii aflaţi sub diagonală sunt anulaţi în sub-etape de eliminare, sugerate în figura 2.2.

În prima sub-etapă de eliminare se anulează toţi coeficienţii a_{i1} unde $i = 2, \ldots, n$, astfel: se înmulţeşte prima ecuaţie cu elementul de multiplicare $-a_{21}/a_{11}$ şi se adună la a doua ecuaţie, se înmulţeşte prima ecuaţie cu elementul de multiplicare $-a_{31}/a_{11}$ şi se

adună la a treia ecuaţie, ş.a.m.d. La sfârşitul acestei prime sub-etape noul sistem de rezolvat, echivalent cu cel iniţial din punct de vedere al soluţiei, este:

$$\begin{cases} a_{11}x_1 + & a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ & a'_{22}x_2 + \cdots + a'_{2n}x_n = b'_2 \\ & \cdots \\ & a'_{n2}x_2 + \cdots + a'_{nn}x_n = b'_n, \end{cases} \tag{2.33}$$

unde, de exemplu, noii coeficienţi ai celei de a doua ecuaţii sunt

$$a'_{2j} = a_{2j} + pa_{1j}, \tag{2.34}$$

unde $p = -a_{21}/a_{11}$ este primul multiplicator folosit pentru anularea coeficientului a_{21}, iar noul termen liber este

$$b'_2 = b_2 + pb_1. \tag{2.35}$$

În a doua sub-etapă de eliminare se urmăreşte anularea coeficienţilor matricei aflaţi pe coloana a doua, sub diagonală. Pentru aceasta se înmulţeşte ecuaţia a doua cu elementul de multiplicare $p = -a'_{32}/a'_{22}$ şi se adună la ecuaţia a treia, apoi elementul de multiplicare este $p = -a'_{42}/a'_{22}$, ş.a.m.d.

În total sunt $n-1$ sub-etape de eliminare la sfârşitul cărora sistemul echivalent obţinut este de forma

$$\begin{cases} a_{11}x_1 + & a_{12}x_2 + \cdots + & a_{1n}x_n = b_1 \\ & a_{22}x_2 + \cdots + & a_{2n}x_n = b_2 \\ & \cdots & \\ & & a_{nn}x_n = b_n. \end{cases} \tag{2.36}$$

În mod intenţionat în sistemul (2.36) au fost omise notaţiile de tip "prim" deoarece operaţiile se fac "în loc", peste valorile vechi sunt scrise valorile noi, adică relaţiile de tipul (2.34) se vor implementa ca

$$a_{2j} = a_{2j} + pa_{1j}, \tag{2.37}$$

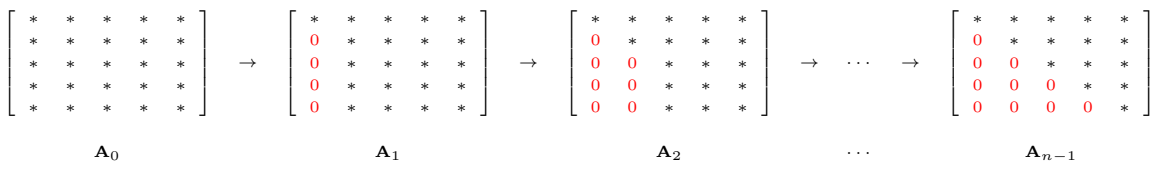


Figura 2.2: Eliminare în metoda Gauss: pentru un sistem de dimensiune n există $n-1$ sub-etape de eliminare. În final matricea este superior triunghiulară. Matricea iniţială este notată $\mathbf{A}_0$ iar matricea superior triunghiulară obţinută este notată $\mathbf{A}_{n-1}$. În realitate, transformările sunt memorate "în loc", în acelaşi tablou bidimensional.

unde simbolul = reprezintă acum o atribuire.

Vom construi acum pseudocodul etapei de eliminare. Modificarea ecuaţiei a doua în prima sub-etapă de eliminare poate fi descrisă astfel:

; anularea elementului a_{21}
$p = -a_{21}/a_{11}$; element de multiplicare
pentru $j = 1, n$; parcurge coloanele
$a_{2j} = a_{2j} + pa_{1j}$
•
$b_2 = b_2 + pb_1$

Dacă în secvenţa de mai sus înlocuim 2 cu i atunci se obţine secvenţa ce permite anularea elemenului a_{i1}. Inserată într-un ciclu cu contor în care i parcurge liniile de la 2 la n, se obţine secvenţa ce descrie prima sub-etapă de eliminare:

; prima sub-etapă de eliminare
pentru $i = 2, n$; parcurge liniile
$p = -a_{i1}/a_{11}$; element de multiplicare
pentru $j = 2, n$; parcurge coloanele
$a_{ij} = a_{ij} + pa_{1j}$
•
$b_i = b_i + pb_1$
•

În ciclul în j contorul începe cu valoarea 2 pentru a evita calcule inutile pentru valorile a_{i1}, al căror rezultat este nul. Aceasta însemnă că, de fapt, zerourile nu sunt memorate în matricea $\mathbf{A}$. În triunghiul inferior vor rămâne valorile avute exact înainte de anulare.

Dacă în pseudocodul de mai sus se înlocuieşte 1 cu k şi 2 cu $k + 1$ se obţine secvenţa de cod corespunzătoare sub-etapei de eliminare k. Aceasta, repetată pentru cele $n - 1$ valori posibile conduce la următorul pseudocod pentru etapa de eliminare:

; etapa de eliminare din metoda Gauss
pentru $k = 1, n - 1$; parcurge sub-etape ale eliminării
pentru $i = k + 1, n$; parcurge liniile
$p = -a_{ik}/a_{kk}$; element de multiplicare
pentru $j = k + 1, n$; parcurge coloanele
$a_{ij} = a_{ij} + pa_{kj}$
•
$b_i = b_i + pb_k$

•

•

În *etapa de substituţie regresivă* sistemul triunghiular obţinut (2.36) se rezolvă uşor, în ordinea descrescătoare a indicelui componentelor vectorului soluţie. Din ultima ecuaţie rezultă

$$x_n = b_n/a_{nn}, \tag{2.38}$$

iar apoi se calculează pe rând x_{n-1} din penultima ecuaţie, x_{n-2} din antepenultima ş.a.m.d.

În general, din ecuaţia i

$$a_{ii}x_i + a_{i,i+1}x_{i+1} + \cdots + a_{in}x_n = b_i, \tag{2.39}$$

se calculează componenta x_i:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}}. \tag{2.40}$$

Algoritmul etapei de substituţie regresivă este descris de următorul pseudocod:

```
; etapa de retrosubstituţie
x_n = b_n/a_nn
pentru i = n − 1, 1, −1
        s = 0
        pentru j = i + 1, n
                s = s + a_ij x_j
        •
        x_i = (b_i − s)/a_ii
•
```

Algoritmul complet al procedurii Gauss este următorul.

```
procedură Gauss(n, a, b, x)
; rezolvă sistemul algebric liniar ax = b prin metoda Gauss
întreg n                          ; dimensiunea sistemului
tablou real a[n][n]               ; matricea coeficienţilor - indici de la 1
tablou real b[n]                  ; vectorul termenilor liberi
tablou real x[n]                  ; vectorul soluţie
întreg i, j, k
real p, s
; etapa de eliminare
pentru k = 1, n − 1               ; parcurge sub-etape ale eliminării
```

; aici se poate introduce pivotarea
pentru $i = k+1, n$; parcurge liniile
$p = -a_{ik}/a_{kk}$; element de multiplicare
pentru $j = k+1, n$; parcurge coloanele
$a_{ij} = a_{ij} + pa_{kj}$
•
$b_i = b_i + pb_k$
•
•
; etapa de retrosubstituţie
$x_n = b_n/a_{nn}$
pentru $i = n-1, 1, -1$
$s = 0$
pentru $j = i+1, n$
$s = s + a_{ij}x_j$
•
$x_i = (b_i - s)/a_{ii}$
•
retur

Algoritmul poate fi îmbunătăţit prin folosirea unei strategii de pivotare la fiecare etapă de eliminare. Detalii despre pivotare sunt prezentate în paragraful 2.4.4.

2.4.3 Evaluarea algoritmului

Algoritmul prezentat anterior va fi analizat acum conform criteriilor prezentate în capitolul 1.

Din punct de vedere al timpului de calcul, complexitatea este cubică deoarece în etapa de eliminare există trei cicluri imbricate. Într-adevăr, considerând ca operaţii elementare orice operaţii algebrice (adunări, scăderi, înmulţiri, împărţiri), rezultă că în etapa de eliminare se execută un număr de operaţii egal cu

$$T_e = \sum_{k=1}^{n-1}[2(n-k)+3](n-k) \approx \sum_{k=1}^{n-1} 2(n-k)^2 = 2\frac{(n-1)n(2n-1)}{6} \approx \frac{2n^3}{3}. \qquad (2.41)$$

În etapa de substituţie se execută un număr de operaţii egal cu

$$T_s = \sum_{i=1}^{n-1}[2(n-i)+2] \approx \sum_{i=1}^{n-1}[2(n-i)] = 2\frac{n(n-1)}{2} \approx n^2. \qquad (2.42)$$

Etapa de eliminare a algoritmului Gauss este cea mai costisitoare din punct de vedere al timpului de calcul, complexitatea ei fiind $T_e = \mathrm{O}(2n^3/3)$, cu un ordin de mărime mai mare decât complexitatea etapei de substituţie regresivă care este $T_s = \mathrm{O}(n^2)$. În consecinţă, complexitatea algoritmului Gauss este cubică, dată de etapa de eliminare $T_{\text{Gauss}} = \mathrm{O}(2n^3/3)$.

Din punct de vedere al necesarului de memorie, considerând locaţia ocupată de un număr real ca locaţie elementară de memorie, rezultă (din inspectarea declaraţiilor din pseudocod) că este nevoie de $n^2 + 2n + 2$ locaţii de memorie. În consecinţă $M = \mathrm{O}(n^2)$, algoritmul este pătratic din punct de vedere al necesarului de memorie.

Algoritmul Gauss prezentat în acest paragraf este un algoritm costisitor, el poate fi folosit până la valori de cel mult 1000 ale dimensiunii sistemului, foarte puţin pentru necesităţile problemelor reale. El poate deveni un algoritm eficient dacă este adaptat unor matrice cu structuri speciale (rare), caz analizat în paragraful 2.6.

Din punct de vedere al erorilor, în algoritmul Gauss apar, pe lângă erorile inerente, numai erori de rotunjire. Cu cât sistemul este de dimensiune mai mare, cu atât cresc numărul de operaţii efectuate şi erorile acumulate datorită rotunjirii. O diminuare a erorilor de rotunjire se poate obţine dacă se includ în algoritm strategii de pivotare.

Din punct de vedere al stabilităţii, algoritmul Gauss poate să nu fie stabil chiar dacă problema matematică este bine formulată şi bine condiţionată (numărul de condiţionare al matricei **A** este mic). Acest lucru se întâmplă atunci când numărul de condiţionare al matricei **U** este mare. Remediul îl constituie în acest caz pivotarea.

2.4.4 Strategii de pivotare

Elementele diagonale a_{kk} obţinute în urma etapei de eliminare se numesc pivoţi.

Deoarece transformările elementare făcute nu modifică determinantul sistemului şi deoarece determinanul unei matrice tridiagonale este egal cu produsul elementelor de pe diagonală, înseamnă că determinatul matricei este egal cu produsul pivoţilor. Dacă problema este bine formulată matematic, atunci determinantul matricei coeficienţilor este diferit de zero, ceea ce înseamnă că toţi pivoţii sunt nenuli.

Pe parcursul algoritmului Gauss prezentat anterior, elementele de multiplicare se calculează ca $p = -a_{ik}/a_{kk}$. Valoare a_{kk} folosită pentru calculul elementului de multiplicare este un pivot, ea rămânând în diagonala matricei triunghiulare. Se poate întâmpla însă ca pivotul să fie nul chiar dacă problema este bine formulată matematic, caz în care algoritmul Gauss prezentat anterior eşuează. În acest caz se poate face o operaţie de permutare (de linii sau/şi de coloane), astfel încât pe poziţia kk să fie adus un element nenul.

Pivotarea este o operaţie de permutare care urmăreşte obţinerea unor valori nenule pentru pivoţi. Ea este o operaţie care trebuie făcută înainte de calculul multiplicatorului.

Există mai multe strategii de pivotare, cele mai des întâlnite fiind: pivotarea pe linii, pe coloane, totală, diagonală.

Pivotarea pe linii (parţială) constă în schimbarea a două linii între ele (linia k şi o linie aflata sub ea), ceea ce este echivalent cu scrierea ecuaţiilor în altă ordine. Este cea mai simplă strategie de pivotare. Pivotul este căutat numai pe pe coloana de sub elementul diagonal.

Pivotarea pe coloane constă în schimbarea a două coloane între ele (coloana k şi o coloană aflată la dreapta ei), ceea ce este echivalent cu renumerotarea variabilelor. Pivotul este căutat numai pe linie, la dreapta elementului diagonal. Permutările făcute trebuie memorate deoarece la finalul algoritmului soluţia trebuie reconstituită în ordinea iniţială.

Pivotarea totală (completă sau maximală) caută pivotul şi pe linii şi pe coloane, sub şi la dreapta elementului diagonal. După aflarea lui vor fi permutate şi două linii şi două coloane. Algoritmul acestei permutări este cel mai costisitor şi nu neaparat cel mai eficient.

Pivotarea diagonală caută pivotul pe diagonală. Vor fi permutate apoi două linii şi două coloane. O astfel de pivotare poate fi utilă în cazul în care matricea este simetrică.

Pseudocodul strategiei de pivotare pe linii este următorul:

$p = 0$
<u>pentru</u> $i = k, n$; parcurge coloana k, de la elementul diagonal în jos
    <u>dacă</u> $|a_{ik}| > p$ <u>atunci</u>
        $l = i$; memorează poziţia potenţialului pivot
        $p = |a_{ik}|$
    •
•
<u>dacă</u> $p = 0$ <u>atunci</u>

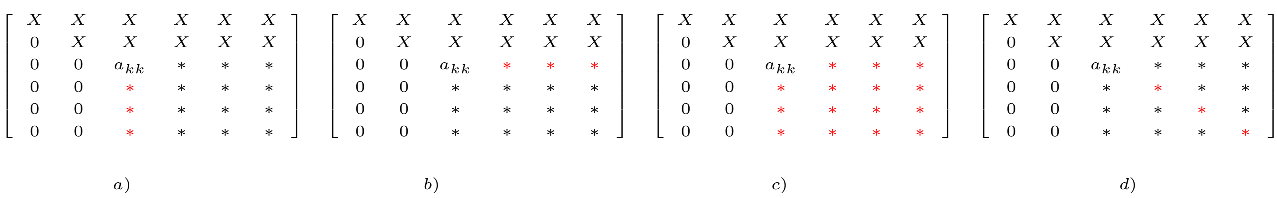


Figura 2.3: Zona de căutare a pivotului în pivotarea pe linii (a), pe coloane (b), totală (c), diagonală (d). Cu X sunt marcate elementele nenule ale căror valori nu se vor mai modifica.

```
    scrie "problema este prost formulată matematic"
altfel
    pentru j = k, n      ; permută linia l cu linia k
        p = a_kj
        a_kj = a_lj
        a_lj = p
    •
    p = b_k              ; permută termenii liberi
    b_k = b_l
    b_l = p
•
```

Pivotarea este absolut necesară dacă pe parcursul algoritmului Gauss se întâlneşte un pivot nul. Ea are însă şi un efect benefic asupra stabilităţii şi acurateţii soluţiei. Pentru a ilustra această afirmaţie, să reluăm exemplul discutat la paragraful 1.3.6.

$$\begin{cases} 10^{-20}x + y &= 1 \\ x + y &= 0, \end{cases} \tag{2.43}$$

având soluţia corectă $(x, y) \approx (-1, 1)$. Dupa eliminarea Gauss, sistemul triunghiular devine

$$\begin{cases} 10^{-20}x + y &= 1 \\ (1 - 10^{20})y &= -10^{20}. \end{cases} \tag{2.44}$$

Presupunând că zeroul maşinii este de ordinul 10^{-16} rezultă că sistemul ce se va rezolva la retrosubstituţie va fi

$$\begin{cases} 10^{-20}x + y &= 1 \\ 10^{20}y &= -10^{20}. \end{cases} \tag{2.45}$$

Rezultatul final va fi $(x, y) = (0, 1)$, foarte departe de soluţia adevărată. Deşi sistemul iniţial are matricea coeficienţilor bine condiţionată $\kappa(\mathbf{A}) \approx 2.6$, matricea $\mathbf{U}$ este extrem de prost condiţionată $\kappa(\mathbf{U}) = 10^{40}$, ceea ce duce la eşecul rezolvării, în ciuda efectuării cu acurateţe a etapei de eliminare. Acest sistem nu este stabil la retrosubstituţie şi în consecinţă, întreaga rezolvare este compromisă. Deşi problema este bine condiţionată, algoritmul fără pivotare nu este stabil.

Aplicarea pivotării este o etapă esenţială în rezolvarea directă a sistemelor de ecuaţii algebrice liniare. Pivotarea parţială asigura stabilitatea procedurii, cu un efort de implementare nesemnificativ. Pivotarea totală este rareori aplicată în practică deoarece căutarea pivoţilor şi pe linii şi pe coloane duce la o creştere semnificativă a timpului de calcul, nerealizând decât o îmbunătăţire nesemnificativă a acurateţii soluţiei.

2.4.5 Cazul sistemelor multiple

De multe ori este necesar să se rezolve mai multe sisteme de ecuaţii algebrice liniare care au aceeaşi matrice a coeficienţilor, dar termeni liberi diferiţi. Un asfel de caz este cel al rezolvării circuitelor rezistive neliniare, în care pentru rezolvare se foloseşte metoda tangentelor paralele.

Fie m sisteme de ecuaţii algebrice liniare

$$\mathbf{A}\mathbf{x}^{(1)} = \mathbf{b}^{(1)}, \quad \mathbf{A}\mathbf{x}^{(2)} = \mathbf{b}^{(2)}, \quad \cdots, \quad \mathbf{A}\mathbf{x}^{(m)} = \mathbf{b}^{(m)}, \tag{2.46}$$

care au aceeaşi matrice a coeficienţilor $\mathbf{A} \in \mathbb{R}^{n\times n}$ dar termeni liberi diferiţi $\mathbf{b}^{(k)} \in \mathbb{R}^{n\times 1}$, unde $k = 1, m$. Se doreşte rezolvarea celor m sisteme

$$\mathbf{A}\mathbf{x}^{(k)} = \mathbf{b}^{(k)}, \quad k = 1, \ldots, m, \tag{2.47}$$

adică găsirea soluţiilor $\mathbf{x}^{(k)} \in \mathbb{R}^{n\times 1}$, în condiţiile în care se cunoaşte matricea $\mathbf{A}$ şi termenii liberi $\mathbf{b}^{(k)}$.

Dacă notăm cu

$$\mathbf{B} = [\mathbf{b}^{(1)} \quad \mathbf{b}^{(2)} \quad \cdots \quad \mathbf{b}^{(m)}] \quad \in \mathbb{R}^{n\times m} \tag{2.48}$$

matricea dreptunghiulară a termenilor liberi şi cu

$$\mathbf{X} = [\mathbf{x}^{(1)} \quad \mathbf{x}^{(2)} \quad \cdots \quad \mathbf{x}^{(m)}] \quad \in \mathbb{R}^{n\times m} \tag{2.49}$$

matricea dreptunghiulară a necunoscutelor, atunci relaţiile (2.47) se scriu compact

$$\mathbf{A}\mathbf{X} = \mathbf{B}. \tag{2.50}$$

Varianta I - aplicarea succesivă a algoritmului Gauss

Cea mai proasta idee de a rezolva o astfel de problemă este de a apela de m ori succesiv algoritmul Gauss. Efortul total de calcul ar fi $m(2n^3/3+n^2) \approx 2mn^3/3$. Deoarece matricea coeficienţilor este aceeaşi, etapa de eliminare este repetată inutil, de m ori.

Varianta II - adaptarea algoritmului Gauss pentru *rezolvarea simultană* a sistemelor

Presupunând că în momentul rezolvării toţi termenii liberi sunt cunoscuţi, atunci putem modifica algoritmul Gauss, astfel încât la eliminare să se prelucreze toţi termenii liberi, iar la retrosubstituţie, să se calculeze "simultan" componentele tuturor soluţiilor. Algoritmul complet al unei astfel de proceduri este următorul.

`procedură` Gauss_multiplu(n, m, a, B, X)
; rezolvă sismultan sistemele algebrice liniare $aX = B$ prin metoda Gauss

```
întreg n                          ; dimensiunea sistemului
întreg m                          ; numărul de sisteme
tablou real a[n][n]               ; matricea coeficienţilor - indici de la 1
tablou real B[n][m]               ; matricea termenilor liberi
tablou real X[n][m]               ; matricea soluţie
întreg i, j, k
real p, s
; etapa de eliminare
pentru k = 1, n − 1               ; parcurge sub-etape ale eliminării
    ; aici se poate introduce pivotarea
    pentru i = k + 1, n           ; parcurge liniile
        p = −a_ik/a_kk            ; element de multiplicare
        pentru j = k + 1, n       ; parcurge coloanele
            a_ij = a_ij + p a_kj
        •
        pentru j = 1, m           ; parcurge coloanele termenilor liberi
            b_ij = b_ij + p b_kj
        •
    •
•
; etapa de retrosubstituţie
pentru k = 1, m
    x_nk = b_nk/a_nn
    pentru i = n − 1, 1, −1
        s = 0
        pentru j = i + 1, n
            s = s + a_ij x_jk
        •
        x_ik = (b_ik − s)/a_ii
    •
•
retur
```

În etapa de eliminare se face acum un număr de operaţii egal cu

$$\begin{aligned} T_e &= \sum_{k=1}^{n-1}[2(n-k)+2m+1](n-k) \approx \sum_{k=1}^{n-1}[2(n-k)^2+2m(n-k)] = \\ &= 2\frac{(n-1)n(2n-1)}{6} + 2m\frac{n(n-1)}{2} \approx \frac{2n^3}{3} + mn^2. \end{aligned} \tag{2.51}$$

În etapa de substituţie se face un număr de operaţii egal cu

$$T_s = m\sum_{i=1}^{n-1}[2(n-i)+2] \approx m\sum_{i=1}^{n-1}[2(n-i)] = 2m\frac{n(n-1)}{2} \approx mn^2. \tag{2.52}$$

În consecinţă, efortul total de calcul în aceasta variantă este de ordinul $O(2n^3/3 + 2mn^2)$, mai mic decât în cazul primei variante.

Varianta III - *rezolvarea succesivă* a sistemelor folosind calculul inversei

Dacă termenii liberi nu sunt cunoscuţi simultan, atunci metoda anterioară nu se poate aplica. O nouă variantă ar fi calculul inversei $\mathbf{A}^{-1}$ şi aflarea fiecărei soluţii $\mathbf{x}^{(k)} = \mathbf{A}^{-1}\mathbf{b}^{(k)}$ imediat ce este cunoscut termenul liber.

Calculul inversei se poate face folosind procedura Gauss pentru sisteme multiple rezolvate simultan, în care matricea termenilor liberi este chiar matricea unitate. Dacă $\mathbf{B} = \mathbf{I}$, atunci soluţia sistemului $\mathbf{AX} = \mathbf{B}$ va fi chiar $\mathbf{X} = \mathbf{A}^{-1}$. Aceasta idee este descrisă de următorul pseudocod.

```
funcţie invA(n, a)
; calculează inversa matricei a
întreg n                          ; dimensiunea matricei
tablou real a[n][n]               ; matricea, indici de la 1
; alte declaraţii
....
pentru i = 1, n
      pentru j = 1, n
            B_ij = 0
      •
      B_ii = 1
•
Gauss_multiplu(n, n, a, B, X)
întoarce X                        ; X este inversa matricei
```

Complexitatea acestui calcul este obţinută din complexitatea procedurii Gauss multiplu, pentru cazul particular $n = m$, deci $2n^3/3 + 2mn^2 = 8n^3/3$.

Calculul inversei este o operaţie foarte costisitoare din punct de vedere al timpului de calcul.

În final, pentru calculul fiecărei soluţii, matricea inversă trebuie înmulţită cu vectorul termenilor liberi, algoritmul acestei înmulţiri având o complexitate de ordinul $2n^2$, aşa cum poate fi estimat cu uşurinţă din pseudocodul următor:

```
funcţie produs_Mv (n, M, v)
; calculează produsul dintre o matrice pătrată M şi un vector coloană v
întreg n                          ; dimensiunea problemei
tablou real M[n][n]               ; matricea, indici de la 1
tablou real v[n]                  ; vectorul
tablou real p[n]                  ; rezultatul p = Mv
; alte declaraţii
....
pentru i = 1, n
      p_i = 0
      pentru j = 1, n
            p_i = p_i + M_ij v_j
      •
•
întoarce p
```

Efortul total de calcul la rezolvarea succesivă a m sisteme algebrice liniare folosind calculul inversei, va fi $O(8n^3/3+2mn^2)$. Efortul de calcul este tot de ordin cubic, dar mai mare decât la rezolvarea simultană a sistemelor. Există însă şi o variantă mai eficientă de rezolvare succesivă a sistemelor algebrice. Ea este bazată însă nu pe calculul inversei ci pe factorizarea matricei coeficienţilor.

2.5 Metoda factorizării LU

Metoda factorizării LU (*Lower-Upper*) este o metodă directă de rezolvare a unui sistem algebric liniar de dimensiune n

$$\mathbf{Ax} = \mathbf{b}, \tag{2.53}$$

bazată pe descompunerea matricei coeficienţilor într-un produs de două matrice

$$\mathbf{A} = \mathbf{LU}, \tag{2.54}$$

unde $\mathbf{L}$ este o matrice inferior triunghiulară iar $\mathbf{U}$ este o matrice superior triunghiulară (fig. 2.4).

Sistemul de rezolvat (2.53) este echivalent cu

$$\mathbf{LUx} = \mathbf{b}. \tag{2.55}$$

Dacă se notează

$$\mathbf{y} = \mathbf{Ux}, \tag{2.56}$$

$$\mathbf{A} = \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} * & 0 & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 & 0 \\ * & * & * & 0 & 0 & 0 \\ * & * & * & * & 0 & 0 \\ * & * & * & * & * & 0 \\ * & * & * & * & * & * \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & 0 & * \end{bmatrix}$$

Figura 2.4: Structura termenilor în descompunerea $\mathbf{A} = \mathbf{LU}$. Cu $*$ sunt marcate elementele nenule.

atunci rezolvarea sistemului (2.55) este echivalentă cu rezolvarea succesivă a sistemelor

$$\begin{aligned} \mathbf{Ly} &= \mathbf{b}, \\ \mathbf{Ux} &= \mathbf{y}. \end{aligned} \tag{2.57}$$

Rezolvarea sistemelor (2.57) va fi simplă deoarece ele au o structură particulară iar procedura de rezolvare va fi adaptată acestei structuri.

Pe scurt, rezolvarea sistemului se face prin *factorizarea* matricei coeficienţilor, urmată de rezolvarea a două sisteme de ecuaţii cu structuri particulare, prin *substituţie progresivă şi regresivă*.

2.5.1 Un exemplu simplu

Cea mai simplă metodă de factorizare provine din algoritmul Gauss. Matricea $\mathbf{U}$ este chiar matricea coeficienţilor obţinută la sfârşitul etapei de eliminare al algoritmului Gauss. Pentru exemplul prezentat în paragraful 2.4.1, matricea $\mathbf{U}$ este

$$\mathbf{U} = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 7 & -1 \\ 0 & 0 & -2/7 \end{bmatrix}. \tag{2.58}$$

Matricea $\mathbf{L}$ are 1 pe diagonală, iar elementele nenule se obţin din ultimele valori nenule ale elementelor anulate în Gauss (-2 şi 4 în relaţia (2.29) şi -9 în relaţia (2.30)), împărţite la pivoţi:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -2/1 & 1 & 0 \\ 4/1 & -9/7 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 4 & -9/7 & 1 \end{bmatrix}. \tag{2.59}$$

Se verifică uşor că $\mathbf{LU} = \mathbf{A}$, unde

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -1 \\ -2 & 3 & 1 \\ 4 & -1 & -3 \end{bmatrix}. \tag{2.60}$$

Rezolvarea sistemului $\mathbf{Ly} = \mathbf{b}$ devine

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 4 & -9/7 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ -2 \end{bmatrix}, \tag{2.61}$$

adică

$$\begin{cases} y_1 & & & = & -1 \\ -2y_1 & + & y_2 & = & 0 \\ 4y_1 & - & 9/7y_2 + y_3 & = & -2 \end{cases} \tag{2.62}$$

de unde

$$\begin{aligned} y_1 &= -1 \\ y_2 &= 2y_1 = -2 \\ y_3 &= -2 - 4y_1 + 9/7y_2 = -4/7. \end{aligned} \tag{2.63}$$

Rezolvarea sistemului $\mathbf{Ux} = \mathbf{y}$ devine

$$\begin{bmatrix} 1 & 2 & -1 \\ 0 & 7 & -1 \\ 0 & 0 & -2/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ -4/7 \end{bmatrix}, \tag{2.64}$$

adică

$$\begin{cases} x_1 + 2x_2 - x_3 = -1 \\ 7x_2 - x_3 = -2 \\ -2/7x_3 = -4/7. \end{cases} \tag{2.65}$$

de unde

$$\begin{aligned} x_3 &= (-4/7)/(-2/7) = 2, \\ x_2 &= (-2 + x_3)/7 = 0, \\ x_1 &= -1 - 2x_2 + x_3 = 1. \end{aligned} \tag{2.66}$$

2.5.2 Variante de factorizare

Nu există o descompunere unică a matricei $\mathbf{A}$ în factorii $\mathbf{L}$ şi $\mathbf{U}$. Variantele considerate standard sunt: *Doolittle*, în care termenii diagonali ai matricei $\mathbf{L}$ sunt unitari ($l_{ii} = 1$), *Crout*, în care termenii diagonali ai matricei $\mathbf{U}$ sunt unitari ($u_{ii} = 1$) şi *Cholesky* în care factorii sunt transpuşi unul altuia ($\mathbf{L} = \mathbf{U}^T$).

Variantele Doolittle şi Crout se aplică la orice matrice nesingulară, iar varianta Cholesky se aplică doar matricelor simetrice şi pozitiv definite, la care se salvează simetria şi după factorizare.

Pentru a exemplifica aceste variante, să considerăm cazul descompunerii în factori a unei matrice de dimeniune 2:

$$\begin{bmatrix} 3 & 2 \\ 6 & 1 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}. \tag{2.67}$$

Relaţiile între componentele celor doi factori sunt

$$\begin{cases} l_{11}u_{11} = 3 \\ l_{11}u_{12} = 2 \\ l_{21}u_{11} = 6 \\ l_{21}u_{12} + l_{22}u_{22} = 1. \end{cases} \tag{2.68}$$

Un astfel de sistem este nedeterminat. Dar, dacă fixăm oricare două valori, restul de valori vor rezulta în mod unic. De aceea, pentru a găsi o factorizare LU unică, este necesar să se stabilească nişte restricţii suplimentare pentru matricele-factori. De exemplu, dacă alegem $l_{11} = l_{22} = 1$ obţinem o soluţie posibilă (numită varianta Doolittle), iar dacă alegem $u_{11} = u_{22} = 1$ obţinem o altă soluţie posibilă (numită varianta Crout). În cazul exemplului de mai sus

$$\begin{bmatrix} 3 & 2 \\ 6 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 0 & -3 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 6 & -3 \end{bmatrix} \begin{bmatrix} 1 & 2/3 \\ 0 & 1 \end{bmatrix}. \tag{2.69}$$

Pentru o matrice simetrică şi pozitiv definită se poate impune condiţia de păstrare a simetriei după factorizare, ca de exemplu

$$\begin{bmatrix} 9 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 2/3 & \sqrt{5}/3 \end{bmatrix} \begin{bmatrix} 3 & 2/3 \\ 0 & \sqrt{5}/3 \end{bmatrix}. \tag{2.70}$$

2.5.3 Algoritmul variantei Doolittle

În cele ce urmează vom demonstra că algoritmul Gauss, extrem de puţin modificat, generează şi factorizarea LU a matricei coeficienţilor, în varianta Doolittle.

Diferitele etape prin care trece matricea $\mathbf{A}$ în algoritmul Gauss sunt etape de transformare liniară, respectiv de înmulţire cu o matrice. Mai precis, dacă notăm matricea iniţială cu

$$\mathbf{A}_0 = \mathbf{A}, \tag{2.71}$$

atunci, după prima sub-etapă de eliminare se obţine matricea $\mathbf{A}_1$, care are zerouri pe prima coloană, sub diagonală (fig. 2.2). Această etapă de eliminare este echivalentă, din punct de vedere algebric, cu înmulţirea la stânga a matricei iniţiale $\mathbf{A}$ cu o matrice elementară notată $\mathbf{E}_1$. Similar, efectuarea celei de a doua etape de eliminare este echivalentă cu înmuţirea cu o matrice elementară $\mathbf{E}_2$ ş.a.m.d:

$$\begin{aligned} &\mathbf{A}_1 = \mathbf{E}_1\mathbf{A}_0, \\ &\mathbf{A}_2 = \mathbf{E}_2\mathbf{A}_1 = \mathbf{E}_2\mathbf{E}_1\mathbf{A}_0, \\ &\dots \\ &\mathbf{A}_{n-1} = \mathbf{E}_{n-1}\mathbf{A}_{n-2} = \mathbf{E}_{n-1}\mathbf{E}_{n-2}\cdots\mathbf{E}_2\mathbf{E}_1\mathbf{A}_0. \end{aligned} \tag{2.72}$$

Matricea obţinută după ultima etapă de eliminare este superior triunghiulară, deci putem nota

$$\mathbf{U} = \mathbf{A}_{n-1}. \tag{2.73}$$

Dacă notăm produsul matricelor elementare de eliminare cu

$$\mathbf{E} = \mathbf{E}_{n-1}\mathbf{E}_{n-2}\cdots\mathbf{E}_2\mathbf{E}_1, \tag{2.74}$$

atunci relaţia (2.72) se scrie

$$\mathbf{U} = \mathbf{E}\mathbf{A}. \tag{2.75}$$

Pentru finalizarea demonstraţiei, vom arăta că matricea $\mathbf{E}$ este nesingulară şi, mai mult, inversa ei este triunghiular inferioară şi, în consecinţă,

$$\mathbf{L} = \mathbf{E}^{-1}. \tag{2.76}$$

Pentru acesta trebuie să analizăm structura matricelor de eliminare. Vom scrie relaţiile pentru cazul $n = 3$, dar generalizarea este imediată.

Matricea elementară $\mathbf{E}_1$ este cea care realizează prima sub-etapă de eliminare, deci

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & a'_{32} & a'_{33} \end{bmatrix} = \mathbf{E}_1 \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}. \tag{2.77}$$

Se deduce uşor că $\mathbf{E}_1$ are 1 pe diagonală, iar pe prima coloană, sub diagonală, sunt exact elementele de multiplicare folosite pentru anularea coeficienţilor:

$$\mathbf{E}_1 = \begin{bmatrix} 1 & 0 & 0 \\ -a_{21}/a_{11} & 1 & 0 \\ -a_{31}/a_{11} & 0 & 1 \end{bmatrix}. \tag{2.78}$$

În general, o matrice de eliminare $\mathbf{E}_k$ este inferior triunghiulară, are 1 pe diagonală, iar pe coloana k, sub diagonală, sunt exact elementele de multiplicare folosite pentru anularea coeficienţilor în sub-etapa k de eliminare. O astfel de matrice este inversabilă, iar inversa ei are 1 pe diagonală, iar pe coloana k, sub diagonală, sunt exact opusele elementelor de multiplicare folosite pentru anularea coeficienţilor în etapa k de eliminare. De exemplu

$$\mathbf{E}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ a_{21}/a_{11} & 1 & 0 \\ a_{31}/a_{11} & 0 & 1 \end{bmatrix}, \quad \mathbf{E}_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & a'_{32}/a'_{22} & 1 \end{bmatrix}. \tag{2.79}$$

În consecinţă, matricea $\mathbf{E}$ este inversabilă, iar inversa ei este

$$\mathbf{E}^{-1} = \mathbf{E}_1^{-1}\mathbf{E}_2^{-1}\cdots\mathbf{E}_{n-2}^{-1}\mathbf{E}_{n-1}^{-1}. \tag{2.80}$$

Prin înmulţire, se obţine o matrice care are 1 pe diagonală, iar sub fiecare diagonală k sunt opusele elementelor de multiplicare folosite în etapa k de eliminare:

$$\mathbf{E}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ a_{21}/a_{11} & 1 & 0 \\ a_{31}/a_{11} & a'_{32}/a'_{22} & 1 \end{bmatrix}. \tag{2.81}$$

Matricea $\mathbf{E}^{-1}$ este matricea inferior triunghiulară căutată (matricea $\mathbf{L}$ în varianta Doolittle). Elementele nenule şi diferite de 1 ale matricei $\mathbf{L}$ pot fi memorate exact în triunghiul inferior al tabloului real bidimensional iniţial, aşa cum se face în secvenţa următoare.

```
; etapa de eliminare din metoda Gauss cu memorarea opuselor elementelor
; de multiplicare în triunghiul inferior al matricei
pentru k = 1, n - 1                ; parcurge sub-etape ale eliminării
    pentru i = k + 1, n            ; parcurge liniile
        p = -a_ik / a_kk           ; element de multiplicare
        pentru j = k + 1, n        ; parcurge coloanele
            a_ij = a_ij + p a_kj
        •
        a_ik = -p
    •
•
```

De fapt, putem chiar să renunţam la variabila p, obţinând următorul pseudocod pentru factorizarea "în loc" a unei matrice.

```
procedură factorizare_LU(n, a)
; factorizează "in loc" matricea a
; varianta Doolittle
; declaraţii
...
pentru k = 1, n - 1                ; parcurge sub-etape ale eliminării
    pentru i = k + 1, n            ; parcurge liniile
        a_ik = a_ik / a_kk         ; element de multiplicare
        pentru j = k + 1, n        ; parcurge coloanele
            a_ij = a_ij - a_ik a_kj
        •
    •
•
retur
```

Prin factorizare "în loc" a matricei se înţelege faptul că elementele semnificative ale factorilor (adică elementele diferite de zero ale lui $\mathbf{U}$ şi elementele diferite de zero aflate sub diagonală ale matricei $\mathbf{L}$) sunt memorate în locul valorilor iniţiale ale matricei $\mathbf{A}$. Se mai scrie "$\mathbf{A} = \mathbf{L} + \mathbf{U} - \mathbf{I}$", unde egalul este cel în sens aritmetic, dar matricea $\mathbf{A}$ conţine valorile după factorizare.

2.5.4 Calculul soluţiei după factorizare

Sistemul de rezolvat (2.53) este echivalent cu

$$\mathbf{LUx} = \mathbf{b}. \tag{2.82}$$

Dacă se notează

$$\mathbf{y} = \mathbf{Ux}, \tag{2.83}$$

atunci rezolvarea sistemului (2.82) este echivalentă cu rezolvarea succesivă a sistemelor

$$\mathbf{Ly} = \mathbf{b}, \tag{2.84}$$

$$\mathbf{Ux} = \mathbf{y}. \tag{2.85}$$

Sistemul (2.84), a cărui soluţie o putem scrie formal $\mathbf{y} = \mathbf{L}^{-1}\mathbf{b}$, este un sistem cu structură inferior triunghiulară, a cărui rezolvare se face prin *substituţie progresivă*:

$$\left\{\begin{array}{l} l_{11}y_1 = b_1, \\ l_{21}y_1 + l_{22}y_2 = b_2, \\ \cdots \\ l_{n1}y_1 + l_{n2}y_2 + \cdots l_{nn}y_n = b_n, \end{array}\right. \Rightarrow \left\{\begin{array}{l} y_1 = b_1/l_{11}, \\ y_2 = (b_2 - l_{21}y_1)/l_{22}, \\ \cdots \\ y_n = (b_n - l_{n1}y_1 - \cdots - l_{n,n-1}y_{n-1})/l_{nn}. \end{array}\right. \tag{2.86}$$

Scris compact, soluţia se calculează la substituţie regresivă ca

$$y_1 = b_1/l_{11}, \tag{2.87}$$

$$y_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij}y_j\right)/l_{ii}, \quad i = 2, \ldots, n. \tag{2.88}$$

Sistemul (2.85), a cărui soluţie o putem scrie formal $\mathbf{x} = \mathbf{U}^{-1}\mathbf{y}$, este un sistem cu structură superior triunghiulară, a cărui rezolvare se face prin *substituţie regresivă*, ca la metoda Gauss:

$$x_n = y_n/u_{nn}, \tag{2.89}$$

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)/u_{ii}, \quad i = n-1, \ldots, 1. \tag{2.90}$$

Pe baza formulelor (2.87) ÷ (2.90) putem scrie acum algoritmul rezolvării propriu-zise a sistemului, presupunând că matricea coeficienţilor este factorizată în loc în varianta Doolittle.

```
procedură rezolvă_LU(n, a, b, x)
; rezolvă sistemul de ecuaţii ax = b prin factorizare LU
; matricea este presupusă a fi deja factorizată "în loc"
; varianta Doolittle
; declaraţii
...
; substituţie progresivă
y_1 = b_1                          ; formula (2.87), unde l_11 = 1
pentru i = 2, n
      s = 0
      pentru j = 1, i - 1
            s = s + a_ij y_j       ; formula (2.88), unde L este memorat în a
      •
      y_i = b_i - s ; deoarce l_ii = 1
•
; substituţie regresivă
x_n = y_n / a_nn                   ; formula (2.89), unde U este memorat în a
pentru i = n - 1, 1, -1
      s = 0
      pentru j = i + 1, n
            s = s + a_ij x_j
      •
      x_i = (y_i - s) / a_ii
•
retur
```

2.5.5 Evaluarea algoritmului. Strategii de pivotare.

Deoarece algoritmul de factorizare este derivat din Gauss, putem evalua complexitatea lui prin similitudine cu acesta. Factorizarea propriu-zisă are complexitatea etapei de eliminare Gauss, deci $T_f = \mathrm{O}(2n^3/3)$, iar rezolvările prin substituţie progresivă şi regresivă au împreună $T_s = \mathrm{O}(2n^2)$.

Din punct de vedere al necesarului de memorie, efortul este dat de memorarea tabloului bidimensional, deci $\mathrm{M} = \mathrm{O}(n^2)$.

Ca la metoda Gauss, în factorizarea LU nu există erori de trunchiere, iar erorile de rotunjire pot fi micşorate dacă se aplică strategii de pivotare de tipul celor descrise la paragraful 2.4.4. Acestea au ca efect şi îmbunătăţirea stabilităţii algoritmului.

Pivotarea poate fi descrisă algebric prin folosirea unor matrice de permutare. *O matrice de permutare este o matrice care are exact un element egal cu 1 pe fiecare linie şi pe fiecare coloană, şi 0 în rest.* Ea reprezintă o permutare a liniilor şi/sau coloanelor matricei unitate. Atunci când o matrice de permutare înmulţeşte la stânga (respectiv la dreapta) o matrice oarecare, rezultatul este o matrice în care sunt permutate liniile (respectiv coloanele) matricei iniţiale.

Pivotarea pe linie poate fi descrisă prin înmulţirea la stânga cu o matrice de permutare notată în cele ce urmează cu $\mathbf{P}$, iar pivotarea pe coloane poate fi descrisă prin înmulţirea la dreapta cu o matrice de permutare ce va fi notată cu $\mathbf{Q}$. Se poate demonstra că inversa unei matrice de permutare este o matrice de permutare şi că produsul a două matrice de permutare este de asemenea o matrice de permutare [3].

Astfel, dacă înaintea primul pas al etapei de eliminare se efectuează o permutare parţială, relaţia (2.71) se scrie

$$\mathbf{A}_0 = \mathbf{P}_1\mathbf{A}. \tag{2.91}$$

Presupunând că la fiecare etapă de eliminare se efectuează o permutare parţială, relaţiile (2.72) se scriu

$$\begin{aligned} &\mathbf{A}_1 = \mathbf{E}_1\mathbf{A}_0 = \mathbf{E}_1\mathbf{P}_1\mathbf{A}, \\ &\mathbf{A}_2 = \mathbf{E}_2\mathbf{P}_2\mathbf{A}_1 = \mathbf{E}_2\mathbf{P}_2\mathbf{E}_1\mathbf{P}_1\mathbf{A}, \\ &\cdots \\ &\mathbf{A}_{n-1} = \mathbf{E}_{n-1}\mathbf{P}_{n-1}\cdots\mathbf{E}_2\mathbf{P}_2\mathbf{E}_1\mathbf{P}_1\mathbf{A}. \end{aligned} \tag{2.92}$$

În argumentarea factorizării Doolittle, ultima relaţie se scrie

$$\mathbf{U} = \mathbf{E}_{n-1}\mathbf{P}_{n-1}\cdots\mathbf{E}_2\mathbf{P}_2\mathbf{E}_1\mathbf{P}_1\mathbf{A}. \tag{2.93}$$

Aceasta pune în evidenţă factorii L şi U pentru o permutare pe linii a matricei $\mathbf{A}$. De exemplu (scriind pentru comoditate pentru $n = 4$):

$$\begin{aligned} \mathbf{U} &= \mathbf{E}_3\mathbf{P}_3\mathbf{E}_2\mathbf{P}_2\mathbf{E}_1\mathbf{P}_1\mathbf{A} = \\ &= \underbrace{\mathbf{E}_3}_{\mathbf{E}'_3}\underbrace{\mathbf{P}_3\mathbf{E}_2\mathbf{P}_3^{-1}}_{\mathbf{E}'_2}\underbrace{\mathbf{P}_3\mathbf{P}_2\mathbf{E}_1\mathbf{P}_2^{-1}\mathbf{P}_3^{-1}}_{\mathbf{E}'_1}\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1\mathbf{A} = \\ &= \underbrace{\mathbf{E}'_3\mathbf{E}'_2\mathbf{E}'_1}_{\mathbf{L}^{-1}}\underbrace{\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1}_{\mathbf{P}}\mathbf{A} \end{aligned} \tag{2.94}$$

Factorizarea cu pivotare pe linii se scrie deci compact ca

$$\mathbf{PA} = \mathbf{LU}, \tag{2.95}$$

unde toate elementele subdiagonale ale matricei **L** sunt în modul mai mici sau egale cu 1, drept consecinţă a modului în care a fost ales pivotul. Interpretarea formulei (2.95) ar putea fi făcută astfel. Mai întâi se permută liniile matricei **A** în conformitate cu matricea de permutare **P** apoi se aplică algoritmul de rezolvare pentru matricea **PA**. Am văzut că de fapt pivotarea parţială nu se realizează aşa, pentru că matricea **P** nu este cunoscută apriori. În implementarea practică **P** nu este reprezentată explicit ca o matrice, ci liniile sunt permutate la fiecare etapă.

Permutarea coloanelor se reprezintă algebric prin înmulţirea la dreapta cu o matrice de permutare **Q**. Factorizarea LU cu pivotare totală se poate scrie formal ca

$$\mathbf{PAQ} = \mathbf{LU}, \tag{2.96}$$

Pivotarea totală este rareori aplicată în practică deoarece căutarea pivoţilor şi pe linii şi pe coloane duce la o creştere semnificativă a timpului de calcul, nerealizând decât o îmbunătăţire nesemnificativă a stabilităţii.

2.5.6 Cazul sistemelor multiple

Cazul sistemelor multiple a fost prezentat mai întâi în paragraful 2.4.5. Metoda factorizării LU este utilă în acest caz deoarece factorizarea se face o singură dată, iar substituţiile se fac pentru cele m sisteme, pe măsură ce sunt cunoscuţi termenii liberi. Efortul total de rezolvare prin factorizare a m sisteme algebrice de dimensiune n va fi deci $T = \mathrm{O}(2n^3/3 + 2mn^2)$, mai mic decât cel necesar calculului inversei (tabelul 2.1).

2.5.7 Varianta Cholesky

Dacă matricea **A** este simetrică, atunci este de dorit ca şi factorizarea ei LU să păstreze această simetrie, adică

$$\mathbf{U} = \mathbf{L}^T. \tag{2.97}$$

Tabelul 2.1: Efort de calcul pentru rezolvarea sistemelor multiple.

Nr. sisteme	Metoda	Complexitate T
1	Gauss	$2n^3/3 + n^2$
	LU	$2n^3/3 + 2n^2$
m - simultan	Gauss	$2n^3/3 + 2mn^2$
m - succesiv	folosind inversa	$8n^3/3 + 2mn^2$
	LU	$2n^3/3 + 2mn^2$

Decompunerea LU care satisface condiţia (2.97) se numeşte *factorizare Cholesky* şi ea se poate realiza doar dacă $\mathbf{A}$ este pozitiv definită.

Pentru a justifica această afirmaţie, să considerăm $\mathbf{A}$ o matrice nesingulară şi simetrică şi presupunem că există $\mathbf{L}$ o matrice triunghiular inferioară, nesingulară, astfel încât

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T. \tag{2.98}$$

Fie un vector coloană arbitrar $\mathbf{x}$ nenul. Atunci $\mathbf{y} = \mathbf{L}^T\mathbf{x}$ va fi un vector coloană nenul (pentru că altfel $\mathbf{x} = (\mathbf{L}^T)^{-1}\mathbf{y}$ ar rezulta nul). Atunci

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T\mathbf{L}\mathbf{L}^T\mathbf{x} = \mathbf{y}^T\mathbf{y} = \sum_{i=1}^{n} y_i^2 > 0.$$

În consecinţă, numai matricile care satisfac $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ pentru vectori coloana $\mathbf{x}$ nenuli pot admite o descompunere Cholesky. Aceste matrici sunt prin definiţie matrici pozitiv definite.

Demonstraţia de mai sus arată că pozitiv definirea matricei este o condiţie necesară pentru factorizarea Cholesky. Se poate arăta însă că ea este şi o condiţie suficientă:

Teoremă: Dacă A este o matrice simetrică şi pozitiv definită, atunci factorizarea ei Cholesky există în mod unic, adică există în mod unic o matrice triunghiular inferioară L cu elementele diagonale pozitive, astfel încât $\mathbf{A} = \mathbf{L}\mathbf{L}^T$.

Pentru a înţelege modul de generare al matricei $\mathbf{L}$, vom începe prin partiţionarea matricelor astfel

$$\begin{bmatrix} \alpha & \mathbf{a}^T \\ \mathbf{a} & \hat{\mathbf{A}} \end{bmatrix} = \begin{bmatrix} \lambda & \mathbf{0} \\ \mathbf{l} & \hat{\mathbf{L}} \end{bmatrix} \begin{bmatrix} \lambda & \mathbf{l}^T \\ \mathbf{0} & \hat{\mathbf{L}}^\mathbf{T} \end{bmatrix}. \tag{2.99}$$

Pentru intuirea mai clară a blocurilor, am notat scalarii cu litere greceşti, vectorii cu litere mici aldine, iar matricile cu litere mari aldine. Din calculul elementului aflat pe poziţia (1,1) rezultă $\alpha = \lambda^2$ de unde

$$\lambda = \sqrt{\alpha} \tag{2.100}$$

este primul element din $\mathbf{L}$. Din calculul blocului aflat pe poziţia (2,1) rezultă $\mathbf{a} = \lambda\mathbf{l}$, de unde

$$\mathbf{l} = \mathbf{a}/\lambda \tag{2.101}$$

este prima coloană, aflată sub diagonală, a matricei $\mathbf{L}$. Din calculul blocului aflat pe poziţia (2,2) rezultă $\hat{\mathbf{A}} = \mathbf{l}\mathbf{l}^T - \hat{\mathbf{L}}\hat{\mathbf{L}}^T$ de unde

$$\hat{\mathbf{L}}\hat{\mathbf{L}}^T = \hat{\mathbf{A}} - \mathbf{l}\mathbf{l}^T. \tag{2.102}$$

Matricea din membrul drept al acestei relaţii este complementul Schur al lui α:

$$\mathbf{S} = \hat{\mathbf{A}} - \mathbf{l}\mathbf{l}^T = \hat{\mathbf{A}} - \mathbf{a}\mathbf{a}^T/\alpha. \tag{2.103}$$

Mai mult, se poate demonstra că $\mathbf{S}$ este simetrică şi pozitiv definită şi, în consecinţă $\hat{\mathbf{L}}\hat{\mathbf{L}}^T$ este factorizarea ei Cholesky.

Astfel, echivalentul primei etape de eliminare Gauss, se poate scrie in cazul factorizării Cholesky ca

$$\mathbf{A} = \mathbf{A}_0 = \begin{bmatrix} \lambda & \mathbf{0} \\ \mathbf{l} & \hat{\mathbf{L}} \end{bmatrix} \begin{bmatrix} \lambda & \mathbf{l}^T \\ \mathbf{0} & \hat{\mathbf{L}}^\mathbf{T} \end{bmatrix} = \begin{bmatrix} \lambda & \mathbf{0} \\ \mathbf{l} & \mathbf{I} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix} \begin{bmatrix} \lambda & \mathbf{l}^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \mathbf{L}_1\mathbf{A}_1\mathbf{L}_1^T. \tag{2.104}$$

Similar,

$$\begin{aligned} \mathbf{A}_1 &= \mathbf{L}_2\mathbf{A}_2\mathbf{L}_2^T, \\ &\vdots \\ \mathbf{A}_{n-1} &= \mathbf{L}_n\mathbf{A}_n\mathbf{L}_n^T, \end{aligned} \tag{2.105}$$

unde $\mathbf{A}_n = \mathbf{I}$.

Rezultă că, descompunerea finală a matricei $\mathbf{A}$

$$\mathbf{A} = \underbrace{\mathbf{L_1}\mathbf{L}_2 \cdots \mathbf{L}_n}_{\mathbf{L}} \underbrace{\mathbf{L}_n^T\mathbf{L}_{n-1}^T \cdots \mathbf{L}_1^T}_{\mathbf{L}^T} \tag{2.106}$$

este unică deoarece la fiecare pas mărimile au fost determinate în mod unic.

În ce priveşte implementarea factorizării Cholesky, numai jumătate din matricea $\mathbf{A}$ trebuie memorată. Nu numai necesarul de memorie este înjumătătit faţă de cazul general, dar şi efortul de factorizare este mai mic. Pentru a justifica acest lucru, vom schiţa o variantă de algoritm care nu foloseşte o schemă particulară de memorarea a matricei.

Un mic exerciţiu algebric arată că formulele de calcul pentru coloana k a matricei $\mathbf{L}$ sunt

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}, \quad k = 1, \ldots, n \tag{2.107}$$

$$l_{ik} = (a_{ik} - \sum_{j=1}^{k-1} l_{ij}l_{kj})/l_{kk}, \quad i = k+1, \ldots, n. \tag{2.108}$$

Acestea sunt implementate de pseudocodul următor:

`procedură` factorizare_LU_Cholesky(n, a, l)
; factorizează matricea a, presupusă simetrică şi pozitiv definită

; întoarce matricea triunghiular inferioară l
; varianta Cholesky
; declaraţii
...
pentru $k = 1, n$; parcurge sub-etape ale eliminării
  pentru $i = k, n$; calculează coloana, sub diagonală
    $s = a_{ik}$
    pentru $j = 1, k-1$
      $s = s - l_{ij} l_{kj}$
    •
    dacă $i = k$
      $l_{kk} = \sqrt{s}$
    altfel
      $l_{ik} = s / l_{kk}$
    •
  •
•
retur

Efortul de calcul corespunzător este jumătate din efortul unei factorizări LU pentru matrice oarecare (egal cu efortul unei eliminări Gauss)

$$T_e \approx \sum_{k=1}^{n}[2k(n-k)] = -2\sum_{k=1}^{n}[(n-k-n)(n-k)] = -2\left[\sum_{k=1}^{n}(n-k)^2 - n\sum_{k=1}^{n}(n-k)\right] =$$
$$= -2\left[\frac{(n-1)n(2n-1)}{6} - n\frac{n(n-1)}{2}\right] \approx -2\left(\frac{2n^3}{6} - \frac{n^3}{2}\right) = \frac{n^3}{3}. \quad (2.109)$$

În ce priveşte stabilitatea, algoritmul Cholesky este întotdeauna stabil şi nu are nevoie de pivotare. Aceasta se datorează proprietăţilor speciale ale matricei $\mathbf{A}$, care fiind pozitiv definită este şi diagonal dominantă.

2.6 Cazul matricelor rare

Algoritmii de rezolvare prezentaţi până acum (bazaţi în esenţă pe eliminarea Gauss) au complexitate destul de mare. Atât timpul de calcul este ridicat (ordin cubic), dar şi spaţiul de memorie este mare (ordin pătratic, datorită memorării matricei coeficienţilor).

În mod uzual, numerele reale se stochează în dublă precizie pe 8B, iar numerele întregi

pe 4B[1]. Un model mic al unei probleme, având de exemplu 1000 grade de libertate, va necesita o matrice a coeficienţilor cu 10^6 numere reale, deci 8 MB. Modele rafinate ale problemelor reale pot avea miloane de necunoscute. O problema cu un milion de necunoscute ar necesita un spaţiu de memorare de 1 TB şi nu ar încăpea în memoria calculatoarelor uzuale. Pe un PC obişnuit, nu s-ar putea lucra nici măcar cu o matrice de dimensiune 10000. Din fericire însă, formularea tuturor problemelor reale de inginerie poate fi făcută astfel încât ecuaţiile care se scriu nu implică toate necunoscutele simultan. De fapt, pentru o ecuaţie, legăturile sunt chiar foarte puţine comparate cu numărul total de necunoscute. De exemplu, în scrierea unei anumite ecuaţii nodale pentru un circuit, în ecuaţie apar doar termeni corespunzători laturilor care sunt incidente la nodul pentru care se scrie ecuaţia. Matricea conductanţelor nodale ar avea toate elementele nenule doar dacă graful circuitului ar fi un poligon complet, ceea ce nu se întâmplă în nici o aplicaţie reală.

Astfel de matrice nu se memorează în forma lor totală, ca tablouri bidimensionale, ci se vor memora doar elementele lor nenule. Mai mult, algoritmii de calcul se vor adapta acestei scheme de memorare, rezultând avantaje atât din punct de vedere al necesarului de memorie cât şi al timpului de calcul.

O matrice care conţine un număr foarte mare de elemente nule se numeşte matrice rară. Despre o matrice care nu este rară se spune că este *matrice densă sau plină*.

Se defineşte *densitatea unei matrice* ca fiind raportul dintre numărul de elemente nenule şi numărul total de elemente al matricei. De obicei, algoritmii care exploatează raritatea matricelor devin avantajoşi pentru valori mici ale densităţii. Nu se poate preciza o valoare exactă a densităţii care să demarcheze matricile rare de matricile pline.

Dacă, pentru o anumită matrice care are şi elemente nule, se poate elabora un algoritm care exploatează această structură şi care este mai eficient decât algoritmul conceput pentru matricea plină, atunci aceasta este o matrice rară.

De exemplu, pentru un circuit cu $N = 1000$ noduri, presupunând că fiecare nod este conectat în medie cu alte 4, atunci matricea conductanţelor nodale va avea în medie 5 elemente nenule pe linie (4 termeni nediagonali şi 1 termen diagonal), şi deci va avea densitatea $d = 5N/N^2 = 5/N = 0.5\%$. Aceasta este cu siguranţă o matrice pentru care merită aplicate tehnici de matrice rare.

[1] 1 Byte = 1 octet = 8 biţi. Vezi şi http://en.wikipedia.org/wiki/Byte pentru precizări asupra prefixului folosit pentru multipli de B.

2.6.1 Formate de memorare a matricelor rare

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

Cea mai naturală metodă ar fi cea în care se memorează doar valorile nenule şi poziţia lor în matrice, dată prin indici de linie (r_{idx}) şi de coloană (c_{idx}). Aşa se face de exemplu în formatul de fişier ASCII al site-ului *Matrix Market*[2]. Astfel, pentru un tablou bidimensional de dimensiune $m \times n$, în loc să se memoreze toate cele mn valori (inclusiv zerouri) într-un spaţiu de $M_{plin} = 8mn$ B, sunt necesare doar n_{nz} locaţii de memorie pentru numere reale şi $2n_{nz}$ locaţii de memorie pentru numere întregi, deci în total $M_{rar,coord} = 8 * n_{nz} + 4 * 2n_{nz} = 16n_{nz}$ B.

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

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

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

Memorarea poate fi şi mai eficientă decât atât. Informaţia din vectorul r_idx poate fi comprimată, indicându-se doar locul unde se schimbă valoarea lui r_idx. Acesta este *formatul (vechi) Yale*, cunoscut şi sub sigla *CRS* - Compressed Row Storage. O matrice $\mathbf{M}$, de dimensiuni $m \times n$ cu un număr de n_{nz} de elemente nenule, va fi stocată cu ajutorul a trei tablouri unidimensionale astfel:

- un tablou unidimensional *val*, de dimensiune n_{nz}, care conţine toate valorile nenule, de la stânga la dreapta şi de sus în jos;
- un talou unidimensional r_ptr, de dimensiune $m+1$ (câte un element pentru fiecare linie, plus un element adiţional care marchează sfârşitul tabloului), care conţine indicii ce indică în tabloul *val* locurile în care încep liniile. Mai precis, linia i a matricei iniţiale are valorile în *val* de la poziţia $r\_ptr(i)$ la poziţia $r\_ptr(i+1) - 1$;
- un talou unidimensional r_idx, de dimensiune n_{nz}, care conţine, pentru fiecare element din *val*, indicele coloanei pe care se află.

[2]Site-ul Matrix Market este acesibil la http://math.nist.gov/MatrixMarket.

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

De exemplu, aceeaşi matrice $\mathbf{M}$ de dimensiune 3×4, având 6 elemente nenule, va fi memorată într-un vector *val* de dimensiune 6, un vector *r_ptr* de dimensiune $3 + 1$ şi un vector *c_idx* de dimensiune 6, astfel

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

Un raţionament similar se poate face pe coloane, formatul numindu-se *CCS - Compressed Column Storage*, fiind formatul folosit de colecţia de matrice *Harwell - Boeing*, disponibilă de asemenea pe site-ul *Matrix Market*. Necesarul de memorie este $M_{rar,CCS} = 8n_{nz} + 4(n+1) + 4n_{nz} = 12n_{nz} + 4(n+1)$ B.

Acelaşi exemplu, memorat în format CCS este:

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

Dacă matricile au o structură particulară, atunci se poate alege o variantă şi mai eficientă, care să ţină cont de structură. Un exemplu ilustrativ este cel al *matricelor bandă* ale căror sub-diagonale se memorează în locaţii consecutive. De exemplu, o *matrice tridiagonală* are elemente nenule numai pe diagonala principală (notată q în fig. 2.5) şi pe cele două subdiagonale vecine (p şi r).

Aşa cum sugerează notaţia din figura 2.5, o matrice tridiagonală poate fi memorată cu ajutorul a trei vectori, eventual grupaţi într-o matrice cu 3 linii şi n coloane:

$$\mathbf{M}_{\text{rar}} = \begin{bmatrix} 0 & p_2 & p_3 & \cdots & p_{n-1} & p_n \\ q_1 & q_2 & q_3 & \cdots & q_{n-1} & q_n \\ r_1 & r_2 & r_3 & \cdots & r_{n-1} & 0 \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} q_1 & r_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ p_2 & q_2 & r_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & p_3 & q_3 & r_3 & \cdots & 0 & 0 & 0 \\ \cdots & & & & & & & \\ \cdots & & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & p_{n-1} & q_{n-1} & r_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & p_n & q_n \end{bmatrix}$$

Figura 2.5: Structura unei matrice tridiagonale.

Acest mod de stocare este cunoscut şi sub sigla *CDS* - Compressed Diagonal Storage. Se observă într-un astfel de mod de stocare vor fi incluse şi elemente nule, fără semnificaţie, dar numărul lor este nesemnificativ.

2.6.2 Metode directe pentru matrice rare

Memorarea rară a matricelor nu are nici o valoare dacă algoritmii de calcul nu sunt adaptaţi structurii de date. De exemplu, algoritmul Gauss descris în paragraful 2.4.2 aplicat unei matrice tridiagonale ar face o mulţime de calcule inutile, mai precis operaţii algebrice cu zerouri, ale căror rezultate ar fi tot zerouri. Algoritmul poate fi regândit cu uşurinţă pentru matricea tridiagonală. Matricea coeficienţilor are la inceputul etapei k de eliminare următoarea structură (exemplificată pentru $k = 4$:

$$\begin{bmatrix} * & * & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & * & * & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & * & * & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & q_k & r_k & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & p_{k+1} & q_{k+1} & r_{k+1} & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{k+2} & q_{k+2} & r_{k+2} & \cdots & 0 & 0 \\ \ldots & & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & p_n & q_n \end{bmatrix}$$

Se observă că singurul element de multiplicare ce trebuie calculat este $m = -p_{k+1}/q_k$, singura modificare suferind-o ecuaţia $k+1$, mai precis doar un singur termen în membrul stâng al acesteia $q_{k+1} = q_{k+1} + m * r_k$, şi temenul liber corespunzător.

În ceea ce priveşte etapa de retrosubstituţie, matricea triangularizată va avea doar două diagonale, calculul se va face tot prin substituţie regresivă, ultima componentă fiind calulată mai întâi

$$x_n = b_n/q_n, \tag{2.110}$$

iar o componentă oarecare x_i va rezulta din ecuaţia i după eliminare

$$q_i x_i + r_i x_{i+1} = b_i \quad \Rightarrow \quad x_i = (b_i - r_i x_{i+1})/q_i, \quad i = n-1, \ldots, 1. \tag{2.111}$$

Algoritmul următor implementează metoda Gauss adaptată unei matrice tridiagonale, memorată cu ajutorul a trei vectori, aşa cum a fost descris mai sus.

`procedură` Gauss_tridiag(n, p, q, r, b, x)
; rezolvă sistemul algebric liniar $ax = b$ prin metoda Gauss
; matricea a este tridiagonală, memorată în p, q, r
`întreg` n ; dimensiunea sistemului

```
tablou real p[n], q[n], r[n]        ; "matricea" coeficienţilor - indici de la 1
tablou real b[n]                    ; vectorul termenilor liberi
tablou real x[n]                    ; vectorul soluţie
întreg i, k
; etapa de eliminare din metoda Gauss
pentru k = 1, n − 1                 ; parcurge sub-etape ale eliminării
      m = −p_{k+1}/q_k              ; element de multiplicare
      q_{k+1} = q_{k+1} + m r_k     ; modifică element în linia k + 1
      b_{k+1} = b_{k+1} + m b_k     ; modifică termenul liber al ecuaţiei k + 1
•
; etapa de retrosubstituţie
x_n = b_n/q_n
pentru i = n − 1, 1, −1
      x_i = (b_i − r_i x_{i+1})/q_i
•
retur
```

Etapa de eliminare are acum o complexitate de ordinul $5n$, iar etapa de retrosubstituţie de ordinul $3n$, complexitatea totală fiind liniară atât din punct de vedere al timpului de calcul $T = \mathrm{O}(8n)$, cât şi din punct de vedere al necesarului de memorie $T = \mathrm{O}(5n)$.

În cazul unor matrice rare care nu sunt tridiagonale şi nici nu au o structură particulară, algoritmul Gauss trebuie adaptat unor tehnici de memorare de tip CRS sau CCS. În timpul etapei de eliminare matricea se poate umple, astfel încât pivotarea urmăreşte nu numai stabilitatea numerică (pivoţi nenuli de modul cât mai mare pentru acumularea unor erori de rotunjire cât mai mici), ci şi minimizarea umplerilor, adică a elementelor nenule nou apărute. Găsirea pivotării optime care ar alege dintre toate pivotările posibile la un moment dat ar conduce la un algoritm non-polinomial, lipsit de importanţă practică. Strategiile de pivotare care se implementează folosesc tehnici euristice, care dau o soluţie pseudo-optimală dar au un grad de complexitate polinomial.

Motivul cel mai important pentru care calculul explicit al inversei unei matrice trebuie evitat este acela că, în marea majoritate a aplicaţiilor practice, inversa unei matrice este plină, chiar dacă matricea este rară. Fenomenul prin care elemente iniţial nule devin nenule prin calcul, se numeşte *fenomen de umplere*. La matrice rare de dimensiuni mari inversarea este practic imposibilă datorită acestui fenomen.

Din fericire, factorizarea unei matrice rare poate salva raritatea dacă matricea are o anumită structură. De exemplu o matrice cu structura $\mathbf{A}_1$ din figura 2.6 prin factorizare rămâne rară, în schimb matricea $\mathbf{A}_2$ se umple prin factorizare. Cele două matrice pot reprezenta acelaşi sistem, ultima ecuaţie din $\mathbf{A}_1$ fiind prima ecuaţie în $\mathbf{A}_2$ şi ultima ne-

$$\mathbf{A}_1 = \begin{bmatrix} * & 0 & \cdots & 0 & 0 & * \\ 0 & * & \cdots & 0 & 0 & * \\ \cdots & & & & & \\ 0 & 0 & \cdots & * & 0 & * \\ 0 & 0 & \cdots & 0 & * & * \\ * & * & \cdots & * & * & * \end{bmatrix} \qquad \mathbf{A}_2 = \begin{bmatrix} * & * & * & \cdots & * & * \\ * & * & 0 & \cdots & 0 & 0 \\ * & 0 & * & \cdots & 0 & 0 \\ \cdots & & & & & \\ * & 0 & \cdots & 0 & * & 0 \\ * & 0 & \cdots & 0 & 0 & * \end{bmatrix}$$

Figura 2.6: Matricea $\mathbf{A}_1$ are factorii LU rari, în timp ce matricea $\mathbf{A}_2$ are factorii LU plini.

cunoscută în $\mathbf{A}_1$ fiind prima necunoscută în $\mathbf{A}_2$. Una din tehnicile euristice de pivotare este cea sugerată de aceste două cazuri şi anume, pivotarea urmăreşte plasarea la sfârşit a ecuaţiilor "pline".

Structura matricei joacă deci un rol important în conceperea algoritmului de rezolvare.

Deoarece factorizarea generează umpleri, un algoritm eficient va trebui să ştie în avans unde apar umplerile astfel încât să aloce memorie pentru structurile de date ce memorează factorii L şi U. Această etapă se numeşte *factorizare simbolică*. Abia după ce se efectuează factorizarea simbolică, are loc factorizarea numerică. Factorizarea simbolică este o etapă mai puţin costisitoare dar mai complicat de elaborat. Ea se bazează pe construirea unor grafuri asociate matricei şi deducerea unor arbori de eliminare. Un document ce ilustrează în detaliu aceste concepte este [13], iar o carte excelentă dedicată exclusiv acestui subiect este [2].

În concluzie, pentru a obţine algoritmi performanţi pentru o anumită problemă, trebuie folosite structuri de date eficiente, care să valorifice eventualele proprietăţi particulare ale datelor (de exemplu raritatea matricelor), iar algoritmii de calcul trebuie adaptaţi acestor structuri de date. Metodele directe de rezolvare a sistemelor rare de ecuaţii algebrice au patru paşi principali: permutarea sistemului în vederea asigurării unei stabilităţi maxime, factorizarea simbolică, factorizarea numerică şi rezolvarea celor două sisteme triunghiulare.

2.7 Metode iterative staţionare - generalităţi

Metodele iterative pentru rezolvarea sistemelor algebrice liniare se bazează pe generarea unui şir de aproximaţii ale soluţiei, care se doreşte a fi convergent către soluţia exactă. În consecinţă, din punct de vedere teoretic, soluţia poate fi obţinută cu o astfel de metodă numai într-un număr infinit de paşi. Cum algoritmul poate face numai un număr finit de paşi, înseamnă că aceste metode sunt afectate şi de erori de trunchiere,

nu numai de erori de rotunjire. Aparent, acesta ar fi un dezavantaj faţă de metodele directe care găsesc soluţia, teoretic într-un număr finit de paşi. În realitate, metodele iterative au proprietatea remarcabilă de a face autocorecţia erorilor astfel încât se poate întâmpla ca soluţia obţinută printr-o metodă iterativă să fie mai precisă decât soluţia obţinută printr-o metodă directă. Metodele iterative sunt atractive şi pentru faptul că oferă utilizatorului posibilitatea de a decide el însuşi asupra compromisului dintre timpul de calcul şi acurateţea soluţiei, calculând mai multe iteraţii dacă se doreşte o soluţie mai precisă, sau oprind algoritmul mai repede atunci când se doreşte un timp mai scurt de rezolvare.

Un avantaj major al metodelor iterative este acela că, aplicate unor matrice rare, ele nu genereaza umpleri ale acestora, aşa cum se întâmplă în cazul metodelor directe. Dacă memoria disponibilă pentru rezolvarea unei probleme printr-o metodă directă nu este suficientă, atunci o metodă iterativă este singura opţiune.

Dezavantajul metodelor iterative este acela că se poate întâmpla ca ele să nu fie convergente pentru o problemă dată, caz în care problema trebuie reformulată înainte de a fi rezolvată cu o metodă iterativă. Reformularea se poate face cu ajutorul unei matrice de transformare, numită *precondiţionator*. Un precondiţionator bun îmbunătăţeşte convergenţa metodei iterative suficient de mult pentru a amortiza costul construirii şi aplicării precondi-ţionatorului. Fără precondiţionare, o metodă iterativă s-ar putea chiar să eşueze.

2.7.1 Ideea de bază a metodelor iterative staţionare

Ideea metodelor iterative staţionare pentru rezolvarea unui sistem algebric liniar de dimensiune n

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.112}$$

este de a construi un şir de aproximaţii $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}, \ldots$ care să conveargă către soluţia exactă a ecuaţiei (2.112), notată $\mathbf{x}^*$:

$$\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*, \quad \text{unde} \quad \mathbf{A}\mathbf{x}^* = \mathbf{b}. \tag{2.113}$$

Fiecare aproximaţie corespunzătoare unei iteraţii k este un vector cu n componente:

$$\mathbf{x}^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} \in \mathbb{R}^{n\times 1}.$$

Un astfel de algoritm are nevoie de o iniţializare $\mathbf{x}^{(0)}$, un mod de generare a şirului de iteraţii şi un criteriu de oprire.

Iniţializarea şirului de iteraţii este, în principiu, arbitrară. Dacă însă sunt cunoscute apriori informaţii despre soluţie, de exemplu o încadrare a ei, atunci este recomandat ca iniţializarea să fie făcută cât mai aproape de soluţie, algoritmul iterativ putând fi mai rapid convergent.

Şirul de iteraţii se generează *recursiv*, adică soluţia la o nouă iteraţie se calculează exclusiv în funcţie de soluţia la iteraţia anterioară:

$$\mathbf{x}^{(k)} = F(\mathbf{x}^{(k-1)}), \tag{2.114}$$

unde F este o funcţie continuă a cărei construcţie o vom preciza în cele ce urmează. Trecând la limită pentru $k \to \infty$ relaţia (2.114) rezultă că soluţia exactă $\mathbf{x}^*$ va satisface relaţia

$$\mathbf{x}^* = F(\mathbf{x}^*), \tag{2.115}$$

ceea ce înseamnă că $\mathbf{x}^*$ este punct fix pentru aplicaţia F.

În concluzie, soluţia exactă a sistemului de ecuaţii este şi punct fix pentru F. Rezolvarea sistemului de ecuaţii algebrice liniare se face prin căutarea unui punct fix pentru F.

Este evident atunci că trebuie să existe o legătură între sistemul algebric iniţial, definit de matricea coeficienţilor $\mathbf{A}$ împreună cu vectorul termenilor liberi $\mathbf{b}$ şi funcţia F. Construcţia aplicaţiei F se bazează pe descompunerea matricei $\mathbf{A}$ într-o diferenţă de două matrice

$$\mathbf{A} = \mathbf{B} - \mathbf{C}. \tag{2.116}$$

Sistemul de rezolvat (2.112) este echivalent cu

$$\mathbf{Bx} = \mathbf{Cx} + \mathbf{b} \tag{2.117}$$

sau

$$\mathbf{x} = \mathbf{Mx} + \mathbf{u}, \tag{2.118}$$

unde

$$\begin{aligned} \mathbf{M} &= \mathbf{B}^{-1}\mathbf{C}, \\ \mathbf{u} &= \mathbf{B}^{-1}\mathbf{b}. \end{aligned} \tag{2.119}$$

Matricea $\mathbf{M} \in \mathbb{R}^{n\times n}$ definită de (2.119) se numeşte *matrice de iteraţie*. Relaţia (2.118) sugerează modul de definire a aplicaţiei F:

$$F(\mathbf{x}) = \mathbf{Mx} + \mathbf{u}, \tag{2.120}$$

iar calculul recursiv (2.114) poate fi scris explicit ca

$$\mathbf{x}^{(k)} = \mathbf{B}^{-1}\mathbf{C}\mathbf{x}^{(k-1)} + \mathbf{B}^{-1}\mathbf{b}. \tag{2.121}$$

În paragraful 2.4.5 s-a arătat că trebuie evitat calculul explicit al inversei unei matrice. De aceea, orice algoritm iterativ nu implementează relaţia (2.121) ci

$$\mathbf{B}\mathbf{x}^{(k)} = \mathbf{C}\mathbf{x}^{(k-1)} + \mathbf{b}, \tag{2.122}$$

care reprezintă rezolvarea unui nou sistem de ecuaţii algebrice liniare, având $\mathbf{B}$ ca matrice a coeficienţilor.

Încă nu am precizat exact cum se descompune matricea coeficienţilor în (2.116). Există o infinitate de moduri de a face acest lucru. Ideea este de a avea o matrice $\mathbf{B}$ cu o structură particulară, astfel încât sistemul (2.122) să poată fi rezolvat cu un efort de calcul mic, folosind o rezolvare adaptată structurii particulare a matricei $\mathbf{B}$. De exemplu, în metoda Jacobi $\mathbf{B}$ este diagonală şi rezolvarea sistemului (2.122) este imediată, iar în metoda Gauss-Seidel $\mathbf{B}$ este inferior triunghiulară, rezolvarea sitemului (2.122) făcându-se prin substituţie progresivă.

2.7.2 Criteriul de oprire

În ceea ce priveşte condiţia de oprire, se pune problema dacă şirul de iteraţii este convergent sau nu. Convergenţa şirului de iteraţii către soluţia exactă $\mathbf{x}^*$ este echivalentă cu convergenţa şirului $\mathbf{x}^{(k)} - \mathbf{x}^*$ către zero. Ideal ar fi ca iteraţiile să fie oprite atunci când eroarea $\|\mathbf{x}^{(k)} - \mathbf{x}^*\| \leq \varepsilon$, unde ε este o valoare impusă de utilizator. Cum eroarea nu poate fi calculată deoarece nu se cunoaşte soluţia exactă, în practică se implementează o condiţie de oprire bazată de criteriul de convergenţă Cauchy şi anume construcţia şirului iteraţiilor este oprită atunci când

$$\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \leq \varepsilon, \tag{2.123}$$

iar soluţia numerică aproximativă este cea obţinută la ultima iteraţie calculată.

Se poate întâmpla însă ca şirul iteraţiilor să nu fie convergent. Din acest motiv, criteriul de oprire va include şi o condiţie legată de numărul maxim de iteraţii admis. Dacă eroarea dorită ε nu se atinge într-un număr de iteraţii mai mic decât o limită maximă impusă, atunci procedeul iterativ se consideră neconvergent şi rezultatul obţinut nu este de încredere. S-ar putea întâmpla ca procedeul iterativ să fie convergent mai lent, astfel încât să fie nevoie de mai multe iteraţii pentru obţinerea unei precizii dorite. Este responsabilitatea utilizatorului de a urmări convergenţa procesului iterativ şi, în cazul unui proces iterativ convergent, de a stabili un compromis între timpul de calcul şi acurateţe.

În consecinţă, procedurile iterative de rezolvare a sistemelor algebrice liniare vor avea ca parametri de intrare, pe lângă mărimile ce definesc sistemul (coeficienţi şi termeni liberi), o eroare ce reprezintă criteriul dorit de oprire a

iteraţiilor şi un număr maxim de iteraţii, util pentru a asigura oprirea procedurii în caz de neconvergenţă.

Este evident că valoarea actuală a erorii folosită în momentul apelului unei astfel de proceduri iterative nu are sens să fie mai mică decât zeroul maşinii.

2.7.3 Intermezzo despre vectori şi valori proprii

Pentru înţelegerea mai uşoară a paragrafului următor, vom reaminti câteva noţiuni legate de vectori şi valori proprii. Aceste noţiuni sunt folosite ca instrumente de analiză în metodele iterative.

Prin definiţie, un vector propriu $\mathbf{v}$ al unei matrice pătrate reale $\mathbf{M}$, de dimensiune n este acel vector nenul, pentru care există un scalar λ astfel încât

$$\mathbf{M}\mathbf{v} = \lambda\mathbf{v}. \tag{2.124}$$

Altfel spus, dacă ne imaginăm reprezentarea geometrică a unui vector în spaţiu, prin aplicarea matricei $\mathbf{M}$ asupra lui, vectorul nu se roteşte (pentru că dacă λ este real, atunci $lambda\mathbf{v}$ este coliniar cu $\mathbf{v}$). El îşi poate schimba cel mult sensul, dar nu direcţia. Se observă imediat că vectorii proprii ai unei matrice nu sunt unici. Dacă $\mathbf{v}$ este un vector propriu, atunci şi vectorul scalat $\alpha\mathbf{v}$ este de asemenea vector propriu. Mărimea λ se numeşte valoare proprie a matricei $\mathbf{M}$ asociată vectorului propriu $\mathbf{v}$.

Metodele iterative depind adesea de înmulţirea dintre o matrice şi un vector, operaţie care se face în mod repetat. Dacă $\mathbf{v}$ este un vector propriu a lui $\mathbf{M}$, atunci înmulţirea repetată cu $\mathbf{M}$ conduce la

$$\mathbf{M}^k\mathbf{v} = \lambda^k\mathbf{v}. \tag{2.125}$$

Dacă $|\lambda| < 1$, atunci norma vectorului rezultant $\|\mathbf{M}^k\mathbf{v}\|$ va tinde către 0 când k tinde la infinit (un exemplu ilustrativ este prezentat în fig. 2.7), în timp ce dacă $|\lambda| > 1$, atunci norma vectorului rezultant va tinde către infinit.

Dacă matricea $\mathbf{M}$ este simetrică, atunci se poate demonstra că ea are n vectori proprii liniar independenţi, notaţi $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \ldots, \mathbf{v}^{(n)}$. Evident că această mulţime nu este unică, fiecare vector propriu putând fi scalat. Fiecare vector propriu are asociată o valoare proprie. Aceste valori proprii sunt unice pentru o matrice dată.

Dacă $\mathbf{M}$ este aplicată unui vector $\mathbf{u}$ care nu este vector propriu, atunci acesta poate fi scris ca o combinaţie liniară de alţi vectori al căror comportament este înţeles. Dacă cei n vectori proprii ai lui $\mathbf{M}$ formează o bază, atunci $\mathbf{u} = \sum_{i=1}^{n} \alpha_i\mathbf{v}^{(i)}$. Cum produsul matrice-vector este distributiv, atunci se poate urmări efectul aplicării lui $\mathbf{M}$ asupra fiecărui vector separat:

$$\mathbf{M}^k\mathbf{u} = \alpha_1\lambda_1^k\mathbf{v}^{(1)} + \cdots \alpha_n\lambda_n^k\mathbf{v}^{(n)}. \tag{2.126}$$

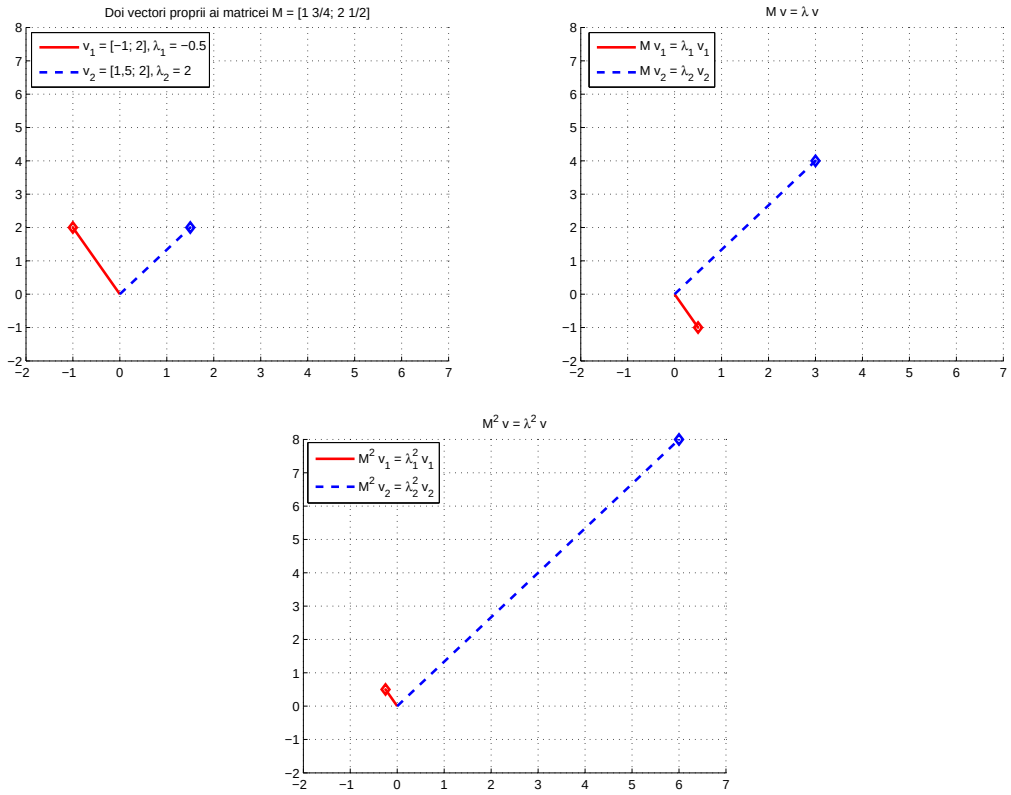


Figura 2.7: Înmulţirea repetată dintre matricea $\mathbf{M} = [1, 3/4; 2, 1/2]$ şi vectorii ei proprii $\mathbf{v}_1 = [-1, 2]$ (corespunzător valorii proprii $\lambda_1 = -0.5$) şi $\mathbf{v}_2 = [1.5, 2]$ ($\lambda_2 = 2$): sus stânga $\mathbf{v}_1$ şi $\mathbf{v}_2$; sus dreapta $\mathbf{M}\mathbf{v}_1 = \lambda_1\mathbf{v}_1$ şi $\mathbf{M}\mathbf{v}_2 = \lambda_2\mathbf{v}_2$; jos $\mathbf{M}^2\mathbf{v}_1 = \lambda_1^2\mathbf{v}_1$ şi $\mathbf{M}^2\mathbf{v}_2 = \lambda_2^2\mathbf{v}_2$.

Dacă toate valorile proprii sunt subunitare în modul, atunci norma vectorului rezultant va tinde către zero. E suficient ca o singură valoare proprie să fie în modul mai mare ca 1, pentru ca norma vectorului rezultant să tindă către infinit.

De aceea, se acordă o deosebită importanţă *razei spectrale a unei matrice*, definită ca modulul celei mai mari valori proprii

$$\rho(\mathbf{M}) = \max_i |\lambda_i|. \tag{2.127}$$

Valorile proprii sunt rădăcinile polinomului caracteristic. Pentru a justifica acest lucru, din (2.125) rezultă că valorile componentelor vectorilor proprii satisfac sistemul de ecuaţii algebrice liniare

$$(\lambda\mathbf{I} - \mathbf{M})\mathbf{v} = \mathbf{0} \tag{2.128}$$

Cum vectorii proprii sunt nenuli, rezultă că în mod necesar matricea coeficienţilor acestui sistem trebuie să fie singulară, altfel vectorii proprii ar fi nuli. Condiţia de singularitate

este echivalentă cu anularea determinantului matricei, care este exact *polinomul caracteristic al matricei*:

$$\det(\lambda \mathbf{I} - \mathbf{M}) = 0. \tag{2.129}$$

Membrul stâng al relaţiei (2.129) este un polinom de gradul n în λ, cu coeficienţi reali. Conform teoremei fundamentale a algebrei, ecuaţia are exact n soluţii (reale sau în perechi complex conjugate), care sunt valorile proprii ale matricei.

2.7.4 Convergenţă

Estimarea convergenţei procesului iterativ poate fi făcută apriori, folosind următoarele teoreme.

Teorema 1: Condiţia necesară şi suficientă ca procesul iterativ să fie convergent este ca raza spectrală a matricei de iteraţie să fie strict subunitară

Intuitiv, această teoremă poate fi înţeleasă pe baza relaţiei dintre eroarea soluţiei şi matricea de iteraţie. Din (2.114), (2.115) şi (2.120) rezultă că eroarea la o iteraţie k este:

$$\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^* = F(\mathbf{x}^{(k-1)}) - F(\mathbf{x}^*) = \mathbf{M}\mathbf{x}^{(k-1)} + \mathbf{u} - \mathbf{M}\mathbf{x}^* - \mathbf{u} = \mathbf{M}\mathbf{e}^{(k-1)}, \tag{2.130}$$

şi în consecinţă eroarea depinde de matricea de iteraţie ridicată la puterea k şi de eroarea iniţiala $\mathbf{e}^{(0)}$:

$$\mathbf{e}^{(k)} = \mathbf{M}\mathbf{e}^{(k-1)} = \mathbf{M}^2\mathbf{e}^{(k-2)} = \cdots = \mathbf{M}^k\mathbf{e}^{(0)}. \tag{2.131}$$

După cum a fost ilustrat în paragraful 2.7.3, dacă toate valorile proprii ale matricei $\mathbf{M}$ au modulul strict subunitar, atunci norma erorii $\|\mathbf{e}^{(k)}\|$ va tinde către zero atunci când k tinde la infinit, procedeul iterativ fiind convergent.

Teorema 2: O condiţie suficientă ca procesul iterativ să fie convergent este ca norma matricei de iteraţie să fie strict subunitară.

Această teoremă se poate demonstra cu uşurintă pe baza primei teoreme şi a relaţiei

$$\rho(\mathbf{M}) \leq \|\mathbf{M}\|. \tag{2.132}$$

O altă demonstraţie poate fi făcută pornind de la (2.131) şi aplicând proprietăţile normei:

$$\|\mathbf{e}^{(k)}\| \leq \|\mathbf{M}\|^k \|\mathbf{e}^{(0)}\|. \tag{2.133}$$

Când k tinde la infinit şi norma matricei de iteraţie este subunitară, majorantul normei erorii tinde la zero. În consecinţă, norma erorii tinde la zero, procesul fiind convergent.

Mai mult, şi diferenţa dintre două aproximaţii consecutive scade dacă norma matricei de iteraţie este strict subunitară:

$$\begin{aligned} \|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| &= \|\mathbf{M}\mathbf{x}^{(k-1)} + \mathbf{u} - \mathbf{M}\mathbf{x}^{(k-2)} - \mathbf{u}\| = \|\mathbf{M}(\mathbf{x}^{(k-1)} - \mathbf{x}^{(k-2)})\| \leq \\ &\leq \|\mathbf{M}\|\|\mathbf{x}^{(k-1)} - \mathbf{x}^{(k-2)}\|, \end{aligned} \tag{2.134}$$

astfel încât utilizarea unui criteriu de oprire Cauchy este pe deplin justificată.

Fie o margine a erorii absolute notată cu a_k, unde $\|\mathbf{e}^{(k)}\| \leq a_k$. Din relaţia (2.133) rezultă că

$$a_k = \|\mathbf{M}\|^k a_0, \tag{2.135}$$

de unde

$$\log(a_k) = k \log \|\mathbf{M}\| + \log a_0. \tag{2.136}$$

Mărimea $R(\mathbf{M}) = -\log \|\mathbf{M}\|$ se numeşte rata de convergenţă. Rezultă că

$$\log(a_k) = -kR(\mathbf{M}) + \log a_0. \tag{2.137}$$

Într-un grafic ce ar ilustra dependenţa erorii în funcţie de numărul de iteraţii, în care eroarea este reprezentată în scară logaritmică, rata de convergenţa este panta dreptei ce mărgineşte superior graficul.

Din (2.137) scrisă pentru două iteraţii consecutive, rezultă că

$$R(\mathbf{M}) = \log(a_{k-1}) - \log(a_k), \tag{2.138}$$

ceea ce înseamnă că rata de convergenţă reprezintă numărul de cifre semnificative corecte ce se câştigă la fiecare iteraţie. Inversa ratei de convergenţă reprezintă numărul de iteraţii necesar pentru câştigarea unei cifre semnificative corecte.

De exemplu, pentru o matrice având $\|\mathbf{M}\| = 10^{-3}$, rata de convergenţă este 3, deci la fiecare iteraţie numărul de cifre semnificative corecte creşte cu 3. Aceasta este o convergenţă extrem de rapidă. Pentru o matrice cu $\|\mathbf{M}\| = 10^{-1}$, la fiecare iteraţie se câştigă o cifră semnificativă.

Este de remarcat că alegerea valorii iniţiale nu are nici o influenţă asupra convergenţei sau neconvergenţei procesului iterativ. În cazul unui proces iterativ convergent, valoarea iniţială afectează doar numărul de iteraţii necesar pentru atingerea unei erori impuse.

2.7.5 Algoritm general

Algoritmul general al unei metode iterative este dat de următorul pseudocod simplificat

`procedură` metodă_iterativă$(n, B, C, b, x0, er, maxit, x)$
...
xv = **x0** ; iniţializează şirul iteraţiilor
$k = 0$; iniţializare contor iteraţii
`repetă`
t = **C** $*$ **xv** + **b**

```
    metodă_directă (n, B, t, x)
    d = ||xv − x||
    xv = x                          ; actualizează soluţia veche
    k = k + 1
cât timp d > er şi k ≤ maxit
retur
```

Acest pseudocod nu este scris conform regulilor din capitolul 1. El sugerează doar paşii importanţi ai unei metode iterative. Instrucţiunile scrise cu litere aldine, de exemplu $\mathbf{xv} = \mathbf{x0}$ ascund operaţii de atribuiri pe componente. De asemenea, instrucţiunea $\mathbf{t} = \mathbf{C}*\mathbf{xv}+\mathbf{b}$ reprezintă mai întâi înmulţirea dintre o matrice şi un vector, urmată de adunarea cu un vector.

Se observă de asemenea că nu sunt memorate toate valorile aproximative din şirul de iteraţii. Deoarece o iteraţie se calculează exclusiv în funcţie de iteraţia anterioară, este suficientă memorarea doar a unei iteraţii vechi (notate $\mathbf{xv}$ în algoritm).

Estimarea gradului de complexitate a unui astfel de algoritm poate fi făcută doar pentru o singură iteraţie. Efortul de calcul depinde însă de structura matricelor în care a fost descompusă matricea coeficienţilor, fiind consumat mai ales în secvenţa în care se calculează noul termen liber $\mathbf{t}$ şi care implică produsul dintre o matrice şi un vector (având o complexitate pătratică în cazul cel mai defavorabil, în care matricea $\mathbf{C}$ este plină) şi procedura de rezolvare directă, care în general are o complexitate liniară deoarece $\mathbf{B}$ are o structură rară, particulară. Intuitiv, ne aşteptăm ca procedeul iterativ să fie cu atât mai rapid convergent cu cât $\mathbf{B}$ conţine mai multă informaţie din $\mathbf{A}$. Dacă $\mathbf{B}$ este mai rară (în cazul extrem $\mathbf{B}$ reţine doar elementele diagonale din $\mathbf{A}$), atunci rezolvarea cu metoda directă este foarte rapidă, deci efortul per iteraţie este mic, dar convergenţă este lentă.

Următoarele două paragrafe detaliază această procedură în funcţie de alegerea precisă a descompunerii matricei coeficienţilor.

2.8 Metoda Jacobi

În metoda Jacobi descompunerea matricei $\mathbf{A}$ se face astfel încât matricea $\mathbf{B}$ este diagonală.

2.8.1 Un exemplu simplu

Să considerăm acelaşi exemplu de la paragraful 2.4.1. Ideea metodei Jacobi este de a păstra în membru stâng termenii diagonali.

$$\left\{\begin{array}{rcrcrcr} x & + & 2y & - & z & = & -1 \\ -2x & + & 3y & + & z & = & 0 \\ 4x & - & y & - & 3z & = & -2 \end{array}\right. \Leftrightarrow \left\{\begin{array}{rcl} x & = & -2y + z - 1 \\ 3y & = & 2x - z \\ -3z & = & -4x + y - 2 \end{array}\right. \tag{2.139}$$

Aceasta sugerează următorul calcul recursiv al şirului de iteraţii

$$\begin{aligned} x^{(n)} &= -2y^{(v)} + z^{(v)} - 1 \\ y^{(n)} &= 2x^{(v)} - z^{(v)} \\ z^{(n)} &= -4x^{(v)} + y^{(v)} - 2 \end{aligned} \tag{2.140}$$

Dacă iniţializarea este nulă, atunci şirul iteraţiilor Jacobi va fi $[0,0,0]^T$, $[-1,0,-2]^T$, $[-3,0,2]^T$, etc.

2.8.2 Algoritmul metodei

Dacă notăm $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ (fig. 2.8), atunci descompunerea $\mathbf{A} = \mathbf{B} - \mathbf{C}$ în metoda Jacobi este

$$\mathbf{B} = \mathbf{D}, \quad \mathbf{C} = -(\mathbf{L} + \mathbf{U}), \tag{2.141}$$

iar relaţia (2.122) ce reprezintă calculul recursiv al noii iteraţii în funcţie de cea veche prin rezolvarea unui sistem algebric liniar devine

$$\mathbf{D}\mathbf{x}^{(k)} = -(\mathbf{L} + \mathbf{U})\mathbf{x}^{(k-1)} + \mathbf{b}. \tag{2.142}$$

Ecuaţia i a sistemului (2.142) este

$$a_{ii}\mathbf{x}_i^{(k)} = -\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k-1)} + b_i. \tag{2.143}$$

Se observă că fiecare ecuaţie este tratată izolat de celelalte.

$$\underset{\mathbf{A}}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}} = \underset{\mathbf{L}}{\begin{bmatrix} 0 & 0 & 0 & 0 \\ a_{21} & 0 & 0 & 0 \\ a_{31} & a_{32} & 0 & 0 \\ a_{41} & a_{42} & a_{43} & 0 \end{bmatrix}} + \underset{\mathbf{D}}{\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix}} + \underset{\mathbf{U}}{\begin{bmatrix} 0 & a_{12} & a_{13} & a_{14} \\ 0 & 0 & a_{23} & a_{24} \\ 0 & 0 & 0 & a_{34} \\ 0 & 0 & 0 & 0 \end{bmatrix}}$$

Figura 2.8: Partiţionarea matricei în metodele iterative.

Rezolvarea sistemului (2.142) este imediată, soluţia la o nouă iteraţie $\mathbf{x}^{(n)}$ calculându-se numai în funcţie de soluţia la iteraţia anterioară $\mathbf{x}^{(v)}$, nefiind necesară memorarea tuturor iteraţiilor:

$$\mathbf{x}_i^{(n)} = (b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} x_j^{(v)})/a_{ii} \quad i = 1, \ldots, n. \tag{2.144}$$

Fiecare componentă nouă poate fi calculată independent de celelalte componente noi, motiv pentru care metoda Jacobi se mai numeşte şi *metoda deplasărilor simultane*.

Pseudocodul procedurii Jacobi, în care pentru calculul normei se foloseşte norma euclidiană, este următorul.

```
procedură Jacobi(n, a, b, x0, er, maxit, x)
; rezolvă sistemul algebric liniar ax = b, de dimensiune n prin metoda Jacobi
întreg n                    ; dimensiunea sistemului
tablou real a[n][n]         ; matricea coeficienţilor, indici de la 1
tablou real b[n]            ; vectorul termenilor liberi
; mărimi specifice procedurilor iterative
tablou real x0[n]           ; iniţializarea soluţiei
real er                     ; eroarea folosită de criteriul de oprire
întreg maxit                ; număr maxim de iteraţii admis
tablou real xv[n]           ; aproximaţia anterioară ("veche")

pentru i = 1, n
    xv_i = x0_i
•
k = 0                       ; iniţializare contor iteraţii
repetă
    d = 0
    pentru i = 1, n
        s = 0
        pentru j = 1, n
            dacă j ≠ i
                s = s + a_ij * xv_j
            •
        •
        x_i = (b_i - s)/a_ii
        d = d + (x_i - xv_i)^2
    •
```

$d = \sqrt{d}$

pentru $i = 1, n$

$xv_i = x_i$; actualizează soluţia veche

•

$k = k + 1$

cât timp $d > er$ şi $k \leq maxit$

retur

2.8.3 Aspecte legate de convergenţă

Convergenţa procedurii Jacobi depinde de matricea de iteraţie (2.119) care este în acest caz

$$\mathbf{M} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}). \tag{2.145}$$

Pentru o problemă bine formulată matematic şi bine condiţionată, algoritmul Jacobi poate fi sau nu convergent. Acest lucru poate fi ilustrat grafic pentru un sistem de două ecuaţii cu două necunoscute. Din punct de vedere matematic nu contează ordinea în care sunt scrise ecuaţiile. Această ordine poate avea însă efect asupra convergenţei sau neconvergenţei procedeului iterativ. Dacă notăm cu Δ_1 dreapta corespunzătoare primei ecuaţii şi cu Δ_2 dreapta corespunzătoare celei de a doua ecuaţii, atunci deplasările simultane din metoda Jacobi pot decurge ca în fig. 2.9, find convergente în cazul din stânga şi neconvergente în cazul din dreapta.

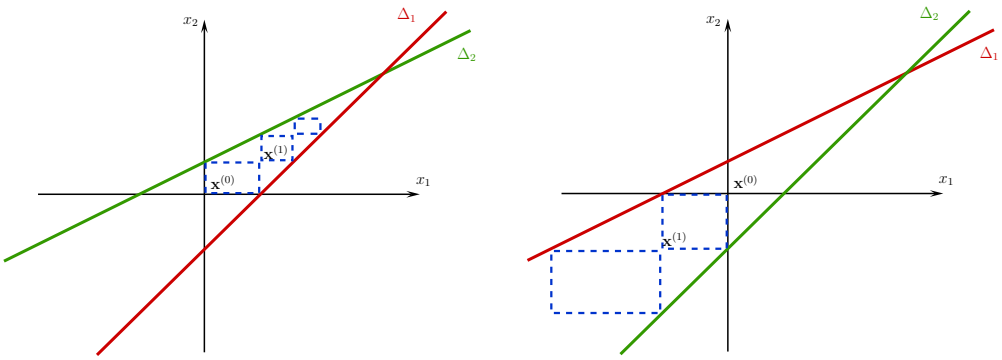


Figura 2.9: Schimbarea ordinii ecuaţiilor din sistem înseamnă schimbarea matricei de iteraţie, deci a proprietătilor de convergenţă ale metodei Jacobi.

În general, condiţiile de convergenţă sunt date de teoremele prezentate în paragraful 2.7.4, unde matricea de iteraţie este (2.145). Un alt rezultat util este următorul. **Dacă matricea coeficienţilor este diagonal dominantă, atunci condiţia suficientă de**

convergenţă este satisfăcută şi algoritmul Jacobi este convergent. Pentru justificarea acestei afirmaţii este suficient să se expliciteze componentele matricei (2.145). Rezultă imediat că $\|\mathbf{M}\| = \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}/a_{ii}|$, valoare mai mică decât 1 dacă matricea este diagonal dominantă.

O analiză riguroasă a convergenţei metodelor Jacobi şi Gauss-Seidel poate fi făcută pentru sisteme care provin din rezolvarea numerică a ecuaţiilor cu derivate parţiale (de exemplu ecuaţia Laplace) [8].

2.9 Metoda Gauss-Seidel

În metoda Gauss-Seidel descompunerea matricei $\mathbf{A}$ se face astfel încât matricea $\mathbf{B}$ este triunghiular inferioară.

2.9.1 Un exemplu simplu

Să considerăm acelaşi exemplu de la paragraful 2.4.1. Ideea metodei Gauss-Seidel este de a păstra în membru stâng termenii diagonali şi sub-diagonali.

$$\left\{\begin{array}{rcrcrcr} x & + & 2y & - & z & = & -1 \\ -2x & + & 3y & + & z & = & 0 \\ 4x & - & y & - & 3z & = & -2 \end{array}\right. \Leftrightarrow \left\{\begin{array}{rcrcrcr} x & & & & & = & -2y+z-1 \\ -2x & + & 3y & & & = & -z \\ 4x & - & y & - & 3z & = & -2 \end{array}\right. \tag{2.146}$$

ceea ce sugerează următorul calcul recursiv al şirului de iteraţii

$$\begin{array}{rcrcrcl} x^{(n)} & & & & & = & -2y^{(v)} + z^{(v)} - 1 \\ -2x^{(n)} & + & 3y^{(n)} & & & = & -z^{(v)} \\ 4x^{(n)} & - & y^{(n)} & - & 3z^{(n)} & = & -2 \end{array} \tag{2.147}$$

Dacă iniţializarea este nulă, atunci şirul iteraţiilor Gauss-Seidel va fi $[0,0,0]^T$, $[-1,-2,4]^T$, $[7,10,-40]^T$, etc.

2.9.2 Algoritmul metodei

Dacă notăm $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ (fig.2.8), atunci descompunerea $\mathbf{A} = \mathbf{B} - \mathbf{C}$ în metoda Gauss-Seidel este

$$\mathbf{B} = \mathbf{L} + \mathbf{D}, \quad \mathbf{C} = -\mathbf{U}, \tag{2.148}$$

iar relaţia (2.122) ce reprezintă calculul recursiv al noii iteraţii în funcţie de cea veche prin rezolvarea unui sistem algebric liniar devine

$$(\mathbf{L} + \mathbf{D})\mathbf{x}^{(k)} = -\mathbf{U}\mathbf{x}^{(k-1)} + \mathbf{b}. \tag{2.149}$$

Ecuaţia i a sistemului (2.149) este

$$\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} + a_{ii}x_i^{(k)} = -\sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)} + b_i. \tag{2.150}$$

Deoarece calculul la o iteraţie nouă depinde exclusiv de iteraţia anterioară, nu este necesară memorarea tuturor iteraţiilor, iar relaţia (2.150) se poate scrie ca

$$\sum_{j=1}^{i-1} a_{ij}x_j^{(n)} + a_{ii}x_i^{(n)} = -\sum_{j=i+1}^{n} a_{ij}x_j^{(v)} + b_i, \tag{2.151}$$

în care (k) s-a înlocuit cu (n)-nou şi $(k-1)$ cu (v)-vechi. Rezolvarea sistemului se va face prin substituţie progresivă, conform formulei:

$$\mathbf{x}_i^{(n)} = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(n)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(v)})/a_{ii}. \tag{2.152}$$

Se observă că este respectat *principiul lui Seidel*, conform căruia o valoare nouă a unei necunoscute trebuie folosită imediat în calcule. O componentă nouă nu poate fi calculată independent de celelalte componente noi, motiv pentru care metoda Gauss-Seidel se mai numeşte şi *metoda deplasărilor succesive*. Altfel spus, ecuaţiile sunt examinate secvenţial, iar rezultatele obţinute sunt imediat folosite în calcule.

Pseudocodul procedurii Gauss-Seidel, în care pentru calculul normei se foloseşte norma Chebyshev, este următorul.

```
procedură Gauss-Seidel(n, a, b, x0, er, maxit, x)
; rezolvă sistemul algebric liniar ax = b, de dimensiune n prin metoda Gauss-Seidel
; declaraţii ca la procedura Jacobi
...
pentru i = 1, n
    xv_i = x0_i
•
k = 0                          ; iniţializare contor iteraţii
repetă
    d = 0
    pentru i = 1, n
        s = b_i
        pentru j = 1, n
            dacă j ≠ i
                s = s + a_ij * xv_j
```

•

•

$s = s/a_{ii}$

$p = |x_i - s|$

<u>dacă</u> $p > d$

$d = p$

•

$x_i = s$

•

$k = k + 1$

<u>cât timp</u> $d > er$ <u>şi</u> $k \leq maxit$

<u>retur</u>

Spre deosebire de algoritmul Jacobi, în algoritmul Gauss-Seidel nu este necesară memorarea soluţiei anterioare. O dată calculată noua valoare, vechea valoare nu mai este folosită decât la calculul erorii.

2.9.3 Aspecte legate de convergenţă

Convergenţa procedurii Gauss-Seidel depinde de matricea de iteraţie (2.119) care este în acest caz

$$\mathbf{M} = -(\mathbf{L} + \mathbf{D})^{-1}\mathbf{U}. \tag{2.153}$$

Pentru o problemă bine formulată matematic şi bine condiţionată, algoritmul Gauss-Seidel poate fi sau nu convergent. Acest lucru poate fi ilustrat grafic pentru un sistem de două ecuaţii cu două necunoscute. Din punct de vedere matematic nu contează ordinea în care sunt scrise ecuaţiile. Această ordine poate avea însă efect asupra convergenţei sau neconvergenţei procedeului iterativ. Dacă notăm cu Δ_1 dreapta corespunzătoare primei ecuaţii şi cu Δ_2 dreapta corespunzătoare celei de a doua ecuaţii, atunci deplasările succesive din metoda Gauss-Seidel pot decurge ca în fig. 2.10, find convergente în cazul din stânga şi neconvergente în cazul din dreapta.

Condiţiile generale de convergenţă sunt date de teoremele prezentate în paragraful 2.7.4, unde matricea de iteraţie este (2.153). Ca şi în cazul metodei Jacobi, metoda Gauss-Seidel este convergentă dacă matricea coeficienţilor $\mathbf{A}$ este diagonal dominantă. O altă condiţie suficientă de convergenţă este ca matricea $\mathbf{A}$ să fie simetrică şi pozitiv definită.

În general, dacă metoda Jacobi este convergentă, metoda Gauss-Seidel este mai rapid convergentă. În cazul matricilor simetrice şi pozitiv definite, Gauss-Seidel este de aproximativ două ori mai rapid convergentă decât Jacobi [4].

2.9.4 Evaluarea algoritmului

Efortul de calcul atât al algoritmului Gauss-Seidel cât şi al algoritmului Jacobi depinde de numărul de iteraţii m, număr care depinde de rata de convergenţă a algoritmului, deci de matricea de iteraţie.

Analizând pseudocodurile celor doi algoritmi, constatăm că efortul de calcul per iteraţie este $O(2n^2)$. **Efortul total de calcul al algoritmilor Jacobi şi Gauss-Seidel este $T = O(2mn^2)$.**

Comparând acest efort de calcul cu cel al metodei Gauss, rezultă că metoda Jacobi sau Gauss-Seidel este mai eficientă decât metoda Gauss dacă $2mn^2 < 2n^3/3$, deci dacă $m < n/3$. Practic, dacă algoritmii nu converg în $n/3$ iteraţii atunci este mai bine ca ei să fie opriţi şi rezolvarea să se facă cu o metodă directă.

Din punct de vedere al necesarului de memorie, algoritmul Gauss-Seidel necesită doar structurile de date necesare pentru memorarea sistemului (matricea coeficienţilor şi termenii liberi) şi a vectorului soluţie (ultima iteraţie), în total $M_{GS} = O(n^2 + 2n) \approx O(n^2)$, fiind mai mic decât necesarul de memorie pentru algoritmul Jacobi $M_J = O(n^2 + 3n) \approx O(n^2)$, diferenţa nefiind semnificativă dacă matricile sunt pline.

Dacă însă matricea coeficienţilor este rară, atunci efortul de calcul pe iteraţie se poate diminua, calculul $s = s + a_{ij} * x_j$ făcându-se doar dacă a_{ij} este diferit de zero. Într-un astfel de caz, memorarea matricei coeficienţilor nu se va face însă într-un tablou bidimensional ci în structuri de date de tipul celor descrise în paragraful 2.6.1, deci nici necesarul de memorie nu va mai fi pătratic. Ceea ce este important de remarcat este faptul că nu putem vorbi de umpleri ale matricei coeficienţilor aşa cum se întâmplă în cazul algoritmului Gauss aplicat pentru matrice rare.

Metodele iterative sunt mai potrivite decât metodele directe pentru re-

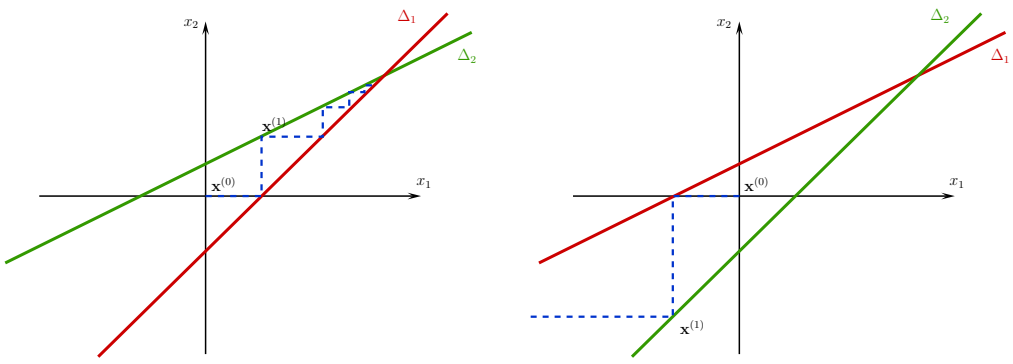


Figura 2.10: Schimbarea ordinii ecuaţiilor din sistem înseamnă schimbarea matricei de iteraţie, deci a proprietătilor de convergenţă ale metodei Gauss-Seidel.

zolvarea sistemelor cu matrice rare, într-un context hardware în care memoria disponibilă nu este suficientă rezolvării prin metode directe.

2.10 Metoda suprarelaxării succesive (SOR)

Interpretarea geometrică a iteraţiilor Gauss-Seidel (fig.2.10) sugerează următorul procedeu de accelerare a convergenţei. Pasul prescris de metoda Gauss-Seidel se amplifică cu un factor supraunitar ω care se numeşte *factor de suprarelaxare*, soluţia nouă calculându-se ca

$$x_i^{(n)} = x_i^{(v)} + \omega(x_i^{(n\_GS)} - x_i^{(v)}). \tag{2.154}$$

Cazul $\omega = 1$ corespunde metodei Gauss-Seidel. Pseudocodul algoritmului suprarelaxării succesive se obţine înlocuind în pseudocodul algoritmului Gauss-Seidel instrucţiunea $x_i = s$ cu $x_i = x_i + \omega(s - x_i)$.

Relaţia (2.154) în care înlocuim formula de calcul a soluţiei Gauss-Seidel, dată de (2.151) devine

$$\begin{aligned} x_i^{(n)} &= \omega x_i^{(n\_GS)} + (1-\omega)x_i^{(v)} = \\ &= \omega(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(n)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(v)})/a_{ii} + (1-\omega)x_i^{(v)}. \end{aligned} \tag{2.155}$$

Aceasta corespunde rezolvării rezolvării sistemului

$$(\mathbf{D} + \omega\mathbf{L})\mathbf{x}^{(n)} = [(1-\omega)\mathbf{D} - \omega\mathbf{U}]\,\mathbf{x}^{(v)} + \omega b. \tag{2.156}$$

Matricea de iteraţie a metodei SOR este în consecinţă

$$\mathbf{M} = (\mathbf{D} + \omega\mathbf{L})^{-1}\,[(1-\omega)\mathbf{D} - \omega\mathbf{U}]\,, \tag{2.157}$$

iar proprietaţile ei de convergenţă depind de valoarea factorului de relaxare ω. S-a demonstrat (teorema lui Kahan) că metoda nu converge dacă factorul de relaxare ω este ales în afara intervalului $(0, 2)$ [1]. Cazul $\omega \in (0, 1)$ corespunde unei subrelaxări şi ea se foloseşte atunci când metoda Gauss-Seidel nu converge. Dacă matricea este simetrică şi pozitiv definită, SOR este garantat convergentă pentru orice valoare $\omega \in (0, 2)$. Alegerea valorii lui ω poate afecta în mod semnificativ rata de convergenţă. În general, este dificil să se calculeze apriori valoarea optimală a factorului de suprarelaxare. Chiar atunci când un astfel de calcul este posibil, efortul de calcul este atât de mare încât este rareori folosit.

Pentru o clasă specială de matrici, numite "consistent ordonate", obţinute prin discretizarea unor ecuaţii cu derivate parţiale prin parcurgerea sistematică a nodurilor reţelei

de discretizare se poate demonstra că valoarea optimă pentru ω este

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho^2}}, \tag{2.158}$$

unde ρ este raza spectrală a matricei de iteraţie Jacobi. O astfel de estimare este rareori folosită deoarece estimarea lui ρ este costisitoare. În practică se folosesc tehnici euristice, ce iau în considerare dimensiunea reţelei de discretizare a problemei [1, 8].

În cazul matricelor simetrice, o variantă a metodei SOR, numită SSOR, este folosită ca procedură de precondiţionare pentru metode nestaţionare.

2.11 Algoritmul general al metodelor staţionare

Metodele iterative discutate până acum pentru rezolvarea sistemului $\mathbf{Ax} = \mathbf{b}$ au la bază descompunerea matricei coeficienţilor în $\mathbf{A} = \mathbf{B} - \mathbf{C}$, iteraţiile construindu-se pe baza relaţiei

$$\mathbf{Bx}^{(k+1)} = \mathbf{Cx}^{(k)} + \mathbf{b}. \tag{2.159}$$

Această relaţie a condus la algoritmul general prezentat într-un pseudocod simplificat în paragraful 2.7.5. Dacă se scade $\mathbf{Bx}^{(k)}$ din ambii membri ai relaţiei (2.159), se obţine

$$\mathbf{B}(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \mathbf{b} - \mathbf{Ax}^{(k)}. \tag{2.160}$$

Dacă se defineşte reziduul la iteraţia k

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Ax}^{(k)}, \tag{2.161}$$

atunci membrul drept al relaţiei (2.160) reprezintă reziduul la iteraţia k. În consecinţă, calculul soluţiei la o nouă iteraţie poate fi pus sub forma

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{B}^{-1}\mathbf{r}^{(k)}, \tag{2.162}$$

adică o nouă iteraţie se calculează din iteraţia anterioară, adunând o corecţie calculată în funcţie de reziduul iteraţiei anterioare. Acest reziduu este întotdeauna înmulţit cu matricea $\mathbf{B}^{-1}$, aceeaşi pe parcursul tuturor iteraţiilor. La acest aspect se referă "staţionaritatea" metodelor Jacobi unde $\mathbf{B} = \mathbf{D}$, Gauss-Seidel unde $\mathbf{B} = \mathbf{D} + \mathbf{L}$, SOR unde $\mathbf{B} = \omega^{-1}(\mathbf{D} + \omega\mathbf{L})$. Ca de obicei, nu se face inversarea propriu zisă, ci se rezolvă un sistem algebric liniar, cu matricea coeficienţilor $\mathbf{B}$.

Pseudocodul simplificat care implementează acest mod de efectuare a calculelor este următorul.

`procedură` metodă_iterativă_v2$(n, B, A, b, x0, er, maxit, x)$
...
$\mathbf{xv} = \mathbf{x0}$; iniţializează şirul iteraţiilor
$k = 0$; iniţializare contor iteraţii
`repetă`
$\mathbf{r} = \mathbf{b} - \mathbf{A} \cdot \mathbf{xv}$; calculează reziduu
metodă_directă (n, B, r, z)
$d = \|\mathbf{z}\|$
$\mathbf{x} = \mathbf{xv} + \mathbf{z}$
$\mathbf{xv} = \mathbf{x}$; actualizează soluţia veche
$k = k + 1$
`cât timp` $d > er$ `şi` $k \leq maxit$
`retur`

Algoritmii detaliaţi prezentaţi pentru metodele Jacobi şi Gauss-Seidel au presupus că matricea coeficienţilor este plină. Metodele iterative sunt eficiente însă pentru matrici rare. În cazul matricilor pline, timpul de rezolvare cu metode iterative poate fi comparabil cu timpul de factorizare. Într-o astfel de situaţie, factorizarea este mai utilă deoarece, o dată ce factorii L şi U sunt calculaţi, rezolvarea sistemului poate fi făcută oricând pentru alt termen liber. De aceea un pseudocod simplificat, în care sunt scrise operaţii cu matrice, este mai general, putând fi adaptat unor matrice rare.

2.12 Metoda gradienţilor conjugaţi

Metoda gradienţilor conjugaţi este o metodă iterativă nestaţionară pentru rezolvarea unor sisteme de ecuaţii algebrice de forma

$$\mathbf{Ax} = \mathbf{b}, \tag{2.163}$$

unde matricea $\mathbf{A}$ este *simetrică şi pozitiv definită*[3]. Ca orice metodă iterativă, algoritmul este eficient pentru matrice rare, şi de aceea pseudocodurile descrise vor fi prezentate simplificat, evidenţiind operaţii de algebră liniară, presupuse a fi implementate ţinând cont de raritatea structurilor de date.

[3] O matrice reală $\mathbf{A} \in \mathbb{R}^{n \times n}$ este pozitiv definită dacă ea este simetrică şi dacă $\mathbf{x}^T\mathbf{Ax} > 0$ pentru orice vector real, nenul $\mathbf{x} \in \mathbb{R}^{n \times 1}$.

2.12.1 Forma pătratică asociată

Metoda gradienţilor conjugaţi este simplu de înţeles dacă ea este formulată ca o problemă de minimizare [12]. Pentru acesta este util să se considere o formă pătratică $f : \mathbb{R}^n \to \mathbb{R}$ definită de

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} + c, \tag{2.164}$$

unde $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{x}, \mathbf{b} \in \mathbb{R}^{n\times 1}$, $c \in \mathbb{R}$.

Se poate demonstra următoarea afirmaţie **Dacă A este simetrică şi pozitiv definită, atunci funcţia (2.164) este minimizată de soluţia ecuaţiei (2.163).**

Pentru a justifica această proprietate, este util să se considere gradientul funcţiei f

$$\nabla f(\mathbf{x}) = \frac{1}{2}\mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b}. \tag{2.165}$$

În punctul de minim gradientul este zero

$$\nabla f(\mathbf{x}) = 0 \quad \Rightarrow \quad \frac{1}{2}(\mathbf{A}^T + \mathbf{A})\mathbf{x} = \mathbf{b}. \tag{2.166}$$

Dacă matricea este simetrică, atunci $\mathbf{A}^T = \mathbf{A}$ şi condiţia (2.166) este echivalentă cu (2.163). În consecinţă, soluţia ecuaţiei (2.163) este punct critic pentru f. Dacă, în plus, matricea este pozitiv definită, atunci punctul critic este un punct minim.

Cea mai simplă înţelegere intuitivă a unei matrice pozitiv definite este obţinută prin considerarea variaţiei funcţionalei în cazul particular al unei matrice de dimensiune 2. Fig.2.11 reprezintă forma pătratică asociată unei matrice pozitiv definite. Minimul ei coincide cu soluţia sistemului de rezolvat şi metoda gradienţilor conjugaţi va funcţiona. Fig.2.12 reprezintă forma pătratică asociată unei matrice negativ definite. Maximul ei coincide cu soluţia sistemului de rezolvat şi metoda gradienţilor conjugaţi ar putea fi uşor modificată pentru a maximiza f. Curbele de nivel ale funcţionalei (formei) pătratice asociate unei matrice simetrice pozitiv sau negativ definite de ordin 2 sunt elipse; similar, în general, hipersuprafeţele de nivel în cazul unor sisteme de ordin superior sunt hiperelipsoizi. Fig.2.13 reprezintă forma pătratică asociată unei matrice singulare, pozitiv semidefinite. Minimul ei este atins într-o infinitate de valori, aflate pe o dreaptă. Sistemul de ecuaţii are o infinitate de soluţii, problema nefiind bine formulată matematic. Fig.2.14 reprezintă forma pătratică asociată unei matrice indefinite. Soluţia sistemului de ecuaţii este punct şa pentru această funcţională, fiind punct de minim pentru o direcţie şi punct de maxim pentru altă direcţie. Pentru o astfel de problema, metoda gradienţilor conjugaţi nu poate fi aplicată.

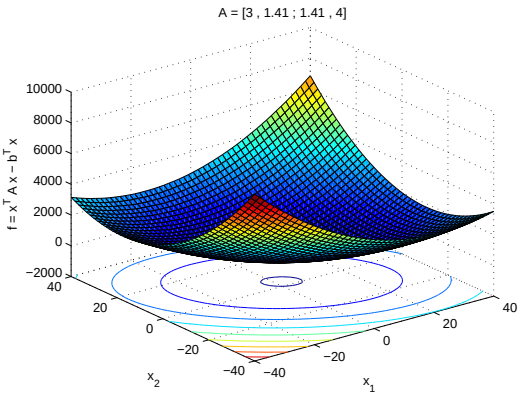


Figura 2.11: Funcţia pătratică asociată unei matrice pozitiv definite.

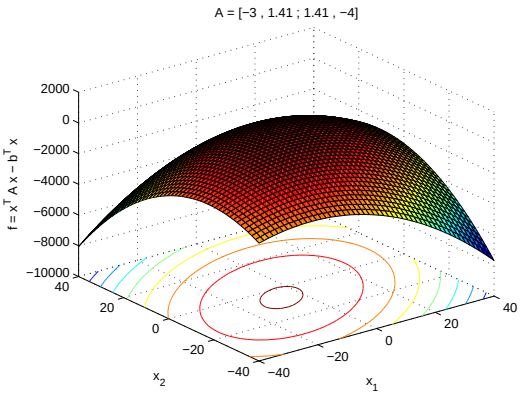


Figura 2.12: Funcţia pătratică asociată unei matrice negativ definite.

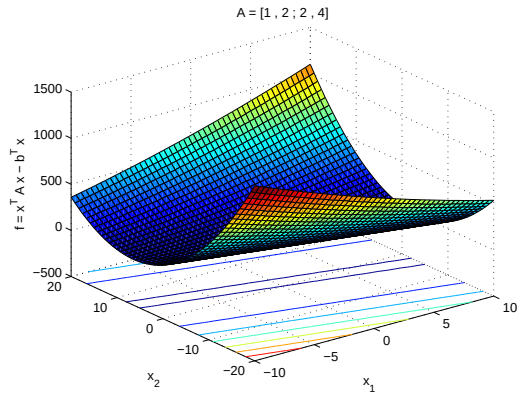


Figura 2.13: Funcţia pătratică asociată unei matrice singulare, pozitiv semidefinite.

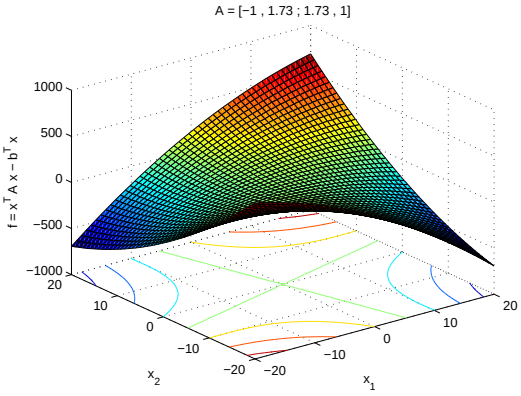


Figura 2.14: Funcţia pătratică asociată unei matrice indefinite.

2.12.2 Metoda celei mai rapide coborâri (a gradientului)

Prima idee de a găsi minimul funcţiei f este de a merge în căutarea lui pe direcţii care corespund ratei celei mai mari de schimbare a funcţiei. Dacă la un moment dat aproximaţia soluţiei este $\mathbf{x}^{(k)}$, atunci direcţia celei mai rapide coborâri este direcţia opusă gradientului în acest punct

$$-\nabla f(\mathbf{x}^{(k)}) = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}. \tag{2.167}$$

Vom defini la fiecare iteraţie eroarea $\mathbf{e}^{(k)}$ şi reziduul $\mathbf{r}^{(k)}$ ca

$$\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}, \tag{2.168}$$

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}. \tag{2.169}$$

Din (2.168) şi (2.163) rezultă că reziduul este eroarea transformată de $\mathbf{A}$ în acelaşi

spaţiu ca al lui $\mathbf{b}$:

$$\mathbf{r}^{(k)} = -\mathbf{A}\mathbf{e}^{(k)}. \tag{2.170}$$

De asemenea, din (2.169) şi (2.167) rezultă că reziduul este direcţia celei mai rapide coborâri:

$$\mathbf{r}^{(k)} = -\nabla f(\mathbf{x}^{(k)}). \tag{2.171}$$

Corecţia primei iteraţii se va face după direcţia reziduului corespunzător iniţializării:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \alpha\mathbf{r}^{(0)}. \tag{2.172}$$

Valoarea scalară α se va alege astfel încât pe direcţia respectivă funcţia să fie minimă. Mai precis $g(\alpha) = f(\mathbf{x}^{(1)})$ să fie minimă. Aceasta implică anularea derivatei de ordinul 1 a funcţiei g:

$$\frac{\mathrm{d}g}{\mathrm{d}\alpha}(\mathbf{x}^{(1)}) = 0 \quad \Rightarrow (\nabla f(\mathbf{x}^{(1)}))^T \cdot \frac{\mathrm{d}\mathbf{x}^{(1)}}{\mathrm{d}\alpha} = 0. \tag{2.173}$$

Din (2.171) rezultă că $\nabla f(\mathbf{x}^{(1)}) = -\mathbf{r}^{(1)}$, iar din (2.172) rezultă că $\mathrm{d}\mathbf{x}^{(1)}/\mathrm{d}\alpha = \mathbf{r}^{(0)}$. În consecinţă, relaţia (2.173) este echivalentă cu faptul că reziduul de la prima iteraţie este ortogonal reziduului iniţial. Urmează că:

$$\begin{aligned}
(\mathbf{r}^{(1)})^T \cdot \mathbf{r}^{(0)} &= 0,\\
(\mathbf{b} - \mathbf{A}\mathbf{x}^{(1)})^T \cdot \mathbf{r}^{(0)} &= 0,\\
[\mathbf{b} - \mathbf{A}(\mathbf{x}^{(0)} + \alpha\mathbf{r}^{(0)})]^T\mathbf{r}^{(0)} &= 0,\\
(\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} - \alpha\mathbf{A}\mathbf{r}^{(0)}))^T \cdot \mathbf{r}^{(0)} &= 0,\\
(\mathbf{b} - \mathbf{A}\mathbf{x}^{(0)})^T \cdot \mathbf{r}^{(0)} - \alpha(\mathbf{A}\mathbf{r}^{(0)})^T\mathbf{r}^{(0)} &= 0,\\
(\mathbf{r}^{(0)})^T\mathbf{r}^{(0)} &= \alpha(\mathbf{r}^{(0)})^T\mathbf{A}^T\mathbf{r}^{(0)}.
\end{aligned}$$

În final, folosind faptul că $\mathbf{A}$ este simetrică, rezultă

$$\alpha = \frac{(\mathbf{r}^{(0)})^T\mathbf{r}^{(0)}}{(\mathbf{r}^{(0)})^T\mathbf{A}\mathbf{r}^{(0)}}. \tag{2.174}$$

Acest raţionament este valabil la fiecare iteraţie, corecţia făcându-se după direcţia ultimului reziduu:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{r}^{(k)}, \tag{2.175}$$

mărimea scalară α_k fiind dedusă din condiţia de ortogonalitate dintre reziduul la iteraţia k şi cel la iteraţia $k+1$:

$$\alpha_k = \frac{(\mathbf{r}^{(k)})^T\mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T\mathbf{A}\mathbf{r}^{(k)}}. \tag{2.176}$$

Acest mod de calcul este descris de următorul algoritm

```
procedură metoda_gradientului_v0(n, A, b, x0, er, maxit, x)
...
xv = x0                          ; iniţializează şirul iteraţiilor
k = 0                            ; iniţializare contor iteraţii
r = b − A · xv                   ; calculează reziduu
cât timp ||r|| > er şi k ≤ maxit
      α = r^T r/(r^T Ar)
      x = xv + αr
      r = b − A · xv
      xv = x                     ; actualizează soluţia veche
      k = k + 1
•
retur
```

În cadrul unei iteraţii, efortul cel mai mare de calcul este dat de produsul dintre o matrice şi un vector. În această variantă, sunt două astfel de produse la fiecare iteraţie.

Dacă înmulţim relaţia (2.175) la stânga cu $-\mathbf{A}$ şi adunăm $\mathbf{b}$ obţinem

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{A}\mathbf{r}^{(k)}, \tag{2.177}$$

O variantă îmbunătăţită a algoritmului este cea care foloseşte această relaţie şi calculează un singur produs matrice vector în cadrul unei iteraţii:

```
procedură metoda_gradientului(n, A, b, x0, er, maxit, x)
...
xv = x0                          ; iniţializează şirul iteraţiilor
k = 0                            ; iniţializare contor iteraţii
r = b − A · xv                   ; calculează reziduu
e = ||r||
cât timp e > er şi k ≤ maxit
      m = A · r
      α = r^T r/(r^T m)
      e = ||r||
      x = xv + αr
      xv = x                     ; actualizează soluţia veche
      k = k + 1
      r = r − αm
•
retur
```

În această variantă de pseudocod, reziduul se calculează recursiv. Dezavantajul este acela că în valoarea reziduului se pot acumula erori de rotunjire. Pentru a corecta acest lucru, este bine ca, periodic, reziduul să se calculeze cu relaţia sa de definiţie, dată de (2.169).

Analiza convergenţei acestei metode se face pe baza *numărului de condiţionare spectrală* care se defineşte ca raportul dintre cea mai mare şi cea mai mică valoare proprie

$$\kappa = \frac{\max \lambda_i}{\min \lambda_i} \geq 1. \tag{2.178}$$

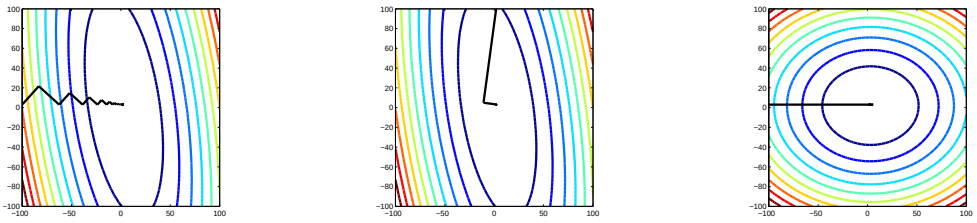
Metoda poate converge rapid, într-un singur pas, dacă punctul de start este ales pe axele elipsoidului sau dacă forma pătratică este sferică (ceea ce corespunde cazului în care toate valorile proprii sunt egale). În rest, convergenţa depinde de numărul de condiţionare κ şi de iniţializare (fig.2.15). Se poate demonstra că eroarea la fiecare iteraţie i este mărginită de

$$\|\mathbf{e}^{(i)}\| \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^i \|\mathbf{e}^{(0)}\| \tag{2.179}$$

2.12.3 Metoda direcţiilor conjugate

Metoda gradientului converge atât de încet deoarece direcţiile de căutare sunt ortogonale şi se întâmplă ca pe parcursul iteraţiilor direcţiile de căutare să se repete. Ar fi de dorit să existe exact n direcţii de căutare, $\mathbf{d}^{(0)}, \mathbf{d}^{(1)}, \ldots, \mathbf{d}^{(n-1)}$ astfel încât soluţia să fie găsită după exact n paşi. La fiecare pas nou

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}, \tag{2.180}$$



$\kappa = 10$ $\kappa = 10$ $\kappa = 1$

Figura 2.15: Convergentă metodei gradientului depinde de iniţializare şi de numărul de condiţionare spectrală. O problemă care are numărul de condiţionare spectrală $\kappa = 1$ (toate valorile proprii sunt egale, iar forma pătratică este sferică) converge într-un singur pas.

iar α_k se va alege astfel încât eroarea $\mathbf{e}^{(k+1)}$ să fie ortogonală pe $\mathbf{d}^{(k)}$ şi, în consecinţă, nici o altă corecţie să nu se mai facă în direcţia lui $\mathbf{d}^{(k)}$:

$$(\mathbf{d}^{(k)})^T\mathbf{e}^{(k+1)} = 0. \tag{2.181}$$

Dacă în (2.180) se scade soluţia exactă din ambii termeni, rezultă

$$\mathbf{e}^{(k+1)} = \mathbf{e}^{(k)} + \alpha_k\mathbf{d}^{(k)}, \tag{2.182}$$

relaţie care înlocuită în (2.181) conduce la

$$\alpha_k = -\frac{(\mathbf{d}^{(k)})^T\mathbf{e}^{(k)}}{(\mathbf{d}^{(k)})^T\mathbf{d}^{(k)}}. \tag{2.183}$$

Relaţia (2.183) nu este utilă deoarece eroarea nu este cunoscută.

Soluţia constă în a face *direcţiile de căutare A-ortogonale* şi nu ortogonale, adică

$$(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(j)} = 0. \qquad j < k. \tag{2.184}$$

Noua cerinţă este ca $\mathbf{e}^{(k+1)}$ să fie A-ortogonal pe $\mathbf{d}^{(k)}$, această condiţie fiind echivalentă cu găsirea unui punct de minim de-a lungul direcţiei $\mathbf{d}^{(k)}$, aşa cum se făcea la metoda gradientului. Într-adevăr, sunt valabile următoarele echivalenţe

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\alpha}f(\mathbf{x}^{(k+1)}) &= 0,\\
(\nabla f(\mathbf{x}^{(k+1)}))^T\frac{\mathrm{d}}{\mathrm{d}\alpha}(\mathbf{x}^{(k+1)}) &= 0,\\
-(\mathbf{r}^{(k+1)})^T\mathbf{d}^{(k)} &= 0,\\
(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{e}^{(k+1)} &= 0.
\end{aligned} \tag{2.185}$$

Rezultă următoarea expresie pentru α_k

$$\alpha_k = -\frac{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{e}^{(k)}}{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)}} = \frac{(\mathbf{d}^{(k)})^T\mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)}}, \tag{2.186}$$

expresie ce se poate calcula. În cazul particular în care direcţiile $\mathbf{d}^{(k)}$ sunt reziduurile, se obţine exact metoda celei mai rapide coborâri.

Această procedură calculează soluţia în exact n paşi. Pentru demonstraţie este util să se exprime eroarea ca o combinaţie liniară a direcţiilor de căutare. Astfel, eroarea iniţială se notează

$$\mathbf{e}^{(0)} = \sum_{j=0}^{n-1}\delta_j\mathbf{d}^{(j)}. \tag{2.187}$$

Valorile δ_j pot fi calculate prin următorul artificiu matematic. Înmulţim la stânga relaţia (2.187) cu $(\mathbf{d}^{(k)})^T\mathbf{A}$ şi folosim faptul că direcţiile sunt A-ortogonale:

$$(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{e}^{(0)} = \sum_{j=0}^{n-1}\delta_j(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(j)} = \delta_k(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)}. \tag{2.188}$$

În consecinţă

$$\delta_k = \frac{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{e}^{(0)}}{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)}} = \frac{(\mathbf{d}^{(k)})^T\mathbf{A}(\mathbf{e}^{(0)} + \sum_{j=0}^{k-1}\alpha_j\mathbf{d}^{(j)})}{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)}} = \frac{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{e}^{(k)}}{(\mathbf{d}^{(k)})^T\mathbf{A}\mathbf{d}^{(k)}}, \tag{2.189}$$

unde am folosit din nou A-ortogonalitatea vectorilor pentru a introduce în paranteză termeni suplimentari ce fac ca expresia finală a lui δ_k să fie exprimată în funcţie de eroarea $\mathbf{e}^{(k)}$. Din (2.186) şi (2.189) rezultă că $\alpha_k = -\delta_k$. În consecinţă eroarea la iteraţia k este

$$\mathbf{e}^{(k)} = \mathbf{e}^{(0)} + \sum_{j=0}^{k-1}\alpha_j\mathbf{d}^{(j)} = \sum_{j=0}^{n-1}\delta_j\mathbf{d}^{(j)} - \sum_{j=0}^{k-1}\delta_j\mathbf{d}^{(j)} = \sum_{j=k}^{n-1}\delta_j\mathbf{d}^{(j)}. \tag{2.190}$$

Este ca şi cum la fiecare iteraţie eroarea are un termen mai puţin, astfel încât după n iteraţii $\mathbf{e}^{(n)} = \mathbf{0}$.

Ceea ce rămâne de făcut acum este găsirea unui set de direcţii A-ortogonale $\mathbf{d}^{(k)}$. Acest lucru se realizează uşor cu ajutorul procedurii Gram-Schmidt conjugate. Această procedură porneşte de la o mulţime de n vectori liniar independenţi $\mathbf{u}^{(0)}, \mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n-1)}$ şi construieşte direcţiile A-conjugate astfel. Prima direcţie este chiar primul vector

$$\mathbf{d}^{(0)} = \mathbf{u}^{(0)}. \tag{2.191}$$

A doua direcţie porneşte de la al doilea vector la care se adaugă un vector orientat pe direcţia anterioară, scalat astfel încât rezultatul să fie A-ortogonal pe direcţia anterioară. Mai precis

$$\mathbf{d}^{(1)} = \mathbf{u}^{(1)} + \beta_{10}\mathbf{d}^{(0)}, \tag{2.192}$$

unde β_{10} se alege astfel încât

$$(\mathbf{d}^{(1)})^T\mathbf{A}\mathbf{d}^{(0)} = 0. \tag{2.193}$$

Această relaţie este echivalentă cu

$$(\mathbf{u}^{(1)} + \beta_{10}\mathbf{d}^{(0)})^T\mathbf{A}\mathbf{d}^{(0)} = 0, \tag{2.194}$$

de unde

$$\beta_{10} = -\frac{(\mathbf{u}^{(1)})^T\mathbf{A}\mathbf{d}^{(0)}}{(\mathbf{d}^{(0)})^T\mathbf{A}\mathbf{d}^{(0)}}. \tag{2.195}$$

Similar, următoarea direcţie este

$$\mathbf{d}^{(2)} = \mathbf{u}^{(2)} + \beta_{20}\mathbf{d}^{(0)} + \beta_{21}\mathbf{d}^{(1)}, \tag{2.196}$$

unde β_{20} şi β_{21} se aleg astfel încât

$$(\mathbf{d}^{(2)})^T\mathbf{A}\mathbf{d}^{(0)} = 0 \quad \text{şi} \quad (\mathbf{d}^{(2)})^T\mathbf{A}\mathbf{d}^{(1)} = 0, \tag{2.197}$$

de unde rezultă

$$\beta_{20} = -\frac{(\mathbf{u}^{(2)})^T\mathbf{A}\mathbf{d}^{(0)}}{(\mathbf{d}^{(0)})^T\mathbf{A}\mathbf{d}^{(0)}}, \quad \beta_{21} = -\frac{(\mathbf{u}^{(2)})^T\mathbf{A}\mathbf{d}^{(1)}}{(\mathbf{d}^{(1)})^T\mathbf{A}\mathbf{d}^{(1)}}. \tag{2.198}$$

În general

$$\mathbf{d}^{(k)} = \mathbf{u}^{(k)} + \sum_{j=0}^{k-1} \beta_{kj}\mathbf{d}^{(j)}, \tag{2.199}$$

unde β_{kj}, definiţi pentru $k > j$, sunt deduşi din condiţiile de A-ortogonalitate impuse direcţiilor, rezultând

$$\beta_{kj} = -\frac{(\mathbf{u}^{(k)})^T\mathbf{A}\mathbf{d}^{(j)}}{(\mathbf{d}^{(j)})^T\mathbf{A}\mathbf{d}^{(j)}}. \tag{2.200}$$

Dificultatea care apare este aceea că toate direcţiile de căutare trebuie memorate pentru a construi o direcţie de căutare nouă.

2.12.4 Metoda gradienţilor conjugaţi

Metoda gradienţilor conjugaţi este metoda direcţiilor conjugate în care direcţiile de căutare sunt construite prin conjugarea reziduurilor:

$$\mathbf{u}^{(k)} = \mathbf{r}^{(k)}. \tag{2.201}$$

Această alegere este justificată din mai multe motive. Unul din motive este intuitiv, inspirat de metoda celei mai rapide coborâri, în care direcţiile de căutare erau direcţiile reziduurilor. Un al doilea motiv este dat de proprietatea reziduului de a fi ortogonal pe direcţiile de căutare anterioare. Într-adevăr, dacă înmulţim la stânga relaţia (2.190) cu $-(\mathbf{d}^k)^T\mathbf{A}$ obţinem

$$-(\mathbf{d}^k)^T\mathbf{A}\mathbf{e}^{(i)} = -\sum_{j=i}^{n-1} \delta_j(\mathbf{d}^k)^T\mathbf{A}\mathbf{d}^{(j)}, \tag{2.202}$$

de unde

$$(\mathbf{d}^k)^T\mathbf{r}^{(i)} = 0, \quad k < i, \tag{2.203}$$

folosind A-ortogonalitatea direcţiilor şi relaţia (2.170). De asemenea, fiecare reziduu este ortogonal pe reziduurile anterioare

$$(\mathbf{r}^k)^T\mathbf{r}^{(i)} = 0, \quad k < i. \tag{2.204}$$

În plus, reziduul $\mathbf{r}^k$ este o combinaţie liniară dintre reziduul anterior şi produsul $\mathbf{A}\mathbf{d}^{(k-1)}$:

$$\mathbf{r}^{(k)} = -\mathbf{A}\mathbf{e}^{(k)} = -\mathbf{A}(\mathbf{e}^{(k-1)} + \alpha_{k-1}\mathbf{d}^{(k-1)}) = \mathbf{r}^{(k-1)} - \alpha_{k-1}\mathbf{A}\mathbf{d}^{(k-1)}. \tag{2.205}$$

Calculând coeficienţii β în acest caz, rezultă

$$\beta_{kj} = -\frac{(\mathbf{r}^{(k)})^T \mathbf{A}\mathbf{d}^{(j)}}{(\mathbf{d}^{(j)})^T \mathbf{A}\mathbf{d}^{(j)}}. \tag{2.206}$$

Pentru a explicita termenul de la numărător, vom înmulţi relaţia (2.205) la stânga cu $(\mathbf{r}^{(i)})^T$:

$$(\mathbf{r}^{(i)})^T \mathbf{r}^{(k+1)} = (\mathbf{r}^{(i)})^T \mathbf{r}^{(k)} - \alpha_k (\mathbf{r}^{(i)})^T \mathbf{A}\mathbf{d}^{(k)}, \tag{2.207}$$

de unde

$$\alpha_k (\mathbf{r}^{(i)})^T \mathbf{A}\mathbf{d}^{(k)} = (\mathbf{r}^{(i)})^T \mathbf{r}^{(k)} - (\mathbf{r}^{(i)})^T \mathbf{r}^{(k+1)}. \tag{2.208}$$

Pentru $i \neq k$, membrul drept al acestei relaţii este nenul doar pentru cazul $i = k+1$, când valoarea lui este $-(\mathbf{r}^{(i)})^T \mathbf{r}^{(i)}/\alpha_{i-1}$. Rezultă că valorile β sunt nenule doar dacă $i = k+1$:

$$\beta_{kj} = \begin{cases} \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{\alpha_{k-1}(\mathbf{d}^{(k-1)})^T \mathbf{A}\mathbf{d}^{(k-1)}} & k = i-1, \\ 0 & k < i-1 \end{cases} \tag{2.209}$$

Din acest motiv, nu mai este necesar ca valorile β să fie notate cu doi indici. Vom renota

$$\beta_k = \beta_{k,k-1} = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{\alpha_{k-1}(\mathbf{d}^{(k-1)})^T \mathbf{A}\mathbf{d}^{(k-1)}} = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{d}^{(k-1)})^T \mathbf{r}^{(k-1)}} = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{r}^{(k-1)})^T \mathbf{r}^{(k-1)}}. \tag{2.210}$$

În concluzie, metoda gradienţilor conjugaţi se bazează pe următoarele şase relaţii:

$$\begin{aligned} \mathbf{d}^{(0)} &= \mathbf{r}^{(0)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)} \\ \alpha_k &= \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{d}^{(k)})^T \mathbf{A}\mathbf{d}^{(k)}} \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \alpha_k \mathbf{A}\mathbf{d}^{(k)} \\ \beta_{k+1} &= \frac{(\mathbf{r}^{(k+1)})^T \mathbf{r}^{(k+1)}}{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}} \\ \mathbf{d}^{(k+1)} &= \mathbf{r}^{(k+1)} + \beta_{k+1} \mathbf{d}^{(k)} \end{aligned}$$

Algoritmul este complet după n iteraţii (fig.2.16). În practică însă metoda este folosită pentru probleme atât de mari încât nu ar fi fezabil să se execute toate cele n iteraţii. De aceea, iteraţiile în algoritmul de mai jos sunt executate într-un ciclu cu test şi nu într-un ciclu cu contor. Din acest motiv, se mai spune că *metoda este semi-iterativă*, şirul iteraţiilor nu tinde către soluţia exactă atunci când numărul iteraţiilor tinde la infinit, ci, într-o aritmetică exactă, soluţia exactă se obţine după exact n iteraţii.

Pseudocodul de mai jos implementează algoritmul descris.

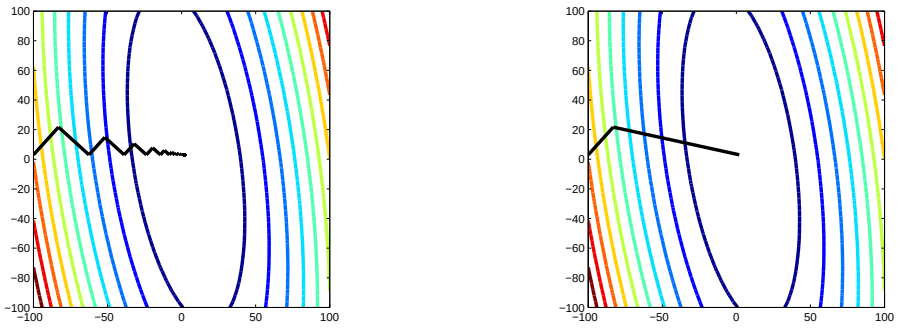


Figura 2.16: Convergenţa metodei gradientului depinde de iniţializare şi de numărul de condiţionare spectrală (stânga). Metoda gradienţilor conjugaţi converge în exact n paşi, indiferent de iniţializare şi de numărul de condiţionare spectrală (dreapta).

`procedură` metoda_gradienţilor_conjugaţi$(n, A, b, x0, er, maxit, x)$
...
$\mathbf{xv} = \mathbf{x0}$; iniţializează şirul iteraţiilor
$k = 0$; iniţializare contor iteraţii
$\mathbf{r} = \mathbf{b} - \mathbf{A} \cdot \mathbf{xv}$; calculează reziduu
`cât timp` $\|\mathbf{r}\| > er$ `şi` $k \leq maxit$
    `dacă` $k = 0$
        $\mathbf{d} = \mathbf{r}$
    `altfel`
        $\beta = \mathbf{r}^T\mathbf{r}/(\mathbf{rv}^T \cdot \mathbf{rv})$
        $\mathbf{d} = \mathbf{r} + \beta\mathbf{dv}$
    •
    $\alpha = \mathbf{r}^T\mathbf{r}/(\mathbf{d}^T\mathbf{Ad})$
    $\mathbf{x} = \mathbf{xv} + \alpha\mathbf{d}$
    $\mathbf{rv} = \mathbf{r}$
    $\mathbf{r} = \mathbf{rv} - \alpha\mathbf{Ad}$
    $\mathbf{xv} = \mathbf{x}$
    $\mathbf{dv} = \mathbf{d}$
    $k = k + 1$
•
`retur`

Evident că algoritmul de mai sus se poate îmbunătăţi, calculând la o iteraţie o singură dată produsul matrice - vector $\mathbf{Ad}$ şi produsul scalar $\mathbf{r}^T\mathbf{r}$.

Faptul că metoda gradienţilor conjugaţi necesită un efort mult mai mic decât metoda

direcţiilor conjugate poate fi justificat şi făcând un raţionament bazat pe spaţii vectoriale. Să notăm cu $\mathcal{D}_k$ subspaţiul vectorial generat de direcţiile de căutare până la iteraţia k:

$$\mathcal{D}_k = \text{span}\{\mathbf{d}^{(0)}, \mathbf{d}^{(1)}, \ldots, \mathbf{d}^{(k-1)}\}. \tag{2.211}$$

În metoda gradienţilor conjugaţi, direcţiile de căutare sunt construite din reziduuri:

$$\begin{aligned} \mathbf{d}^{(0)} &= \mathbf{r}^{(0)}, \\ \mathbf{d}^{(1)} &= \mathbf{r}^{(1)} + \beta_1 \mathbf{d}^{(0)} = \mathbf{r}^{(1)} + \beta_1 \mathbf{r}^{(0)}, \\ \mathbf{d}^{(2)} &= \mathbf{r}^{(2)} + \beta_2 \mathbf{d}^{(1)} = \mathbf{r}^{(2)} + \beta_2 \mathbf{r}^{(1)} + \beta_2 \beta_1 \mathbf{r}^{(0)} \\ &\ldots \end{aligned} \tag{2.212}$$

şi în consecinţă subspaţiul $\mathcal{D}_k$ este în acelaşi timp şi subspaţiul generat de reziduuri

$$\mathcal{D}_k = \text{span}\{\mathbf{r}^{(0)}, \mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(k-1)}\}. \tag{2.213}$$

Pe de altă parte, din relaţia (2.205) se observă că reziduul la iteraţia k este o combinaţie liniară dintre reziduul la iteraţia $k-1$ şi produsul dintre matricea $\mathbf{A}$ şi direcţia de căutare la iteraţia $k-1$. Aceasta înseamnă că subspaţiul $\mathcal{D}_k$ este reuniunea dintre subspaţiul $\mathcal{D}_{k-1}$ şi subspaţiul $\mathbf{A}\mathcal{D}_{k-1}$.

$$\mathcal{D}_k = \mathcal{D}_{k-1} \cup \mathbf{A}\mathcal{D}_{k-1}. \tag{2.214}$$

De aici rezultă că

$$\mathcal{D}_k = \mathcal{D}_0 \cup \mathbf{A}\mathcal{D}_0 \cup \mathbf{A}^2\mathcal{D}_0 \cup \mathbf{A}^{k-1}\mathcal{D}_0, \tag{2.215}$$

deci $\mathcal{D}_k$ este un *subspaţiu Krylov*, creat prin aplicarea repetată a unei matrice unui vector:

$$\begin{aligned} \mathcal{D}_k &= \text{span}\{\mathbf{d}^{(0)}, \mathbf{A}\mathbf{d}^{(0)}, \mathbf{A}^2\mathbf{d}^{(0)} \cdots, \mathbf{A}^{k-1}\mathbf{d}^{(0)}\} = \\ &= \text{span}\{\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \mathbf{A}^2\mathbf{r}^{(0)} \cdots, \mathbf{A}^{k-1}\mathbf{r}^{(0)}\}. \end{aligned} \tag{2.216}$$

Aceasta înseamnă că subspaţiul $\mathbf{A}\mathcal{D}_k$ este inclus în subspaţiul $\mathcal{D}_{k+1}$ şi deoarece reziduul $\mathbf{r}^{(k+1)}$ este ortogonal pe $\mathcal{D}_{k+1}$ (conform relaţiei (2.203)) rezultă că $\mathbf{r}^{(k+1)}$ este deja A-ortogonal pe toate direcţiile de căutare cu excepţia direcţiei $\mathbf{d}^{(k)}$. De aceea procedura Gram-Schmidt necesită la fiecare iteraţie calculul unui singur coeficient.

Algoritmul gradienţilor conjugaţi poate fi dedus şi din algoritmul Lanczos care este dedicat calculului valorilor proprii ale unei matrice simetrice. Mărimile deja calculate în algoritm (reziduurile, direcţiile de căutare, coeficienţii β) pot fi folosiţi pentru estimarea numărului de condiţionare spectrală a matricei $\mathbf{A}$. Detalii legate de acest aspect pot fi găsite în [3].

2.13 Precondiţionare

Convergenţa metodelor nestaţionare depinde de numărul de condiţionare spectrală al matricei coeficienţilor. Una dintre marile descoperiri în domeniul algoritmilor numerici a fost faptul că, în multe cazuri, problema iniţială poate fi transformată într-una echivalentă care are proprietaţi îmbunătăţite considerabil. Problema echivalentă se obţine de exemplu prin *precondiţionarea la stânga* a problemei iniţiale

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.217}$$

adică prin rezolvarea problemei echivalente din punct de vedere al soluţiei

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}, \tag{2.218}$$

unde $\mathbf{M}$ este o matrice nesingulară, numită *matrice de precondiţionare* sau *precondiţionator*. Dacă se rezolvă iterativ sistemul (2.218), convergenţa depinde de proprietăţile matricei $\mathbf{M}^{-1}\mathbf{A}$ şi nu de proprietaţile matricei $\mathbf{A}$ a sistemului iniţial. Dacă precondiţionatorul $\mathbf{M}$ este ales corespunzător, atunci sistemul (2.218) ar putea fi rezolvat mult mai repede decât sistemul (2.217).

Aşa cum a mai fost explicat şi cu alte ocazii, calculul matricei $\mathbf{M}^{-1}\mathbf{A}$ nu se face explicit deoarece nu ar fi eficient, ci prin rezolvarea unui sistem de ecuaţii de forma

$$\mathbf{M}\mathbf{y} = \mathbf{A} \tag{2.219}$$

Este evident că în cazurile extreme $\mathbf{M} = \mathbf{I}$ şi $\mathbf{M} = \mathbf{A}$ nu există nici un câştig în rezolvare. Între aceste două situaţii pot exista precondiţionatori utili, suficient de apropiaţi de $\mathbf{A}$ într-un anumit sens astfel încât iteraţiile sistemului (2.218) să conveargă mai repede decât iteraţiile sistemului (2.217). O condiţie puternică pentru un bun precondiţionator este ca valorile proprii ale matricei $\mathbf{M}^{-1}\mathbf{A}$ să fie apropiate de 1 şi ca norma $\|\mathbf{M}^{-1}\mathbf{A} - \mathbf{I}\|_2$ să fie mică [16].

O problemă echivalentă cu (2.217) se poate obţine si prin *precondiţionarea la dreapta*

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}, \tag{2.220}$$

unde $\mathbf{x} = \mathbf{M}^{-1}\mathbf{y}$. Uneori se folosesc ambele tipuri de precondiţionare simultan.

Dacă matricea $\mathbf{A}$ este simetrică şi pozitiv definită, atunci precondiţionarea se face astfel încât să se păstreze această proprietate. De exemplu, alegem şi matricea de precondiţionare $\mathbf{M}$ simetrică şi pozitiv definită, scrisă cu ajutorul unei matrice $\mathbf{C}$ astfel încât $\mathbf{M} = \mathbf{C}\mathbf{C}^T$. Atunci, sistemul precondiţionat de rezolvat este

$$\left[\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-T}\right]\mathbf{C}^T\mathbf{x} = \mathbf{C}^{-1}\mathbf{b}, \tag{2.221}$$

unde $\mathbf{C}^{-T} = (\mathbf{C}^{-1})^T = (\mathbf{C}^T)^{-1}$, iar matricea din paranteza dreapta este simetrică şi pozitiv definită.

Există o mare varietate de idei de precondiţionare pentru rezolvarea iterativă a sistemelor algebrice liniare. Printre cele mai cunoscute metode de precondiţionare sunt:

- *Precondiţionarea Jacobi (sau scalarea diagonală)* - în care $\mathbf{M} = \text{diag}(\mathbf{A})$ este o matrice diagonală care are pe diagonală valorile matricei iniţiale. Matricea de precondiţionare astfel construită trebuie să fie nesingulară. O generalizare a ei este alegerea $\mathbf{M} = \text{diag}(\mathbf{c})$, unde $\mathbf{c}$ este un vector ales convenabil.

- *Factorizarea incompletă Cholesky sau LU* în care se aplică procedura de factorizare, dar factorii nu sunt calculaţi complet, valorile care ar umple matricea nefiind luate în considerare. Matricea de precondiţionare se obţine ca produsul acestor factori incompleţi.

Aceste două exemple de precondiţionatori nu fac nicio referire la problema iniţială care a generat sistemul (2.217). Cel mai bun sfat general care poate fi dat pentru construcţia precondiţionatorilor este de a examina problema şi de a încerca să se construiască precondiţionatorul pe baza unei versiuni mai simple a problemei. Pe această idee se bazează *metodele multigrid*, care vor fi discutate ulterior. Metodele Jacobi şi SSOR furnizează de multe ori precondiţionatori foarte buni pentru metodele de tip multigrid.

Capitolul 3

Algoritmi numerici pentru analiza circuitelor electrice rezistive liniare

Cel mai simplu exemplu din ingineria electrică ce conduce la un sistem de ecuaţii algebrice liniare îl reprezintă analiza circuitelor electrice rezistive liniare. Un circuit rezistiv liniar este un circuit ce conţine rezistoare, surse ideale de tensiune şi curent precum şi surse comandate liniar. Problema fundamentală a analizei acestor circuite are ca date topologia circuitului şi valorile parametrilor (rezistenţele, valorile surselor), şi urmăreşte calculul curenţilor şi tensiunilor din fiecare latură.

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

De aceea, în acest capitol se va explica modul în care se poate concepe un algoritm pentru rezolvarea problemei analizei circuitelor rezistive liniare folosind tehnica nodală. În prima parte se consideră cazul cel mai simplu, în care fiecare latură a circuitul conţine un rezistor şi eventual o sursă ideală de tensiune. Apoi, se discută modul în care algoritmul trebuie modificat pentru a putea fi aplicat unui circuit general, care conţine şi surse ideale de curent, surse comandate liniar şi laturi alcătuite numai din surse ideale de tensiune.

3.1 Tehnica nodală

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

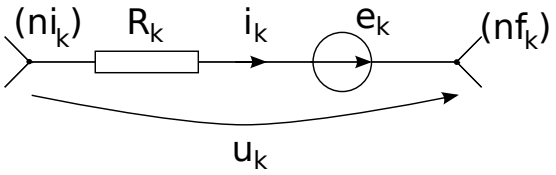


Figura 3.1: Latura standard.

Datele problemei sunt

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

Se cere să se calculeze

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

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

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

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

[1]Notaţia $\sum^A$ reprezintă o sumă alegebrică, adică $\sum^A x_k = \sum \varepsilon_k x_k$, unde $\varepsilon_k = \pm 1$.

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

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

şi L relaţii Joubert, scrise pentru toate laturile

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

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

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

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

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

$$\begin{array}{lcll}
\mathbf{u} & = & [\ u_1 \quad u_2 \quad \ldots \quad u_L\ ]^T \in \mathbb{R}^{L\times 1} & \text{vectorul tensiunilor laturilor,} \\
\mathbf{i} & = & [\ i_1 \quad i_2 \quad \ldots \quad i_L\ ]^T \in \mathbb{R}^{L\times 1} & \text{vectorul curenţilor prin laturi,} \\
\mathbf{v} & = & [\ v_1 \quad v_2 \quad \ldots \quad v_{N-1}\ ]^T \in \mathbb{R}^{(N-1)\times 1} & \text{vectorul potenţialelor nodurilor,} \\
\mathbf{e} & = & [\ e_1 \quad e_2 \quad \ldots \quad e_L\ ]^T \in \mathbb{R}^{L\times 1} & \text{vectorul tensiunilor electromotoare,} \\
\mathbf{R} & = & \mathrm{diag}([\ R_1 \quad R_2 \quad \ldots \quad R_L\ ]) \in \mathbb{R}^{L\times L} & \text{matricea diagonală a rezistenţelor laturilor.}
\end{array} \tag{3.5}$$

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

$$\mathbf{Ai} = \mathbf{0}, \tag{3.6}$$

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

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

Relaţia Kirchhoff II în forma (3.4) se scrie

$$\mathbf{u} = \mathbf{A}^T\mathbf{v}, \tag{3.7}$$

iar relaţiile lui Joubert (3.3) se scriu compact

$$\mathbf{u} = \mathbf{Ri} - \mathbf{e}. \tag{3.8}$$

Dacă matricea $\mathbf{R}$ este inversabilă (lucru adevărat dacă toate rezistenţele sunt presupuse nenule) atunci din (3.8) rezultă că

$$\mathbf{i} = \mathbf{R}^{-1}(\mathbf{u} + \mathbf{e}). \tag{3.9}$$

Înlocuind (3.9) şi (3.7) în (3.6), rezultă ecuaţia matriceală satisfăcută de vectorul potenţialelor

$$\mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T\mathbf{v} = -\mathbf{A}\mathbf{R}^{-1}\mathbf{e}. \tag{3.10}$$

Matricea coeficienţilor sistemului algebric liniar de rezolvat

$$\mathbf{G} = \mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T \quad \in \mathbb{R}^{(N-1)\times(N-1)} \tag{3.11}$$

este numită *matricea conductanţelor nodale*. Termenii ei au următoarea semnificaţie: orice termen diagonal G_{ii} reprezintă suma conductanţelor laturilor care concură la nodul i, iar orice termen nediagonal G_{ij} cu $i \neq j$ reprezintă suma conductanţelor laturilor care unesc direct nodul i cu nodul j, luată cu semnul minus. Altfel scris,

$$G_{ii} = \sum_{k\in(i)} \frac{1}{R_k}, \tag{3.12}$$

$$G_{ij} = -\sum_{k\in(i);k\in(j)} \frac{1}{R_k} \qquad \text{pentru} \quad i \neq j. \tag{3.13}$$

Termenul liber al sistemului de ecuaţii

$$\mathbf{t} = -\mathbf{A}\mathbf{R}^{-1}\mathbf{e} \quad \in \mathbb{R}^{(N-1)\times 1} \tag{3.14}$$

este numit *vectorul injecţiilor de curent*. 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 intră în nodul k, şi cu semnul minus în caz contrar:

$$t_k = \sum_{m\in(k)}{}^{A} \frac{e_m}{R_m}. \tag{3.15}$$

Este important să remarcăm că matricea $\mathbf{G}$ este simetrică şi pozitiv definită dacă rezistenţele sunt pozitive. Deoarece $\mathbf{R}$ este o matrice diagonală, inversa ei este tot o matrice diagonală, având pe diagonală valorile conductanţelor laturilor

$$\mathbf{R}^{-1} = \mathrm{diag}([\ 1/R_1 \quad 1/R_2 \quad \ldots \quad 1/R_L\ ]). \tag{3.16}$$

Transpusa matricei conductanţelor nodale va fi

$$\mathbf{G}^T = \left(\mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T\right)^T = \left(\mathbf{A}^T\right)^T \left(\mathbf{R}^{-1}\right)^T (\mathbf{A})^T = \mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T = \mathbf{G}, \tag{3.17}$$

deci ea este o matrice simetrică. Relaţiile (3.12) şi (3.13) indică faptul că ea este şi diagonal dominantă.

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

$$\mathbf{x}^T\mathbf{G}\mathbf{x} = \mathbf{x}^T\mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T\mathbf{x} = \mathbf{y}^T\mathbf{R}^{-1}\mathbf{y} = \sum_{k=1}^{L}\frac{y_k^2}{R_k} > 0, \tag{3.18}$$

unde am notat $\mathbf{y} = \mathbf{A}^T\mathbf{x}$ un vector coloană de componente y_k, $k = 1, \ldots, L$. Inegalitatea demonstrată în (3.18) este strictă, ea ar putea fi zero doar dacă vectorul $\mathbf{y}$, şi în consecinţă vectorul $\mathbf{x}$ este nul, ceea ce contrazice ipoteza făcută. În concluzie, matricea conductanţelor nodale este pozitiv definită dacă rezistenţele laturilor sunt strict pozitive.

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

3.1.1 Structuri de date

În conceperea unui algoritm, stabilirea structurilor de date ce vor fi folosite este de asemenea o etapă foarte importantă. Numele datelor ce le vom folosi în algoritm vor fi alese în concordanţă cu teoria prezentată în paragraful anterior.

Declaraţii posibile pentru datele problemei sunt

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

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

```
; declaraţii date - varianta B
înregistrare circuit
```

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

Un alt aspect important este acela că matricea conductanţelor nodale şi vectorul injecţiilor de curent sunt rare. De exemplu, presupunând că în medie sunt 4 laturi care concură la un nod, rezultă o densitate a matricei egală cu $5n/n^2 = 5/n$, foarte mică chiar şi pentru probleme mici (pentru $N \approx 1000$ rezultă $d = 0.5\ \%$). De aceea, pentru a concepe un algoritm eficient (atât din punct de vedere al timpului de calcul cât şi al memoriei, vor trebui folosite tehnici de matrice rare pentru stocarea ei. Altfel, programul rezultat poate deveni ineficient chiar la valori mici (≈ 1000) ale numărului de noduri. Totuşi, pentru a păstra simplă descrierea algoritmului, în cele ce urmează vom păstra acelaşi tip de declaraţii şi pentru matricile rare şi vectorii rari. În consecinţă, variabile mai importante care vor apărea în algoritm sunt:

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

3.1.2 Etapa de preprocesare

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

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

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

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

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

Figura 3.2: Contribuţia unei laturi k la matricea conductanţelor nodale (stânga) şi la vectorul injecţiilor de curent (dreapta).

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

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

Există mai multe variante de a asambla sistemul de ecuaţii. O variantă ar fi cea în care se parcurg nodurile şi se scriu ecuaţiile una câte una. Aşa ar face şi o persoană care ar aplica această metodă, cu creionul pe hârtie, pentru a rezolva o problemă de dimensiuni extrem de mici. În această abordare trebuie identificate care sunt laturile ce ating un nod. Numărul lor variază de la nod la nod, iar algoritmul corespunzător este destul de costisitor necesitând analiza grafului circuitului. O altă abordare se bazează pe parcurgerea laturilor şi adunarea contribuţiilor acestora la sistem. Această variantă este mult mai simplu de conceput deoarece fiecare latură are exact două noduri. Din acelaşi motiv şi descrierea circuitului a fost orientată pe laturi şi nu pe noduri. Contribuţia fiecărei laturi la matricea conductanţelor nodale şi la vectorul injecţiilor de curent este ilustrată în fig. 3.2.

Algoritmul metodei nodale este următorul

```
procedură nodalRE_v1 (circuit, G, t)
; asamblează sistemul de ecuaţii pentru un circuit cu laturi de tip
; sursă reală de tensiune, folosind tehnica nodală
```

; parametri de intrare:
; circuit - structură de date ce descrie circuitul
; parametri de ieşire:
; G - matricea conductanţelor nodale şi
; t - vectorul injecţiilor de curent
; declaraţii
....
L = circuit.L ; pentru simplificarea scrierii algoritmului
N = circuit.N
ni = circuit.ni
nf = circuit.nf
R = circuit.R
e = circuit.e
; anulează componentele matricei G şi ale vectorului termenilor liberi t
G = 0
t = 0
; asamblează sistem
pentru $k = 1, L$; parcurge laturi
    $i = \text{ni}_k$; nodul iniţial al laturii k
    $j = \text{nf}_k$; nodul final al laturii k
    $G_{ii} = G_{ii} + 1/R_k$
    $G_{jj} = G_{jj} + 1/R_k$
    $G_{ij} = G_{ij} - 1/R_k$
    $G_{ji} = G_{ji} - 1/R_k$
    $t_i = t_i - e_k/R_k$
    $t_j = t_j + e_k/R_k$
•
retur

O primă observaţie este aceea că, pentru simplificarea pseudocodului, am preferat să nu scriem explicit modul în care se anulează componentele matricei coeficienţilor şi ale vectorului termenilor liberi, înainte de asamblarea propriu-zisă. Pentru a sugera însă că se fac mai multe operaţii asupra componentelor am folosit litere aldine. Atragem atenţia însă că o variantă de tipul

pentru i = 1,N
    pentru j = 1,N
        $G_{ij} = 0$
    •

•

scrisă pentru "instrucţiunea" $\mathbf{G} = \mathbf{0}$ va face ca matricea coeficienţilor să devină o matrice plină. Această instrucţiune trebuie scrisă explicit în funcţie de modul de memorare al matricei rare. De asemenea, pentru a evita repetarea unor calcule, se pot memora valorile $1/R_k$ şi e_k/R_k.

O altă variantă de implementare poate folosi asamblarea matricei de incidenţă. Calculul matricei conductanţelor nodale şi a vectorului injecţiilor de curent se va face apoi folosind produse de matrice sau produse matrice-vectori, scrise pentru matrice rare.

`procedură` nodalRE_v2 (circuit, G, t)
; asamblează sistemul de ecuaţii pentru un circuit cu laturi de tip
; sursă reală de tensiune, folosind tehnica nodală
; parametri de intrare:
; circuit - structură de date ce descrie circuitul
; parametri de ieşire:
; G - matricea conductanţelor nodale şi
; t - vectorul injecţiilor de curent
; declaraţii
....
L = circuit.L ; pentru simplificarea scrierii algoritmului
N = circuit.N
ni = circuit.ni
nf = circuit.nf
R = circuit.R
e = circuit.e
; anulează componentele:
$\mathbf{A} = \mathbf{0}$; matricei incidenţe laturi noduri
$\mathbf{Glat} = \mathbf{0}$; matricei diagonale $\mathbf{R}^{-1}$
; asamblează sistem
`pentru` $k = 1, L$; parcurge laturi
$i = \text{ni}_k$; nodul iniţial al laturii k
$j = \text{nf}_k$; nodul final al laturii k
$A_{ik} = -1$
$A_{jk} = +1$
$\text{Glat}_{kk} = 1/R_k$

•

$\mathbf{G} = \mathbf{A} * \mathbf{Glat} * \mathbf{A}^T$; apel proceduri speciale pentru matrice rare

$\mathbf{t} = -\mathbf{A} * \mathbf{Glat} * \mathbf{e}$
retur

Varianta a doua a algoritmului se poate dovedi mult mai eficientă decât prima variantă dacă implementarea se face în medii de calcul precum Matlab.

3.1.3 Etapa de rezolvare

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

Pentru rezolvare avem la dispoziţie algoritmii pregătiţi în capitolul 1. De exemplu, dacă vrem să folosim metoda Gauss, atunci apelul ei se face astfel

Gauss $(N-1, G, t, v)$
$v_N = 0$

Vectorul soluţie întors reprezintă potenţialele a $N-1$ noduri. Potenţialul ultimului nod a fost considerat zero şi de aceea, pentru completitudine, este adaugată atribuirea $v_N = 0$ imediat după apelul procedurii de rezolvare. O altă observaţie este aceea că matricea conductanţelor nodale este diagonal dominantă, ceea ce înseamnă că, în cazul folosirii metodei Gauss, pivotarea nu este necesară nici pentru diminuarea erorilor de rotunjire.

Algoritmul Gauss prezentat în paragraful 2.4.2 nu este însă conceput pentru matrice rare. Într-o astfel de înlănţuire a pseudocodului, chiar dacă sistemul a fost generat cu tehnici de matrice rare, rezolvarea lui cu această procedură strică raritatea. De fapt, este mai mult decât atât, matricea devine plină cu zerouri, dar memorată cu structuri de matrice rare, ceea ce este mult mai costisitor decât memorarea matricei pline.

În conceperea unui algoritm pentru probleme reale trebuie acordată o atenţie deosebită memorării cu eficienţă a structurilor de date şi efectuarea unor instrucţiuni adecvate cu acestea.

3.1.4 Etapa de postprocesare

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

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

Este posibil ca, datorită erorilor de rotunjire, valorile calculate ale puterilor consumată şi generată să nu fie identice. Acest lucru se întâmplă mai ales dacă matricea sistemului este prost condiţionată numeric, caz ce poate apărea dacă valorile rezistenţelor sunt foarte diferite.

3.2 Tratarea surselor ideale de curent

Tratarea surselor ideale de curent se face cu uşurinţă dacă se consideră latura standard de tipul celei din fig. 3.3.

Şi în acest caz vom presupune că toate rezistenţele R_k sunt nenule. În particular, această latură poate fi o sursă reală de tensiune (dacă $J_k = 0$), un rezistor (dacă $J_k = 0$ şi $E_k = 0$), o sursă ideală de curent (dacă $1/R_k = 0$).

Ecuaţiile Kirchhoff I şi II se scriu sub aceeaşi formă:

$$\mathbf{Ai} = \mathbf{0}, \tag{3.19}$$

$$\mathbf{u} = \mathbf{A}^T\mathbf{v}. \tag{3.20}$$

Relaţiile între tensiunile şi curenţii unei laturi sunt acum

$$u_k = R_k(i_k - J_k) - e_k, \quad k = 1, \ldots, L. \tag{3.21}$$

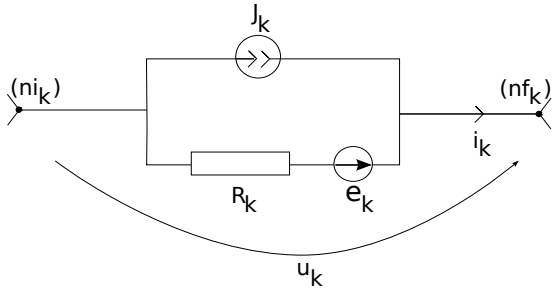


Figura 3.3: Latura standard.

Folosind aceleaşi notaţii (3.5), relaţiile de mai sus se scriu compact

$$\mathbf{u} = \mathbf{R}(\mathbf{i} - \mathbf{J}) - \mathbf{e}, \quad k = 1, \ldots, L, \tag{3.22}$$

unde $\mathbf{J} = (J_k)_{k=1,L}$ este vectorul curenţilor electromotori prin laturile circuitului. Rezultă curenţii prin laturile standard

$$\mathbf{i} = \mathbf{R}^{-1}(\mathbf{u} + \mathbf{e}) + \mathbf{J}, \tag{3.23}$$

şi ecuaţia matriceală în potenţiale:

$$\mathbf{A}\mathbf{R}^{-1}\mathbf{A}^T\mathbf{v} = -\mathbf{A}\mathbf{R}^{-1}\mathbf{e} - \mathbf{A}\mathbf{J}. \tag{3.24}$$

Comparând ecuaţia rezultată cu ecuaţia (3.10) obţinută în cazul în care latura standard conţinea numai o sursă reală de tensiune, se observă că matricea coeficienţilor este neschimbată, iar în membrul drept apare un termen suplimentar, corespunzător sursei ideale de curent. Acum, un element al termenului liber

$$\mathbf{t} = -\mathbf{A}\mathbf{R}^{-1}\mathbf{e} - \mathbf{A}\mathbf{J} \quad \in \mathbb{R}^{(N-1)\times 1} \tag{3.25}$$

se calculează ca suma algebrică a unor termeni de tipul e_m/R_m pentru toate laturile m care concură la nodul k, cu plus dacă săgeata internă a sursei de tensiune electromotoare de pe latura m intra în nodul k, şi cu semnul minus în caz contrar şi suma algebrică a unor termeni de tipul J_m cu plus dacă săgeata internă a sursei de curent de pe latura m intra în nodul k, şi cu semnul minus în caz contrar.

Pentru a simplifica algoritmul, vom presupune că tensiunea electromotoare e_k şi curentul electromotor J_k sunt au sensurile de referinţă la fel faţă de nod. Modificările algoritmului vor afecta etapa de preprocesare unde vor trebui citite şi valorile curenţilor electromotori şi vor trebui corectaţi termenii liberi ai sistemului de rezolvat.

`funcţie` citire_date_REJ ()

...

`citeşte` circuit.N, circuit.L
`pentru` $k = 1$,circuit.L
    `citeşte` circuit.ni$_k$, circuit.nf$_k$, circuit.R_k, circuit.e_k, circuit.J_k
•
`întoarce` circuit

`procedură` nodalREJ_v1 (circuit, G, t)
; asamblează sistemul de ecuaţii pentru un circuit cu laturi de tip
; sursă reală de tensiune în paralel cu sursă ideală de curent,
; folosind tehnica nodală
; parametri de intrare:
; circuit - structură de date ce descrie circuitul
; parametri de ieşire:
; G - matricea conductanţelor nodale şi
; t - vectorul injecţiilor de curent
; declaraţii
....
L = circuit.L ; pentru simplificarea scrierii algoritmului
N = circuit.N
ni = circuit.ni
nf = circuit.nf
R = circuit.R
e = circuit.e
J = circuit.J
; anulează componentele matricei G şi a vectorului termenilor liberi t
G = 0
t = 0
; asamblează sistem
`pentru` $k = 1, L$; parcurge laturi
    i = ni$_k$; nodul iniţial al laturii k
    j = nf$_k$; nodul final al laturii k
    $G_{ii} = G_{ii} + 1/R_k$
    $G_{jj} = G_{jj} + 1/R_k$
    $G_{ij} = G_{ij} - 1/R_k$
    $G_{ji} = G_{ji} - 1/R_k$
    $t_i = t_i - e_k/R_k - J_k$
    $t_j = t_j + e_k/R_k + J_k$
•

retur

procedură nodalREJ_v2 (circuit, G, t)
; asamblează sistemul de ecuaţii pentru un circuit cu laturi de tip
; sursă reală de tensiune în paralel cu sursă ideală de curent,
; folosind tehnica nodală
; parametri de intrare:
; circuit - structură de date ce descrie circuitul
; parametri de ieşire:
; G - matricea conductanţelor nodale şi
; t - vectorul injecţiilor de curent
; declaraţii
....
L = circuit.L ; pentru simplificarea scrierii algoritmului
N = circuit.N
ni = circuit.ni
nf = circuit.nf
R = circuit.R
e = circuit.e
J = circuit.J
; anulează componentele:
$\mathbf{A} = \mathbf{0}$; matricei incidenţelor laturi noduri
$\mathbf{Glat} = \mathbf{0}$; matricei diagonale $\mathbf{R}^{-1}$
; asamblează sistem
pentru $k = 1, L$; parcurge laturi
$i = \text{ni}_k$; nodul iniţial al laturii k
$j = \text{nf}_k$; nodul final al laturii k
$A_{ik} = -1$
$A_{jk} = +1$
$\text{Glat}_{kk} = 1/R_k$
•
$\mathbf{G} = \mathbf{A} * \mathbf{Glat} * \mathbf{A}^T$; apel proceduri speciale pentru matrice rare
$\mathbf{t} = -\mathbf{A} * \mathbf{Glat} * \mathbf{e} - \mathbf{A} * \mathbf{J}$
retur

Etapa de postprocesare trebuie adaptată noii configuraţii a laturii standard:

procedură postprocesare_circuitREJ (circuit, v)
; declaraţii

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

3.3 Tratarea surselor ideale de tensiune

Vom presupune acum că circuitul are şi surse ideale de tensiune. Laturile circuitului sunt fie de tip sursă reală de tensiune în paralel cu sursă ideală de curent (vom numi această latură de tip "REJ"), fie de tip sursă ideală de tensiune (tip "E") (fig.3.4). Pentru a simplifica scrierea algoritmului, vom presupune că sunt numerotate mai întâi laturile de tip REJ şi apoi laturile de tip E. Vom nota cu L_{REJ} şi cu L_E laturile din fiecare categorie şi cu $L = L_{REJ} + L_E$ numărul total de laturi.

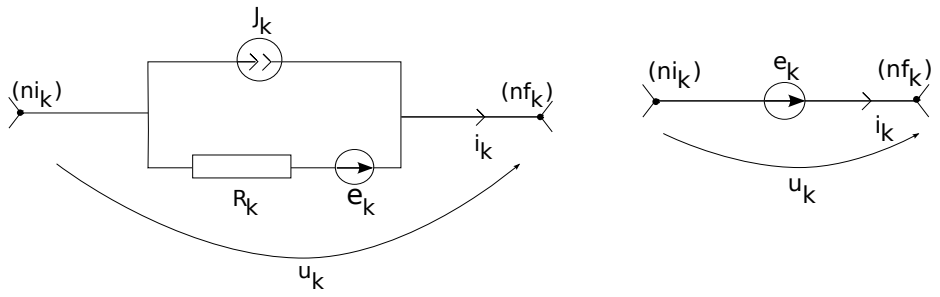


Figura 3.4: Laturi standard.

Ecuaţiile Kirchhoff I şi II se scriu sub aceeaşi formă:

$$\mathbf{Ai} = \mathbf{0}, \tag{3.26}$$

$$\mathbf{u} = \mathbf{A}^T\mathbf{v}. \tag{3.27}$$

Relaţiile între tensiunile şi curenţii unei laturi sunt acum

$$u_k = R_k(i_k - J_k) - e_k, \quad k = 1, \ldots, L_{REJ}, \tag{3.28}$$

$$u_k = -e_k, \quad k = L_{REJ} + 1, \ldots, L_{REJ} + L_E. \tag{3.29}$$

Pentru scrierea compactă a acestor relaţii, vom partiţiona vectorii în două, corespunzător celor două tipuri de laturi:

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_{REJ} \\ \mathbf{u}_E \end{bmatrix}, \quad \mathbf{i} = \begin{bmatrix} \mathbf{i}_{REJ} \\ \mathbf{i}_E \end{bmatrix}, \quad \mathbf{e} = \begin{bmatrix} \mathbf{e}_{REJ} \\ \mathbf{e}_E \end{bmatrix}, \tag{3.30}$$

unde vectorii cu indice REJ au dimensiunea L_{REJ}, iar vectorii cu indice E au dimensiunea L_E.

Formele matriceale ale relaţiilor (3.28) şi (3.29) sunt

$$\mathbf{u}_{REJ} = \mathbf{R}(\mathbf{i}_{REJ} - \mathbf{J}) - \mathbf{e}_{REJ}, \tag{3.31}$$

$$\mathbf{u}_E = -\mathbf{e}_E, \tag{3.32}$$

unde $\mathbf{R}$ este acum o matrice diagonală, de dimensiune L_{REJ}, având pe diagonală rezistenţele laturilor de tip REJ, iar $\mathbf{J}$ este un vector de dimensiune L_{REJ}.

Partiţionarea în două a vectorului curenţilor face utilă partiţionarea în două blocuri a matricei incidenţelor laturi-noduri, astfel încât relaţia (3.26) devine

$$\begin{bmatrix} \mathbf{A}_{REJ} & \mathbf{A}_E \end{bmatrix} \begin{bmatrix} \mathbf{i}_{REJ} \\ \mathbf{i}_E \end{bmatrix} = \mathbf{0}, \tag{3.33}$$

unde $\mathbf{A}_{REJ}$ este o matrice de dimensiune $(N-1) \times L_{REJ}$ ce conţine incidenţa laturilor de tip REJ la noduri, iar matricea $\mathbf{A}_E$ este o matrice de dimensiune $(N-1) \times L_E$ ce conţine incidenţa laturilor de tip E la noduri, de unde rezultă, după realizarea operaţiilor cu blocuri de matrice că

$$\mathbf{A}_{REJ}\mathbf{i}_{REJ} + \mathbf{A}_E\mathbf{i}_E = \mathbf{0}. \tag{3.34}$$

Relaţia (3.27) se explicitează şi ea ca

$$\begin{bmatrix} \mathbf{u}_{REJ} \\ \mathbf{u}_E \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{REJ}^T \\ \mathbf{A}_E^T \end{bmatrix} \mathbf{v}, \tag{3.35}$$

de unde rezultă că

$$\mathbf{u}_{REJ} = \mathbf{A}_{REJ}^T \mathbf{v}, \tag{3.36}$$

$$\mathbf{u}_E = \mathbf{A}_E^T \mathbf{v}. \tag{3.37}$$

Recapitulând, relaţiile matriceale utile sunt:

$$\mathbf{A}_{REJ}\mathbf{i}_{REJ} + \mathbf{A}_E\mathbf{i}_E = \mathbf{0}, \tag{3.38}$$

$$\mathbf{u}_{REJ} = \mathbf{A}_{REJ}^T \mathbf{v}, \tag{3.39}$$

$$\mathbf{u}_E = \mathbf{A}_E^T \mathbf{v}, \tag{3.40}$$

$$\mathbf{u}_{REJ} = \mathbf{R}(\mathbf{i}_{REJ} - \mathbf{J}) - \mathbf{e}_{REJ}, \tag{3.41}$$

$$\mathbf{u}_E = -\mathbf{e}_E. \tag{3.42}$$

Din (3.41) rezultă curenţii prin laturile de tip REJ:

$$\mathbf{i}_{REJ} = \mathbf{R}^{-1}(\mathbf{u}_{REJ} + \mathbf{e}_{REJ}) + \mathbf{J}. \tag{3.43}$$

Această expresie împreună cu (3.39) se substituie în (3.38). Sistemul de rezolvat devine

$$\mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{A}_{REJ}^T\mathbf{v} + \mathbf{A}_E\mathbf{i}_E = -\mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{e}_{REJ} - \mathbf{A}_{REJ}\mathbf{J}, \tag{3.44}$$

$$\mathbf{A}_E^T\mathbf{v} = -\mathbf{e}_E. \tag{3.45}$$

Acest sistem este un sistem algebric liniar, de dimensiune $N - 1 + L_E$. Necunoscutele acestui sistem sunt atât potenţialele nodurilor cât şi valorile curenţilor prin sursele ideale de tensiune:

$$\mathbf{x} = \begin{bmatrix} \mathbf{v} \\ \mathbf{i}_E \end{bmatrix}. \tag{3.46}$$

Sistemul de rezolvat

$$\mathbf{M}\mathbf{x} = \mathbf{p} \tag{3.47}$$

are matricea coeficienţilor

$$\mathbf{M} = \begin{bmatrix} \mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{A}_{REJ}^T & \mathbf{A}_E \\ \mathbf{A}_E^T & \mathbf{0}, \end{bmatrix} \tag{3.48}$$

şi vectorul termenilor liberi

$$\mathbf{p} = \begin{bmatrix} -\mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{e}_{REJ} - \mathbf{A}_{REJ}\mathbf{J} \\ -\mathbf{e}_E \end{bmatrix} \tag{3.49}$$

Matricea coeficienţilor rămâne simetrică dar îşi pierde pozitiv definirea. O prelucrare suplimentară a acestui sistem poate conduce însă la un sistem mai mic şi cu proprietăţi

mai bune. Pentru aceasta, vom presupune că sursele ideale de tensiune formează un subgraf conex. Această ipoteză nu este restrictivă deoarece se ştie că, indiferent cum ar fi plasate sursele ideale de tensiune, se poate găsi un circuit, echivalent cu circuitul iniţial din punct de vedere al grafului de curenţi, care să satisfacă această proprietate. Mai mult, subgraful conex nu are bucle, deoarece se ştie că un model de circuit în care apar bucle formate numai din surse ideale de tensiune nu este o problemă bine formulată (nu are soluţie unică). Vom presupune că nodurile sunt numerotate astfel încât sunt lăsate la sfârşit nodurile care aparţin subgrafului conex al surselor ideale de tensiune. Ideea ce va fi explicitată în relaţii matematice în cele ce urmează este aceea că nodurile care aparţin acestui subgraf conex au potenţiale cunoscute. Pentru fiecare astfel de nod, există o cale de la acest nod la nodul de referinţă (care este şi el un nod al acestui subgraf conex), formată numai din surse ideale de tensiune. Aceste potenţiale vor fi exprimate în funcţie de tensiunile electromotoare ale surselor ideale de tensiune şi apoi substituite în relaţiile Kirchhoff I aduse la forma (3.44). Sistemul rezultat va fi de dimensiune $N-1-L_E$ iar matricea coeficienţilor va reprezenta conductanţe nodale, deci va fi simetrică, pozitiv definită şi diagonal dominantă.

Vom nota partiţionarea vectorului potenţialelor în

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_* \\ \mathbf{v}_c \end{bmatrix}, \tag{3.50}$$

unde $\mathbf{v}_c$ este vectorul potenţialelor nodurilor subgrafului conex format din surse ideale de tensiune (în care nu este inclus nodul de referinţă), având dimensiunea L_E, iar $\mathbf{v}_*$ este vectorul potenţialelor restului de noduri, de dimensiune $N-1-L_E$. Această partiţionare impune o partiţionare suplimentară a matricei $\mathbf{A}_E$, matrice ce conţine incidenţa laturilor de tip E la noduri. Deoarece laturile de tip E sunt acum conectate numai la noduri ale subgrafului conex, înseamnă că blocul superior, care reprezintă incidenţă laturilor de tip E la nodurile de potenţial necunoscut este nul:

$$\mathbf{A}_E = \begin{bmatrix} \mathbf{0} \\ \mathbf{A}_{Ec} \end{bmatrix}, \tag{3.51}$$

unde $\mathbf{A}_{Ec}$ este o matrice pătrată de dimensiunea numărului de noduri de potenţial cunoscut (exceptând nodul de referinţă) ce reprezintă incidenţa laturilor de tip E la nodurile de potenţial cunoscut. Relaţia (3.45) devine

$$\begin{bmatrix} \mathbf{0} & \mathbf{A}_{Ec}^T \end{bmatrix} \begin{bmatrix} \mathbf{v}_* \\ \mathbf{v}_c \end{bmatrix} = -\mathbf{e}_E, \tag{3.52}$$

de unde

$$\mathbf{A}_{Ec}^T \mathbf{v}_c = -\mathbf{e}_E. \tag{3.53}$$

Potenţialele nodurilor subgrafului conex format numai din surse ideale de tensiune sunt

$$\mathbf{v}_c = -\left(\mathbf{A}_{Ec}^T\right)^{-1}\mathbf{e}_E. \tag{3.54}$$

Pentru o problemă de circuit bine formulată, matricea $\mathbf{A}_{Ec}^T$ este inversabilă, fiecare linie a inversei ei reprezentând calea de la un nod la nodul de referinţă, cale aleasă doar din laturi ale subgrafului conex al surselor ideale de tensiune. Această cale este unică deoarece laturile acestui subgraf conex nu formează bucle.

Pentru a înlocui (3.54) în (3.44) este util să partiţionăm matricile ce intervin astfel:

$$\mathbf{G}' = \mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{A}_{REJ}^T = \begin{bmatrix} \mathbf{G}_{**} & \mathbf{G}_{*c} \\ \mathbf{G}_{*c}^T & \mathbf{G}_{cc} \end{bmatrix} \in \mathbb{R}^{(N-1)\times(N-1)} \tag{3.55}$$

$$\mathbf{t}' = -\mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{e}_{REJ} - \mathbf{A}_{REJ}\mathbf{J} = \begin{bmatrix} \mathbf{t}_* \\ \mathbf{t}_{*c} \end{bmatrix} \in \mathbb{R}^{(N-1)\times 1}. \tag{3.56}$$

Matricea $\mathbf{G}_{**}$ este o matrice pătrată de dimensiune $N-1-L_E$, reprezentând conductanţe nodale pentru subgraful care conţine laturi de tip REJ şi nodurile de potenţial necunoscut. Matricea $\mathbf{G}_{*c}$ este o matrice de dimensiune $(N-1-L_E)\times L_E$ care conţine conductanţe nodale nediagonale între nodurile cu potenţial cunoscut şi cele cu potenţial necunoscut. $\mathbf{G}_{cc}$ este o matrice pătrată de dimenensiune L_E care conţine conductanţe nodale pentru subgraful care are laturi de tip RE dar noduri de potenţial cunoscut.

Relaţia (3.44) devine

$$\begin{bmatrix} \mathbf{G}_{**} & \mathbf{G}_{*c} \\ \mathbf{G}_{*c}^T & \mathbf{G}_{cc} \end{bmatrix} \begin{bmatrix} \mathbf{v}_* \\ \mathbf{v}_c \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{A}_{Ec} \end{bmatrix} \mathbf{i}_E = \begin{bmatrix} \mathbf{t}_* \\ \mathbf{t}_{*c} \end{bmatrix}, \tag{3.57}$$

relaţie echivalentă cu următoarele două relaţii matriceale:

$$\mathbf{G}_{**}\mathbf{v}_* + \mathbf{G}_{*c}\mathbf{v}_c = \mathbf{t}_*, \tag{3.58}$$

$$\mathbf{G}_{*c}^T\mathbf{v}_* + \mathbf{G}_{cc}\mathbf{v}_c + \mathbf{A}_{Ec}\mathbf{i}_E = \mathbf{t}_{*c}. \tag{3.59}$$

Înlocuind acum expresia (3.54), rezultă că potenţialele necunoscute se determină prin rezolvarea sistemului

$$\mathbf{G}_{**}\mathbf{v}_* = \mathbf{t}_* + \mathbf{G}_{*c}(\mathbf{A}_{Ec}^T)^{-1}\mathbf{e}_E, \tag{3.60}$$

care este un sistem de dimensiune $N-1-L_E$, cu matricea coeficienţilor simetrică şi pozitiv definită, urmând ca valorile curenţilor prin sursele ideale de tensiune să se determine din a două relaţie matriceală.

Algoritmul corespunzător acestui caz este uşor de conceput dacă el este orientat pe asamblarea matricelor şi partiţionarea lor.

3.4 Tratarea surselor comandate liniar

Un circuit rezistiv liniar poate conţine şi surse comandate liniar. Există patru feluri de surse comandate (fig. 3.5), notate pe scurt "SUCU" (sursă de tensiune comandată în tensiune), "SUCI" (sursă de tensiune comandată în curent, "SICU" (sursă de curent comandată în tensiune) şi "SICI" (sursă de curent comandată în curent).

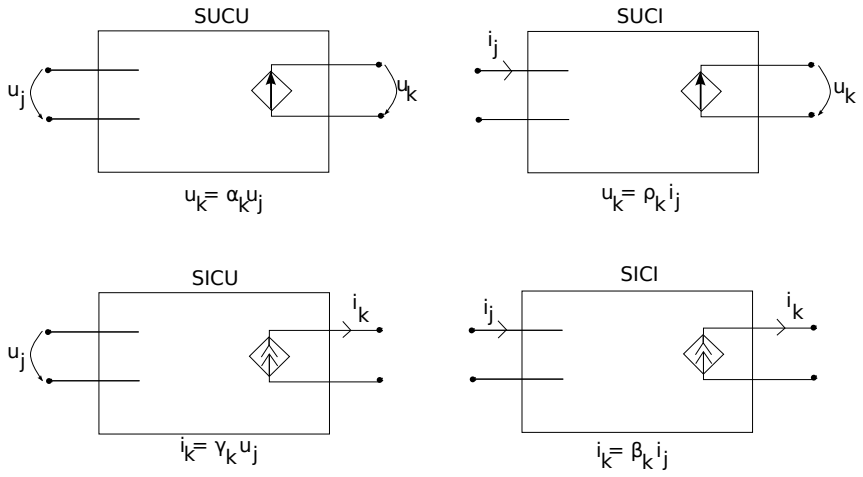


Figura 3.5: Tipuri de surse comandate liniar.

Vom sugera paşii care trebuie făcuţi în asamblarea unui sistem de ecuaţii în cazul în care pe lângă laturi de tip REJ şi E există numai laturi de tip SUCU, presupuse a fi numerotate la sfârşit. Tratarea celorlalte tipuri de surse comandate se face într-o manieră similară. Cu un raţionament similar celui din paragraful anterior, ecuaţiile matriceale sunt:

- Kirchhoff I:

$$\mathbf{A}_{REJ}\mathbf{i}_{REJ} + \mathbf{A}_E\mathbf{i}_E + \mathbf{A}_{SUCU}\mathbf{i}_{SUCU} = \mathbf{0}, \tag{3.61}$$

- Kirchhoff II:

$$\mathbf{u}_{REJ} = \mathbf{A}_{REJ}^T\mathbf{v}, \tag{3.62}$$
$$\mathbf{u}_E = \mathbf{A}_E^T\mathbf{v}, \tag{3.63}$$
$$\mathbf{u}_{SUCU} = \mathbf{A}_{SUCU}^T\mathbf{v}, \tag{3.64}$$

- relaţii constitutive:

$$\mathbf{u}_{REJ} = \mathbf{R}(\mathbf{i}_{REJ} - \mathbf{J}) - \mathbf{e}_{REJ}, \tag{3.65}$$
$$\mathbf{u}_E = -\mathbf{e}_E, \tag{3.66}$$
$$\mathbf{u}_{SUCU} = \alpha\mathbf{S}_{SUCU}\mathbf{v}, \tag{3.67}$$

unde α este o matrice diagonală, de dimensiune egală cu numărul de surse comandate, având pe diagonală parametrii surselor comandate, iar $\mathbf{S}_{SUCU}$ este o matrice de selecţie (topologică), de dimensiune $L_{SUCU} \times (N-1)$ care selectează pentru fiecare latură de tip SUCU, perechea de noduri care determină tensiunea de comandă.

În acest caz, sistemul de rezolvat va avea un număr de $N - 1 + L_E + L_{SUCU}$ necunoscute, reprezentând potenţialele nodurilor, curenţii prin sursele ideale de tensiune şi curenţii prin sursele comandate:

$$\mathbf{x} = \begin{bmatrix} \mathbf{v} \\ \mathbf{i}_E \\ \mathbf{i}_{SUCU} \end{bmatrix}. \tag{3.68}$$

Sistemul de rezolvat

$$\mathbf{M}\mathbf{x} = \mathbf{p} \tag{3.69}$$

are matricea coeficienţilor

$$\mathbf{M} = \begin{bmatrix} \mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{A}_{REJ}^T & \mathbf{A}_E & \mathbf{A}_{SUCU} \\ \mathbf{A}_E^T & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{SUCU}^T - \alpha\mathbf{S}_{SUCU} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{3.70}$$

iar vectorul termenilor liberi

$$\mathbf{p} = \begin{bmatrix} -\mathbf{A}_{REJ}\mathbf{R}^{-1}\mathbf{e}_{REJ} - \mathbf{A}_{REJ}\mathbf{J} \\ -\mathbf{e}_E \\ \mathbf{0} \end{bmatrix}. \tag{3.71}$$

În concluzie, sursele comandate complică problema. Nu numai că relaţiile matematice devin mai complicate, conducând şi la creşterea complexităţii algoritmului, dar sistemul de rezolvat are o matrice a coeficienţilor care nu mai este simetrică.

Capitolul 4

Interpolarea funcţiilor

Formularea celor mai multe probleme de inginerie poate fi scrisă formal

$$\mathbf{y} = f(\mathbf{x}), \tag{4.1}$$

unde $\mathbf{x}$ sunt datele problemei, care sunt din punct de vedere matematic parametri independenţi, iar $\mathbf{y}$ sunt mărimile de interes ce se doresc a fi estimate. În aplicaţiile reale f nu este definită explicit în funcţie de datele problemei. În notaţia de mai sus, f poate reprezenta de exemplu chiar un proces de măsurare a mărimilor $\mathbf{y}$ pentru o anumită stare sau configuraţie complet caracterizată de mărimile $\mathbf{x}$, sau f poate reprezenta programe software complicate, capabile să analizeze configuraţia caracterizată complet de datele $\mathbf{x}$ şi să calculeze printr-un algoritm de postprocesare mărimile $\mathbf{y}$.

Se ajunge astfel de multe ori ca, pentru un set de date (obţinute de exemplu prin măsurători), să se dorească găsirea unei funcţii care să aproximeze aceste date. Dacă se cunoaşte un astfel de set de date se spune că *funcţia este reprezentată prin date*. De exemplu, dacă setul de date este notat $(\mathbf{x}_k, \mathbf{y}_k)$, $k = 0, \ldots, n$, iar $\mathbf{y}_k = f(\mathbf{x}_k)$, atunci se doreşte găsirea unei expresii analitice pentru o funcţie g care să aproximeze aceste date adică $g(\mathbf{x}_k) \approx \mathbf{y}_k$ sau chiar $g(\mathbf{x}_k) = \mathbf{y}_k$. În cele ce urmează, mulţimea (setul) de date $\mathbf{x}_k$ îl vom numi şi reţea (grid) de discretizare.

Pe scurt, *interpolarea unui set de date înseamnă găsirea unei funcţii care să treacă prin punctele mulţimii de date* (fig.4.1). Dacă setul de date are foarte multe puncte, rezultatul interpolării poate reprezenta o funcţie cu multe oscilaţii (fig. 4.1 - dreapta), care se datorează nu vreunui fenomen fizic, ci faptului că mulţimea de date iniţiale este afectată de erori inerente, datorate de exemplu procedeului de măsurare. În acest caz, dacă se ştie că funcţia este netedă, se preferă *aproximarea (regresia) setului de date, adică găsirea unei funcţii care să treacă printre punctele mulţimii de date* (fig.4.2), abordare ce are avantajul că diminuează erorile de măsurare din rezultatul final.

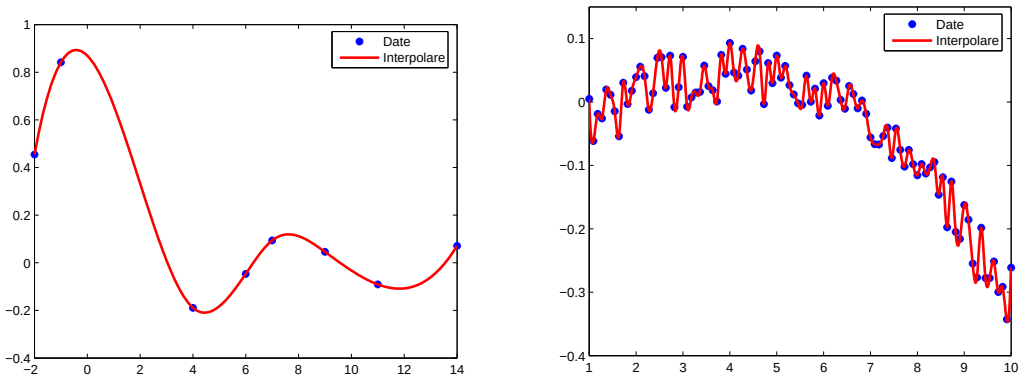


Figura 4.1: Interpolarea unui set de date. În cazul în care setul de date are foarte multe valori, interpolarea poate genera oscilaţii nedorite.

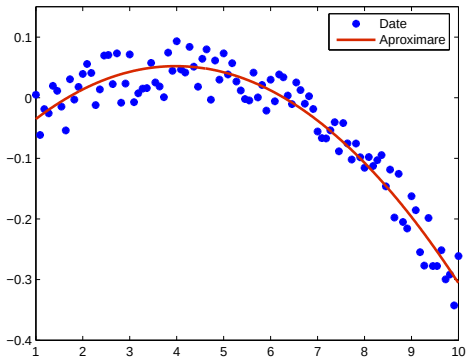


Figura 4.2: Aproximarea unui set de date.

Setul de date nu este neapărat obţinut numai prin măsurători. Să presupunem că există un software capabil să calculeze $f(\mathbf{x})$ pentru orice $\mathbf{x}$ dorit. În acest caz se spune că *funcţia este reprezentată prin cod*. Dacă efortul de evaluare al funcţiei este mare, atunci se preferă evaluarea numerică doar în câteva puncte relevante, obţinându-se astfel un set finit de date, iar pentru acesta se determină o aproximare sau interpolare g ce va fi folosită pentru estimări ulterioare ale funcţiei f.

Până acum nu am precizat nimic despre domeniul de definiţie şi domeniul de valori al funcţiei f. Cazul cel mai simplu este cel în care datele sunt mărimi scalare, reale, iar rezultatul evaluării funcţiei este tot un număr real, adică

$$f, g : [a, b] \to \mathbb{R}. \tag{4.2}$$

Acest caz este numit *cazul scalar unidimensional* (1D). Cazul vectorial unidimensional, în care valorile funcţiei sunt într-un spaţiu multidimensional $f : [a, b] \to \mathbb{R}^m$, $m > 1$ se poate rezolva pe componente, reducându-se la m interpolări/aproximări 1D.

Problema se complică dacă domeniul de definiţie are mai multe dimensiuni. Cazul

$$f, g : [a, b] \times [c, d] \to \mathbb{R} \tag{4.3}$$

îl vom numi *cazul scalar bidimensional* (2D). El este particularizarea cazului scalar n-dimensional (nD) $f, g : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}$. Cazul cel mai general este $f, g : \mathcal{D} \subset \mathbb{R}^n \to \mathbb{R}^m$ care se reduce însă la m situaţii de tip nD.

Ideile metodelor de interpolare/aproximare pot fi cel mai uşor explicate pe cazul 1D, de aceea paragrafele vor presupune că are loc (4.2).

4.1 Distanţa dintre două funcţii

Evident se doreşte ca funcţia $g : [a, b] \to \mathbb{R}$ să aproximeze/interpoleze cât mai bine funcţia $f : [a, b] \to \mathbb{R}$. Din punct de vedere matematic, aceasta înseamnă că distanţa dintre cele două funcţii, calculată ca norma diferenţei lor

$$d(f, g) = \|f - g\| \tag{4.4}$$

să fie cât mai mică. Există mai multe procedee de definire a normei din relaţia (4.4).

O primă posibilitate este aceea de a estima aria cuprinsă între graficele celor două funcţii, adică de a calcula distanţa dintre funcţii ca

$$d_1(f, g) = \frac{1}{b-a} \int_a^b |f(x) - g(x)| \, \mathrm{d}x. \tag{4.5}$$

Dezavantajul acestei abordări este acela că, local, pot exista diferenţe foarte mari între cele două funcţii, fără ca valoarea rezultatului formulei (4.5) să reflecte acest lucru[1].

O a doua variantă, care nu elimină însă dezavantajul menţionat mai sus, calculează abaterea pătratică medie dintre cele două funcţii:

$$d_2(f, g) = \sqrt{\frac{1}{b-a} \int_a^b (f(x) - g(x))^2 \, \mathrm{d}x}. \tag{4.6}$$

Norma (4.6) este norma L_2 din spaţiul funcţiilor de pătrat integrabil şi generalizează norma euclidiană.

O a treia posibilitate este de a evalua norma Chebyshev, care estimează abaterea maximă dintre cele două funcţii:

$$d_3(f, g) = \max_{x \in [a,b]} |f(x) - g(x)|. \tag{4.7}$$

[1] Această afirmaţie se bazează pe proprietatea că, dacă valoarea unei funcţii se modifică într-o mulţime numărabilă de puncte, atunci valoarea integralei ei nu se modifică.

Din punctul de vedere al acurateţii rezultatului aproximării, această normă este cea mai avantajoasă deoarece nu suferă de dezavantajul normelor ce folosesc integrale. Valoarea ei furnizează informaţii despre diferenţa locală maximă dintre cele două funcţii.

Oricare din normele (4.5), (4.6), (4.7) presupun evaluarea funcţiei f în toate punctele domeniului de definiţie, ceea ce nu este posibil într-un algoritm. Funcţia f este cunoscută sau cel mult poate fi evaluată numai într-un număr discret de puncte x_k, $k = 0, \ldots, n$. De aceea, în implementarea numerică a procedurilor de interpolare sau aproximare se vor folosi normele discrete:

$$d_{1d}(f,g) = \sum_{k=0}^{n} |g(x_k) - f(x_k)|, \tag{4.8}$$

$$d_{2d}(f,g) = \sqrt{\sum_{k=0}^{n} (g(x_k) - f(x_k))^2}, \tag{4.9}$$

$$d_{3d}(f,g) = \max_{k=0,n} |g(x_k) - f(x_k)|. \tag{4.10}$$

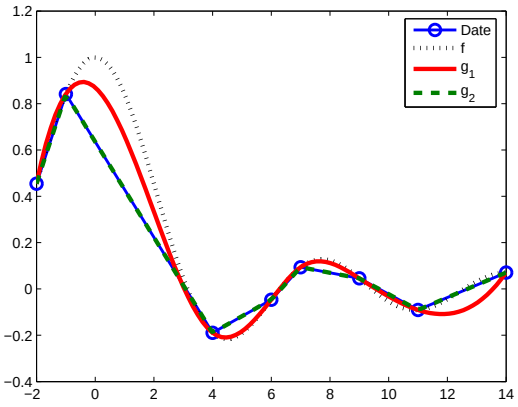


Figura 4.3: Oricare normă discretă este zero pentru diferenţa $f - g_1$ sau $f - g_2$, unde f este funcţia adevărată, iar g_1, g_2 sunt posibile interpolări.

Normele discrete au avantajul că pot fi evaluate cu uşurintă. Dezavantajul lor este acela că se pierde posibilitatea evaluării acurateţii interpolării/aproximării între nodurile reţelei de interpolare. Mai mult, pentru orice funcţie f există o infinitate de funcţii g care nu sunt identice cu f şi pentru care oricare din normele discrete de mai sus sunt zero dacă $g(x_k) = f(x_k)$ pentru toate valorile $k = 0, \ldots, n$ (fig.4.3). Deci, problema interpolării/aproximării devine în acest moment prost formulată. Corectarea formulării se face impunând condiţii suplimentare pentru funcţia g, aşa cum se prezintă în paragraful următor.

4.2 Formularea problemei interpolării

Problema interpolării este aceea de a găsi o funcţie g pentru care distanţa discretă faţă de funcţia f, cunoscută într-un număr finit de puncte $f(x_j) = y_j$, este zero. Această condiţie este echivalentă cu a impune egalitatea valorilor funcţiilor f şi g în toate punctele (nodurile) reţelei de discretizare:

$$g(x_j) = f(x_j), \quad j = 0, \ldots, n, \tag{4.11}$$

echivalent cu a scrie

$$g(x_j) = y_j, \quad j = 0, \ldots, n. \tag{4.12}$$

Relaţiile (4.12) se numesc *condiţii de interpolare* Pentru a face ca problema să fie bine formulată matematic (soluţia să existe şi să fie unică) funcţia g se caută în spaţiul polinoamelor generalizate, adică se caută de forma unei combinaţii liniare de m funcţii φ_k, $k = 1, \ldots, m$ numite *funcţii de bază*:

$$g(x) = \sum_{k=0}^{m} c_k \varphi_k(x). \tag{4.13}$$

Funcţiile de bază pot fi de diferite tipuri şi ele se aleg înainte de rezolvarea propriu-zisă a problemei interpolării. De exemplu, dacă se ştie că funcţia de interpolat are un caracter periodic cu perioada 2π atunci o alegere potrivită a funcţiilor de bază este $\varphi_0(x) = 1$, $\varphi_1(x) = \sin x$, $\varphi_2(x) = \cos x$, $\varphi_3(x) = \sin(2x)$, etc. Alegerea $\varphi_k(x) = x^k$ corespunde cazului polinoamelor scrise în mod clasic.

Astfel, problema interpolării se reduce la problema determinării celor m coeficienţi c_k.

Condiţiile de interpolare (4.12) scrise pentru polinoame de interpolare de tipul (4.13) devin

$$\sum_{k=0}^{m} c_k \varphi_k(x_j) = y_j, \quad j = 0, \ldots, n, \tag{4.14}$$

ceea ce înseamnă că valorile coeficienţilor c_k sunt soluţiile unui sistem de ecuaţii algebrice liniare cu $n+1$ ecuaţii şi $m+1$ necunoscute. Pentru ca sistemul (4.14) să fie bine formulat matematic vom impune ca numărul de funcţii de bază să fie egal cu numărul de puncte al reţelei de discretizare

$$m = n \tag{4.15}$$

şi determinantul matricei coeficienţilor să fie nenul:

$$\Delta = \begin{vmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \cdots & \varphi_n(x_0) \\ \varphi_0(x_1) & \varphi_1(x_1) & \cdots & \varphi_n(x_1) \\ \cdots & & & \\ \varphi_0(x_n) & \varphi_1(x_n) & \cdots & \varphi_n(x_n) \end{vmatrix} \neq 0. \tag{4.16}$$

Determinantul Δ este nenul dacă şi numai dacă punctele x_j sunt distincte şi funcţiile de bază φ_k sunt liniar independente.

Rezumând cele de mai sus, problema interpolării funcţiilor se formulează astfel.

Date:

- un tabel de valori (x_k, y_k), $k = 0, \ldots, n$, unde punctele reţelei de discretizare x_k sunt distincte două câte două;
- $n + 1$ funcţii de bază liniar independente $\varphi_k(x)$, $k = 0, \ldots, n$.

Se cer:

- coeficienţii c_k, $k = 0, \ldots, n$ pentru care sunt satisfăcute condiţiile de intepolare $g(x_j) = y_j$, $j = 0, \ldots, n$ unde $g(x) = \sum_{k=0}^{n} c_k \varphi_k(x)$ este funcţia (în particular polinomul) de interpolare al datelor din tabel.

4.3 Metode de interpolare globală

Metodele de interpolare globală sunt metodele în care funcţiile de bază se definesc compact, printr-o singură expresie pe întreg domeniul de definiţie al funcţiei de interpolat. În aceste metode, gradul polinomului de interpolare este în strânsă legătură cu numărul de puncte din reţeaua de discretizare, mai precis gradul polinomului de interpolare este cu unu mai mic decât numărul de puncte din tabelul de date (de exemplu prin două puncte trece în mod univoc o dreaptă, prin trei puncte o parabolă, etc.). În funcţie de cum se aleg funcţiile de bază se obţin diferite metode de interpolare globală.

4.3.1 Metoda clasică

În metoda clasică funcţiile de bază sunt polinoamele algebrice $\varphi_k(x) = x^k$, $k = 0, \ldots, n$. Atunci polinomul de interpolare (4.13), cu condiţia (4.15) devine

$$g(x) = \sum_{k=0}^{n} c_k x^k = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n, \tag{4.17}$$

iar condiţiile de interpolare (4.12) conduc la rezolvarea sistemului de ecuaţii algebrice liniare

$$\begin{cases} c_0 + c_1 x_0 + c_2 x_0^2 + \cdots + c_n x_0^n = y_0 \\ c_0 + c_1 x_1 + c_2 x_1^2 + \cdots + c_n x_1^n = y_1 \\ \qquad\qquad \cdots \\ c_0 + c_1 x_n + c_2 x_n^2 + \cdots + c_n x_n^n = y_n \end{cases} \tag{4.18}$$

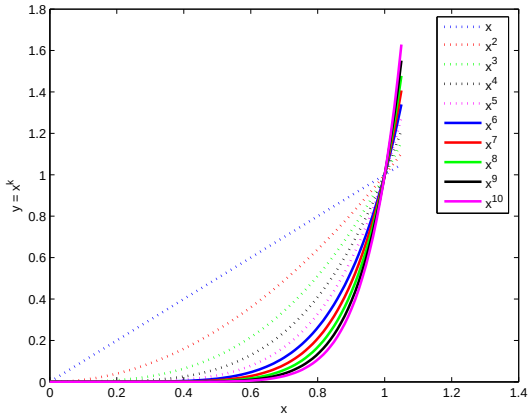


Figura 4.4: Funcţiile de bază x^k devin aproape liniar dependente pentru valori ale lui $k > 5$.

pentru determinarea celor $n + 1$ coeficienţi.

Calculul coeficienţilor polinomului de interpolare se numeşte *etapa de pregătire* iar evaluarea propriu-zisă a polinomului de interpolare se numeşte *etapa de evaluare*.

În metoda clasică, etapa de pregătire constă în asamblarea şi rezolvarea sistemului (4.18). Această etapă necesită un efort mare de calcul, corespunzător rezolvării unui sistem cu matrice plină, de ordinul $O(2n^3/3)$. Etapa de evaluare presupune evaluarea expresiei (4.17), deci un efort de calcul de ordinul $O(2n)$.

Dezavantajul major al acestei metode nu stă în primul rând în efortul mare de calcul necesar etapei de pregătire ci, mai ales în aceea că pentru valori mari ale lui n matricea coeficienţilor sistemului (4.18) este slab condiţionată, pentru că funcţiile de bază devin aproape liniar independente (fig.4.4). Din acest motiv această metodă nu se recoman-dă să fie aplicată pentru polinoame de interpolare de grad mai mare decât 5.

O metodă mai eficientă de interpolare trebuie să folosească funcţii de bază care să fie cât mai deosebite între ele. Mai mult, ar fi convenabil ca sistemul de ecuaţii de rezolvat în etapa de pregătire să fie cât mai simplu de rezolvat. Aşa se întâmplă de exemplu în metoda Lagrange.

4.3.2 Metoda Lagrange

În metoda Lagrange funcţiile de bază sunt polinoamele Lagrange

$$\varphi_k(x) = l_k(x) = \frac{\prod_{i=0, i\neq k}^{n}(x - x_i)}{\prod_{i=0, i\neq k}^{n}(x_k - x_i)}, \tag{4.19}$$

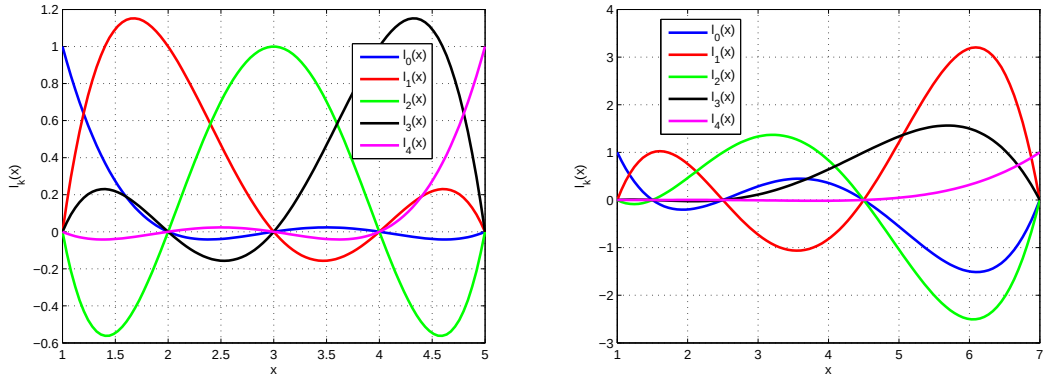


Figura 4.5: Funcţiile Lagrange pentru o reţea de discretizare uniformă (stânga), respectiv neuniformă (dreapta).

iar polinomul de interpolare este

$$g(x) = \sum_{k=0}^{m} c_k l_k(x). \tag{4.20}$$

Fiecare polinom Lagrange $l_k(x)$ poate fi considerat asociat unui nod k, şi are proprietatea că ia valoarea 1 când este evaluat în acel nod şi valoarea 0 când este evaluat în toate celelalte noduri:

$$l_k(x_j) = \begin{cases} 1 & \text{dacă} \quad j = k, \\ 0 & \text{dacă} \quad j \neq k. \end{cases} \tag{4.21}$$

Figura 4.5 ilustrează polinoamele Lagrange pentru o reţea de discretizare cu 5 puncte, uniformă şi, respectiv, neuniformă. Între nodurile reţelei de discretizare polinoamele Lagrange pot avea valori oricât de mari.

Cu această alegere a funcţiilor de bază, condiţiile de interpolare $g(x_j) = y_j$, $j = 0, \ldots, n$ devin

$$\sum_{k=0}^{m} c_k l_k(x_j) = y_j, \tag{4.22}$$

de unde, aplicând (4.21)

$$c_j = y_j, \quad j = 0, \ldots, n. \tag{4.23}$$

Practic, sistemul algebric de rezolvat pentru determinarea coeficienţilor este unul în care matricea coeficienţilor este matricea unitate, şi etapa de pregătire aşa cum a fost definită anterior dispare.

Expresia polinomului Lagrange poate fi scrisă în mod compact ca

$$g(x) = \sum_{k=0}^{n} y_k l_k(x) = \sum_{k=0}^{n} y_k \frac{\prod_{i=0, i\neq k}^{n} (x - x_i)}{\prod_{i=0, i\neq k}^{n} (x_k - x_i)}. \tag{4.24}$$

În particular, polinomul de interpolare liniar pentru o reţea de discretizare cu două puncte este

$$g(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0}, \tag{4.25}$$

iar parabola interpolatoare pentru o reţea de discretizare cu trei puncte este

$$g(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}. \tag{4.26}$$

Implementarea metodei Lagrange se poate face în două variante. Prima din ele este o variantă fără pregătire, aşa cum rezultă în mod natural din prezentarea de mai sus. Pentru orice valoare nouă din domeniul de definiţie, evaluarea polinomului de interpolare se face cu expresia (4.24). Efortul de calcul poate fi estimat direct inspectând expresia (4.24) şi este de ordinul $T_e = \mathrm{O}(4n^2)$ pentru ficare evaluare, rezultând un efort de calcul total pentru evaluarea în m puncte de $T_{L-fp} = \mathrm{O}(4mn^2)$.

O variantă de implementare mai eficientă se bazează pe rescrierea relaţiei (4.24) prin scoaterea forţată a factorului comun

$$p = \prod_{k=0}^{n} (x - x_k), \tag{4.27}$$

polinomul de interpolare fiind

$$g(x) = p \sum_{k=0}^{n} \frac{\alpha_k}{x - x_k}, \tag{4.28}$$

unde coeficienţii notaţi

$$\alpha_k = \frac{y_k}{\prod_{j=0, j \neq k}^{n} (x_k - x_j)} \tag{4.29}$$

vor fi calculaţi în etapa de pregătire. Pseudocodul acestei variante este următorul.

```
procedură Lagrange_pregătire(n, x, y, α)
; pregăteşte coeficienţii din metoda Lagrange
; declaraţii - parametri de intrare
întreg n                    ; dimensiunea problemei - numărul de intervale
tablou real x[n], y[n]      ; tabelul de valori, indici de la zero
; declaraţii - parametri de ieşire
tablou real α[n]            ; coeficienţii
pentru k = 0, n
    α_k = y_k
    pentru j = 0, n
        dacă j ≠ k atunci α_k = α_k/(x_k - x_j)
```

•

•

retur

funcţie Lagrange_evaluare($n, x, y, \alpha, xcrt$)
; evaluează polinomul de interpolare Lagrange în punctul $xcrt$
; declaraţii - parametri de intrare
întreg n ; dimensiunea problemei - numărul de intervale
tablou real $x[n], y[n]$; tabelul de valori, indici de la zero
tablou real $\alpha[n]$; coeficienţii
real $xcrt$; punctul de evaluat
; alte declaraţii
real p, s
$p = 1$
pentru $k = 0, n$
    $\cdots$; vezi comentariul de mai jos
    $p = p * (xcrt - x_k)$
•
$s = 0$
pentru $k = 0, n$
    $s = s + \alpha_k/(xcrt - x_k)$
•
întoarce $s * p$

În acest caz etapa de pregătire are ordinul de complexitate $T_p = \mathrm{O}(2n^2)$, iar etapa de evaluare ordinul $T_e = O(5n)$, rezultând un efort de calcul total $T_{L-cp} = \mathrm{O}(2n^2 + 5mn)$.

Aşa cum este scris, pseudocodul etapei de evaluare eşuează dacă punctul de evaluare coincide cu unul din punctele din tabel, din cauza împărţirii la zero. Un algoritm înţelept tratează şi astfel de cazuri particulare. În acest caz se poate adăuga următoarea linie de cod în spaţiul lăsat liber al pseudocodului de mai sus:

dacă $|xcrt - x_k| <$ zeroul_maşinii() atunci întoarce y_k

Reamintim că testarea egalităţii numerelor reale nu se poate face decât cel mult până la zeroul maşini (vezi discuţia şi algoritmul de la pagina 34).

4.3.3 Eroarea de trunchiere

În cazul în care funcţia de interpolat f este continuă şi derivabilă de un număr finit de ori, se poate obţine o margine a erorii de interpolare

$$e(x) = f(x) - g(x), \tag{4.30}$$

care poate fi privită ca o eroare de trunchiere deoarece interpolarea caută o aproximare într-un spaţiu finit dimensional de funcţii.

Datorită condiţiilor de interpolare (4.11), eroarea de interpolare este nulă în nodurile de interpolare

$$e(x_k) = 0, \quad k = 0, \ldots, n \tag{4.31}$$

şi, în consecinţă, putem considera o aproximare a erorii de interpolare de tipul

$$E(x) = a \prod_{k=0}^{n} (x - x_k), \tag{4.32}$$

unde a ţine seama de abaterile dintre funcţiile f şi g. Notăm diferenţa între eroarea de interpolare $e(x)$ şi aproximarea ei prin polinomul $E(x)$ cu:

$$h(x) = e(x) - E(x) = f(x) - g(x) - a \prod_{k=0}^{n} (x - x_k). \tag{4.33}$$

Este evident că $h(x)$ se anulează în toate nodurile reţelei de interpolare

$$h(x_k) = 0, \quad k = 0, \ldots, n. \tag{4.34}$$

Este interesant că, alegând convenabil constanta a, de exemplu

$$a = \frac{f(\alpha) - g(\alpha)}{\prod_{k=0}^{n} (\alpha - x_k)}, \tag{4.35}$$

unde α este ales arbitrar în intervalul $[a, b]$, $h(x)$ se mai anulează într-un punct şi anume

$$h(\alpha) = 0. \tag{4.36}$$

În consecinţă, h are $n + 2$ rădăcini: α, x_0, ..., x_n. Dacă presupunem f continuă şi derivabilă, atunci şi h este continuă şi derivabilă, iar funcţia

$$h'(x) = f'(x) - g'(x) - aP_n'(x), \tag{4.37}$$

unde $P_n(x) = \prod_{k=0}^{n} (\alpha - x_k)$ are, conform teoremei lui Role, $n + 1$ rădăcini. Dacă h' este continuă şi derivabilă, atunci h'' are n rădăcini. Continuând raţionamentul, va rezulta că $h^{(n+1)}$ va avea o singură rădăcină $\xi \in [a, b]$:

$$h^{(n+1)}(\xi) = 0. \tag{4.38}$$

Dar deoarece $g^{(n+1)}(x) = 0$ rezultă că

$$h^{(n+1)}(x) = f^{(n+1)}(x) - a(n+1)!, \tag{4.39}$$

de unde

$$a = \frac{f^{(n+1)}(\xi)}{(n+1)!}. \tag{4.40}$$

Deoarece $h(\alpha) = 0$, rezultă că

$$e(\alpha) = E(\alpha) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (\alpha - x_k). \tag{4.41}$$

Dar α a fost ales arbitrar, astfel încât, în relaţia de mai sus se poate substitui α cu x.

În concluzie, pentru orice x există un $\xi \in [a, b]$ astfel încât eroarea de interpolare este

$$e(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (x - x_k). \tag{4.42}$$

Să considerăm o margine M_{n+1} a derivatei de ordinul $n+1$ a funcţiei f:

$$|f^{(n+1)}(x)| \leq M_{n+1}, \quad (\forall) x \in [a, b]. \tag{4.43}$$

Rezultă atunci că eroarea de interpolare este mărginită de

$$|e(x)| \leq \frac{M_{n+1}}{(n+1)!} \prod_{k=0}^{n} |x - x_k|. \tag{4.44}$$

Eroarea de interpolare depinde deci de marginea derivatei de un ordin egal cu numărul de puncte de tabel. Această afirmaţie este ilustrată în fig.4.6.

4.3.4 Metoda Newton

În metoda Newton funcţiile de bază sunt alese astfel încât matricea sistemului de rezolvat în etapa de pregătire a coeficienţilor să fie triunghiular inferioară:

$$\begin{aligned} &\varphi_0(x) = 1, \\ &\varphi_1(x) = (x - x_0), \\ &\varphi_2(x) = (x - x_0)(x - x_1), \\ &\cdots \\ &\varphi_n(x) = (x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{4.45}$$

Polinomul de interpolare (4.13) este atunci

$$g(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}), \tag{4.46}$$

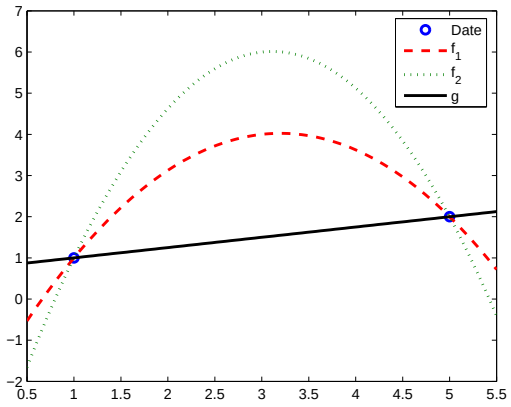


Figura 4.6: Ilustrarea formulei (4.44) în cazul unui tabel de valori cu două puncte. Presupunând că funcţia adevărată este o parabolă, atunci eroarea de interpolare este mai mică pentru parabola f_1, care are derivata de ordinul doi (curbura) mai mică.

iar condiţiile de interpolare (4.14) devin

$$\begin{cases} c_0 = y_0, \\ c_0 + c_1(x_1 - x_0) = y_1, \\ c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1) = y_2, \\ \cdots \\ c_0 + c_1(x_n - x_0) + c_2(x_n - x_0)(x_n - x_1) + \cdots c_n(x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1}) = y_n. \end{cases} \tag{4.47}$$

Sistemul de rezolvat în etapa de pregătire are structura inferior triunghiulară şi se va rezolva prin substituţie progresivă. Coeficienţii reprezintă diferenţe divizate ale funcţiei faţă de submulţimi ale nodurilor de discretizare:

$$\begin{cases} c_0 = y_0 = f[x_0] \\ c_1 = (y_1 - y_0)/(x_1 - x_0) = f[x_0, x_1] \\ c_2 = [y_2 - y_0 - (y_1 - y_0)/(x_1 - x_0)(x_2 - x_0)]/(x_2 - x_0)/(x_2 - x_1) = f[x_0, x_1, x_2], \\ \vdots \\ c_n = \cdots = f[x_0, x_1, x_2, \ldots, x_n]. \end{cases} \tag{4.48}$$

În general, *diferenţa divizată faţa de o submulţime a nodurilor reţelei de discretizare* se defineşte în mod recursiv astfel:

$$f[x_i, x_{i+1}, \ldots, x_{i+m}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+m}] - f[x_i, x_{i+1}, \ldots, x_{i+m-1}]}{x_{i+m} - x_i}, \tag{4.49}$$

unde diferenţa divizată faţă de o submulţime cu un punct este valoarea funcţiei în acel punct $f[x_i] = f(x_i)$.

Următoarele două proprietăţi ale diferenţelor divizate sunt importante în contextul interpolării cu metoda Newton:

- Variabilele independente pot fi permutate într-o diferenţă divizată fără ca valoarea funcţiei să se modifice. Acest lucru este evident pentru diferenţa divizată faţă de o submulţime cu două noduri:

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{f[x_0] - f[x_1]}{x_0 - x_1} = f[x_1, x_0] \tag{4.50}$$

  şi, pe baza relaţiei de recurenţă (4.49), această proprietate poate fi generalizată pentru o mulţime oarecare de noduri.

- Diferenţa divizată faţă de o submulţime cu două puncte identice este derivata funcţiei:

$$f[x_0, x_0] = \lim_{x_1 \to x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(x_0). \tag{4.51}$$

  Legătura dintre diferenţele divizate ale unei funcţii şi derivatele sale poate fi generalizată, aşa cum se va vedea în cele ce urmează.

Polinomul de interpolare (4.46) cu coeficienţii daţi de (4.48) este

$$\begin{aligned} g(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ &+ \cdots + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{4.52}$$

Dacă nodurile de interpolare tind toate către acelaşi punct x_0, atunci polinomul de interpolare $g(x)$ tinde către seria Taylor a funcţiei f în x_0:

$$f[x_0] + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_0](x - x_0)^2 + \cdots f[\underbrace{x_0, x_0, \ldots, x_0}_{n+1}](x - x_0)^n. \tag{4.53}$$

În consecinţă, termeni succesivi ai polinomului interpolării Newton aproximează termeni corespunzători din dezvoltarea în serie Taylor. Acest lucru conferă un avantaj deosebit metodei Newton faţă de metoda Lagrange, pentru că ea permite controlul erorii de trunchie-re, efortul de calcul putând fi adaptat preciziei impuse soluţiei. Mai mult, atunci când se adaugă un punct suplimentar în reţeaua de interpolare, se poate porni de la interpolarea cu un grad mai scăzut şi trebuie doar adăugat un singur termen în sumă, nefiind necesară refacerea totală a calculelor ci doar evaluarea unui singur termen suplimentar.

Dacă $g(x)$ reprezintă interpolarea funcţiei $f(x)$ pe o reţea cu nodurile x_0, x_1, ..., x_n, atunci la adăugarea unui nod nou x_{n+1} (care poate avea orice poziţie faţă de punctele anterioare), noul polinom de interpolare va fi

$$h(x) = g(x) + f[x_0, x_1, \ldots, x_n, x_{n+1}](x - x_0)(x - x_1) \cdots (x - x_n). \tag{4.54}$$

Condiţia de interpolare pentru noul punct este $h(x_{n+1}) = f(x_{n+1})$, deci

$$f(x_{n+1}) = g(x_{n+1}) + f[x_0, x_1, \ldots, x_n, x_{n+1}](x_{n+1} - x_0)(x_{n+1} - x_1)\cdots(x_{n+1} - x_n). \quad (4.55)$$

Deoarece x_{n+1} poate fi ales arbitrar, în (4.55) putem înlocui x_{n+1} cu x, deci

$$f(x) = g(x) + f[x_0, x_1, \ldots, x_n, x](x - x_0)(x - x_1)\cdots(x - x_n), \quad (4.56)$$

de unde rezultă că eroarea de trunchiere făcută la interpolarea funcţiei f pe reţeaua de noduri x_0, x_1, …, x_n este

$$e(x) = f(x) - g(x) = f[x_0, x_1, \ldots, x_n, x] \prod_{k=0}^{n} (x - x_k). \quad (4.57)$$

Am demonstrat însă şi că eroarea de interpolare este dată de formula (4.42):

$$e(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (x - x_k). \quad (4.58)$$

Cum pentru un tabel de valori dat polinomul de interpolare global este unic, rezultă că cele două erori de trunchiere (4.57) şi (4.58) sunt acelaşi, de unde rezultă că, pentru orice diviziune, există ξ astfel încât

$$f[x_0, x_1, \ldots, x_n, x] = \frac{f^{(n+1)}(\xi)}{(n+1)!}. \quad (4.59)$$

Iată deci că, diferenţele divizate pe o reţea cu mai mult de două noduri reprezintă aproximări ale derivatelor de ordin superior, ordinul derivatei fiind cu unu mai mic decât numărul de noduri ale diviziunii. În particular, dacă în relaţia (4.59) toate nodurile tind către x_0, rezultă că are loc:

$$f[\underbrace{x_0, x_0, \ldots, x_0, x_0}_{n+2}] = \frac{f^{(n+1)}(x_0)}{(n+1)!}. \quad (4.60)$$

Algoritmul metodei Newton se bazează pe calculul tabelului diferenţelor divizate în etapa de pregătire, calcul ilustrat de următoarea schemă:

x_0	x_1	x_2	$\cdots$	x_{n-1}	x_n
$y_0 = f[x_0]$	$y_1 = f[x_1]$	$y_2 = f[x_2]$	$\cdots$	$y_{n-1} = f[x_{n-1}]$	$y_n = f[x_n]$
$f[x_0, x_1]$	$f[x_1, x_2]$		$\cdots$	$f[x_{n-1}, x_n]$	
$f[x_0, x_1, x_2]$			$\cdots$	$f[x_{n-2}, x_{n-1}, x_n]$	
$\vdots$					
$f[x_0, x_1, \ldots, x_n]$					

În total sunt calculate $n(n+1)/2$ diferenţe divizate, din care utile pentru polinomul de interpolare sunt doar n, şi anume valorile marcate cu roşu în schema de mai sus. Pseudocodul metodei Newton este următorul.

```
procedură Newton_pregătire(n, x, y, dd)
; pregăteşte diferenţele divizate din metoda Newton
; declaraţii - parametri de intrare
întreg n                  ; dimensiunea problemei - numărul de intervale
tablou real x[n], y[n]    ; tabelul de valori, indici de la zero
; declaraţii - parametri de ieşire
tablou real dd[n][n]      ; diferenţele divizate
pentru j = 0, n
      dd(0, j) = y(j)
•
pentru i = 1, n
      pentru j = 0, n − i
            dd(i, j) = (dd(i − 1, j) − dd(i − 1, j − 1))/(x(j + i) − x(j))
      •
•
retur

funcţie Newton_evaluare(n, x, y, dd, xcrt)
; evaluează polinomul de interpolare Newton în punctul xcrt
; declaraţii - parametri de intrare
întreg n                  ; dimensiunea problemei - numărul de intervale
tablou real x[n], y[n]    ; tabelul de valori, indici de la zero
tablou real dd[n][n]      ; diferenţele divizate
real xcrt                 ; punctul de evaluat
real ycrt                 ; valoarea funcţiei în punctul de evaluat
ycrt = dd(n, 0)
pentru k = n − 1, 0, −1
      ycrt = dd(k, 0) + (xcrt − x_k) ∗ ycrt
•
întoarce ycrt
```

Funcţia de evaluare se bazează pe scrierea polinomului sub forma

$$g(x) = c_0 + (x - x_0)\left[c_1 + (x - x_1)\left[c_2 + \cdots [c_{n-1} + c_n(x - x_n)] \cdots \right]\right]. \tag{4.61}$$

În etapa de pregătire se efectuează $\sum_{i=1}^{n} 2(n-i) = 2n(n-1)/2$ operaţii, această etapă având ordinul de complexitate $T_p = O(n^2)$. Deoarece etapa de evaluare are ordinul de complexitate $T_e = O(3n)$, rezultă un efort de calcul total în metoda Newton $T_N = O(n^2 + 3mn)$.

Rezumând, efortul de calcul pentru cele trei metode de interpolare globală discutate, este prezentat în tabelul 4.1.

Tabelul 4.1: Efortul de calcul pentru metodele de interpolare globală.

Metoda	Pregătire	Evaluare
Clasică	$O(2n^3/3)$	$O(2mn)$
Lagrange	$O(2n^2)$	$O(5mn)$
Newton	$O(n^2)$	$O(3mn)$

În concluzie, metodele clasică, Lagrange, Newton dau teoretic acelaşi rezultat pentru că polinomul de interpolare est unic. Dintre toate, metoda Newton este cea mai eficientă din punct de vedere al timpului de pregătire, cât şi a celui de evaluare. Avantajul major este însă acela că metoda Newton permite controlul erorii de trunchiere. Evaluarea polinomului de interpolare cu controlul erorii de trunchiere este dată de procedura următoare:

`procedură` Newton_evaluare2$(n, x, y, dd, xcrt, ycrt, et)$
; evaluează polinomul de interpolare Newton în punctul $xcrt$
; declaraţii - parametri de intrare
`întreg` n ; dimensiunea problemei - numărul de intervale
`tablou real` $x[n], y[n]$; tabelul de valori, indici de la zero
`tablou real` $dd[n][n]$; diferenţele divizate
`real` $xcrt$; punctul de evaluat
`real` $ycrt$; valoarea funcţiei în punctul de evaluat
`real` et ; eroarea de trunchiere
$ycrt = dd(n-1, 0)$
`pentru` $k = n-2, 0, -1$
    $ycrt = dd(k, 0) + (xcrt - x_k) * ycrt$
•
$et = dd(n, 0)$
`pentru` $k = 0, n-1$
    $et = et * (xcrt - x_k)$
•
`retur`

4.3.5 Un exemplu simplu

Să ilustrăm cele trei metode prezentate până acum pentru tabelul de valori:

x	-1	2	4
y	-6	9	49

Deoarece tabelul are trei puncte, polinomul de interpolare globală va fi de gradul doi.

În metoda clasică, polinomul de interpolare este de forma

$$g(x) = c_0 + c_1 x + c_2 x^2, \tag{4.62}$$

unde cei trei coeficienţi necunoscuţi se determină prin impunerea condiţiilor de interpolare

$$\begin{aligned} g(-1) &= 6, \\ g(2) &= 9, \\ g(4) &= 49, \end{aligned} \tag{4.63}$$

care conduc la rezolvarea sistemului algebric liniar

$$\begin{aligned} c_0 &- c_1 + c_2 = 6, \\ c_0 &+ 2c_1 + 4c_2 = 9, \\ c_0 &+ 4c_1 + 16c_2 = 49. \end{aligned} \tag{4.64}$$

Soluţia acestui sistem este $c_0 = -7$, $c_1 = 2$, $c_2 = 3$, de unde rezultă expresia polinomului de interpolare $g(x) = 3x^2 + 2x - 7$.

În metoda Lagrange, se construiesc polinoamele de interpolare Lagrange:

$$l_0(x) = \frac{(x-2)(x-4)}{(-1-2)(-1-4)} = \frac{x^2 - 6x + 8}{15}, \tag{4.65}$$

$$l_1(x) = \frac{(x+1)(x-4)}{(2+1)(2-4)} = \frac{x^2 - 3x - 4}{-6}, \tag{4.66}$$

$$l_2(x) = \frac{(x+1)(x-2)}{(4+1)(4-2)} = \frac{x^2 - x - 2}{10}, \tag{4.67}$$

iar polinomul de interpolare globală este

$$g(x) = -6l_0(x) + 9l_1(x) + 49l_2(x) = 3x^2 + 2x - 7. \tag{4.68}$$

În metoda Newton se construieşte mai întâi tabelul diferenţelor divizate[2]:

-1	2	4
$f[x_0] = -6$	$f[x_1] = 9$	$f[x_2] = 49$
$f[x_0, x_1] = 5$	$f[x_1, x_2] = 20$	
$f[x_0, x_1, x_2] = 3$		

[2] $f[x_0, x_1] = (9+6)/(2+1) = 5$, $f[x_1, x_2] = (49-9)/(4-2)$, $f[x_0, x_1, x_2] = (20-5)/(4+1)$

Polinomul de interpolare se calculează uşor ca

$$g(x) = -6 + 5(x+1) + 3(x+1)(x-2) = 3x^2 + 2x - 7. \tag{4.69}$$

Dacă adăugăm un nou punct în acest tabel

x	-1	2	4	3
y	-6	9	49	10

putem calcula uşor polinomul de gradul trei. Tabelul diferenţelor divizate se completează cu noile valori

-1		2		4		3
-6		9		49		10
	5		20		39	
		3		19		
			4			

iar polinomul de interpolare este

$$h(x) = 3x^2 + 2x - 7 + 4(x+1)(x-2)(x-4) = 4x^3 - 17x^2 + 10x + 25.$$

Termenul $e(x) = 4(x+1)(x-2)(x-4)$ reprezintă eroarea de trunchiere a polinomului de interpolare de grad doi $g(x) = 3x^2 + 2x - 7$ pentru tabelul de valori ce conţine patru puncte.

4.3.6 Interpolarea Chebyshev

Toate cele trei metode de intepolare polinomială globală descrise mai sus au un foarte mare dezavantaj dacă gridul de interpolare este uniform. Acest lucru a fost pus în evidenţă de Runge care a observat că la interpolarea funcţiei $f : [-5, 5] \to \mathbb{R}$, $f(x) = 1/(1+x^2)$ pe o reţea echidistantă de puncte apar oscilaţii la capetele polinomului de interpolare, oscilaţii cu atât mai mari cu cât gradul polinomului este mai mare (fig. 4.7). *Efectul de oscilaţie a polinomului de interpolare între nodurile reţelei de interpolare se numeşte efect Runge.*

Efectul Runge se poate explica prin faptul că eroarea de trunchiere dată de (4.42) poate fi oricât de mare datorită faptului că $f^{(n+1)}(\xi)$ poate fi oricât de mare.

Efectul Runge se poate elimina dacă nodurile reţelei de interpolare se îndesesc către capetele domeniului. Oscilaţii minime ale polinomului de interpolare se obţin dacă plasarea nodurilor în interiorul domeniului de definiţie se face în concordanţă cu rădăcinile polinoamelor Chebyshev. În intervalul $[-1, 1]$, acestea sunt

$$x_i = \cos\left(\frac{2i+1}{2(n+1)}\pi\right), \quad i = 0, \ldots, n, \tag{4.70}$$

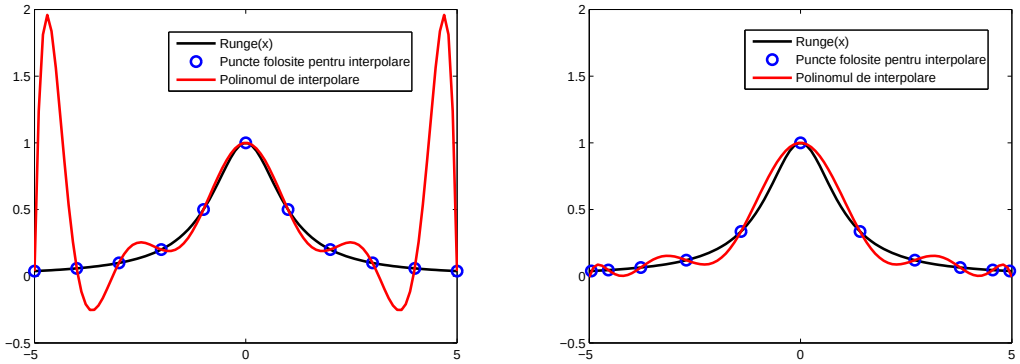


Figura 4.7: Efectul Runge: polinomul de intepolare are oscilaţii la capetele intervalului dacă gridul de discretizare este uniform (stânga). Efectul poate fi eliminat prin plasarea nodurilor de discretizare în conformitate cu rădăcinile polinoamelor Chebyshev (dreapta).

Printr-o transformare liniară, se poate deduce că într-un interval arbitrar $[a, b]$ nodurile Chebyshev trebuie plasate conform formulei

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i+1}{2(n+1)}\pi\right). \tag{4.71}$$

În concluzie, interpolarea Chebyshev este tot interpolarea polinomială globală numai că nodurile reţelei de interpolare sunt alese în concordanţă cu rădăcinile polinoamelor Chebyshev.

De cele mai multe ori însă, utilizatorul nu are libertatea de a alege punctele reţelei de interpolare. De aceea, limitarea acestei metode este aceea că se poate aplica numai funcţiilor definite prin cod şi nu tabelar. În cazul funcţiilor date tabelar, pentru a obţine polinoame de interpolare care să nu prezinte efect Runge, este mai eficient dacă se foloseşte interpolarea pe porţiuni.

4.4 Metode de interpolare polinomială pe porţiuni

Interpolarea polinomială globală poate suferi de efectul Runge, efect care poate fi eliminat în cazul funcţiilor al căror cod este cunoscut prin folosirea unei interpolări Chebyshev. În cazul funcţiilor date tabelar, interpolări satisfăcătoare se obţin doar dacă polinoamele de interpolare se caută pe porţiuni ale domeniului de definiţie. Rezultatul interpolării pe porţiuni este ceea ce în matematică se numeşte "funcţie cu acoladă", spre deosebire de interpolarea globală al cărei rezultat este o singură expresie compactă, valabilă pe tot domeniul de definiţie.

4.4.1 Interpolarea liniară pe porţiuni

Cea mai simplă interpolare pe porţiuni este aceea în care funcţiile de bază $\varphi_k(x)$ sunt polinoame Lagrange $l_k(x)$ definite pe porţiuni. Ele satisfac de asemenea condiţiile $l_k(x_k) = 1$ şi $l_k(x_j) = 0$ pentru $j \neq k$, dar variaţia între nodurile grilei de discretizare este liniară (fig. 4.8):

$$l_0(x) = \begin{cases} \frac{x-x_1}{x_0-x_1} & \text{dacă} \quad x \in [x_0, x_1] \\ 0 & \text{dacă} \quad x \in (x_1, x_n] \end{cases} \tag{4.72}$$

$$l_k(x) = \begin{cases} \frac{x-x_{k-1}}{x_k-x_{k-1}} & \text{dacă} \quad x \in [x_{k-1}, x_k) \\ \frac{x-x_{k+1}}{x_k-x_{k+1}} & \text{dacă} \quad x \in [x_k, x_{k+1}] \\ 0 & \text{dacă} \quad x \in [x_1, x_{k-1}) \cup (x_{k+1}, x_n] \end{cases} \qquad k = 2, n-1 \tag{4.73}$$

$$l_n(x) = \begin{cases} \frac{x-x_{n-1}}{x_n-x_{n-1}} & \text{dacă} \quad x \in [x_{n-1}, x_n] \\ 0 & \text{dacă} \quad x \in [x_0, x_{n-1}) \end{cases} \tag{4.74}$$

Graficul polinomului de interpolare

$$g(x) = \sum_{k=0}^{n} c_k l_k(x) \tag{4.75}$$

este chiar linia poligonală care uneşte punctele reţelei de interpolare (fig. 4.9). Folosirea acestui polinom în afara domeniului de definiţie se numeşte *extrapolare*.

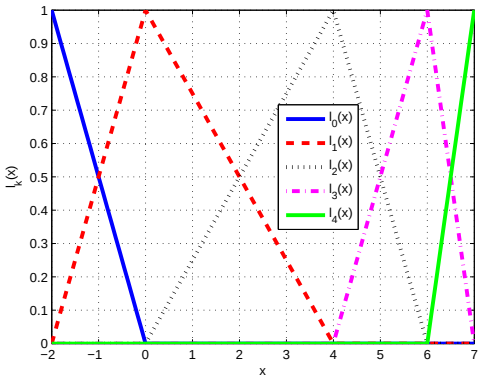


Figura 4.8: Funcţii Lagrange pe porţiuni.

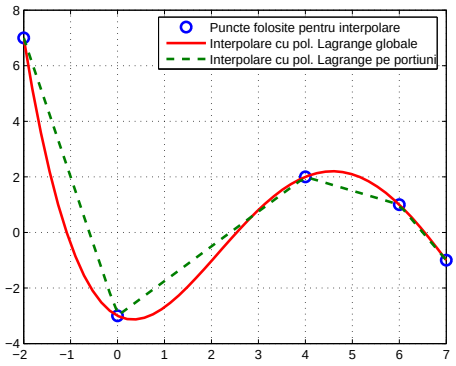


Figura 4.9: Interpolare globală şi interpolare pe porţiuni.

Polinomul de interpolare are o expresie diferită pe fiecare sub-interval definit de reţeaua de discretizare

$$g(x) = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k) + y_k, \quad \text{pentru} \quad x \in [x_k, x_{k+1}], \quad k = 0, \ldots, n-1. \tag{4.76}$$

Polinomul de interpolare este continuu, şi de aceea nu are importanţă în care interval (la stânga sau la dreapta) este inclus un nod al reţelei de interpolare. Procedura de evaluare a polinomului de interpolare liniară pe porţiuni se reduce în ultimă instanţă la evaluarea ecuaţiei unei drepte care trece prin două puncte date, de coordonate (x_k, y_k) şi, respectiv, (x_{k+1}, y_{k+1}). Singura dificultate este de a determina sub-intervalul din care face parte punctul de evaluat $xcrt$.

Pseudocodul următor evaluează un polinom de interpolare liniară pe porţiuni, în care căutarea intervalului în care se află punctul de evaluat se face prin *metoda căutării binare*. Ideea acestei metode este de a elimina la fiecare iteraţie jumătate din numărul de intervale posibile prin compararea punctului de evaluat cu punctul din reţeaua de discretizare care împarte numărul de intervale posibile în două. Pseudocodul acestei metode a fost prezentat la pagina 28.

`funcţie` interpolare_lpp($n, x, y, xcrt$)
; evaluează polinomul de interpolare liniară pe porţiuni în punctul $xcrt$
; declaraţii - parametri de intrare
`întreg` n ; dimensiunea problemei - numărul de intervale
`tablou real` $x[n], y[n]$; tabelul de valori, indici de la zero
`real` $xcrt$; punctul de evaluat
$k = \text{cauta}(n, x, xcrt)$
`întoarce` $(y_{k+1} - y_k)/(x_{k+1} - x_k) * (xcrt - x_k) + y_k$

Interpolarea liniară pe porţiuni are dezavantajul că funcţia obţinută prin interpolare nu este derivabilă în nodurile reţelei de interpolare. Pentru a genera interpolări pe porţiuni care să nu fie numai continue ci şi derivabile este necesară creşterea gradului polinoamelor care se folosesc pe porţiuni.

4.4.2 Interpolarea Hermite

O funcţie de interpolare derivabilă s-ar putea construi dacă s-ar cunoaşte valorile derivatelor funcţiei de interpolat în nodurile reţelei de interpolare.

Se defineşte astfel *problema interpolării Hermite* ca fiind problema interpolării unei funcţii pe o reţea de noduri x_k în care se cunosc valorile funcţiei y_k şi valorile derivatelor acesteia y'_k. Tabelul de valori necesar pentru o interpolare Hermite este deci de tipul

x	x_0	x_1	$\cdots$	x_k	$\cdots$	x_n
y	y_0	y_1	$\cdots$	y_k	$\cdots$	y_n
y'	y'_0	y'_1	$\cdots$	y'_k	$\cdots$	y'_n

Pe fiecare subinterval $[x_k, x_{k+1}]$ unde $k = 0, \ldots, n-1$ trebuie puse patru condiţii de interpolare: două pentru valorile funcţiei şi două pentru valorile derivatei la capete:

$$\begin{cases} g(x_k) = y_k, \\ g(x_{k+1}) = y_{k+1}, \\ g'(x_k) = y'_k, \\ g'(x_{k+1}) = y'_{k+1}. \end{cases} \tag{4.77}$$

Aceste condiţii asigură continuitatea şi derivabilitatea polinomului de interpolare.

Deoarece se doreşte ca impunerea condiţiilor de interpolare să conducă la un polinom unic, rezultă că pe fiecare subinterval trebuie considerat drept polinom de interpolare unul de grad trei. Cei patru coeficienţi ai polinomului de interpolare vor rezulta în mod unic din impunerea condiţiilor de interpolare (4.77). Pentru simplificarea calculelor, este util ca polinomul de interpolare pe porţiuni să fie scris ca

$$g(x) = c_{0k} + c_{1k}(x - x_k) + c_{2k}(x - x_k)^2 + c_{3k}(x - x_k)^3, \quad x \in [x_k, x_{k+1}], \tag{4.78}$$

derivata lui fiind

$$g'(x) = c_{1k} + 2c_{2k}(x - x_k) + 3c_{3k}(x - x_k)^2, \quad x \in [x_k, x_{k+1}]. \tag{4.79}$$

Impunând cele patru condiţii de interpolare rezultă următorul sistem de patru ecuaţii cu patru necunoscute

$$\begin{cases} c_{0k} = y_k, \\ c_{0k} + c_{1k}(x_{k+1} - x_k) + c_{2k}(x_{k+1} - x_k)^2 + c_{3k}(x_{k+1} - x_k)^3 = y_{k+1}, \\ c_{1k} = y'_k, \\ c_{1k} + 2c_{2k}(x_{k+1} - x_k) + 3c_{3k}(x_{k+1} - x_k)^2 = y'_{k+1}, \end{cases} \tag{4.80}$$

a cărui soluţie este

$$\begin{cases} c_{0k} = y_k, \\ c_{1k} = y'_k, \\ c_{2k} = \left[3(y_{k+1} - y_k) - (x_{k+1} - x_k)(2y'_k + y'_{k+1})\right] / (x_{k+1} - x_k)^2, \\ c_{3k} = \left[(y'_{k+1} + y'_k) - 2(2y_{k+1} - y_k)/(x_{k+1} - x_k)\right] / (x_{k+1} - x_k)^2. \end{cases} \tag{4.81}$$

Problema interpolării Hermite este aceea că de cele mai multe ori nu se cunosc informaţii despre derivatele funcţiei. Evaluarea derivatelor se face numeric şi această etapă este numită etapă de pregătire a interpolării Hermite. Există mai multe variante de a aproxima numeric derivatele. O primă variantă, cunoscută sub numele de *interpolare Bessel*, este de a folosi formule simple pentru aproximarea derivatelor. De exemplu, la extremităţi se

poate considera o aproximare liniară pe ultimul subinterval şi atunci valorile se aproximează cu

$$y_0' = \frac{y_1 - y_0}{x_1 - x_0} \quad y_n' = \frac{y_n - y_{n-1}}{x_n - x_{n-1}}, \tag{4.82}$$

iar în nodurile interioare, derivând expresia unei parabole ce interpolează trei noduri consecutive, se poate deduce o formulă aproximativă de tipul

$$y_k' = \frac{\beta^2 y_{k+1} + (\alpha^2 - \beta^2) y_k - \alpha^2 y_{k-1}}{\alpha\beta(\alpha + \beta)h}, \tag{4.83}$$

unde am notat $x_{k+1} - x_k = \alpha h$, $x_k - x_{k-1} = \beta h$. Această abordare[3] nu este tocmai naturală pentru că interpolarea Hermite foloseşte pe porţiuni polinoame de ordinul 3 şi nu de ordinul 1 sau 2 cum sunt presupuse în formulele (4.82) şi (4.83).

Metoda cea mai avantajoasă de evaluare a derivatelor este cea în care ele se deduc din impunerea unor condiţii de continuitate suplimentare pentru derivata a doua a polinomului de interpolare în nodurile reţelei de interpolare. În acest caz, interpolarea Hermite se numeşte *interpolare spline cubică clasică*. În nodurile interioare se impune condiţia:

$$g''(x_k - 0) = g''(x_k + 0), \quad k = 1, \ldots, n-1. \tag{4.84}$$

Din (4.79) rezultă expresia derivatei a doua a polinomului de interpolare

$$g''(x) = 2c_{2k} + 6c_{3k}(x - x_k), \quad x \in [x_k, x_{k+1}]. \tag{4.85}$$

Condiţia (4.84) devine

$$2c_{2,k-1} + 6c_{3,k-1}(x_k - x_{k-1}) = 2c_{2k}, \quad k = 1, \ldots, n-1. \tag{4.86}$$

Simplificând relaţia cu 2 şi înlocuind expresiile coeficienţilor date de (4.81), relaţiile (4.86) devin

$$\begin{aligned} &\frac{1}{x_k - x_{k-1}} y_{k-1}' + 2\left(\frac{1}{x_k - x_{k-1}} + \frac{1}{x_{k+1} - x_k}\right) y_k' + \frac{1}{x_{k+1} - x_k} y_{k+1}' = \\ &= 3\frac{y_k - y_{k-1}}{(x_k - x_{k-1})^2} + 3\frac{y_{k+1} - y_k}{(x_{k+1} - x_k)^2}. \end{aligned} \tag{4.87}$$

Cele $n-1$ relaţii (4.87) conţin $n+1$ necunoscute. Pentru unicitate se impun încă două condiţii la capete. O variantă este să se impună valorile derivatelor y_0' şi y_n' ca la interpolarea Bessel, caz în care se spune că s-au impus *condiţii forţate la capete*. O altă variantă este aceea în care la capete se impune ca derivata a doua a polinomului de interpolare să fie zero:

$$g''(x_0) = 0, \quad g''(x_n) = 0. \tag{4.88}$$

[3] Formulele (4.82) şi (4.83) sunt formule de derivare numerică, problemă explicată în detaliu în capitolul următor.

În acest caz se spune că s-au impus *condiţii naturale la capete*. După impunerea acestor condiţii, sistemul (4.87) devine bine formulat. El este un sistem tridiagonal din a cărui rezolvare rezultă valorile derivatelor necesare calculului coeficienţilor polinomului de interpolare Hermite. Din impunerea condiţiei naturale la capătul din stânga rezultă următoarea relaţie ce trebuie să existe între derivatele funcţiei:

$$2y'_0 + y'_1 = 3\frac{y_1 - y_0}{x_1 - x_0}, \tag{4.89}$$

iar din impunerea condiţiei la capătul din dreapta rezultă

$$y'_{n-1} + 2y'_n = 3\frac{y_n - y_{n-1}}{x_n - x_{n-1}}. \tag{4.90}$$

Oricare ar fi modul de tratare a extremităţilor, interpolarea spline cubică clasică are avantajul că minimizează curbura pătratică medie a polinomului de intepolare, mai precis expresia

$$\int_a^b \left(g''(x)\right)^2 \, \mathrm{d}x \tag{4.91}$$

este minimă faţa de orice altă interpolare polinomială pe porţiuni[4]. Datorită acestei proprietăţi, în această interpolare nu apar oscilaţii nedorite între punctele reţelei de interpolare.

Algoritmul interpolării spline cubice este următorul:

`funcţie` pregăteşte_spline$(n, x, y, yder)$
; calculează derivatele funcţiei în nodurile reţelei de discretizare
; declaraţii - parametri de intrare
`întreg` n ; dimensiunea problemei - numărul de intervale
`tablou real` $x[n], y[n]$; tabelul de valori, indici de la zero
`tablou real` $yder[n]$; parametri de ieşire
`tablou real` $p[n], q[n], r[n]$; matricea tridiagonală asamblată
; asamblează matricea tridiagonală
$q_0 = 2$
$r_0 = 1$
$b_0 = 3(y_1 - y_0)/(x_1 - x_0)$
`pentru` $k = 1, n-1$
    $p_k = 1/(x_k - x_{k-1})$
    $q_k = 2/(x_k - x_{k-1}) + 2/(x_{k+1} - x_k)$
    $r_k = 1/(x_{k+1} - x_k)$

[4]În limba engleză *spline* înseamnă, printre altele, o fâşie de lemn, destul de elastică, folosită pentru desenarea curbelor, ca un florar.

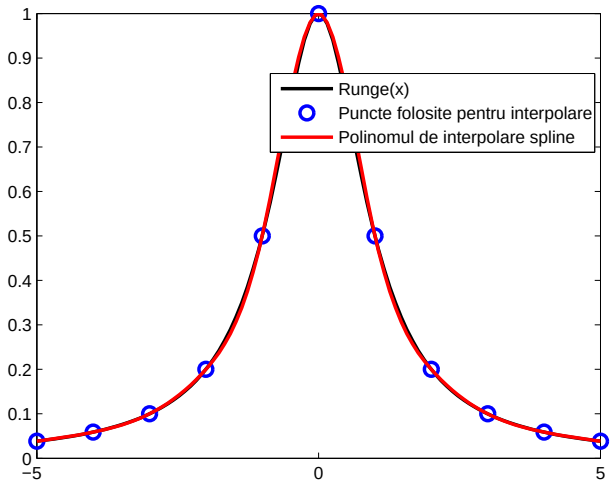


Figura 4.10: Interpolarea *spline* a funcţiei Runge pe o reţea uniformă de puncte.

$$b_k = 3(y_k - y_{k-1})/(x_k - x_{k-1})^2 + 3(y_{k+1} - y_k)/(x_{k+1} - x_k)^2$$

•

$p_n = 1$
$q_n = 2$
$b_n = 3(y_n - y_{n-1})/(x_n - x_{n-1})$
Gauss_tridiag$(n+1, p, q, r, b, yder)$; procedură descrisă la pagina 86

`funcţie` aproximare_spline$(n, x, y, yder, xcrt)$
; evaluează polinomul de aproximare spline în punctul $xcrt$
; declaraţii - parametri de intrare
`întreg` n ; dimensiunea problemei - numărul de intervale
`tablou real` $x[n], y[n], yder[n]$; tabelul de valori, indici de la zero
`real` $xcrt$; punctul de evaluat
$k =$ cauta$(n, x, xcrt)$
$c0k = y_k$
$c1k = yder_k$
$h = x_{k+1} - x_k$
$c2k = (3(y_{k+1} - y_k) - h(2 * yder_k + yder_{k+1}))/h^2$
$c3k = (yder_{k+1} + yder_k - 2(y_{k+1} - y_k)/h)/h^2$
$hcrt = xcrt - x_k$
`întoarce` $c0k + c1k * hcrt + c2k * hcrt^2 + c3k * hcrt^3$

Rezultatul rulării unui astfel de algoritm este prezentat în fig. 4.10.

Capitolul 5

Derivarea numerică

Necesitatea evaluării derivatelor funcţiilor este importantă în problemele de inginerie. Ele apar, de exemplu, în relaţii utile pentru evaluarea unor mărimi (curent, tensiune), în definirea senzitivităţilor unor mărimi de interes în raport cu anumiţi parametri consideraţi variabili, senzitivităţi utile în proiectarea optimală a dispozitivelor. Derivarea numerică se bazează pe interpolare. Formulele de derivare numerică provin din derivarea polinomului de interpolare şi ele se reduc la calcule aritmetice ce folosesc formule relativ simple.

Un alt exemplu în care apar derivate este cel în care modelul matematic al problemei conduce la ecuaţii sau sisteme de ecuaţii diferenţiale, foarte puţine dintre acestea putând fi rezolvate prin metode analitice. Rezolvarea ecuaţiilor diferenţiale folosind formule de derivare numerică pentru derivatele ce intervin este cunoscută sub numele de *metoda diferenţelor finite*.

5.1 Formularea problemei derivării numerice

Vom presupune mai întâi cazul unui funcţii reale ce depinde de o singură variabilă reală. Problema derivării unei astfel de funcţii se formulează astfel.

Se dă funcţia $f : [a, b] \to \mathbb{R}$ (cunoscută prin date sau prin cod) şi **se cere** evaluarea numerică a derivatei $f'(x_0)$, unde $x_0 \in [a, b]$.

Matematic, derivata se defineşte ca o limită

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}. \tag{5.1}$$

O astfel de expresie nu poate fi calculată cu un calculator numeric deoarece evaluarea unei limite ar presupune o procedură cu un număr infinit de paşi. Metodele de derivare numerică vor fi afectate în consecinţă şi de erori de trunchiere, nu numai de erori inerente şi de rotunjire. Mai mult, pe măsură ce x se apropie de x_0 şi $f(x)$ se apropie de $f(x_0)$,

ceea ce înseamnă că membrul drept care trebuie evaluat tinde către cazul nedeterminat 0/0. Este de aşteptat atunci ca evaluarea unei astfel de limite cu o formulă aproximativă de tipul $(f(x) - f(x_0))/(x - x_0)$ să ridice probleme numerice atunci când x este ales prea aproape de x_0. Pe de altă parte, evaluarea derivatei printr-o diferenţă divizată poate introduce erori inacceptabile atunci când x nu este suficient de apropiat de x_0.

Problema derivării numerice este strâns legată de problema interpolării sau aproximării funcţiilor. Orice fel de derivată poate fi reprezentată ca fiind derivata funcţiei aproximative obţinute prin interpolare sau aproximare. Dacă notăm cu $g(x)$ interpolarea sau aproximarea funcţiei $f(x)$ atunci derivata numerică $g'(x)$ va aproxima derivata exactă $f'(x)$:

$$g'(x) = \frac{\mathrm{d}g}{\mathrm{d}x} \approx f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x}. \tag{5.2}$$

Evident că precizia cu care se face interpolarea sau aproximarea va determina precizia de calcul a derivatei numerice.

5.2 Formule de derivare numerică

5.2.1 Formule de derivare cu eroare de ordinul 1

Cea mai simplă şi mai robustă metodă de interpolare este metoda interpolării liniare pe porţiuni. Funcţia g este derivabilă în interiorul intervalelor definite de reţeaua de interpolare (Fig.5.1). Derivata ei este o funcţie constantă pe porţiuni (Fig.5.2).

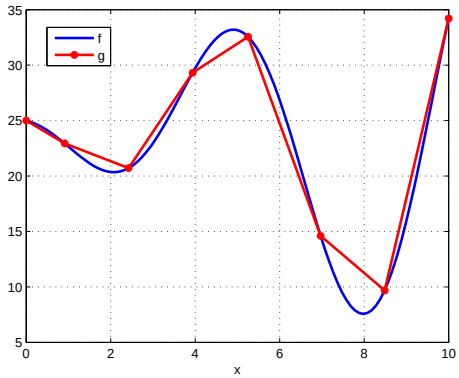


Figura 5.1: Funcţia reală f şi interpolarea ei pe porţiuni g.

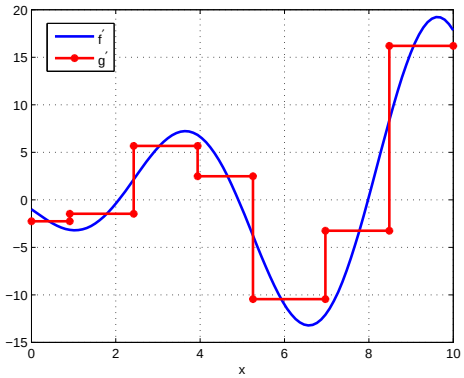


Figura 5.2: Aproximarea derivatei f' cu valoarea g'.

În nodurile de interpolare funcţia nu este derivabilă. Într-un astfel de nod am putea prelungi fie valoarea derivatei din dreapta, obţinând o formulă de *derivare progresivă de*

ordinul 1:

$$y'_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}, \tag{5.3}$$

fie valoarea derivatei din stânga, obţinând o formulă de *derivare regresivă de ordinul 1*:

$$y'_k = \frac{y_k - y_{k-1}}{x_k - x_{k-1}}. \tag{5.4}$$

Ordinul erorii de trunchiere specifice acestei aproximări se poate estima cu ajutorul dezvoltării în serie Taylor a funcţiei f în jurul punctului x_k:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(\xi)}{2!}(x - x_k)^2. \tag{5.5}$$

Evaluând această expresie în $x = x_{k+1}$ rezultă că

$$y_{k+1} = y_k + f'(x_k)(x_{k+1} - x_k) + \frac{f''(\xi)}{2!}(x_{k+1} - x_k)^2, \tag{5.6}$$

de unde

$$\frac{y_{k+1} - y_k}{x_{k+1} - x_k} - f'(x_k) = \frac{f''(\xi)}{2!}(x_{k+1} - x_k). \tag{5.7}$$

Presupunând că funcţia pe care o derivăm are derivata a doua mărginită de o constantă M_2, unde $|f''(x)| \leq M_2$, şi notând cu $h = x_{k+1} - x_k$, rezultă că eroarea de trunchiere a formulei de derivare numerică progresivă folosită este

$$|e_t| \leq \frac{M_2}{2} h, \tag{5.8}$$

şi se notează pe scurt $|e_t| = \mathrm{O}(h)$. Se spune că eroarea este de ordinul 1 (1 este puterea lui h, sau 1 este gradul polinomului de interpolare folosit).

Pentru derivata regresivă (5.4) concluzia este similară, rezultatul obţinându-se din evaluarea dezvoltării (5.5), care se mai scrie si sub forma

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \mathrm{O}((x - x_k)^2) \tag{5.9}$$

în punctul $x = x_{k-1}$. Notând $x_k - x_{k-1} = h$ rezultă

$$y_{k-1} = y_k - f'(x_k)h + \mathrm{O}(h^2), \tag{5.10}$$

de unde

$$\frac{y_k - y_{k-1}}{h} - f'(x_k) = \mathrm{O}(h). \tag{5.11}$$

5.2.2 Formule de derivare cu eroare de ordinul 2

Abordarea din paragraful anterior are dezavantajul că derivata funcţiei interpolatoare este discontinuă în nodurile reţelei de interpolare. Acest lucru poate fi corectat dacă se măreşte ordinul polinomului de interpolare folosit local pentru deducerea formulelor de derivare numerică.

Să considerăm de exemplu trei puncte consecutive ale reţelei de interpolare şi polinomul de interpolare de grad doi care trece prin acestea. Expresia acestui polinom (Lagrange) este

$$\begin{aligned} g(x) &= \frac{(x-x_k)(x-x_{k+1})}{(x_{k-1}-x_k)(x_{k-1}-x_{k+1})}y_{k-1} + \frac{(x-x_{k-1})(x-x_{k+1})}{(x_k-x_{k-1})(x_k-x_{k+1})}y_k + \\ &+ \frac{(x-x_{k-1})(x-x_k)}{(x_{k+1}-x_{k-1})(x_{k+1}-x_k)}y_{k+1}, \end{aligned} \tag{5.12}$$

iar derivata lui are forma

$$\begin{aligned} g'(x) &= \frac{(x-x_k)+(x-x_{k+1})}{(x_{k-1}-x_k)(x_{k-1}-x_{k+1})}y_{k-1} + \frac{(x-x_{k-1})+(x-x_{k+1})}{(x_k-x_{k-1})(x_k-x_{k+1})}y_k + \\ &+ \frac{(x-x_{k-1})+(x-x_k)}{(x_{k+1}-x_{k-1})(x_{k+1}-x_k)}y_{k+1}. \end{aligned} \tag{5.13}$$

Pentru a simplifica scrierea, vom nota $h_1 = x_k - x_{k-1}$ şi $h_2 = x_{k+1} - x_k$. Evaluând polinomul derivat (5.13) în x_k se obţine formula de *derivare centrată de ordinul 2*

$$y'_k = -\frac{h_2}{h_1(h_1+h_2)}y_{k-1} - \frac{h_1-h_2}{h_1h_2}y_k + \frac{h_1}{h_2(h_1+h_2)}y_{k+1}. \tag{5.14}$$

Evaluând polinomul derivat (5.13) în punctele x_{k-1} şi x_{k+1} se obţine formula de *derivare progresivă de ordinul 2*

$$y'_{k-1} = -\frac{2h_1+h_2}{h_1(h_1+h_2)}y_{k-1} + \frac{h_1+h_2}{h_1h_2}y_k - \frac{h_1}{h_2(h_1+h_2)}y_{k+1}, \tag{5.15}$$

şi, respectiv, de *derivare regresivă de ordinul 2*

$$y'_{k+1} = -\frac{h_2}{h_1(h_1+h_2)}y_{k-1} - \frac{h_1+h_2}{h_1h_2}y_k + \frac{h_1+2h_2}{h_2(h_1+h_2)}y_{k+1}. \tag{5.16}$$

În cazul unui pas echidistant $h_1 = h_2 = h$, formulele de derivare de ordinul 2 devin cele din relaţiile (5.17) pentru derivarea centrată, (5.18) pentru derivarea progresivă şi, respectiv, (5.19) pentru derivarea regresivă:

$$y'_k = \frac{y_{k+1}-y_{k-1}}{2h}, \tag{5.17}$$

$$y'_{k-1} = \frac{-3y_{k-1}+4y_k-y_{k+1}}{2h}, \tag{5.18}$$

$$y'_{k+1} = \frac{-y_{k-1}-4y_k+3y_{k+1}}{2h}. \tag{5.19}$$

Ordinul erorii de trunchiere specifice acestei aproximări se poate estima cu ajutorul dezvoltării în serie Taylor în care se reţin mai mulţi termeni. De exemplu, dezvoltând funcţia f în vecinătatea lui x_k şi păstrând inclusiv termenul de ordin 2, eroarea de trunchiere a seriei este de ordin 3:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(x_k)}{2!}(x - x_k)^2 + \mathrm{O}((x - x_k)^3). \tag{5.20}$$

Dacă evaluăm această dezvoltare în punctele x_{k-1} şi x_{k+1} unde presupunem pas echidistant $x_{k+1} - x_k = x_k - x_{k-1} = h$, rezultă:

$$f(x_{k+1}) = f(x_k) + f'(x_k)h + \frac{f''(x_k)}{2!}h^2 + \mathrm{O}(h^3), \tag{5.21}$$

$$f(x_{k-1}) = f(x_k) - f'(x_k)h + \frac{f''(x_k)}{2!}h^2 - \mathrm{O}(h^3). \tag{5.22}$$

Scăzând aceste două relaţii rezultă

$$f(x_{k+1}) - f(x_{k-1}) = 2f'(x_k)h + \mathrm{O}(h^3), \tag{5.23}$$

de unde rezultă că formula de derivare centrată are eroarea de trunchiere de ordinul 2:

$$\frac{y_{k+1} - y_{k-1}}{2h} - f'(x_k) = \mathrm{O}(h^2). \tag{5.24}$$

Formulele de ordinul 2 sunt mai precise decât formulele de ordinul 1, şi această proprietate poate fi ilustrată în cazul unui pas echidistant $x_{k+1} - x_k = x_k - x_{k-1} = h$ (Fig.5.3). Panta secantei care uneşte punctele $k-1$ şi $k+1$ este mai apropiată de derivata în punctul x_k decât panta secantei care uneşte punctul k cu punctul $k-1$ sau cu punctul $k+1$. În cazul în care f ar fi funcţie de gradul doi, derivata centrată este chiar valoarea adevărată (eroare de trunchiere este 0).

5.3 Algoritmi numerici pentru derivarea funcţiilor

Algoritmul pentru evaluarea derivatei unei funcţii într-un punct depinde de modul în care este cunoscută funcţia.

Dacă funcţia este dată tabelar (prin date), atunci se evaluează derivatele în punctele din tabel, iar dacă se doreşte evaluarea derivatei într-un punct oarecare, se poate face o interpolare liniară pe porţiuni a acestor valori. Pseudocodul acestei abordări este prezentat în paragraful 5.3.1.

Dacă funcţia este dată printr-un cod, atunci avem libertatea de a alege pasul de derivare. Teoretic, cu cât pasul de derivare este mai mic, cu atât eroarea este mai mică.

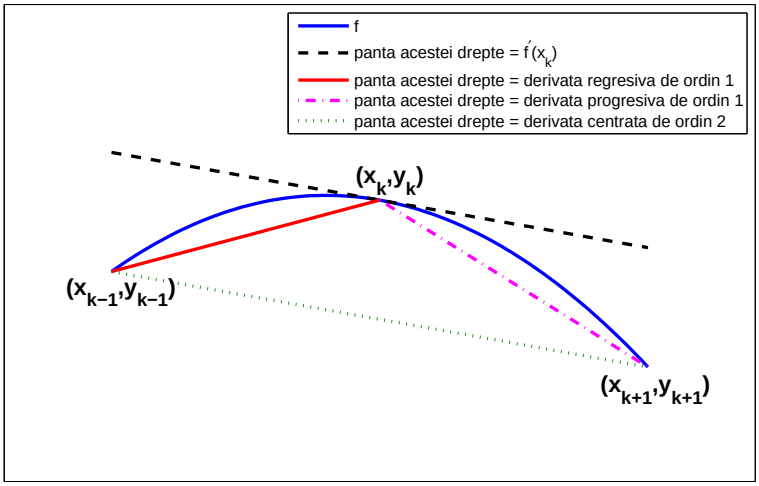


Figura 5.3: Semnificaţia geometrică a celor mai simple formule de derivare numerică.

Această afirmaţie este adevărată doar pentru eroarea de trunchiere, lucru demonstrat anterior pentru formule de derivare de ordinul 1 şi 2. În calculele numerice apar şi erori de rotunjire care sunt cu atât mai mari cu cât pasul de derivare este mai mic. În acest caz este necesară estimarea unui pas optim de derivare, aşa cum este prezentat în paragraful 5.3.2.

5.3.1 Cazul funcţiilor date tabelar

Deoarece, în general, un polinom de interpolare pe porţiuni nu este derivabil în nodurile de interpolare, rezultând o derivată care nu este continuă (ca în Fig. 5.2), se preferă ca aproximarea derivatei unei funcţii pe un domeniu să fie interpolarea (de exemplu liniară) pe porţiuni a derivatelor din nodurile de interpolare, calculate cu formule de derivare numerică. Pseudocodul următor ilustrează această abordare.

`procedură` pregătire_derivate(n, x, y, dy)
; calculează derivatele numerice în noduri
; declaraţii
`întreg` n ; numărul de puncte din tabel minus 1
`tablou real` $x[n], y[n]$; tabelul de valori, indici de la 0
`tablou real` $dy[n]$; valorile derivatelor
; alte declaraţii
...
; la capătul din stânga diferenţe progresive de ordinul 2

$h1 = x_1 - x_0$
$h2 = x_2 - x_1$
$dy_0 = -(2h1 + h2)/(h1(h1 + h2))dy_0 + (h1 + h2)/(h1h2)dy_1 - h1/(h2(h1 + h2))dy_2$
; la capătul din dreapta diferenţe regresive de ordinul 2
$h1 = x_{n-1} - x_{n-2}$
$h2 = x_n - x_{n-1}$
$dy_n = -h2/(h1(h1 + h2))dy_{n-2} - (h1 + h2)/(h1h2)dy_{n-1} + (h1 + 2h2)/(h2(h1 + h2))dy_n$
; în nodurile interioare diferenţe centrate de ordinul 2
pentru $k = 1, n - 1$
    $h1 = x_k - x_{k-1}$
    $h2 = x_{k+1} - x_k$
    $dy_k = -h2/(h1(h1 + h2))dy_{k-1} - (h1 - h2)/(h1h2)dy_k + h1/(h2(h1 + h2))dy_{k+1}$
•

Evaluarea derivatei funcţiei într-un punct oarecare *xcrt* se face apelând procedura de interpolare liniară pe porţiuni prezentată în paragraful 4.4.1 pentru un tabel de valori format din derivatele în noduri:

$rez =$ interpolare_lpp$(n, x, dy, xcrt)$

5.3.2 Cazul funcţiilor date prin cod

În cazul în care funcţia este dată prin cod, pasul de discretizare folosit pentru orice formulă de derivare numerică trebuie ales de utilizator. Eroarea de trunchiere este cu atât mai mică cu cât pasul este mai mic. Dar, pe lângă eroarea de trunchiere, există şi o eroare de rotunjire, care este cu atât mai mare cu cât pasul este mai mic, deoarece în formulele derivatelor se fac scăderi cu numere apropiate.

Teste numerice făcute pentru calculul numeric al derivatei funcţiei sinus în punctul $\pi/4$ arată că variaţia erorii în funcţie de pasul de discretizare este cea din Fig.5.4.

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

Pasul optim de derivare se poate estima. În cele ce urmează vom estima pasul optim de derivare în cazul formulei de derivare progresivă de ordinul 1.

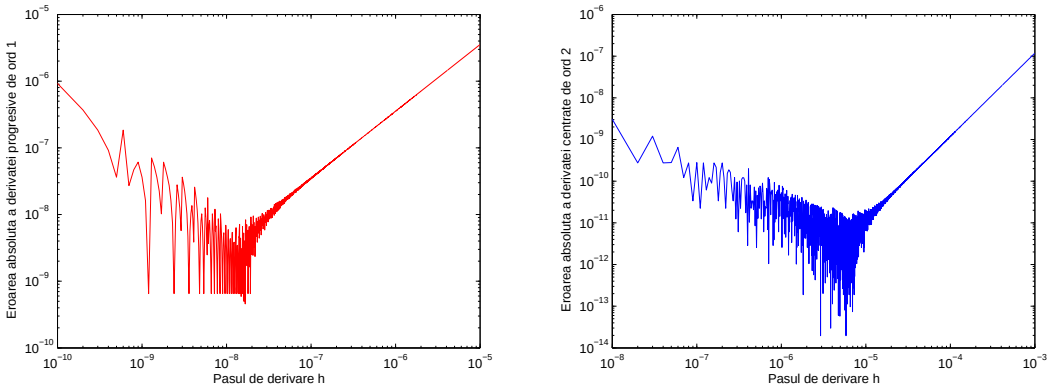


Figura 5.4: Eroarea totală în funcţie de pasul de derivare pentru formula de derivare progresivă de ordinul 1 (stânga) şi formula de derivare centrată de ordinul 2 (dreapta).

În paragraful 5.2.1 am arătat că eroarea de trunchiere în cazul folosirii derivatei numerice

$$y'_k = \frac{y_{k+1} - y_k}{x_{k+1} - x_k} = \frac{y_{k+1} - y_k}{h} \tag{5.25}$$

este mărginită:

$$|e_t| = |y'_k - f'(x_k)| \leq \frac{M_2}{2} h, \tag{5.26}$$

unde M_2 este o margine superioară a derivatei de ordinul 2 a funcţiei de derivat: $|f''(x)| \leq M_2$. Să notăm cu

$$a_t = \frac{M_2}{2} h \tag{5.27}$$

marginea erorii de trunchiere.

Să presupunem că valorile y_k sunt reprezentate cu precizia maximă admisibilă în calculator, deci ele sunt afectate doar de erori de rotunjire. În consecinţă $|e_{y_k}/y_k| < \text{eps}$, unde eps este zeroul maşinii. Presupunem de asemenea că pasul de discretizare h nu este afectat de erori $e_h = 0$. Aplicând formula erorii la împărţire din tabelul 1.1 rezultă că eroarea de rotunjire a relaţiei (5.25) este majorată de

$$|e_r| \leq \frac{|e_{y_{k+1}}| + |e_{y_k}|}{h} \leq \frac{\text{eps}\,(|y_{k+1}| + |y_k|)}{h} \tag{5.28}$$

Dacă notăm o margine superioară M_0 pentru valorile y_{k+1} şi y_k, atunci eroarea de rotunjire este majorată de

$$|e_r| \leq \frac{2M_0\,\text{eps}}{h}. \tag{5.29}$$

Să notăm cu

$$a_r = \frac{2M_0\,\text{eps}}{h} \tag{5.30}$$

marginea erorii de rotunhire.

Eroarea totală $e = e_t + e_r$ va fi majorată de o expresie ce depinde de pasul de derivare $|e| = |e_t + e_r| \leq |e_t| + |e_r| \leq a_t + a_r = m(h)$ unde

$$m(h) = \frac{M_2}{2}h + \frac{2M_0\,\text{eps}}{h}, \tag{5.31}$$

care îsi va atinge valoarea minimă pentru un pas de derivare optim

$$\min_h(m(h)) = m(h_{\text{optim}}). \tag{5.32}$$

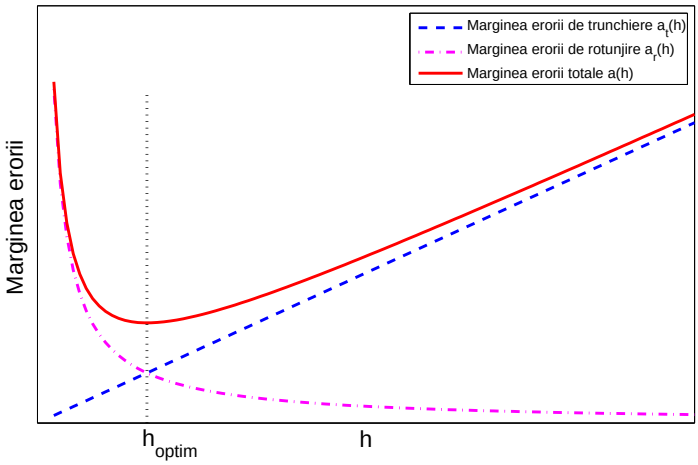


Figura 5.5: Pasul optim de derivare numerică este pasul de la care încep să predomine erorile de trunchiere.

Punând condiţia de minim pentru expresia (5.31), $m'(h) = 0$, rezultă valoarea optimă a pasului de derivare

$$h_{\text{optim}} = 2\sqrt{\frac{M_0\text{eps}}{M_2}}. \tag{5.33}$$

În concluzie, pasul optim de derivare depinde de eroarea de rotunjire, de marginea funcţiei şi de marginea derivatei a doua. El este pasul pentru care marginea erorii totale este minimă şi nu trebuie confundat cu pasul pentru care eroarea totală este minimă, pas care s-ar putea să fie mai mic decât pasul optim, dar care nu poate fi estimat apriori.

Algoritmul de mai jos calculează derivata progresivă de ordinul 1 a unei funcţii date prin cod, folosind un pas de derivare optim. Pentru calculul pasului optim, trebuie estimată derivata de ordinul 2. Pentru estimarea ei se foloseşte o formulă de derivare în trei puncte, formulă ce va fi justificată în paragraful 5.4:

$$M_2 \approx \frac{f(x) - 2f(x+h) + f(x+2h)}{h^2}. \tag{5.34}$$

Se consideră că valoarea pasului optim este rezonabilă dacă raportul dintre două aproximaţii consecutive ale acestuia este cuprins în intervalul [0.5,2]. S-ar putea ca algoritmul să nu fie convergent, de aceea s-a introdus un contor de iteraţii.

```
funcţie derivata_f(x, h0, maxit)
; calculează derivata numerică a funcţiei f
; folosind diferenţe progresive de ordinul 1
; şi pas optim de derivare
real x                                  ; punctul în care se va evalua derivata
real h0                                 ; pasul iniţial
întreg maxit                            ; numărul maxim de iteraţii pentru calculul pasului
; alte declaraţii
...
eps = zeroul_maşinii ()                 ; vezi pagina 34
f0 = f(x)                               ; valoarea funcţiei în punctul de derivare
h = h0
k = 0
repetă
      k = k + 1
      f1 = f(x + h)
      f2 = f(x + 2h)
      M2 = |f0 − 2f1 + f2|/h^2          ; estimarea marginii derivatei a doua
      dacă M2 < eps                     ; testează dacă derivata a doua este zero
            hoptim = h                  ; funcţia e liniară, eroarea de trunchiere e zero
      altfel
            M0 = max(|f0|, |f1|, |f2|)  ; estimarea marginii funcţiei
            hoptim = 2*sqrt(M0 * eps/M2)
      r = h/hoptim                      ; rata de modificare a pasului
      h = hoptim
până când (k > maxit) sau (r > 0.5 şi r < 2)
întoarce (f(x + hoptim) − f0)/hoptim
```

5.4 Derivate de ordin superior

Derivatele de ordin superior pot fi privite ca aplicaţii recursive ale derivatei de ordinul unu. O posibilitate de calcul a derivatelor de ordin superior este aceea de a folosi abordarea din paragraful 5.3.1. De exemplu, se pot calcula derivatele de ordinul doi în noduri apelând

pregătire_derivate$(n, x, dy, d2y)$

iar evaluarea derivatei de ordinul doi într-un punct oarecare se poate face apelând

$rez =$ interpolare_lpp$(n, x, d2y, xcrt)$

Experimentele numerice arată că, procedând astfel, acurateţea rezultatului se pierde pe măsură ce ordinul derivatei creşte.

O altă posibilitate este de a deduce formule de derivare pornind de la polinomul de interpolare iniţial. De exemplu, folosind expresia (5.13) rezultă că derivata de ordin 2 a polinomului de interpolare este

$$\begin{aligned} g''(x) &= \frac{2}{(x_{k-1}-x_k)(x_{k-1}-x_{k+1})}y_{k-1} + \frac{2}{(x_k-x_{k-1})(x_k-x_{k+1})}y_k + \quad (5.35)\\ &+ \frac{2}{(x_{k+1}-x_{k-1})(x_{k+1}-x_k)}y_{k+1}, \quad (5.36)\end{aligned}$$

formulă care devine în cazul particular al unui pas uniform $x_{k+1} - x_k = x_k - x_{k-1} = h$

$$g''(x) = \frac{y_{k-1} - 2y_k + y_{k+1}}{h^2}. \quad (5.37)$$

Pentru evaluarea erorii unei astfel de formule se poate folosi seria Taylor cu un număr suplimentar de termeni. Astfel, adunând dezvoltările evaluate în x_{k+1} şi x_{k-1}:

$$f(x_{k+1}) = f(x_k) + f'(x_k)h + \frac{f''(x_k)}{2!}h^2 + \frac{f'''(x_k)}{3!}h^3 + \mathrm{O}(h^4), \quad (5.38)$$

$$f(x_{k-1}) = f(x_k) - f'(x_k)h + \frac{f''(x_k)}{2!}h^2 - \frac{f'''(x_k)}{3!}h^3 - \mathrm{O}(h^4). \quad (5.39)$$

se obţine

$$f(x_{k+1}) + f(x_{k-1}) = 2f(x_k) + f''(x_k)h^2 + \mathrm{O}(h^4). \quad (5.40)$$

de unde

$$\frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} - f''(x_k) = \mathrm{O}(h^2). \quad (5.41)$$

Dacă s-ar dori estimarea unei derivate de ordinul 3, atunci derivând încă o dată (5.35) s-ar obţine zero, rezultat inutil. În concluzie, polinomul de interpolare iniţial trebuie să aibă un grad suficient de mare pentru ca derivata de ordin superior care interesează să fie nenulă. Acest lucru necesită considerarea a mai mult de 3 puncte consecutive din tabel. Interpolarea de grad mai mare ar putea conduce însă la apariţia fenomenului Runge care va afecta şi acurateţea derivatelor numerice. De aceea, calculul derivatelor de ordin superior trebuie evitat pe cât posibil în rezolvarea problemelor cu ajutorul calculatorului.

5.5 Derivate parţiale

Estimarea derivatelor parţiale ale unei funcţii se face folosind aceleaşi abordări prezentate pentru derivatele totale. De exemplu, să considerăm o funcţie reală ce depinde de două variabile, definită pe un domeniu dreptunghiular

$$f(x,y) : [a,b] \times [c,d] \to \mathbb{R}.$$

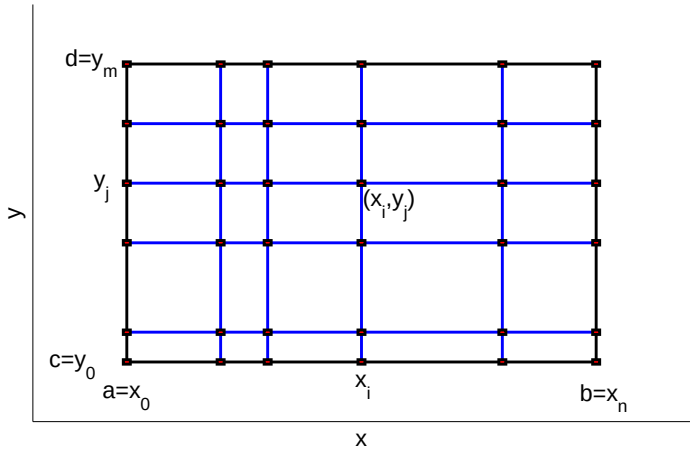


Figura 5.6: Discretizarea domeniului de definiţie al funcţiei $f(x,y) : [a,b] \times [c,d] \to \mathbb{R}$.

Domeniul de definiţie se discretizează acum cu ajutorul a două reţele de puncte, una pe axa Ox, iar cealaltă pe axa Oy:

$$a = x_0, x_1, \ldots, x_i, \ldots, x_n = b$$

$$c = y_0, y_1, \ldots, y_j, \ldots, y_m = d$$

Produsul cartezian al acestor două reţele unidimensionale dau reţeaua bidimensională de discretizare a domeniului de calcul (Fig.5.6), un nod al acestei reţele bidimensionale fiind caracterizat de coordonatele (x_i, y_j), unde $i = 0, n$, $j = 0, m$. Tabelul de valori care se dă este de această dată bidimensional:

$$f(x_i, y_j) = z_{i,j}, \quad i = 0, n \quad j = 0, m. \tag{5.42}$$

Evaluarea numerică a derivatelor parţiale se reduce la evaluarea numerică a unor derivate obişnuite, deoarece

$$\left.\frac{\partial f}{\partial x}\right|_{x=x_0, y=y_0} = \lim_{x \to x_0} \frac{f(x,y_0) - f(x_0,y_0)}{x - x_0} = \lim_{x \to x_0} \frac{f1(x) - f1(x_0)}{x - x_0} = \frac{\mathrm{d}f1}{\mathrm{d}x}, \tag{5.43}$$

unde $f1(x) = f(x, y_0)$ este o funcţie care depinde de o singură variabilă reală x. Similar,

$$\left.\frac{\partial f}{\partial y}\right|_{x=x_0,y=y_0} = \lim_{y\to y_0} \frac{f(x_0,y) - f(x_0,y_0)}{y - y_0} = \lim_{y\to y_0} \frac{f2(y) - f2(y_0)}{y - y_0} = \frac{\mathrm{d}f2}{\mathrm{d}y}, \tag{5.44}$$

unde $f2(y) = f(x_0, y)$ este o funcţie care depinde de o singură variabilă reală y.

De aceea, formulele de derivare progresivă de ordinul 1 pentru calculul derivatelor parţiale sunt:

$$\frac{\partial g}{\partial x}(x_i, y_j) = \frac{z_{i+1,j} - z_{i,j}}{x_{i+1} - x_i}, \quad \frac{\partial g}{\partial y}(x_i, y_j) = \frac{z_{i,j+1} - z_{i,j}}{y_{j+1} - y_j}. \tag{5.45}$$

Un alt exemplu îl constituie discretizarea operatorului Laplace

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{5.46}$$

În cazul în care paşii de discretizare pe cele două reţele sunt egali: $x_{i+1} - x_i = y_{j+1} - y_j = h$, pentru orice $i = 0, n-1$, $j = 0, m-1$ şi folosind formula de derivare în trei puncte de ordinul 2 dată de (5.37), rezultă că o aproximare numerică a Laplaceanului funcţiei f într-un punct (x_i, y_j) poate fi calculată ca

$$\Delta g = \frac{z_{i-1,j} - 2z_{i,j} + z_{i+1,j}}{h^2} + \frac{z_{i,j-1} - 2z_{i,j} + z_{i,j+1}}{h^2} = \frac{z_{i-1,j} + z_{i+1,j} + z_{i,j-1} + z_{i,j+1} - 4z_{i,j}}{h^2}, \tag{5.47}$$

punctele care intervin în această formulă fiind marcate în Fig.5.7.

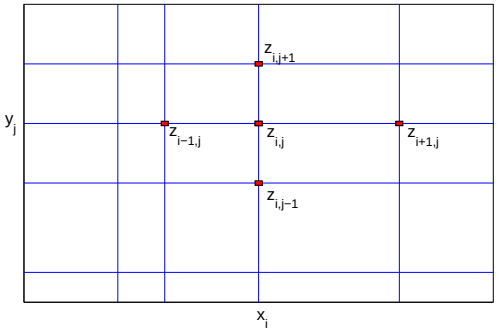


Figura 5.7: În discretizarea operatorului Laplace în punctul (x_i, y_j) apar numai valorile nodurilor marcate.

Dacă astfel de raţionamente se folosesc pentru rezolvarea numerică a problemelor de inginerie formulate cu ajutorul unor ecuaţii cu derivate obişnuite, simple (ODE) sau parţiale (PDE), atunci se spune că pentru rezolvarea problemei se aplică *metoda diferenţelor finite*.

5.6 Metoda diferenţelor finite

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

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

Se obişnuieşte să se spună că paşii 1 şi 2 reprezintă etapa de *preprocesare*, pasul 3 reprezintă *rezolvarea*, iar pasul 4 *postprocesarea*. În cele ce urmează vom exemplifica metoda diferenţelor finite pentru ecuaţii diferenţiale simple.

5.6.1 Rezolvarea unei ecuaţii diferenţiale ordinare

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

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

$$Ri(t) + u(t) = E. \tag{5.48}$$

Înlocuind în această relaţie dependenţa dintre tensiunea şi curentul prin condensator

$$i(t) = C\frac{\mathrm{d}u(t)}{\mathrm{d}t}, \tag{5.49}$$

rezultă ecuaţia diferenţială de ordinul unu, satisfăcută de u:

$$RC\frac{\mathrm{d}u(t)}{\mathrm{d}t} + u(t) = E. \tag{5.50}$$

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

$$u(0) = u_0 = 0. \tag{5.51}$$

Avantajul acestui exemplu este acela că are o soluţie analitică

$$u(t) = (u_0 - E)\exp(-t/\tau) + E. \tag{5.52}$$

unde

$$\tau = RC, \tag{5.53}$$

este *constanta de timp* a circuitului.

Vom rezolva acum ecuaţia diferenţială cu diferite scheme de derivare numerică. Ecuaţia (5.50) o rescriem ca

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} + \frac{1}{\tau}u(t) = \frac{1}{\tau}E. \tag{5.54}$$

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

$$t_{k+1} - t_k = h, \quad \text{pentru} \quad k = 1, \ldots, N-1. \tag{5.55}$$

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

$$u_k \approx u(t_k). \tag{5.56}$$

Varianta I - Utilizarea diferenţelor finite progresive de ordinul 1

Vom discretiza ecuaţia (5.54) prin scrierea ei în momentul de timp t_k şi folosind pentru derivată o formulă de diferenţe finite progresive de ordinul 1:

$$\frac{u_{k+1} - u_k}{h} + \frac{1}{\tau}u_k = \frac{1}{\tau}E, \tag{5.57}$$

de unde

$$u_{k+1} - u_k + \frac{h}{\tau}u_k = \frac{h}{\tau}E. \tag{5.58}$$

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

$$u_{k+1} = u_k\left(1 - \frac{h}{\tau}\right) + \frac{h}{\tau}E. \tag{5.59}$$

Implementarea acestei relaţii este descrisă de pseudocodul următor.

```
procedură mdf_ode_p1(u0,E,tau,h,N,u)
; rezolvă ecuaţia du(t)/dt + 1/τ u(t) = 1/τ E cu metoda diferenţelor finite
; d/dt se discretizează folosind diferenţe progresive de ordinul 1
real u0                    ; condiţia iniţială - dată
real E                     ; coeficient în ecuaţie - dată
real tau                   ; constantă de timp - dată
```

```
real h                          ; pas de discretizare al intervalului de timp - dat
întreg N                        ; număr de valori de timp - dat
tablou real u[N]                ; soluţia discretă - rezultat
u(1) = u0
pentru k = 1,N-1
      u(k+1) = u(k)*(1-h/tau) + h*E/tau
•
retur
```

Această metodă, cunoscută şi sub numele de *metoda Euler explicită*, este instabilă pentru $h > \tau$ (Fig.5.10).

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

Vom discretiza ecuaţia (5.54) prin scrierea ei în momentul de timp t_k şi folosind pentru derivată o formulă de diferenţe finite regresive de ordinul 1:

$$\frac{u_k - u_{k-1}}{h} + \frac{1}{\tau} u_k = \frac{1}{\tau} E, \tag{5.60}$$

de unde

$$u_k - u_{k-1} + \frac{h}{\tau} u_k = \frac{h}{\tau} E. \tag{5.61}$$

Se observă că mărimea u_k poate fi calculată explicit cu formula

$$u_k = \left(u_{k-1} + \frac{h}{\tau} E \right) / \left(1 + \frac{h}{\tau} \right). \tag{5.62}$$

Implementarea acestei relaţii este descrisă de pseudocodul următor.

```
procedură mdf_ode_r1(u0,E,tau,h,N,u)
; rezolvă ecuaţia du(t)/dt + 1/τ u(t) = 1/τ E cu metoda diferenţelor finite
; d/dt se discretizează folosind diferenţe regresive de ordinul 1
real u0                         ; condiţia iniţială - dată
real E                          ; coeficient în ecuaţie - dată
real tau                        ; constantă de timp - dată
real h                          ; pas de discretizare al intervalului de timp - dat
întreg N                        ; număr de valori de timp - dat
tablou real u[N]                ; soluţia discretă - rezultat
u(1) = u0
pentru k = 2,N
      u(k) = (h*E/tau + u(k-1))/(1 + h/tau)
•
retur
```

Metoda nu mai este instabilă pentru $h > \tau$ (Fig.5.10). Această metodă este cunoscută şi sub numele de *metoda Euler implicită*[1] pentru rezolvarea ecuaţiilor diferenţiale ordinare.

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

Vom discretiza ecuaţia (5.54) prin scrierea ei în momentul de timp t_k şi folosind pentru derivată o formulă de diferenţe finite centrate de ordinul 2 dată de relaţia (5.17):

$$\frac{u_{k+1} - u_{k-1}}{2h} + \frac{1}{\tau} u_k = \frac{1}{\tau} E, \tag{5.63}$$

de unde

$$-u_{k-1} + \frac{2h}{\tau} u_k + u_{k+1} = \frac{2h}{\tau} E. \tag{5.64}$$

Se observă că mărimea u_k nu mai poate fi calculată explicit. Relaţia (5.64) poate fi scrisă pentru $k = 2, \ldots, N-1$ şi reprezintă $N-2$ ecuaţii cu N necunoscute. Pentru a obţine un sistem bine formulat, trebuie să adăugăm încă două relaţii. Acestea sunt condiţiile la capete. La $t = 0$ vom impune condiţia iniţială:

$$u(1) = u_0, \tag{5.65}$$

iar pentru ultimul punct $k = N$ vom scrie ecuaţia discretizată dar în care vom folosi pentru derivată o formulă de diferenţe regresive de ordinul 2 dată de relaţia (5.19):

$$\frac{u_{N-2} - 4u_{N-1} + 3u_N}{2h} + \frac{1}{\tau} u_N = \frac{1}{\tau} E, \tag{5.66}$$

de unde

$$u_{N-2} - 4u_{N-1} + \left(3 + \frac{2h}{\tau}\right) u_N = \frac{2h}{\tau} E. \tag{5.67}$$

Implementarea acestei metode este descrisă de pseudocodul următor.

```
procedură mdf_ode_c2(u0,E,tau,h,N,u)
; rezolvă ecuaţia du(t)/dt + 1/τ u(t) = 1/τ E cu metoda diferenţelor finite
; d/dt se discretizează folosind diferenţe centrate de ordinul 2
real u0                          ; condiţia iniţială - dată
real E                           ; coeficient în ecuaţie - dată
real tau                         ; constantă de timp - dată
real h                           ; pas de discretizare al intervalului de timp - dat
întreg N                         ; număr de valori de timp - dat
tablou real u[N]                 ; soluţia discretă - rezultat
tablou real A[N,N]               ; matricea coeficienţilor sistemului asamblat - stocata rar
```

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

```
tablou real tl[N]                    ; vectorul termenilor liberi ai sistemului asamblat
A = 0                                ; pseudocod simplificat
tl = 0
A(1,1) = 1                           ; conditia initiala tl(1) = u0
pentru k = 2,N-1                     ; noduri interioare
      A(k,k-1) = -1
      A(k,k) = 2*h/tau
      A(k,k+1) = 1
      tl(k) = 2*h*E/tau
•
A(N,N-2) = 1                         ; conditia la tmax = N*h
A(N,N-1) = -4
A(N,N) = (3 + 2*h/tau)
tl(N) = 2*h*E/tau
u = "A^-1 tl"                        ; rezolva sistemul algebric liniar asamblat
retur
```

Metoda nu este instabilă şi este mai precisă decât implementările anterioare (Fig.5.8, 5.9, 5.10). Acest lucru era de aşteptat deoarece formulele de derivare numerică de ordinul 2 sunt mai precise decât cele de ordinul 1. Creşterea acurateţii se face însă pe seama creşterii complexităţii algoritmului.

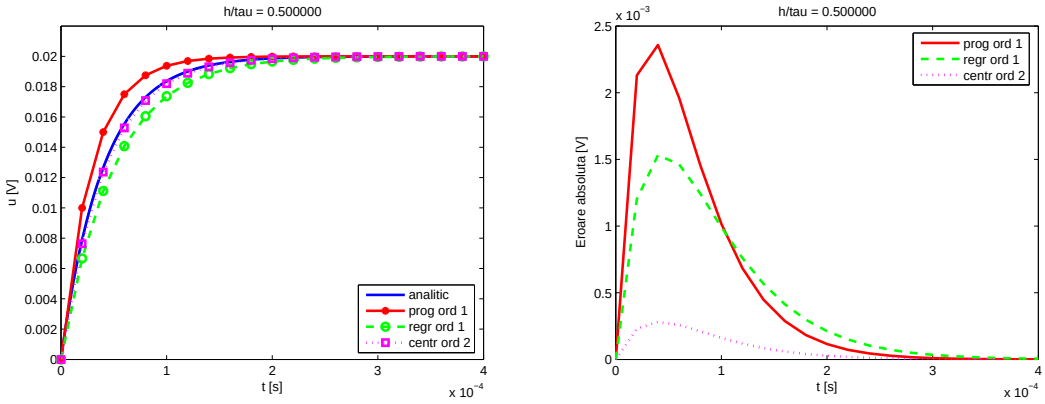


Figura 5.8: Cazul $h = \tau/2$.

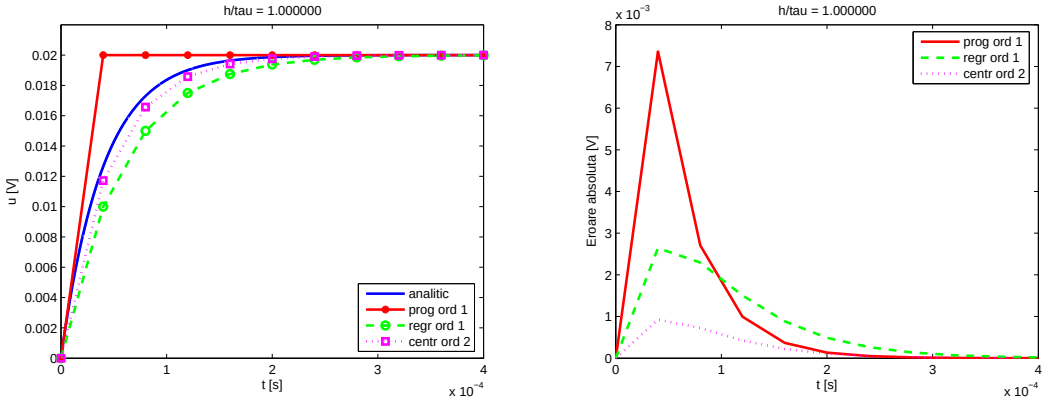


Figura 5.9: Cazul $h = \tau$.

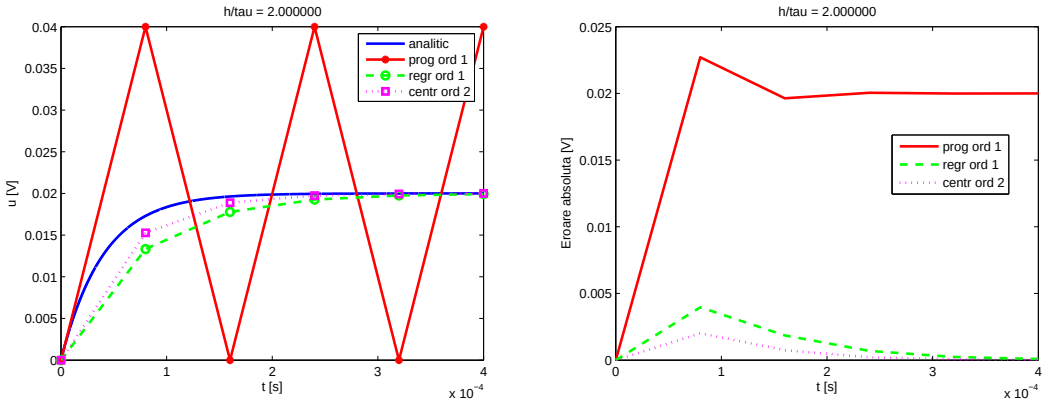


Figura 5.10: Cazul $h = 2\tau$.

5.6.2 Rezolvarea unei ecuaţii cu derivate parţiale de tip eliptic

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

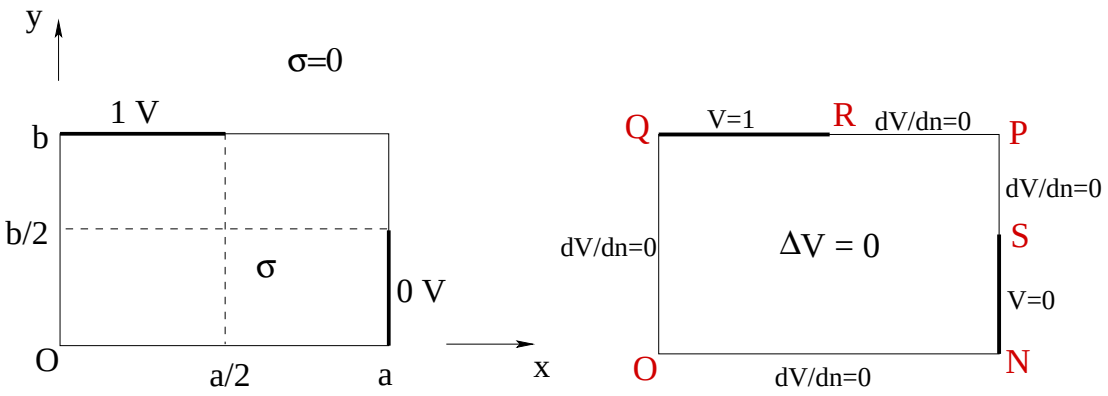


Figura 5.11: Problema 2D de regim electrocinetic: domeniul de calcul (stânga), condiţii de frontieră (dreapta).

Vom considera un conductor omogen, de conductivitate σ, situat într-un mediu perfect izolant. Conductorul are o dimensiune după axa Oz mult mai lungă decât celelalte două, astfel încât problema poate fi tratată ca bidimensională. Fig.5.11 reprezintă o secţiune perpendiculară pe direcţia dimensiunii foarte mari. Această secţiune este un dreptunghi, de dimensiuni a, b. Conductorul are două borne supraconductoare, una aflată la potenţial $V_0 = 1$V, iar cealaltă aflată la potenţial nul. Bornele sunt plasate ca în figură. Dorim să reprezentăm liniile echipotenţiale şi spectrul liniilor de curent. Ar fi interesant să calculăm şi mărimi globale cum ar fi puterea disipată în domeniu precum şi rezistenţa pe unitatea de lungime a acestui rezistor. La aceste aspecte vom reveni după ce vom discuta problema integrării numerice.

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

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

Ecuaţia de ordinul doi satisfăcută de potenţialul electrocinetic este:

$$-\text{div}\,(\sigma\,\text{grad}\,V) = 0 \tag{5.68}$$

unde $V : \mathcal{D} \to \mathbb{R}$ este potenţialul definit pe domeniul bidimensional $\mathcal{D} = \text{ONPQ}$. Ecuaţia (5.68) este o ecuaţie eliptică, de tip Laplace generalizată.

Deoarece domeniul este omogen (conductivitatea σ are aceeaşi valoare în orice punct

al domeniului), ecuaţia se simplifică la o ecuaţie de tip Laplace:

$$\Delta V = 0, \tag{5.69}$$

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

$$\Delta V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2}. \tag{5.70}$$

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

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0, \tag{5.71}$$

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

Pentru buna formulare a problemei, trebuie impuse pentru potenţial condiţiile de frontieră. Frontierele supraconductoare (SN şi QR) au potenţial constant, şi în consecinţă pe ele trebuie impuse condiţii Dirichlet în concordanţă cu valorile potenţialului. Pe frontierele de lângă izolant (NOQ şi RPS), trebuie impuse condiţii Neumann nule[2].

Condiţiile de frontieră impuse potenţialului sunt:

$$V = V_0 \quad \text{pe} \quad \text{QR} \quad \text{(Dirichlet)} \tag{5.72}$$

$$V = 0 \quad \text{pe} \quad \text{SN} \quad \text{(Dirichlet)} \tag{5.73}$$

$$\frac{\partial V}{\partial n} = 0 \quad \text{pe} \quad \text{NOQ} \cup \text{RPS} \quad \text{(Neumann)} \tag{5.74}$$

Şi pentru o astfel de problemă se pot găsi rezolvări analitice bazate, de exemplu, pe metoda separării variabilelor sau pe aproximarea liniilor de câmp.

Faţă de cazul unidimensional discutat în exemplul anterior, aici reţeaua de discretizare este una bidimensională, iar în loc de aproximarea unei derivate simple de ordinul 1, acum avem de aproximat derivate parţiale de ordinul 2.

Reţeaua de discretizare bidimensională poate fi privită ca fiind obţinută din produsul cartezian dintre o reţea de discretizare pe axa Ox şi o reţea de discretizare pe axa Oy:

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

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

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

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

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

$$V(x_i, y_j) \overset{not}{=} V_{i,j} \quad i = 1, n_x, \quad j = 1, n_y. \tag{5.75}$$

Forma discretizată (aproximativă) a ecuaţiei potenţialului este obţinută aproximând prin diferenţe finite ecuaţia (5.71). Forma specifică depinde de tipul nodului: interior sau pe frontiera Neumann. Nodurile de pe frontiera Dirichlet nu ridică probleme, ecuaţia asociată unui astfel de nod constând în simpla atribuire a valorii potenţialului nodului respectiv.

Ecuaţia asociată unui nod interior:

Ecuaţia aproximativă pentru un nod interior se deduce înlocuind derivatele parţiale după x şi după y cu formule de tipul (5.37).

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

$$\frac{\partial^2 V}{\partial x^2}(x_i, y_j) = \frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{h_x^2}, \tag{5.76}$$

$$\frac{\partial^2 V}{\partial y^2}(x_i, y_j) = \frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{h_y^2}. \tag{5.77}$$

Ecuaţia discretizată asociată unui nod interior devine

$$\frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{h_x^2} + \frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{h_y^2} = 0, \tag{5.78}$$

sau

$$2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) V_{i,j} - \frac{1}{h_x^2} V_{i+1,j} - \frac{1}{h_x^2} V_{i-1,j} - \frac{1}{h_y^2} V_{i,j+1} - \frac{1}{h_y^2} V_{i,j-1} = 0. \tag{5.79}$$

Relaţia (5.79) arată că potenţialul într-un nod interior este o combinaţie liniară a potenţialelor nodurilor învecinate. Dacă paşii de discretizare pe cele două axe ar fi egali ($h_x = h_y$), atunci potenţialul într-un nod interior este media aritmetică a potenţialelor celor patru noduri vecine.

În cazul unor reţele neuniforme, folosind notaţiile din Fig.5.12-dreapta, ecuaţia discretizată asociată unui nod interior este

$$V_O\left(\frac{1}{h_B h_D} + \frac{1}{h_A h_C}\right) - V_A \frac{1}{h_A(h_A + h_C)} - V_B \frac{1}{h_B(h_B + h_D)} - \\ -V_C \frac{1}{h_C(h_A + h_C)} - V_D \frac{1}{h_D(h_B + h_D)} = 0, \tag{5.80}$$

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

Ecuaţia asociată unui nod pe frontieră dreaptă:

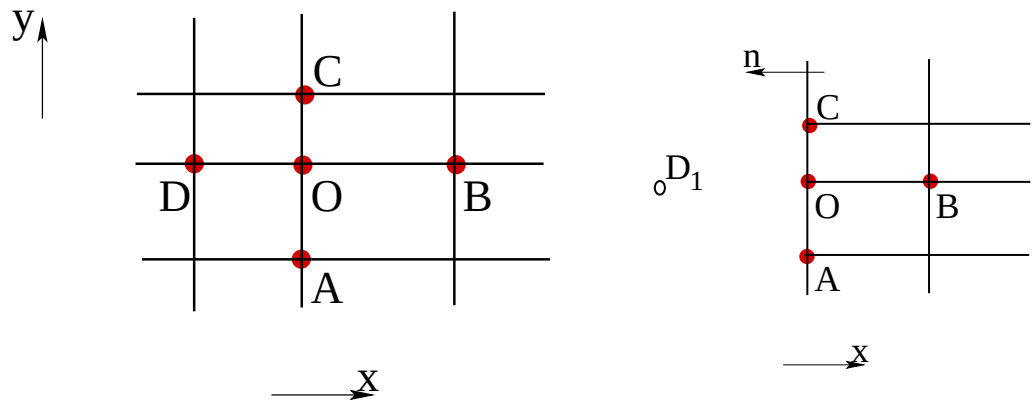


Figura 5.12: Nodurile marcate intervin în scrierea ecuaţiilor: stânga - cazul unui nod interior; dreapta - cazul unui nod pe frontieră Neumann dreaptă.

În cazul unui punct pe frontieră O (fig. 5.12- dreapta), să notăm cu $g = -\sigma \frac{\partial V}{\partial n}$ condiţia de frontieră Neumann şi cu g_O valoarea ei medie în punctul O:

$$g_O = \frac{g_{OA} h_A + g_{OC} h_C}{h_A + h_C} \tag{5.81}$$

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

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

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

$$\frac{\partial V}{\partial x} = \frac{g}{\sigma}. \tag{5.82}$$

Scriind relaţia aproximativă (5.17) pentru condiţia de frontieră Neumann, rezultă că:

$$V_{D_1} = V_B - 2h_B \frac{g_O}{\sigma}. \tag{5.83}$$

Pentru nodul O se scrie acum o ecuaţie de tipul (5.80), ca pentru un nod interior şi, înlocuind expresia (5.83), rezultă ecuaţia finală:

$$V_O \left(\frac{1}{h_B^2} + \frac{1}{h_A h_C} \right) - V_A \frac{1}{h_A(h_A + h_C)} - V_B \frac{1}{h_B^2} - V_C \frac{1}{h_C(h_A + h_C)} = -\frac{g_O}{\sigma h_B}. \tag{5.84}$$

În cazul unei condiţii Neumann nule, relaţia devine:

$$V_O \left(\frac{1}{h_B^2} + \frac{1}{h_A h_C} \right) - V_A \frac{1}{h_A(h_A + h_C)} - V_B \frac{1}{h_B^2} - V_C \frac{1}{h_C(h_A + h_C)} = 0. \tag{5.85}$$

Ecuaţia asociată unui nod pe frontieră - colţ exterior:

Nodurile situate în colţuri reprezintă o problemă pentru metoda diferenţelor finite. Aceasta deoarece pentru un colţ nu poate fi definită direcţia normalei[3]. Dacă notăm $g = -\sigma\frac{\partial V}{\partial n}$, şi notăm cu g_{OA} valoarea lui g pe segmentul OA (fără capătul O) şi cu g_{OB} valoarea lui g pe segmentul OB (fără capătul O), atunci vom presupune că în nodul O (fig. 5.13) este cunoscută derivata după o direcţie care rezultă din prelungirea în O a valorilor funcţiei g.

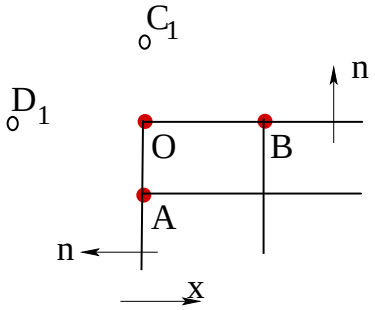


Figura 5.13: Nodurile marcate intervin în scrierea ecuaţiei unui nod pe frontieră colţ exterior.

Pentru deducerea ecuaţiei vom considera două noduri "fantomă" C_1 şi D_1, situate în exterior, la distanţele $h_D = \|OD_1\|$ şi $h_C = \|OC_1\|$ de punctul O. Pentru simplitate putem considera $h_D = h_B$ şi $h_C = h_A$.

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

$$V_{C_1} = V_A - 2h_A\frac{g_{OB}}{\sigma}. \tag{5.86}$$

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

$$V_{D_1} = V_B - 2h_B\frac{g_{OA}}{\sigma}. \tag{5.87}$$

Pentru nodul O se scrie acum o ecuaţie de tipul (5.80), ca pentru un nod interior şi, înlocuind expresiile (5.86) şi (5.87), rezultă ecuaţia finală:

$$V_O\left(\frac{1}{h_A^2}+\frac{1}{h_B^2}\right) - V_A\frac{1}{h_A^2} - V_B\frac{1}{h_B^2} = -\frac{g_{OB}}{\sigma h_A} - \frac{g_{OA}}{\sigma h_B}. \tag{5.88}$$

În cazul unor condiţii Neumann nule, relaţia devine:

$$V_O\left(\frac{1}{h_A^2}+\frac{1}{h_B^2}\right) - V_A\frac{1}{h_A^2} - V_B\frac{1}{h_B^2} = 0. \tag{5.89}$$

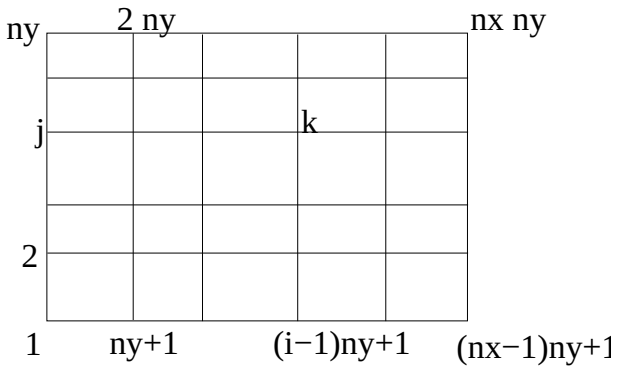


Figura 5.14: Numerotarea nodurilor.

În concluzie, discretizarea problemei conduce la un sistem de ecuaţii algebrice liniar, prin a cărui rezolvare vom obţine valorile potenţialelor în nodurile reţelei de discretizare. Pentru asamblarea acestui sistem cu ajutorul unui algoritm, este util ca nodurile să fie numerotate astfel încât necunoscutele să reprezinte un vector, nu o matrice cum sugerează notaţiile de până acum. Vom numerota nodurile de jos în sus şi de la stânga la dreapta, ca în Fig.5.14. Se poate verifica uşor că relaţia între numărul nodului k şi poziţiile proiecţiilor sale pe cele două axe i şi j este

$$k = (i-1)n_y + j, \quad i = 1, \ldots, n_x, \quad j = 1, \ldots, n_y. \tag{5.90}$$

unde n_x şi n_y reprezintă numărul de puncte de discretizare pe axele Ox, respectiv Oy.

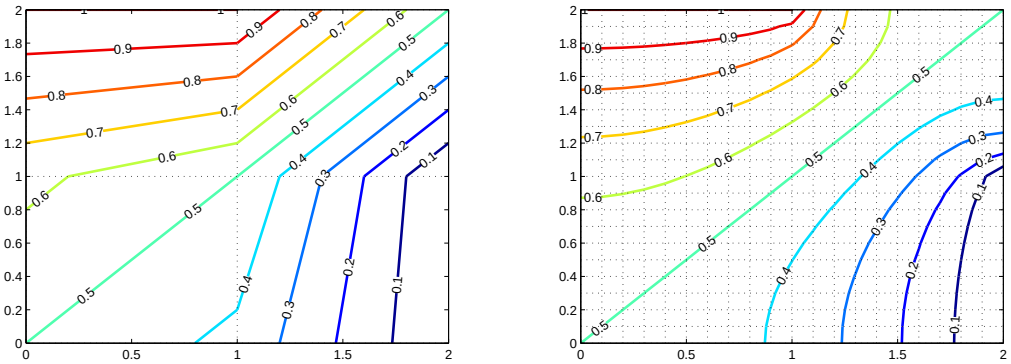


Figura 5.15: Linii echipotenţiale pentru o reţea cu $n_x = n_y = 3$ (stânga) şi o reţea cu $n_x = n_y = 21$ (dreapta).

[3]În realitate, colţul reprezintă o modelare brutală a realităţii. Trecerea de la o suprafaţă la alta se face prin racordări.

5.6.3 Rezolvarea unei ecuaţii cu derivate parţiale de tip hiperbolic

În acest paragraf vom rezolva cu metoda diferenţelor finite cel mai simplu exemplu de ecuaţie hiperbolică, şi anume *ecuaţia undei scalare*:

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2}, \tag{5.91}$$

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

Buna formulare a acestei probleme necesită impunerea

- *condiţiei iniţiale* $u(x,0) = h_0(x)$,
- *condiţiilor la capete* - relaţii în care intervin mărimile $u(0,t) = h_1(t)$ şi $u(a,t) = h_2(t)$.

Se poate demonstra că soluţia ecuaţiei undei scalare (5.91) este

$$u(x,t) = ud(x,t) + ui(x,t) + U_0, \tag{5.92}$$

unde $ud(x,t)$ se numeşte *undă directă* şi este o funcţie care se poate scrie sub forma

$$ud(x,t) = f(x - vt), \tag{5.93}$$

iar $ui(x,t)$ se numeşte *undă inversă* şi este o funcţie care se poate scrie sub forma

$$ud(x,t) = g(x + vt). \tag{5.94}$$

Funcţiile f şi g rezultă în mod univoc, din impunerea condiţiilor iniţiale şi condiţiilor la capete. Mărimea $f(x - vt)$ este o undă directă deoarece o anumită valoare a acestei funcţii, într-un anumit punct x şi într-un anumit moment de timp t, se va regăsi după un interval de timp Δt în punctul $x + \Delta x$, unde $\Delta x = v\Delta t$. Unda directă se propagă deci în sensul pozitiv al axei Ox. Un raţionament similar se poate face pentru unda inversă.

În conceperea algoritmului bazat pe metoda diferenţelor finite avem nevoie de o discretizare spaţială şi de una temporală. Pentru a simplifica prezentarea, vom presupune că ambele discretizări sunt uniforme şi vom nota cu Δx pasul discretizării spaţiale şi cu Δt pasul discretizării temporale. Vom nota cu N numărul de puncte de discretizare spaţiale şi cu M numărul de paşi de timp simulaţi. În consecintă, timpul maxim de simulare este $T = M\Delta t$.

Ca exemplu, vom considera condiţii iniţiale nule şi condiţii Dirichlet la ambele capete[4]:

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

Mărimea $u(x,t)$ este discretizată atât în spaţiu cât şi în timp. Prin rezolvarea cu metoda diferenţelor finite, vom obţine aproximaţii pentru valorile reale $u(x_k, t_j)$.

$$u(x_k, t_j) \approx u_k^{(j)}, \quad k = 1, \ldots, N, \quad j = 1, \ldots, M. \tag{5.95}$$

Alegând pentru derivatele ce intervin în (5.91) formule de derivare ce provin din polinomul de interpolare de ordin doi, rezultă următoarea relaţie discretizată:

$$\frac{u_k^{(j-1)} - 2u_k^{(j)} + u_k^{(j+1)}}{(\Delta t)^2} = v^2 \frac{u_{k-1}^{(j)} - 2u_k^{(j)} + u_{k+1}^{(j)}}{(\Delta x)^2}, \tag{5.96}$$

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

$$u_k^{(j+1)} = \left(\frac{v\Delta t}{\Delta x}\right)^2 \left(u_{k-1}^{(j)} - 2u_k^{(j)} + u_{k+1}^{(j)}\right) + 2u_k^{(j)} - u_k^{j-1}, \quad k = 2, \ldots, N-1, \quad j = 0, \ldots, M-1. \tag{5.97}$$

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

Rezultatul simulării cu metoda diferenţelor finite este prezentat în Fig.5.16. În momentul reflexiei apare o undă inversă deoarece condiţia Dirichlet la capătul din dreapta impune ca mărimea să fie zero. În consecinţă, unda inversă la capătul din dreapta este exact opusul undei directe, şi ea se propagă în sens contrar axei Ox.

Atunci când o problemă necesită atât o discretizare spaţială cât şi una temporală, pentru ca soluţia numerică să fie stabilă este necesar să fie îndeplinită *condiţia lui Courant*

$$|v|\Delta t \leq \Delta x. \tag{5.98}$$

[4]În problemele în care apare propagare, este util uneori ca frontiera domeniului spaţial să fie modelată ca o frontieră absorbantă, "invizibilă" din punct de vedere al propagării mărimilor în domeniul spaţial modelat. Condiţia de frontieră absorbantă pentru capătul din dreapta al problemei studiate este

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0.$$

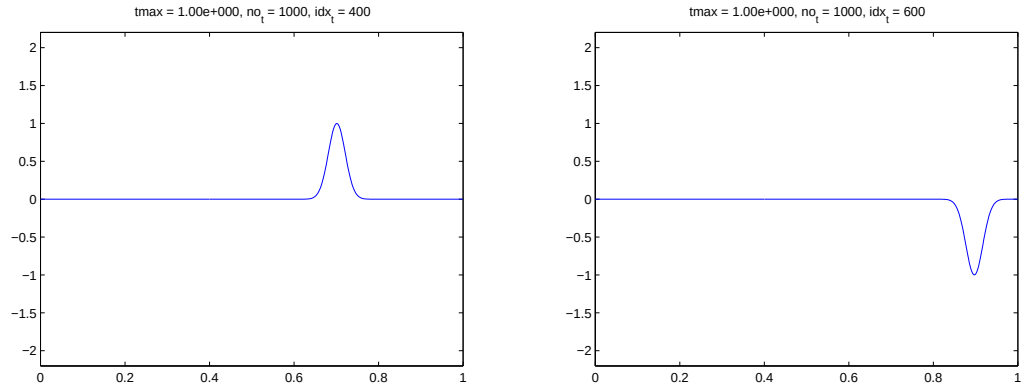


Figura 5.16: Propagarea unui impuls Gauss (stânga) şi rezultatul reflexiei după atingerea unei frontiere pe care s-a impus condiţie Dirichlet nulă (dreapta).

În concluzie, în rezolvarea ecuaţiilor diferenţiale cu metoda diferenţelor finite, trebuie avute în vedere aspectul stabilităţii algoritmului şi aspectul instabilităţii numerice datorate intrării în zona în care predomină erorile de rotunjire. Stabilitatea se analizează diferit dacă ecuaţia este cu derivate ordinare (ODE) sau cu derivate parţiale (PDE), dar această analiză depăşeşte scopul acestui capitol introductiv despre problema derivării numerice. Formulat pe scurt, trebuie ca paşii de discretizare trebuie să fie suficient de mici pentru ca algoritmul să fie stabil, dar nu prea mici ca să nu se intre în zona erorilor de rotunjire. De asemenea, un algoritm eficient ar trebui să poată să folosească paşi de discretizare neuniformi, adaptaţi tipului de variaţie a soluţiei.

Anexa A

Fişiere utile pentru implementarea în C

```
// fisier nrutil.h
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<malloc.h>
#include<time.h>

typedef float **MATRIX;
typedef float *VECTOR;
typedef int **IMATRIX;
typedef int *IVECTOR;

void nrerror (char error_text[]);
VECTOR vector (int nl, int nh);
IVECTOR ivector (int nl, int nh);
MATRIX matrix (int nrl, int nrh, int ncl, int nch);
IMATRIX imatrix (int nrl, int nrh, int ncl, int nch);
void free_vector (VECTOR v, int nl, int nh);
void free_ivector (IVECTOR v, int nl, int nh);
void free_matrix (MATRIX m, int nrl, int nrh, int ncl, int nch);
void free_imatrix (IMATRIX m, int nrl, int nrh, int ncl, int nch);

// fisier nrutil.c
#include "nrutil.h"

void
```

```
nrerror (char error_text[])
{
  fprintf (stderr, "Run-time error...\n");
  fprintf (stderr, "%s\n", error_text);
  fprintf (stderr, "...now exiting to system...\n");
  exit (1);
}

VECTOR
vector (int nl, int nh)
{
  VECTOR v;

  v = (float *) malloc ((unsigned) (nh - nl + 1) * sizeof (float));
  if (!v)
    nrerror ("allocation failure in vector()");
  return v - nl;
}

IVECTOR
ivector (int nl, int nh)
{
  IVECTOR v;

  v = (int *) malloc ((unsigned) (nh - nl + 1) * sizeof (int));
  if (!v)
    nrerror ("allocation failure in ivector()");
  return v - nl;
}

MATRIX
matrix (int nrl, int nrh, int ncl, int nch)
{
  int i;
  MATRIX m;

  m = (float **) malloc ((unsigned) (nrh - nrl + 1) * sizeof (float *));
  if (!m)
    nrerror ("allocation failure 1 in matrix()");
  m -= nrl;
```

```
  for (i = nrl; i <= nrh; i++)
    {
      m[i] = (float *) malloc ((unsigned) (nch - ncl + 1) * sizeof (float));
      if (!m[i])
nrerror ("allocation failure 2 in matrix()");
      m[i] -= ncl;
    }
  return m;
}

IMATRIX
imatrix (int nrl, int nrh, int ncl, int nch)
{
  int i;
  IMATRIX m;

  m = (int **) malloc ((unsigned) (nrh - nrl + 1) * sizeof (int *));
  if (!m)
    nrerror ("allocation failure 1 in imatrix()");
  m -= nrl;

  for (i = nrl; i <= nrh; i++)
    {
      m[i] = (int *) malloc ((unsigned) (nch - ncl + 1) * sizeof (int));
      if (!m[i])
nrerror ("allocation failure 2 in imatrix()");
      m[i] -= ncl;
    }
  return m;
}

void
free_vector (VECTOR v, int nl, int nh)
{
  free ((char *) (v + nl));
}

void
free_ivector (IVECTOR v, int nl, int nh)
```

```
{
  free ((char *) (v + nl));
}

void
free_matrix (MATRIX m, int nrl, int nrh, int ncl, int nch)
{
  int i;

  for (i = nrh; i >= nrl; i--)
    free ((char *) (m[i] + ncl));
  free ((char *) (m + nrl));
}

void
free_imatrix (IMATRIX m, int nrl, int nrh, int ncl, int nch)
{
  int i;

  for (i = nrh; i >= nrl; i--)
    free ((char *) (m[i] + ncl));
  free ((char *) (m + nrl));
}
```

Referinţe

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine şi H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994. http://www.netlib.org/linalg/html_templates/Templates.html.

[2] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006. Part of the SIAM Book Series on the Fundamentals of Algorithms, http://www.cise.ufl.edu/research/sparse/CSparse/.

[3] Gene Golub şi Charles van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996.

[4] D. Ioan, I. Munteanu, B. Ionescu, M. Popescu, R. Popa, M. Lăzărescu şi G. Ciuprina. *Metode numerice în ingineria electrică*. MatrixROM, Bucureşti, 1998.

[5] Valeriu Iorga şi Boris Jora. *Metode numerice*. Editura Albastră, 2004. Cluj-Napoca.

[6] B. Jora, C. Popeea şi S. Barbulea. *Metode de calcul numeric în automatică*. Ed. Enciclopedică, 1996. Bucureşti.

[7] Cleve Moler. *Numerical Computing with MATLAB*. SIAM, 2004. http://www.mathworks.com/moler/.

[8] A.M. Morega. *Modelarea numerică pentru probleme la limită în inginerie*. MatrixROM, Bucureşti, 1998.

[9] William Press, Saul Teukolsky, William Vetterling şi Brian Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, a doua ed., 1992. http://www.nrbook.com/a/bookcpdf.php.

[10] 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/.

[11] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing, 1996. http://www-users.cs.umn.edu/~saad/books.html.

[12] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Rap. tehn., School of Computer Science, Carnegie Mellon University, Pittsburgh, 1994. http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf.

[13] G.W. Stewart. Building an old-fashioned sparse solver. Rap. tehn., University of Maryland, Institute of Advanced Computer Studies, Department of Computer Science, 2003. http://http://www.cs.umd.edu/ stewart/.

[14] Anca Tomescu, I.B.L. Tomescu şi F.M.G. Tomescu. *Modelarea numerică a câmpului electromagnetic. Preliminarii. Câmpuri statice şi staţionare*. MatrixRom, 2003. Bucureşti.

[15] F.M.G. Tomescu. *Fundamentals of Electrical Engineering, Electric Circuits*. MatrixRom, 2011. Bucureşti.

[16] Lloyd Trefethen şi David Bau. *Numerical Linear Algebra*. SIAM - Society for Industrial and Applied Philadelphia, 1997.

[17] Henk A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, 2003.

Lucrarea de față este o primă parte a unei prezentări articulate a metodelor fundamentale de calcul numeric de interes pentru ingineri, cu aplicații directe în ingineria electrică. Acest prim volum este structurat urmând o alternare și întrepătrundere a unei abordări formale a subiectului cu exemplificări antrenante și ilustrative, într-o manieră clară, ușor accesibilă, fără a renunța însă la rigoarea adecvată materiei prezentate. Lucrarea include extinderi către subiecte particulare de interes curent, și se constituie într-o construcție completă și riguroasă, în care domeniul abordat este prezentat clar, cu ilustrări pertinente, acompaniate de o prezentare grafică excelentă. Ca abordare nouă, deosebită, de o reală valoare pedagogică, a unei materii de mare utilitate în activitatea inginerului modern, lucrarea va fi bine-venită pentru cei interesați în aplicarea metodelor de calcul numeric la rezolvarea problememlor științifice și inginerești.

București, 22 septembrie 2013

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

Aveți în mână o lucrare referitoare la Metodele numerice, scrisă de o inginer și adresată inginerilor de formație electrică. Asta nu o face mai puțin riguroasă, dar precis o face mai orientată spre aplicații, deci mai utilă pentru cei care vor sa depășească bariera înțelegerii conceptuale și vor să intre în domeniul dezvoltării de noi proceduri informatice sau măcar în cel al (re)folosirii lor profesionale. Autoarea continuă și consolidează abordarea tradițională dezvoltată in Laboratorul de Metode Numerice (LMN) din Facultatea de Inginerie Electrică. Aceasta se bazează pe observația că gândirea algoritmică este o deprindere clar diferită, atât de gândirea matematică – deductivă, cât și de cea științifică - inductivă. Înțelegerea algoritmilor și concepția acestora presupune un mod specific de gândire, iar daca problemele care trebuie rezolvate nu sunt doar de tipul așezării reginelor pe tabla de șah, ci sunt unele numerice, atunci soluția trebuie bazată pe fundamente matematice solide. Toate acestea ar rămâne pur speculative, dacă nu se au in vedere și aplicațiile inginerești. Aceasta este scopul lucrării, autoarea fiind nu numai profesor, ci și un talentat cercetător.

Lucrarea, este rezultatul unei îndelungi experiențe de lucru cu studenții și masteranzii, dar este marcată și de activitatea de cercetare și dezvoltare. Așa se explică faptul că în această carte sunt prezentate doar aspecte fundamentale, simple, dar profunde. Sunt evitate cele complicate sau foarte avansate, pentru ca țelul lucrării este de a transmite un mod de gândire si mai puțin o cantitate mare de informații. După ce acesta este însușit, lucrurile avansate pot fi găsite pe net sau în monografii. Dar fără acest prim pas, n-ai ști nici ce și nici cum să cauți. Este un model de curs universitar, pentru primul ciclu universitar, care te ajută să poți învăța ulterior singur. Recomand cu căldură, celor care îți doresc să devină adevărați profesioniști în ramuri ale ingineriei electrice, in cel mai larg sens să parcurgă aceasta lucrare, în toate detaliile ei.

București, 9 octombrie 2013

Prof.dr.ing Daniel IOAN,
Seful Laboratorului de Metode Numerice
Facultatea de Inginerie Electrică,
Universitatea Politehnica – București