

# **Analiză numerică**

O introducere bazată pe MATLAB

Radu Tiberiu Trîmbițaș

---

## Prefață

---

Lloyd N. Trefethen a propus următoarea definiție a Analizei numerice:

*Analiza numerică este studiul algoritmilor pentru rezolvarea problemelor matematicii continue.*

Cuvântul cheie este acela de *algoritmi*. Deși foarte multe lucrări nu evidențiază acest lucru, în centrul atenției Analizei numerice stă proiectarea și analiza algoritmilor de rezolvare a unei anumite clase de probleme.

Problemele sunt cele din *matematica continuă*. „Continuu” înseamnă aici faptul că variabilele ce intervin aici sunt reale sau complexe; opusul lui continuu este discret. Pe scurt, am putea spune că Analiza numerică este Algoritmică continuă, în contrast cu Algoritmica clasică, care este Algoritmică discretă.

Este clar că deoarece numerele reale și complexe nu pot fi reprezentate exact în calculator, ele trebuie să fie approximate printr-o reprezentare finită. Din acest moment intervin erorile de rotunjire și iar este clar că studiul lor este unul din obiectivele importante ale Analizei numerice. Au existat și mai există încă opinii care susțin că acesta este cel mai important obiectiv. Un argument în sprijinul acestei idei, înafară de omniprezența erorilor, este dat de metodele exacte de rezolvare a sistemelor de ecuații liniare, cum ar fi eliminarea gaussiană.

Dar, cele mai multe probleme ale matematicii continue nu pot fi rezolvate prin algoritmi aşa-zиşi finiti, chiar presupunând prin absurd că am lucra în aritmetică cu precizie exactă. Un prim exemplu care se poate da aici este problema rezolvării unei ecuații polinomiale. Acest lucru se evidențiază la problemele de valori și vectori proprii, dar apar în orice problemă ce presupune termeni neliniari sau derivate – determinarea zerourilor, cuadraturi, ecuații diferențiale și integrale, optimizare și.m.d.

Chiar dacă erorile de rotunjire ar dispare, Analiza numerică ar rămâne. Aproximarea numerelor, obiectivul aritmetică în virgulă flotantă, este un subiect dificil și obosit. Un obiectiv mai profund al Analizei numerice este aproximarea necunoscutelor, nu a cantităților cunoscute. Scopul este convergența rapidă a aproximățiilor și mândria specialiștilor din acest domeniu este aceea că, pentru multe probleme s-au inventat algoritmi care converg extrem de

rapid. Dezvoltarea pachetelor de calcul simbolic a micșorat importanța erorilor de rotunjire, fără a micșora importanța vitezei de convergență a algoritmilor.

Definiția de mai sus nu surprinde câteva aspecte importante : că acești algoritmi sunt implementați pe calculatoare, a căror arhitectură poate fi o parte importantă a problemei; că fiabilitatea și eficiența sunt obiective supreme; că anumiți specialiști în analiza numerică scriu programe și alții demonstrează teoreme<sup>1</sup>; și lucrul cel mai important, că toată munca este *aplicată*, aplicată cu succes la mii de aplicații, pe milioane de computere din toată lumea. „Problemele matematice continue” sunt problemele pe care știința și ingineria sunt construite; fără metode numerice, știința și ingineria, aşa cum sunt ele practicate astăzi ar ajunge repede în impas. Ele sunt de asemenea problemele care au preocupat cei mai mulți matematicieni de la Newton până azi. La fel ca și cei ce se ocupă de matematica pură, specialiștii în Analiza numerică sunt moștenitorii marii tradiții a lui Euler, Lagrange, Gauss și a altor mari matematicieni.

Din motivele amintite mai sus am încercat să realizez o lucrare în care să existe un echilibru între teorie, aspectele algoritmice și implementările practice. S-a utilizat MATLAB 2019. Mulțumesc echipei de la Mathworks Inc. pentru amabilitatea de a-mi pune la dispoziție documentația și kit-ul de instalare.

Pentru a descărca sursele din această carte și soluțiile problemelor trimitem cititorul la pagina de web a autorului: <http://www.math.ubbcluj.ro/~tradu>.

Radu Tiberiu Trîmbițăș  
Cluj-Napoca, aprilie 2020

---

<sup>1</sup>există mulți specialiști foarte buni care le fac pe amândouă

---

## Cuprins

---

<b>1. Introducere în MATLAB</b>	<b>1</b>
1.1. Lansarea MATLAB și sistemul de help . . . . .	1
1.2. Modul calculator . . . . .	2
1.3. Matrice . . . . .	5
1.3.1. Generarea matricelor . . . . .	5
1.3.2. Indexarea și notația „;” . . . . .	10
1.3.3. Operații în sens matricial și în sens tablou . . . . .	12
1.3.4. Analiza datelor . . . . .	15
1.3.5. Operatori relaționali și logici . . . . .	17
1.4. Programarea în MATLAB . . . . .	21
1.4.1. Fluxul de control . . . . .	21
1.4.2. Fișiere M . . . . .	24
1.4.3. Argumente funcție . . . . .	29
1.4.4. Număr variabil de argumente . . . . .	32
1.4.5. Variabile globale . . . . .	33
1.4.6. Recursivitate . . . . .	34
1.4.7. Alte tipuri numerice . . . . .	37
1.4.8. Controlul erorilor . . . . .	39
1.5. Toolbox-urile Symbolic . . . . .	40
Probleme . . . . .	46
<b>2. Grafică în MATLAB</b>	<b>49</b>
2.1. Grafice bidimensionale . . . . .	49
2.1.1. Grafice de bază . . . . .	49
2.1.2. Axe și adnotarea . . . . .	54
2.1.3. Mai multe grafice pe aceeași figură . . . . .	58
2.2. Grafice tridimensionale . . . . .	59

2.3. Salvarea și imprimarea graficelor . . . . .	66
2.4. Facilități grafice noi în MATLAB 7 . . . . .	68
Probleme . . . . .	69
<b>3. Elemente de Teoria erorilor și aritmetică în virgulă flotantă</b>	<b>71</b>
3.1. Probleme numerice . . . . .	72
3.2. Măsuri ale erorii . . . . .	73
3.3. Eroarea propagată . . . . .	74
3.4. Reprezentarea în virgulă flotantă . . . . .	75
3.4.1. Parametrii reprezentării . . . . .	75
3.4.2. Anularea . . . . .	77
3.5. Standardizarea reprezentării în virgulă flotantă . . . . .	79
3.5.1. Parametrii standardului . . . . .	79
3.5.2. Cantități speciale . . . . .	80
3.6. Numere în virgulă flotantă în MATLAB . . . . .	80
3.7. Condiționarea unei probleme . . . . .	84
3.8. Condiționarea unui algoritm . . . . .	87
3.9. Eroarea globală . . . . .	87
3.10. Probleme prost condiționate și probleme incorrect puse . . . . .	88
3.11. Stabilitatea . . . . .	90
3.11.1. Notații asimptotice . . . . .	90
3.11.2. Precizie și stabilitate . . . . .	91
3.11.3. Analiza regresivă a erorilor . . . . .	93
Probleme . . . . .	94
<b>4. Rezolvarea numerică a sistemelor de ecuații algebrice liniare</b>	<b>97</b>
4.1. Elemente de Analiză matricială . . . . .	97
4.2. Condiționarea unui sistem liniar . . . . .	104
4.3. Metode exacte . . . . .	109
4.3.1. Metoda eliminării a lui Gauss . . . . .	109
4.4. Metode bazate pe factorizare . . . . .	112
4.4.1. Descompunerea LU . . . . .	112
4.4.2. Descompunere LUP . . . . .	116
4.4.3. Factorizarea Cholesky . . . . .	117
4.4.4. Descompunerea QR . . . . .	121
4.5. Rafinarea iterativă . . . . .	127
4.6. Algoritmul lui Strassen pentru înmulțirea matricelor . . . . .	127
4.7. Rezolvarea sistemelor de ecuații liniare în MATLAB . . . . .	131
4.7.1. Sisteme pătratice . . . . .	131
4.7.2. Sisteme supradeterminate . . . . .	132
4.7.3. Sisteme subdeterminate . . . . .	132
4.7.4. Factorizarea LU și Cholesky . . . . .	134
4.7.5. Factorizarea QR . . . . .	135
4.8. Rezolvarea iterativă a sistemelor algebrice liniare . . . . .	138
Probleme . . . . .	149

<b>5. Aproximarea funcțiilor</b>	<b>153</b>
5.1. Aproximație prin metoda celor mai mici pătrate . . . . .	156
5.1.1. Produse scalare . . . . .	156
5.1.2. Ecuațiile normale . . . . .	157
5.1.3. Eroarea în metoda celor mai mici pătrate. Convergență . . . . .	160
5.1.4. Exemple de sisteme ortogonale . . . . .	163
5.1.5. Exemple de polinoame ortogonale . . . . .	165
5.2. Polinoame și potrivirea datelor în MATLAB . . . . .	179
5.3. Interpolare polinomială . . . . .	186
5.3.1. Spațiul $H^n[a, b]$ . . . . .	186
5.3.2. Interpolare Lagrange . . . . .	189
5.3.3. Interpolare Hermite . . . . .	192
5.3.4. Expresia erorii de interpolare . . . . .	196
5.3.5. Convergența interpolării Lagrange . . . . .	200
5.4. Calculul eficient al polinoamelor de interpolare . . . . .	206
5.4.1. Metode de tip Aitken . . . . .	206
5.4.2. Metoda diferențelor divizate . . . . .	208
5.4.3. Diferențe finite: formula lui Newton progresivă și regresivă . . . . .	212
5.4.4. Metoda baricentrică . . . . .	215
5.4.5. Diferențe divizate cu noduri multiple . . . . .	220
5.5. Interpolare spline . . . . .	222
5.5.1. Spline liniare . . . . .	223
5.5.2. Interpolarea cu spline cubice . . . . .	224
5.5.3. Proprietăți de minimalitate ale funcțiilor spline cubice . . . . .	229
5.6. Interpolare în MATLAB . . . . .	231
Probleme . . . . .	234
<b>6. Aproximare uniformă</b>	<b>239</b>
6.1. Polinoamele lui Bernstein . . . . .	240
6.2. B-spline . . . . .	244
6.2.1. Noțiuni și rezultate de bază . . . . .	244
6.2.2. Algoritmul de evaluare a unui B-spline . . . . .	246
6.2.3. Aplicații în grafica pe calculator . . . . .	247
6.2.4. Exemple . . . . .	249
6.3. Funcții spline cu variație diminuată . . . . .	254
6.4. Operatori liniari și pozitivi . . . . .	257
6.5. Cea mai bună aproximare uniformă . . . . .	262
Probleme . . . . .	263
<b>7. Aproximarea funcționalelor liniare</b>	<b>265</b>
7.1. Introducere . . . . .	265
7.2. Derivare numerică . . . . .	268
7.3. Integrare numerică . . . . .	269
7.3.1. Formula trapezului și formula lui Simpson . . . . .	271
7.3.2. Formulele Newton-Cotes cu ponderi și formule de tip Gauss . . . . .	275

7.3.3. Proprietăți ale cuadraturilor gaussiene . . . . .	278
7.4. Cuadraturi adaptive . . . . .	284
7.5. Cuadraturi iterate. Metoda lui Romberg . . . . .	285
7.6. Cuadraturi adaptive II . . . . .	289
7.7. Integrare numerică în MATLAB . . . . .	292
Probleme . . . . .	294
<b>8. Rezolvarea numerică a ecuațiilor neliniare</b>	<b>297</b>
8.1. Ecuații neliniare . . . . .	297
8.2. Iterații, convergență și eficiență . . . . .	298
8.3. Metoda șirurilor Sturm . . . . .	299
8.4. Metoda falsei poziții . . . . .	301
8.5. Metoda secantei . . . . .	303
8.6. Metoda lui Newton . . . . .	305
8.7. Metoda aproximățiilor succesive . . . . .	307
8.8. Metoda lui Newton pentru rădăcini multiple . . . . .	310
8.9. Ecuații algebrice . . . . .	311
8.10. Metoda lui Newton în $\mathbb{R}^n$ . . . . .	311
8.11. Metode quasi-Newton . . . . .	314
8.11.1. Interpolare liniară . . . . .	315
8.11.2. Metode de modificare . . . . .	315
8.12. Ecuații neliniare în MATLAB . . . . .	318
Probleme . . . . .	321
<b>9. Valori și vectori proprii</b>	<b>323</b>
9.1. Valori proprii și rădăcini ale polinoamelor . . . . .	323
9.2. Terminologie și descompunere Schur . . . . .	324
9.3. Iterația vectorială . . . . .	328
9.4. Metoda QR – teoria . . . . .	330
9.5. Metoda QR – practica . . . . .	333
9.5.1. Metoda QR clasică . . . . .	333
9.5.2. Deplasare spectrală . . . . .	338
9.5.3. Metoda QR cu pas dublu . . . . .	342
9.6. Valori și vectori proprii în MATLAB . . . . .	345
Probleme . . . . .	349
<b>10. Rezolvarea numerică a ecuațiilor diferențiale ordinare</b>	<b>351</b>
10.1. Ecuații diferențiale . . . . .	351
10.2. Metode numerice . . . . .	352
10.3. Descrierea locală a metodelor cu un pas . . . . .	353
10.4. Exemple de metode cu un pas . . . . .	354
10.4.1. Metoda lui Euler . . . . .	354
10.4.2. Metoda dezvoltării Taylor . . . . .	355
10.4.3. Metode de tip Euler îmbunătățite . . . . .	356
10.5. Metode Runge-Kutta . . . . .	357

10.6. Descrierea globală a metodelor cu un pas . . . . .	363
10.6.1. Stabilitatea . . . . .	364
10.6.2. Convergența . . . . .	366
10.6.3. Asimptotica erorii globale . . . . .	367
10.7. Monitorizarea erorilor și controlul pasului . . . . .	369
10.7.1. Estimarea erorii globale . . . . .	369
10.7.2. Estimarea erorii de trunchiere . . . . .	371
10.7.3. Controlul pasului . . . . .	373
10.8. Ecuații diferențiale ordinare în MATLAB . . . . .	383
10.8.1. Rezolvitori . . . . .	383
10.8.2. Exemple non-stiff . . . . .	384
10.8.3. Opțiuni . . . . .	386
10.8.4. Ecuații stiff . . . . .	388
10.8.5. Tratarea evenimentelor . . . . .	395
10.8.6. Ecuații implice . . . . .	402
Probleme . . . . .	403
<b>11. Aproximări în mai multe variabile</b>	<b>411</b>
11.1. Aproximarea funcțiilor de mai multe variabile pe un domeniu rectangular . . . . .	411
11.2. Integrarea numerică a funcțiilor de mai multe variabile . . . . .	419
11.2.1. Considerații de implementare . . . . .	423
11.3. Aproximări în mai multe variabile în MATLAB . . . . .	424
11.3.1. Interpolarea funcțiilor de mai multe variabile în MATLAB . . . . .	424
11.3.2. Calculul integralelor duble în MATLAB . . . . .	427
Probleme . . . . .	429
<b>Bibliografie</b>	<b>431</b>
<b>Indice</b>	<b>435</b>



---

## Lista surselor MATLAB

---

1.1	Funcția stat . . . . .	26
1.2	Funcția sqrtn . . . . .	28
1.3	Funcția fd_deriv . . . . .	30
1.4	Funcțiacompanb . . . . .	32
1.5	Funcția momente . . . . .	33
1.6	Funcția koch . . . . .	35
1.7	Fulgul lui Koch . . . . .	36
2.1	Reprezentarea grafică a unei funcții implice . . . . .	62
3.1	Calculul lui eps - varianta 1 . . . . .	81
3.2	Calculul lui eps - varianta 2 . . . . .	81
4.1	Rezolvă sistemul $Ax = b$ prin metoda eliminării a lui Gauss cu pivot scalat pe coloană	113
4.2	Descompunere LUP . . . . .	118
4.3	Substituție directă . . . . .	118
4.4	Substituție inversă . . . . .	119
4.5	Descompunere Cholesky . . . . .	120
4.6	Factorizare QR utilizând reflexii Householder . . . . .	124
4.7	Rezolvarea sistemului $Ax = b$ prin metoda QR . . . . .	124
4.8	Algoritmul lui Strassen pentru înmulțirea a două matrice . . . . .	128
4.9	Jacobi method for linear systems . . . . .	143
4.10	Metoda SOR . . . . .	145
4.11	Determinarea parametrului optim de relaxare . . . . .	146
5.1	Calculul polinoamelor Legendre cu relația de recurență . . . . .	168
5.2	Calculul coeficienților Legendre . . . . .	168
5.3	Aproximare în sensul celor mai mici pătrate cu polinoame Legendre . . . . .	169
5.4	Aproximare în sensul celor mai mici pătrate cu polinoame Cebîșev de speță I	174
5.5	Calculul polinoamelor Cebîșev de speță I cu relația de recurență . . . . .	174
5.6	Aproximare în sensul celor mai mici pătrate cu polinoame Cebîșev de speță I – continuare: calculul c	
5.7	Aproximantă Cebîșev discretă . . . . .	175

5.8 Coeficienții aproximantei Cebîșev discrete . . . . .	175
5.9 Test aproximante mcmmp . . . . .	178
5.10 Exemplu de aproximare în sensul celor mai mici pătrate . . . . .	186
5.11 Interpolare Lagrange . . . . .	190
5.12 Calculul polinoamelor fundamentale Lagrange folosind facilități MATLAB. Rezultatul este returnat . . . . .	190
5.13 Contraexemplul lui Runge . . . . .	204
5.14 Generarea tablelei diferențelor divizate . . . . .	211
5.15 Calculul formei Newton a polinomului de interpolare Lagrange . . . . .	211
5.16 Interpolare Lagrange baricentrică . . . . .	218
5.17 Ponderi baricentrice . . . . .	219
5.18 Interpolare Lagrange baricentrică cu noduri Cebîșev de speță I . . . . .	219
5.19 Interpolare Lagrange baricentrică cu noduri Cebîșev de speță II . . . . .	220
5.20 Generarea tablelei de diferențe divizate cu noduri duble . . . . .	223
6.1 Implementarea algoritmului Cox-deBoor pentru calculul B-splinelor . . . . .	250
6.2 Implementarea algoritmului de Casteljau pentru curbe Bézier . . . . .	251
6.3 Calculul funcției spline cu variație diminuată . . . . .	257
7.1 Aproximarea unei integrale prin formula repetată a trapezului . . . . .	272
7.2 Aproximarea unei integrale prin formula repetată a lui Simpson . . . . .	273
7.3 Calculul nodurilor și coeficienților unei formule de cuadratură gaussiene . . . . .	281
7.4 Aproximarea unei integrale cu o formulă de tip Gauss . . . . .	281
7.5 Generare formulă Gauss-Legendre . . . . .	282
7.6 Generare formulă Gauss-Cebîșev de speță I . . . . .	282
7.7 Generare formulă Gauss-Cebîșev de speță a II-a . . . . .	282
7.8 Generare formulă Gauss-Hermite . . . . .	282
7.9 Generare formulă Gauss-Laguerre . . . . .	283
7.10 Generare formulă Gauss-Jacobi . . . . .	283
7.11 Cuadratură adaptivă . . . . .	285
7.12 Metoda lui Romberg . . . . .	290
7.13 Cuadratura adaptivă, variантă . . . . .	291
8.1 Secant method for nonlinear equations in $\mathbb{R}$ . . . . .	306
8.2 Metoda lui Newton pentru ecuații neliniare în $\mathbb{R}$ . . . . .	308
8.3 Metoda lui Newton în $\mathbb{R}$ și $\mathbb{R}^n$ . . . . .	313
8.4 Metoda lui Broyden pentru sisteme neliniare . . . . .	317
9.1 Tranformarea RQ a unei matrice Hessenberg . . . . .	335
9.2 Trecerea la forma Hessenberg . . . . .	336
9.3 Metoda QR simplă . . . . .	337
9.4 Metoda QR cu partitōnare și tratarea cazurilor $2 \times 2$ . . . . .	339
9.5 Eigenvalues of a $2 \times 2$ matrix . . . . .	340
9.6 Iterația QR . . . . .	340
9.7 Metoda QR cu deplasare spectrală și partitōnare . . . . .	341
9.8 Metoda QR cu deplasare spectrală, partitōnare și tratarea valorilor proprii complexe . . . . .	342
9.9 Iterație QR și partitōnare . . . . .	343
9.10 Metoda QR cu dublu pas, partitōnare și tratarea matricelor $2 \times 2$ . . . . .	344
9.11 Iterație QR cu dublu pas și transformare Hessenberg . . . . .	345
10.1 Metoda Runge-Kutta de ordinul 4 . . . . .	360

10.2 Implementarea unei metode Runge-Kutta cu pas constant cu tabelă Butcher . . . . .	362
10.3 Inițializare tabelă Butcher RK4 . . . . .	363
10.4 Sistemul lui Rössler . . . . .	388
10.5 Problemă stiff cu informații despre jacobian . . . . .	396
10.6 Problema celor două coruri . . . . .	399
10.7 Funcțiile <code>f ox2</code> și <code>events</code> pentru problema de urmărire . . . . .	402
11.1 Interpolant bidimensional produs tensorial de tip Lagrange . . . . .	416
11.2 Interpolant bidimensional sumă booleană de tip Lagrange . . . . .	416
11.3 Aproximarea unei integrale duble pe dreptunghi . . . . .	425



# CAPITOLUL 1

---

## Introducere în MATLAB

---

MATLAB<sup>1</sup> este un sistem interactiv destinat calculelor numerice. Prima versiune MATLAB a fost scrisă în anii '70 de Cleve Moler. MATLAB ușurează sarcina utilizatorului de a rezolva problemele numerice. Aceasta permite concentrarea asupra părții creațoare a rezolvării problemei și încurajează experimentele. MATLAB utilizează algoritmi cunoscuți și testați, în care utilizatorul poate avea încredere. Operațiile puternice se pot realiza ușor cu un număr mic de comenzi (de multe ori una sau două). Vă puteți programa propriul set de funcții pentru aplicația dumneavoastră. De asemenea, sunt disponibile facilități grafice excelente, iar imaginile pot fi inserate în documente LATEX sau Word. Pentru o introducere mai detaliată în MATLAB a se vedea [32, 46, 41].

### 1.1. Lansarea MATLAB și sistemul de help

Sub sistemul de operare Windows, MATLAB se lansează dând un click dublu pe iconul corespunzător sau selectând programul din meniul de start. Prompterul din fereastra de comandă este indicat prin >>. MATLAB poate fi utilizat în mai multe moduri: ca un calculator avansat (când comenzi sunt introduse în linia de comandă de la tastatură), ca un limbaj de programare de nivel înalt și sub formă de rutine apelate dintr-un limbaj de programare, de exemplu C.

Informațiile de help pot fi obținute în mai multe moduri:

- din linia de comandă utilizând comanda 'help subiect';
- dintr-o fereastră de help separată, deschisă prin meniul Help;
- utilizând MATLAB helpdesk memorat pe disc sau CD.

---

<sup>1</sup>MATLAB® este o marcă înregistrată a Mathworks Inc., Natick MA

Comanda `help help` da o scurtă descriere a sistemului de `help`, iar `help` fară nici un parametru dă o listă a subiectelor de `help`. Primele linii arată astfel

`HELP topics:`

```
matlab\general - General purpose commands.
matlab\ops      - Operators and special characters.
matlab\lang      - Programming language constructs.
matlab\elmat     - Elementary matrices and matrix manipulation.
matlab\elfun      - Elementary math functions.
matlab\specfun   - Specialized math functions.
matlab\matfun     - Matrix functions - numerical linear algebra.
```

Pentru a obține informații de `help` despre funcțiile elementare se tastează

```
>> help elfun
```

Pentru a obține doar un ecran la un moment dat se poate introduce întâi comanda `more on`, adică

```
>> more on
>> help elfun
```

Pentru a trece la următoarea pagină se poate apăsa orice tastă.

O altă facilitate utilă este utilizarea unei comenzi de forma `lookfor` cuvant-cheie, care cauță în fișierele `help` un cuvânt cheie. Propunem cititorului să testeze `lookfor factorization`, care dă informații despre rutinile de factorizare a matricelor, deosebit de utile în algebră liniară.

Pentru începători și cei care predau MATLAB demonstrațiile sunt foarte utile. Un set cuprinsător se poate lansa prin comanda

```
>> demo
```

Atenție, ea șterge toate variabilele!

În afară de facilitatea de `help` on-line, există un sistem bazat pe hipertext, care dă detalii asupra celor mai multe comenzi și exemple. El este disponibil prin comanda `doc`.

## 1.2. Modul calculator

Operațiile aritmetice de bază sunt `+` `-` `*` `/` și ridicarea la putere `^`. Ordinea implicită a operațiilor se poate schimba cu ajutorul parantezelor.

MATLAB recunoște mai multe tipuri de numere:

- întregi, cum ar fi 1362 sau -217897;
- reale, de exemplu 1.234, -10.76;
- complexe, cum ar fi  $3.21 - 4.3i$ , unde  $i = \sqrt{-1}$ ;
- Inf, desemnează infinitul;

- NaN, Not a Number, care se obține ca rezultat al unei operații ilegale sau al unei neterminate din analiza matematică ( $0/0$ ,  $\infty/\infty$ ,  $\infty - \infty$ , etc.).

Notația cu exponent este de asemenea utilizată:

$$\begin{aligned}-1.3412e + 03 &= -1.3412 \times 10^3 = -1341.2 \\-1.3412e - 01 &= -1.3412 \times 10^{-1} = -0.13412\end{aligned}$$

Toate calculele se realizează în virgulă flotantă. Formatul în care MATLAB afișează numerele este controlat de comanda format. Tastați help format pentru o listă completă. Tabela următoare dă câteva exemple.

Comanda	Exemple de ieșiri
format short	31.4162(4 zecimale)
format short e	31.416e+01
format long e	3.141592653589793e+000
format short g	31.4162(4 zecimale)
format bank	31.42(2 zecimale)

Comanda format compact elimină liniile goale de la ieșire și permite să se afișeze mai multă informație.

Numele de variabile în MATLAB sunt formate din secvențe de litere și cifre, prima fiind o literă. Exemple: x, y, z525, TotalGeneral. Se face distincție între literele mari și cele mici. Există și nume speciale, a căror folosire trebuie evitată, cum ar fi:

- eps =  $2.2204e-16 = 2^{-54}$  este epsilon-ul mașinii (vezi capitolul 3) care reprezintă cel mai mare număr cu proprietatea că  $1+eps$  nu poate fi distins de 1;
- pi =  $\pi$ .

Dacă se fac calcule cu numere complexe folosirea variabilelor i și j este contraindicată, deoarece ele desemnează unitatea imaginară. Dăm câteva exemple:

```
>>x = 3-2^4
x =
-13
>>y = x*5
y =
-65
>>eps
ans =
2.2204e-016
```

Variabila specială ans păstrează valoarea ultimei expresii evaluate. Ea poate fi utilizată în expresii, la fel ca orice altă variabilă.

```
>>3-2^4
ans =
-13
```

cos, sin, tan, csc, sec, cot acos, asin, atan, atan2, asec, acsc, acot cosh, sinh, tanh, sech, csch, coth acosh, asinh, atanh, asech,acsch, acoth log, log2, log10, exp, pow2, nextpow2 ceil, fix, floor, round abs, angle, conj, imag, real mod, rem, sign	Funcții trigonometrice Funcții trigonometrice inverse  Funcții hiperbolice Funcții hiperbolice inverse  Funcții exponențiale  Rotunjiri Complexe Rest, semn
airy, bessel*, beta*, erf*, expint, gamma*, legendre	Funcții matematice
factor, gcd, isprime, lcm, primes, nchoosek, perms, rat, rats	Funcții din teoria numerelor
cart2sph, cart2pol, pol2cart, sph2cart	Transformări de coordonate

Tabela 1.1: Funcții elementare și funcții matematice speciale ("fun\*" indică existența mai multor funcții al căror nume începe cu "fun")

```
>>ans*5
ans =
-65
```

Dacă dorim să suprimăm afișarea ultimei expresii evaluate, vom pune caracterul „;” la sfârșitul expresiei. Pe o linie de comandă se pot introduce mai multe expresii. Ele pot fi separate prin virgulă, caz în care valoarea expresiei terminată cu virgulă va fi afișată, sau cu „;”, caz în care valoarea expresiei nu va fi afișată.

```
>> x=-13; y = 5*x, z = x^2+y, z2 = x^2-y;
y =
-65
z =
104
```

Dacă dorim să salvăm variabile, o putem face cu comanda

```
>>save nume-fisier lista-variabile
```

unde variabilele din lista-variabile sunt separate prin blanc. Se pot folosi în numele de variabile construcții de tip wildcard, desemnate prin \*. Rezultatul salvării se păstrează în fișierul nume-fisier de tip .mat, în format binar, specific MATLAB. Variabilele salvate pot fi încărcate prin

```
>>load nume-fisier
```

Se pot face salvări și încărcări și în format ascii, în dublă precizie sau prin adăugare la un fișier existent. Pentru detalii a se vedea help save și help load.

Lista variabilelor utilizate în sesiunea curentă se poate vizualiza cu whos:

```
>>whos
  Name      Size            Bytes  Class
  ans       1x1              8  double array
  i         1x1              8  double array
  v         1x3              24  double array
  x         1x1              8  double array
  y         1x1              8  double array
  z         1x1              8  double array
  z2        1x1              8  double array
Grand total is 7 elements using 72 bytes
```

**Comanda**

```
>>diary nume-fisier
```

salvează toate comenzi și rezultatele afișate pe ecran (cu excepția celor ale comenziilor grafice) în fișierul nume-fisier. Acest proces de „jurnalizare” se termină prin

```
>>diary off
```

## 1.3. Matrice

Matricele sunt tipuri de date fundamentale în MATLAB. Ele sunt de fapt tablouri multidimensionale în dublă precizie. Cele mai folosite sunt matricele bidimensionale, care sunt tablouri bidimensionale cu  $m$  linii și  $n$  coloane. Vectorii linie ( $m = 1$ ) și coloană ( $n = 1$ ) sunt cazuri particulare de matrice bidimensionale.

### 1.3.1. Generarea matricelor

Există mai multe moduri de a genera matrice. Unul dintre ele este cel explicit, care utilizează parantezele pătrate. Ca separatori între elemente se folosesc blancul sau virgula în interiorul unei linii și punctul și virgula sau „newline” pentru a separa liniile:

```
>> A = [5 7 9
1 -3 -7]
A =
    5     7     9
    1    -3    -7
>> B = [-1 2 5; 9 0 5]
B =
   -1     2     5
    9     0     5
>> C = [0, 1; 3, -2; 4, 2]
C =
    0     1
    3    -2
    4     2
```

Dimensiunea unei matrice se poate obține cu comanda `size`:

<code>zeros</code>	Matricea nulă
<code>ones</code>	Matrice formată din elemente 1
<code>eye</code>	Matricea identică
<code>repmat</code>	Replicarea și pavarea tablourilor
<code>rand</code>	Numere aleatoare distribuite uniform
<code>randn</code>	Numere aleatoare distribuite normal
<code>linspace</code>	Vector de elemente echidistante
<code>logspace</code>	Vector de elemente spațiate logaritmic

Tabela 1.2: Funcții pentru generarea de matrice

```
>> v = size(A)
v =
    2      3
>> [r, c] = size(A)
r =
    2
c =
    3
```

Prima formă returnează un vector cu două elemente ce conține numărul de linii și respectiv de coloane. A doua pune dimensiunile în variabile separate.

MATLAB are un set util de funcții pentru construirea unor matrice speciale, vezi tabela 1.2. Matricele de zerouri, de elemente 1 și matricele identice se obțin cu funcțiile `zeros`, `ones` și respectiv `eye`. Toate au aceeași sintaxă. De exemplu, `zeros(m, n)` sau `zeros([m, n])` produce o matrice  $m \times n$  de zerouri, în timp ce `zeros(n)` produce o matrice  $n \times n$ . Exemple:

```
>> zeros(2)
ans =
    0      0
    0      0
>> ones(2, 3)
ans =
    1      1      1
    1      1      1
>> eye(3, 2)
ans =
    1      0
    0      1
    0      0
```

O situație comună se întâlnește atunci când se dorește construirea unei matrice identice sau nule având o dimensiune egală cu a unei matrice date  $A$ . Aceasta se poate face cu `eye(size(A))`. O funcție înrudită cu `size` este funcția `length`: `length(A)` este cea mai mare dintre dimensiunile lui  $A$ . Astfel, pentru un vector  $n \times 1$  sau  $1 \times n$ ,  $x$ , `length(x)` returnează  $n$ .

Funcțiile `rand` și `randn` generează matrice de numere (pseudo-)aleatoare, utilizând aceeași sintaxă ca și `eye`. Funcția `rand` produce o matrice de numere aleatoare având

distribuția uniformă pe intervalul [0,1]. Funcția `randn` generează o matrice de numere aleatoare având distribuția normală standard. Apelate fără argumente, ambele funcții produc un singur număr aleator.

```
>> rand
ans =
    0.4057
>> rand(3)
ans =
    0.9355    0.8936    0.8132
    0.9169    0.0579    0.0099
    0.4103    0.3529    0.1389
```

În simulările și experimentele cu numere aleatoare este important ca secvențele de numere aleatoare să fie reproductibile. Numerele produse de `rand` depind de starea generatorului. Starea se poate seta prin comanda `rand('state', j)`. Pentru  $j=0$  generatorul `rand` este setat în starea inițială (starea de la lansarea MATLAB). Pentru întregi  $j$  nenuli, generatorul este setat pe a  $j$ -a stare. Starea lui `randn` se setează în același mod. Perioadele lui `rand` și `randn`, adică numărul de termeni generați înainte ca secvențele să înceapă să se repete este mai mare decât  $2^{1492} \approx 10^{449}$ .

Matricele se pot construi și în formă de bloc. Din matricea  $B$ , definită prin  $B=[1 \ 2 \ ; \ 3 \ 4]$ , putem crea

```
>> C=[B, zeros(2); ones(2), eye(2)]
C =
    1     2     0     0
    3     4     0     0
    1     1     1     0
    1     1     0     1
```

Matricele diagonale pe blocuri se pot defini utilizând funcția `blkdiag`, care este mai ușor de utilizat decât notația cu paranteze pătrate. Exemplu:

```
>> A=blkdiag(2*eye(2), ones(2))
A =
    2     0     0     0
    0     2     0     0
    0     0     1     1
    0     0     1     1
```

Funcția `repmat` permite construirea de matrice prin repetarea de subblocuri: `repmat(A, m, n)` crează o matrice de  $m$  pe  $n$  blocuri în care fiecare bloc este o copie a lui  $A$ . Dacă  $n$  lipsește, valoarea sa implicită este  $m$ . Exemplu:

```
>> A=repmat(eye(2), 2)
A =
    1     0     1     0
    0     1     0     1
    1     0     1     0
    0     1     0     1
```

<code>reshape</code>	Schimbarea dimensiunii
<code>diag</code>	Matrice diagonale și diagonale ale matricelor
<code>blkdiag</code>	Matrice diagonală pe blocuri
<code>tril</code>	Extragerea părții triunghiulare inferior
<code>triu</code>	Extragerea părții triunghiulare inferior
<code>fliplr</code>	Rotire matrice în jurul axei de simetrie verticale
<code>flipud</code>	Rotire matrice în jurul axei de simetrie orizontale
<code>rot90</code>	Rotația unei matrice cu 90 de grade

Tabela 1.3: Funcții de manipulare a matricelor

Sunt disponibile și comenzi pentru manipularea matricelor; vezi tabela 1.3.

Funcția `reshape` schimbă dimensiunile unei matrice: `reshape(A, m, n)` produce o matrice  $m \times n$  ale cărei elemente sunt luate coloană cu coloană din  $A$ . De exemplu:

```
>>A=[1 4 9; 16 25 36], B=reshape(A, 3, 2)
A =
    1      4      9
    16     25     36
B =
    1      25
    16     9
    4      36
```

Funcția `diag` lucrează cu diagonalele unei matrice și poate avea ca argument o matrice sau un vector. Pentru un vector  $x$ , `diag(x)` este matricea cu diagonala principală  $x$ :

```
>>diag([1,2,3])
ans =
    1      0      0
    0      2      0
    0      0      3
```

Mai general, `diag(x, k)` pune  $x$  pe diagonala cu numărul  $k$ , unde  $k = 0$  înseamnă diagonala principală,  $k > 0$  specifică diagonale situate deasupra diagonalei principale, iar  $k < 0$  diagonale dedesubtul diagonalei principale:

```
>> diag([1,2],1)
ans =
    0      1      0
    0      0      2
    0      0      0
>> diag([3 4],-2)
ans =
    0      0      0      0
    0      0      0      0
    3      0      0      0
    0      4      0      0
```

Pentru o matrice  $A$ ,  $\text{diag}(A)$  este vectorul coloană format din elementele de pe diagonala principală a lui  $A$ . Pentru a produce o matrice diagonală având aceeași diagonala ca  $A$  se va utiliza  $\text{diag}(\text{diag}(A))$ . Analog cazului vectorial,  $\text{diag}(A, k)$  produce un vector coloană construit din a  $k$ -a diagonala a lui  $A$ . Astfel dacă

```
A =
2      3      5
7     11     13
17    19    23
```

atunci

```
>> diag(A)
ans =
    2
    11
    23
>> diag(A, -1)
ans =
    7
    19
```

$\text{tril}(A)$  obține partea triunghiulară inferior a lui  $A$  (elementele situate pe diagonala principală și dedesubtul ei și în rest zero). Analog lucrează  $\text{triu}(A)$  pentru partea triunghiulară superior. Mai general,  $\text{tril}(A, k)$  dă elementele situate pe diagonala a  $k$ -a a lui  $A$  și dedesubtul ei, în timp ce  $\text{triu}(A, k)$  dă elementele situate pe a  $k$ -a diagonala a lui  $A$  și deasupra ei. Pentru  $A$  ca mai sus:

```
>> tril(A)
ans =
    2      0      0
    7     11      0
   17    19    23
>> triu(A, 1)
ans =
      0      3      5
      0      0     13
      0      0      0
>> triu(A, -1)
ans =
    2      3      5
    7     11     13
    0     19    23
```

MATLAB posedă un set de funcții pentru generarea unor matrice speciale. Aceste matrice au proprietăți interesante care le fac utile pentru construirea de exemple și testarea algoritmilor. Ele sunt date în tabela 1.4. Vom exemplifica funcțiile `hilb` și `vander` în secțiunea 4.2. Funcția `gallery` asigură accesul la o colecție bogată de matrice de test creată de Nicholas J. Higham [34]. Pentru detalii vezi `help gallery`.

compan	matrice companion
gallery	colecție de matrice de test
hadamard	matrice Hadamard
hankel	matrice Hankel
hilb	matrice Hilbert
invhilb	inversa matricei Hilbert
magic	pătrat magic
pascal	matricea Pascal (coeficienți binomiali)
rosser	matrice simetrică pentru testarea valorilor proprii
toeplitz	matrice Toeplitz
vander	matrice Vandermonde
wilkinson	matricea lui Wilkinson pentru testarea valorilor proprii

Tabela 1.4: Matrice speciale

### 1.3.2. Indexarea și notația „:”

Pentru a permite accesul și atribuirea la nivel de *submatrice*, MATLAB are o notație puternică bazată pe caracterul „:”. Ea este utilizată pentru a defini vectori care acționează ca indici. Pentru scalarii  $i$  și  $j$ ,  $i:j$  desemnează vectorul linie cu elementele  $i, i+1, \dots, j$  (pasul este 1). Un pas diferit,  $s$ , se specifică prin  $i:s:j$ . Exemple:

```
>> 1:5
ans =
    1     2     3     4     5
>> 4:-1:-2
ans =
    4     3     2     1     0    -1    -2
>> 0:.75:3
ans =
    0    0.7500    1.5000    2.2500    3.0000
```

Elementele individuale ale unei matrice se accesează prin  $A(i, j)$ , unde  $i \geq 1$  și  $j \geq 1$  (indicii zero sau negativi nu sunt admisi în MATLAB). Notația  $A(p:q, r:s)$  desemnează submatricea constând din intersecția liniilor de la  $p$  la  $q$  și coloanelor de la  $r$  la  $s$  a lui  $A$ . Ca un caz special, caracterul „:” singur, ca specificator de linie și coloană, desemnează toate elementele din acea linie sau coloană:  $A(:, j)$  este a  $j$ -a coloană a lui  $A$ , iar  $A(i, :)$  este a  $i$ -a linie. Cuvântul cheie `end` utilizat în acest context desemnează ultimul indice în dimensiunea specificată; astfel  $A(end, :)$  selectează ultima linie a lui  $A$ . În fine, o submatrice arbitrară poate fi selectată specificând indicii de linie și coloană individuali. De exemplu,  $A([i, j, k], [p, q])$  produce submatricea dată de intersecția liniilor  $i, j$  și  $k$  și coloanelor  $p$  și  $q$ . Iată câteva exemple ce utilizează matricea

```
>> A = [
    2     3     5
    7    11    13
   17    19    23
]
```

a primelor nouă numere prime:

```
>> A(2,1)
ans =
    7
>> A(2:3,2:3)
ans =
    11     13
    19     23
>> A(:,1)
ans =
    2
    7
    17
>> A(2,:)
ans =
    7     11     13
>> A([1 3],[2 3])
ans =
    3     5
    19    23
```

Un caz mai special este  $A(:)$  care desemnează un vector coloană ce conține toate elementele lui  $A$ , așezate coloană după coloană, de la prima la ultima

```
>> B=A(:)
B =
    2
    7
   17
    3
   11
   19
    5
   13
   23
```

Când apare în partea stângă a unei atribuiri,  $A(:)$  completează  $A$ , păstrându-i forma. Cu astfel de notație, matricea de numere prime  $3 \times 3$ ,  $A$ , se poate defini prin

```
>> A=zeros(3); A(:)=primes(23); A=A'
A =
    2     3     5
    7    11    13
   17    19    23
```

Funcția `primes` returnează un vector de numere prime mai mici sau egale cu argumentul ei. Transpunerea  $A = A'$  (vezi secțiunea 1.3.3) este necesară pentru a ordona numerele prime după linii nu după coloane.

Legată de notația „`:`” este funcția `linspace`, care în loc de increment acceptă număr de puncte: `linspace(a,b,n)` generează  $n$  puncte echidistante între  $a$  și  $b$ . Dacă  $n$  lipsește, valoarea sa implicită este 100. Exemplu:

```
>> linspace(-1,1,9)
ans =
Columns 1 through 7
-1.0000   -0.7500   -0.5000   -0.2500    0    0.2500    0.5000
Columns 8 through 9
 0.7500    1.0000
```

Notăția  $[]$  înseamnă matricea vidă,  $0 \times 0$ . Atribuirea lui  $[]$  unei linii sau unei matrice este un mod de a șterge o linie sau o coloană a matricei:

```
>>A(2,:)=[]
A =
  2     3     5
 17    19    23
```

Același efect se obține cu  $A = A([1, 3], :)$ . Matricea vidă este de asemenea utilă ca indicator de poziție într-o listă de argumente, aşa cum se va vedea în §1.3.4.

Operația	Sens matricial	Sens tablou
Adunare	+	+
Scădere	-	-
Înmulțire	*	.*
Împărțire stângă	\	.\
Împărțire uzuală	/	./
Ridicare la putere	^	.

Tabela 1.5: Operații pe matrice și tablouri

### 1.3.3. Operații în sens matricial și în sens tablou

Pentru scalarii  $a$  și  $b$ , operațiile  $+$ ,  $-$ ,  $/$  and  $^$  produc rezultate evidente. Pe lângă operatorul uzual de împărțire, cu semnificația  $\frac{a}{b}$ , MATLAB are operatorul de împărțire stângă (backslash \), cu semnificația  $\frac{b}{a}$ . Pentru matrice, toate aceste operații pot fi realizate în sens matricial (în conformitate cu regulile algebrei matriciale) sau în sens tablou (element cu element). Tabela 1.5 dă lista lor.

Operațiile de adunare și scădere sunt identice atât în sens matricial cât și în sens tablou. Produsul  $A * B$  este înmulțirea matricială uzuală. Operatorii de împărțire / și \ definesc soluții ale sistemelor liniare:  $A \ B$  este soluția  $X$  a lui  $A * X = B$ , în timp ce  $A / B$  este soluția lui  $X * B = A$ . Exemple:

```
>> A=[2,3,5; 7,11,13; 17,19,23]
A =
  2     3     5
  7    11    13
 17    19    23
>> A=[1 2; 3 4], B=ones(2)
A =
```

```

1      2
3      4
B =
1      1
1      1
>> A+B
ans =
2      3
4      5
>> A*B
ans =
3      3
7      7
>> A\B
ans =
-1     -1
1      1

```

Înmulțirea și împărțirea în sens tablou, sau pe elemente, se specifică precedând operatorul cu un punct. Dacă  $A$  și  $B$  sunt matrice de aceeași dimensiune, atunci  $C = A.*B$  însemnă  $C(i, j) = A(i, j) * B(i, j)$  iar  $C = A.\B$  înseamnă  $C(i, j) = B(i, j) / A(i, j)$ . Pentru  $A$  și  $B$  ca în exemplul precedent:

```

>> A.*B
ans =
1      2
3      4
>> B./A
ans =
1.0000    0.5000
0.3333    0.2500

```

Inversa unei matrice pătratice nesingulare se obține cu funcția `inv`, iar determinantul unei matrice pătratice cu `det`.

Ridicarea la putere  $^$  este definită ca putere a unei matrice, dar forma cu punct face ridicarea la putere element cu element. Astfel, dacă  $A$  este o matrice pătratică, atunci  $A^2$  este  $A*A$ , dar  $A.^2$  se obține ridicând la pătrat fiecare element al lui  $A$ :

```

>> A^2, A.^2
ans =
7      10
15     22
ans =
1      4
9      16

```

Operația  $.^$  permite ca exponentul să fie un tablou când dimensiunile bazei și exponentului coincid, sau când baza este un scalar:

```

>> x=[1 2 3]; y=[2 3 4]; z=[1 2; 3 4];
>> x.^y

```

```

ans =
     1      8      81
>> 2.^x
ans =
     2      4      8
>> 2.^z
ans =
     2      4
     8     16

```

Ridicarea la putere a matricelor este definită pentru toate puterile, nu numai pentru întregii pozitivi. Dacă  $n < 0$  este un întreg, atunci  $A^n$  este definit prin  $\text{inv}(A)^{(-n)}$ . Pentru p neîntreg,  $A^p$  este evaluată utilizând valorile proprii ale lui A; rezultatul poate fi incorect sau imprecis dacă A nu este diagonalizabilă sau când A este prost condiționată din punct de vedere al valorilor proprii.

Transpusa conjugată a matricei A se obține cu  $A'$ . Dacă A este reală, atunci aceasta este transpusa obișnuită. Transpusa fără conjugare se obține cu  $A.'$ . Alternativele funcționale  $\text{ctranspose}(A)$  și  $\text{transpose}(A)$  sunt uneori mai convenabile.

În cazul particular al vectorilor coloană x și y,  $x' * y$  este produsul scalar, care se poate obține și cu  $\text{dot}(x, y)$ . Produsul vectorial a doi vectori se poate obține cu  $\text{cross}$ . Exemplu:

```

>> x=[-1 0 1]'; y=[3 4 5]';
>> x'*y
ans =
     2
>> dot(x,y)
ans =
     2
>> cross(x,y)
ans =
    -4
     8
    -4

```

La adunarea dintre un scalar și o matrice, MATLAB va expanda scalarul într-o matrice cu toate elementele egale cu acel scalar. De exemplu

```

>> [4,3;2,1]+4
ans =
     8      7
     6      5
>> A=[1 -1]-6
A =
    -5     -7

```

Totuși, dacă atribuirea are sens fără expandare, atunci va fi interpretată în acest mod. Astfel, dacă comanda precedentă este urmată de  $A=1$ , A va deveni scalarul 1, nu  $\text{ones}(1, 2)$ . Dacă o matrice este înmulțită sau împărțită cu un scalar, operația se realizează element cu element:

```
>> [3 4 5; 4 5 6]/12
```

max	Maximul
min	Minimul
mean	Media
median	Mediana
std	Abaterea medie pătratică
var	Dispersia
sort	Sortare în ordine crescătoare
sum	Suma elementelor
prod	Produsul elementelor
cumsum	Suma cumulată
cumprod	Produsul cumulat
diff	Diferența elementelor

Tabela 1.6: Funcții de bază pentru analiza datelor

```
ans =
0.2500    0.3333    0.4167
0.3333    0.4167    0.5000
```

Funcțiile de matrice în sensul algebrei liniare au numele terminat în m: expm, funm, logm, sqrtm. De exemplu, pentru  $A = [2 \ 2; \ 0 \ 2]$ ,

```
>> sqrt(A)
ans =
1.4142    1.4142
0          1.4142
>> sqrtm(A)
ans =
1.4142    0.7071
0          1.4142
>> ans*ans
ans =
2.0000    2.0000
0          2.0000
```

### 1.3.4. Analiza datelor

Tabela 1.6 dă funcțiile de bază pentru analiza datelor. Cel mai simplu mod de utilizare al lor este să fie aplicate unui vector, ca în exemplele

```
>> x=[4 -8 -2 1 0]
x =
4      -8      -2      1      0
>> [min(x) max(x)]
ans =
-8      4
>> sort(x)
```

```

ans =
-8     -2      0      1      4
>>sum(x)
ans =
-5

```

Funcția `sort` sortează crescător. Pentru un vector real `x`, se poate face sortarea descrescătoare cu `-sort(-x)`. Pentru vectori complecsi, `sort` sortează după valorile absolute.

Dacă argumentele sunt matrice, aceste funcții acționează pe coloane. Astfel, `max` și `min` returnează un vector ce conține elementul maxim și respectiv cel minim al fiecarei coloane, `sum` returnează un vector ce conține sumele coloanelor, iar `sort` sortează elementele din fiecare coloană a unei matrice în ordine crescătoare. Funcțiile `min` și `max` pot returna un al doilea argument care specifică în care componente sunt situate elementul minim și cel maxim. De exemplu, dacă

```

A =
0     -1      2
1      2     -4
5     -3     -4

```

atunci

```

>>max(A)
ans =
5      2      2
>>[m, i]=min(A)
m =
0     -3     -4
i =
1      3      2

```

Așa cum ne arată acest exemplu, dacă există două sau mai multe elemente minimele într-o coloană, se returnează numai indicele primului. Cel mai mic element dintr-o matrice se poate obține aplicând `min` de două ori succesiv:

```

>>min(min(A))
ans =
-4

```

sau utilizând

```

>> min(A(:))
ans =
-4

```

Funcțiile `max` și `min` pot fi făcute să acționeze pe linie printr-un al treilea argument:

```

>>max (A, [], 2)
ans =
2
2
5

```

Argumentul 2 din `max(A, [], 2)` specifică maximul după a doua dimensiune, adică după indicele de coloană. Al doilea argument vid `[]` este necesar, deoarece `max` sau `min` cu două argumente returnează maximul sau minimul celor două argumente:

```
>>max(A, 0)
ans =
    0     0     2
    1     2     0
    5     0     0
```

Funcțiile `sort` și `sum` pot fi și ele făcute să acționeze pe linii, printr-un al doilea argument. Pentru detalii a se vedea `help sort` sau `doc sort`.

Funcția `diff` calculează diferențe. Aplicată unui vector  $x$  de lungime  $n$  produce vectorul  $[x(2)-x(1) \ x(3)-x(2) \ \dots \ x(n)-x(n-1)]$  de lungime  $n-1$ . Exemplu

```
>>x=(1:8).^2
x =
    1     4     9     16     25     36     49     64
>>y=diff(x)
y =
    3     5     7     9     11     13     15
>>z=diff(y)
z =
    2     2     2     2     2     2
```

### 1.3.5. Operatori relaționali și logici

Operatorii relaționali în MATLAB sunt: `==` (egal), `~` (diferit), `<` (mai mic), `>` (mai mare), `<=` (mai mic sau egal) și `>=` (mai mare sau egal). De notat că un singur `=` = înseamnă atribuire.

Comparația între scalari produce 1 dacă relația este adevărată și 0 în caz contrar. Comparările sunt definite între matrice de aceeași dimensiune și între o matrice și un scalar, rezultatul fiind în ambele cazuri o matrice de 0 și 1. La comparația matrice-matrice se compară perechile corespunzătoare de elemente, pe când la comparația matrice-scalar se compară scalarul cu fiecare element. De exemplu:

```
>> A=[1 2; 3 4]; B = 2*ones(2);
>> A == B
ans =
    0     1
    0     0
>>A > 2
ans =
    0     0
    1     1
```

Pentru a testa dacă dacă matricele  $A$  și  $B$  sunt identice, se poate utiliza expresia `isequal(A, B)`:

<code>ischar</code>	Testează dacă argumentul este sir de caractere(string)
<code>isempty</code>	Testează dacă argumentul este vid
<code>isequal</code>	Testează dacă tablourile sunt identice
<code>isfinite</code>	Testează dacă elementele unui tablou sunt finite
<code>isieee</code>	Testează dacă mașina utilizează aritmetică IEEE
<code>isinf</code>	Testează dacă elementele unui tablou sunt <code>inf</code>
<code>islogical</code>	Testează dacă argumentul este un tablou logic
<code>isnan</code>	Test de NaN
<code>isnumeric</code>	Testează dacă argumentul este numeric
<code>isreal</code>	Testează dacă argumentul este tablou real
<code>issparse</code>	Testează dacă argumentul este tablou rar

Tabela 1.7: O selecție de funcții logice `is*`

```
>> isequal (A, B)
ans =
0
```

Mai există și alte funcții logice înrudite cu `isequal` și al căror nume începe cu `is`. O selecție a lor apare în tabela 1.7; pentru o listă completă a se tasta `doc is`. Funcția `isnan` este utilă deoarece testul `x == NaN` produce întotdeauna 0 (false), chiar dacă `x` este NaN! (Un NaN este prin definiție diferit de orice și nu are o relație de ordine cu nimic, vezi secțiunea 3.4.)

Operatorii logici în MATLAB sunt: `&` (și), `|` (sau), `~` (not), `xor` (sau exclusiv), `all` (adevărat dacă toate elementele unui vector sunt nenule), `any` (adevărat dacă cel puțin un element al unui vector este nenul). Dăm câteva exemple:

```
>> x = [-1 1 1]; y = [1 2 -3];
>> x>0 & y>0
ans =
0     1     0
>> x>0 | y>0
ans =
1     1     1
>> xor(x>0,y>0)
ans =
1     0     1
>> any(x>0)
ans =
1
>> all(x>0)
ans =
0
```

De notat că `xor` trebuie apelat ca o funcție: `xor(a,b)`. Operatorii logici `and`, `or`, `not` și cei relationali pot fi apelați și în formă funcțională: `and(a,b), ..., eq(a,b), ...` (vezi `help ops`).

Precedența operatorilor este rezumată în tabela 1.8 (vezi `help precedence`). MATLAB evaluează operatorii de precedență egală de la stânga la dreapta. Precedența se poate

Nivel de precedență	Operator
1 (cea mai mare)	transpusa ( $\cdot'$ ), putere( $\cdot^{\wedge}$ ), transpusa conjugată complexă( $'$ ), putere matricială( $\wedge$ )
2	plus unar (+), minus unar (-), negație ( $\sim$ )
3	înmulțire ( $\cdot \ast$ ), împărțire dreaptă ( $\cdot /$ ), împărțire stângă ( $\cdot \backslash$ ), înmulțire matricială ( $\ast$ ), împărțire dreaptă matricială ( $/\backslash$ ), împărțire stângă matricială ( $\backslash /$ )
4	adunare (+), scădere (-)
5	două puncte (:)
6	mai mic (<), mai mic sau egal (<=), mai mare (>), mai mare sau egal (>=), egal (==), diferit ( $\neq$ )
7	și logic ( $\&$ )
8 (cea mai mică)	sau logic ( $\mid$ )

Tabela 1.8: Precedența operatorilor

modifica cu ajutorul parantezelor.

De notat că versiunile MATLAB anterioare lui MATLAB 6 aveau aceeași precedență pentru and și or (spre deosebire de majoritatea limbajelor de programare). MathWorks recomandă folosirea parantezelor pentru a garanta obținerea rezultatelor identice în toate versiunile MATLAB.

Pentru matrice `all` returnează un vector linie ce conține rezultatul lui `all` aplicat fiecărei coloane. De aceea `all(all(A==B))` este un alt mod de a testa egalitatea matricelor A și B. Funcția `any` lucrează analog; de exemplu, `any(any(A==B))` returnează 1 dacă A și B au cel puțin un element egal și 0 în caz contrar.

Comanda `find` returnează indicii corespunzători elementelor nenule ale unui vector. De exemplu,

```
>> x = [-3 1 0 -inf 0];
>> f = find(x)
f =
    1     2     4
```

Rezultatul lui `find` poate fi apoi utilizat pentru a selecta doar acele elemente ale vectorului:

```
>> x(f)
ans =
   -3     1    -Inf
```

Cu x ca în exemplul de mai sus, putem utiliza `find` pentru a obține elementele finite ale lui x,

```
>> x(find(isfinite(x)))
ans =
   -3     1     0     0
```

și să înlocuim componente negative ale lui x cu zero:

```
>> x(find(x<0))=0
x =
    0     1     0     0     0
```

Când `find` se aplică matricei A, vectorul de indici corespunde lui A privită ca un vector coloană obținut din așezarea coloanelor una peste alta (adică A(:)), și acest vector poate fi utilizat pentru a indexa A. În exemplul următor se utilizează `find` pentru a face zero toate elementele lui A care sunt mai mici decât elementele corespunzătoare ale lui B:

```
>> A = [4 2 16; 12 4 3], B = [12 3 1; 10 -1 7]
A =
    4      2      16
   12      4       3
B =
    12      3       1
    10     -1       7
>> f = find(A<B)
f =
    1
    3
    6
>> A(f) = 0
A =
    0      0      16
   12      4       0
```

În cazul matricelor, putem utiliza `find` sub forma `[i, j] = find(A)`, care returnează vectorii i și j ce conțin indicii de linie și coloană ale elementelor nenule.

Rezultatele operatorilor logici și ale funcțiilor logice din MATLAB sunt tablouri de elemente 0 și 1, care sunt exemple de tablouri logice. Astfel de tablouri pot fi create și prin aplicarea funcției `logical` unui tablou numeric. Tablourile logice pot fi utilizate la indexare. Fie exemplul

```
>> clear
>> y = [1 2 0 -3 0]
y =
    1      2      0     -3      0
>> i1 = logical(y)
Warning: Values other than 0 or 1 converted to logical 1 (Type
"warning off MATLAB:conversionToLogical" to suppress
this warning.)
>> i1 =
    1      1      0      1      0
>> i2 = (y~=0)
i2 =
    1      1      0      1      0
>> i3 = [1 1 0 1 0]
i3 =
    1      1      0      1      0
>> whos
```

```

Name      Size          Bytes  Class
i1       1x5           5    logical array
i2       1x5           5    logical array
i3       1x5          40   double array
y        1x5          40   double array
Grand total is 20 elements using 90 bytes
>> y(i1)
ans =
     1     2    -3
>> y(i2)
ans =
     1     2    -3
>> isequal(i2,i3)
ans =
     1
>> y(i3)
??? Subscript indices must either be real positive
integers or logicals.

```

Acest exemplu ilustrează regula că A(M), unde M este un tablou logic de aceeași dimensiune ca și A, extrage elementele lui A corespunzând elementelor lui M cu partea reală nenulă. Chiar dacă i2 are aceleși elemente ca i3 (și la comparație ele ies egale), doar tabloul logic i2 poate fi utilizat la indexare.

Un apel la `find` poate fi uneori evitat dacă argumentul său este un tablou logic. În exemplul precedent, `x(find(isfinite(x)))` poate fi înlocuit cu `x(isfinite(x))`. Se recomandă utilizarea lui `find` pentru claritate.

## 1.4. Programarea în MATLAB

### 1.4.1. Fluxul de control

MATLAB are patru structuri de control: instrucțiunea `if`, instrucțiunea de ciclare `for`, instrucțiunea de ciclare `while` și instrucțiunea `switch`. Cea mai simplă formă a instrucțiunii `if` este

```

if expresie
    instrucțiuni
end

```

unde secvența *instrucțiuni* este executată dacă părțile reale ale elementelor lui *expresie* sunt toate nenele. Secvența de mai jos interschimbă x și y dacă x este mai mare decât y:

```

if x > y
    temp = y;
    y = x;
    x = temp;
end

```

Atunci când o instrucțiune `if` este urmată în aceeași linie de alte instrucțiuni, este nevoie de o virgulă pentru a separa `if`-ul de instrucțiunea următoare:

```
if x > 0, x = sqrt(x); end
```

Alternativa se implementează cu `else`, ca în exemplul

```
a = pi^exp(1); c = exp(pi);
if a >= c
    b = sqrt(a^2-c^2)
else
    b = 0
end
```

În fine, se pot introduce teste suplimentare cu `elseif` (de notat că nu este nici un spațiu între `else` și `if`):

```
>> if a >= c
    b = sqrt(a^2-c^2)
elseif a^c > c^a
    b = c^a/a^c
else
    b = a^c/c^a
end
```

Într-un test `if` de forma „`if condiție1 & condiție2`”, `condiție2` nu este evaluată dacă `condiție1` este falsă (așa-numită evaluare prin scurtcircuit). Acest lucru este util când evaluarea lui `condiție2` ar putea da o eroare — probabil din cauza unei variabile nedefinite sau a unei depășiri de indice.

Ciclul `for` este una dintre cele mai utile construcții MATLAB, deși codul este mai compact fără ea. Sintaxa ei este

```
for variabilă = expresie
    instrucțiuni
end
```

De obicei, `expresie` este un vector de forma `i:s:j`. Instrucțiunile sunt executate pentru `variabilă` egală cu fiecare element al lui `expresie` în parte. De exemplu, suma primilor 25 de termeni ai seriei armonice  $1/i$  se calculează prin

```
>> s = 0;
>> for i = 1:25, s = s + 1/i; end, s
s =
    3.8160
```

Un alt mod de a defini `expresie` este utilizarea notației cu paranteze pătrate:

```
for x = [pi/6 pi/4 pi/3], disp([x, sin(x)]), end
0.5236    0.5000
0.7854    0.7071
1.0472    0.8660
```

Ciclurile `for` pot fi imbricate, indentarea ajutând în acest caz la creșterea lizibilității. Editorul-debuger-ul MATLAB poate realiza indentarea automată. Codul următor construiește o matrice simetrică 5 pe 5,  $A$ , cu elementul  $(i, j)$  egal cu  $i/j$  pentru  $j \geq i$ :

```
n = 5; A = eye(n);
for j=2:n
    for i = 1:j-1
        A(i,j)=i/j;
        A(j,i)=i/j;
    end
end
```

Expresia din ciclul `for` poate fi o matrice, în care caz lui *variabilă* î se atribuie succesiv coloanele lui *expresie*, de la prima la ultima. De exemplu, pentru a atribui lui `x` fiecare vector al bazei canonice, putem scrie `for x=eye(n), ... , end`.

Ciclul `while` are forma

```
while expresie
    instrucțiuni
end
```

Secvența *instrucțiuni* se execută atât timp cât *expresie* este adevărată. Exemplul următor aproximează cel mai mic număr nenul în virgulă flotantă:

```
x = 1;
while x>0, xmin = x; x = x/2; end, xmin
xmin =
4.9407e-324
```

Execuția unui ciclu `while` sau `for` poate fi terminată cu o instrucțiune `break`, care dă controlul primei instrucțiuni de după `end`-ul corespunzător. Construcția `while 1, ..., end`, reprezintă un ciclu infinit, care este util atunci când nu este convenabil să se pună testul la începutul ciclului. (De notat că, spre deosebire de alte limbi, MATLAB nu are un ciclu „repeat-until”.) Putem rescrie exemplul precedent mai concis prin

```
x = 1;
while 1
    xmin = x;
    x = x/2;
    if x == 0, break, end
end
xmin
```

Într-un ciclu imbricat un `break`iese în ciclul de pe nivelul anterior.

Instrucțiunea `continue` cauzează trecerea controlului la execuția unui ciclu `for` sau `while` următoarei iterații, sărind instrucțiunile rămase din ciclu. Un exemplu trivial este:

```
for i=1:10
    if i < 5, continue, end
    disp(i)
end
```

care afișează întregii de la 5 la 10.

Structura de control cu care încheiem este instrucțiunea `switch`. Ea constă din „`switch` *expresie*” urmată de o listă de instrucțiuni „`case` *expresie* *instrucțiuni*”, terminată optional cu „`otherwise` *instrucțiuni*” și urmată de `end`. Exemplul următor evaluează *p*-norma unui vector `x` pentru trei valori ale lui *p*:

```

switch(p)
case 1
    y = sum(abs(x));
case 2
    y = sqrt(x'*x);
case inf
    y = max(abs(x));
otherwise
    error('p poate fi 1, 2 sau inf.')
end

```

Funcția `error` generează un mesaj de eroare și oprește execuția curentă. Expresia ce urmează după `case` poate fi o listă de valori delimitate de acolade. Expressia din `switch` poate coincide cu orice valoare din listă:

```

x = input('Enter a real number: ')
switch x
case {inf, -inf}
    disp('Plus or minus infinity')
case 0
    disp('Zero')
otherwise
    disp('Nonzero and finite')
end

```

Construcția `switch` din MATLAB se comportă diferit de cea din C sau C++ : odată ce MATLAB a selectat un grup de expresii `case` și instrucțiunile sale au fost executate, se dă controlul primei instrucțiuni de după `switch`, fără a fi nevoie de instrucțiuni `break`.

### 1.4.2. Fișiere M

Fișierele M din MATLAB sunt echivalentele programelor, funcțiilor, subrutinelor și procedurilor din alte limbaje de programare. Ele oferă următoarele avantaje:

- experimentarea algoritmului prin editare, în loc de a retipări o listă lungă de comenzi;
- înregistrarea permanentă a unui experiment;
- construirea de utilitate, care pot fi utilizate repetat;
- schimbul de fișiere M.

Multe fișiere M scrise de entuziaști pot fi obținute de pe Internet, pornind de la pagina de web <http://www.mathworks.com>. Există și grupul de știri `comp.soft-sys.matlab`, dedicat MATLAB. (Grupurile de știri pot fi citite în mai multe moduri, inclusiv printr-un web browser.) Se pot obține detalii tastând `info` la prompterul MATLAB.

Un fișier M este un fișier text cu extensia (tipul) `.m` ce conține comenzi MATLAB. Ele sunt de două tipuri:

**Fișiere M de tip script** (sau fișiere de comenzi) — nu au nici un argument de intrare sau ieșire și operează asupra variabilelor din spațiul de lucru.

**Fișiere M de tip funcție** — conțin o linie de definiție `function` și pot accepta argumente de intrare și returna argumente de ieșire, iar variabilele lor interne sunt locale funcției (înafără de cazul când sunt declarate `global`).

Un fișier script permite memorarea unei secvențe de comenzi care sunt utilizate repetat sau vor fi necesare ulterior.

Script-ul de mai jos utilizează numerele aleatoare pentru a simula un joc. Să considerăm 13 cărți de pică care sunt bine amestecate. Probabilitatea de a alege o carte particulară din pachet este 1/13. Acțiunea de extragere a unei cărți se implementează prin generarea unui număr aleator. Jocul continuă prin punerea cărții înapoi în pachet și reamestecare până când utilizatorul apăsa o tastă diferită de `r` sau s-a atins numărul de repetări (20).

```
%JOCMARTI
%Simularea unui joc de carti

rand('state',sum(100*clock));
for k=1:20
    n=ceil(13*rand);
    fprintf('Cartea extrasă: %3.0f\n',n)
    disp(' ')
    disp('Apasati r si Return pentru a continua')
    r=input('sau orice literă pentru a termina: ','s');
    if r~=r', break, end
end
```

Linia

```
rand('state',sum(100*clock));
```

resetează de fiecare dată generatorul la o stare diferită.

Primele două linii ale acestui fișier script încep cu simbolul `%` și deci sunt linii de comentariu. Oricătre ori MATLAB întâlnește un `%` va ignora restul liniei. Aceasta ne permite să inserăm texte explicative care vor face fișierele M mai ușor de înțeles. Începând cu versiunea 7 se admit blocuri de comentarii, adică comentarii care să se întindă pe mai multe linii. Ele sunt delimitate prin operatorii `%{` și `%}`. Ei trebuie să fie singuri pe linie, ca în exemplul:

```
%{
Comentariu bloc
pe două lini
%}
```

Dacă script-ul de mai sus este memorat în fișierul `joccarti.m`, tastând `joccarti` se obține:

```
>> joccarti
Cartea extrasă: 7
```

```
Apasati r si Return pentru a continua
sau orice literă pentru a termina: r
```

---

Cartea extrașa: 3

Apasati r si Return pentru a continua  
sau orice litera pentru a termina: a  
>>

Fișierele M de tip funcție permit extinderea limbajului MATLAB prin scrierea de funcții proprii care acceptă și returnează argumente. Ele se pot utiliza în același mod ca funcțiile MATLAB existente, cum ar fi sin, eye, size, etc.

---

### Sursa MATLAB 1.1 Funcția stat

---

```
function [med, abmp] = stat(x)
%STAT Media si abaterea medie patratica a unei selectii
%      [MED,ABMP] = STAT(X) calculeaza media si abaterea
%      medie patratica a selectiei X

n = length(x);
med = sum(x)/n;
abmp = sqrt(sum((x-med).^2)/n);
```

---

Sursa MATLAB 1.1 dă o funcție simplă care calculează media și abaterea medie pătratică a unei selecții (vector). Acest exemplu ilustrează unele facilități ale funcțiilor. Prima linie începe cu cuvântul cheie function urmat de argumentele de ieșire, [med, abmp] și de simbolul =. În dreapta = urmează numele funcției, stat, urmat de argumentele de intrare, în cazul nostru x, între paranteze. (În general, putem avea orice număr de argumente de intrare și de ieșire.) Numele de funcție trebuie să fie la fel ca al fișierului .m în care funcția este memorată – în cazul nostru stat.m.

A doua linie a unui fișier funcție se numește linie H1 sau help 1. Se recomandă ca ea să aibă următoarea formă: să înceapă cu un %, urmat fără nici un spațiu de numele funcției cu litere mari, urmat de unul sau mai multe spații și apoi o scurtă descriere. Descrierea va începe cu o literă mare, se va termina cu un punct, iar dacă este în engleză se vor omite cuvintele "the" și "a". Când se tastează help nume\_functie, toate liniile, de la prima linie de comentariu până la prima linie care nu este de comentariu (de obicei o linie goală, pentru lizibilitatea codului sursă) sunt afișate pe ecran. Deci, aceste linii descriu funcția și argumentele sale. Se convine ca numele de funcție și de argumente să se scrie cu litere mari. Pentru exemplul stat.m avem

```
>>help stat
STAT media si abaterea medie patratica a unei selectii
      [MED,ABMP] = STAT(X) calculeaza media si abaterea
      medie patratica a selectiei X
```

Se recomandă documentarea *tuturor* funcțiilor utilizator în acest mod, oricât de scurte ar fi. Este util ca în liniile de comentariu din text să apară date screrii funcției și datele când s-au făcut modificări. Comanda help lucrează similar și pe fișiere script.

Funcția stat se apelează la fel ca orice funcție MATLAB:

```
>> [m, a]=stat(1:10)
```

```
m =
    5.5000
a =
    2.8723
>> x=rand(1,10);
[m,a]=stat(x)
m =
    0.5025
a =
    0.1466
```

O funcție mai complicată este `sqrtn`, ilustrată în sursa 1.2. Dându-se  $a > 0$ , ea calculează  $\sqrt{a}$  cu metoda lui Newton,

$$x_{k+1} = \frac{1}{2} \left( x_k + \frac{a}{x_k} \right), \quad x_1 = a,$$

afișând și iterațiile. Dăm exemple de utilizare:

```
>> [x,it]=sqrtn(2)
      k           x_k          er. relativă
      1: 1.500000000000000e+000  3.33e-001
      2: 1.41666666666665e+000  5.88e-002
      3: 1.4142156862745097e+000 1.73e-003
      4: 1.4142135623746899e+000 1.50e-006
      5: 1.4142135623730949e+000 1.13e-012
      6: 1.4142135623730949e+000 0.00e+000
x =
    1.4142
it =
    6
>> x=sqrtn(2,1e-4)
      k           x_k          er. relativă
      1: 1.500000000000000e+000  3.33e-001
      2: 1.41666666666665e+000  5.88e-002
      3: 1.4142156862745097e+000 1.73e-003
      4: 1.4142135623746899e+000 1.50e-006
x =
    1.4142
```

Acest fișier M utilizează comanda `return`, care dă controlul apelantului. Spre deosebire de alte limbaje de programare, nu este necesar să se pună `return` la sfârșitul unei funcții sau al unui script. Funcția `nargin` returnează numărul de argumente de intrare cu care funcția a fost apelată și permite atribuirea de valori implicate argumentelor nespecificate. Dacă apelul lui `sqrtn` nu a furnizat o valoare pentru `tol`, atunci `tol` primește valoarea `eps`. Numărul de argumente la ieșire este returnat de funcția `nargout`.

Un fișier M de tip funcție poate conține alte funcții, numite subfuncții, care pot să apară în orice ordine după funcția principală (sau primară). Subfuncțiile sunt vizibile numai din funcția principală sau din alte subfuncții. Ele realizează calcule care trebuie separate de

---

**Sursa MATLAB 1.2 Funcția sqrt**

---

```

function [x,iter] = sqrt(a,tol)
%SQRTN Radical cu metoda lui Newton.
%X = SQRTN(A,TOL) calculeaza radacina patrata a lui
%A prin metoda lui Newton(sau a lui Heron).
%presupunem ca A >= 0.
%TOL este toleranta (implicit EPS).
[X,ITER] = SQRTN(A,TOL) returneaza numarul de
%iteratii ITER necesare.

if nargin < 2, tol = eps; end

x = a;
iter = 0;
xdiff = inf;
fprintf(' k er. relativă\n')
for k=1:50
    iter = iter + 1;
    xold = x;
    x = (x + a/x)/2;
    xdiff = abs(x-xold)/abs(x);
    fprintf('%2.0f: %20.16e %9.2e\n', iter, x, xdiff)
    if xdiff <= tol, return, end
end
error('Nu s-a atins precizia dupa 50 de iteratii.')

```

---

funcția principală, dar nu sunt necesare în alte fișiere M, sau supraîncarcă funcții cu același nume (subfuncțiile au prioritate mai mare). Help-ul pentru o subfuncție se poate specifica punând numele funcției urmat de "/" și numele subfuncției. Pentru exemple a se vedea sursele MATLAB 5.3 și 5.6 din capitolul 5.

Pentru a crea și edita fișiere M avem două posibilități. Putem utiliza orice editor pentru fișiere ASCII sau putem utiliza MATLAB Editor/Debugger. Sub Windows el se apelează prin comanda `edit` sau din opțiunile de meniu File-New sau File-Open. Sub Unix se apelează doar prin comanda `edit`. Editorul/debugger-ul MATLAB are diverse facilități care ajută utilizatorul, cum ar fi indentarea automată a ciclurilor și structurilor de control, evidențierea sintaxei prin culori, verificarea perechilor de paranteze și apostrofuri.

Cele mai multe funcții MATLAB sunt fișiere M păstrate pe disc, dar există și funcții predefinite conținute în interpreterul MATLAB. Calea MATLAB (MATLAB path) este o listă de directorii care specifică unde căută MATLAB fișierele M. Un fișier M este disponibil numai dacă este pe calea MATLAB. Drumul poate fi setat și modificat prin comenzile `path` și `addpath`, sau prin utilitarul (fereastra) `path Browser`, care se apelează din opțiunea de meniu File-Set Path sau tastând `pathtool`. Un script (dar nu și o funcție) care nu este pe calea de căutare se poate executa cu `run` urmat de calea completă până la fișierul M. Un fișier

M se poate afișa pe ecran cu comanda `type`.

Un aspect important al MATLAB este dualitatea comenzi-funcții. În afară de forma clasică, nume, urmat de argumente între paranteze, funcțiile pot fi apelate și sub forma nume, urmat de argumente separate prin spații. MATLAB presupune în al doilea caz că argumentele sunt siruri de caractere. De exemplu apelurile `format long` și `format ('long')` sunt echivalente.

Începând cu versiunea 7, MATLAB permite definirea de funcții imbricate, adică funcții conținute în corpul altor funcții. În exemplul care urmează, funcția F2 este imbricată în funcția F1:

```
function x = F1(p1,p2)
...
F2(p2)
    function y = F2(p3)
        ...
    end
...
end
```

Ca orice altă funcție, o funcție imbricată are propriul său spațiu de lucru în care se memorează variabilele pe care le utilizează. Ea are de asemenea acces la spațiul de lucru al tuturor funcțiilor în care este imbricată. Astfel, de exemplu, o variabilă care are o valoare atribuită ei de funcția exterioară poate fi citită și modificată de o funcție imbricată la orice nivel în funcția exterioară. Variabilele create într-o funcție imbricată pot fi citite sau modificate în orice funcție care conține funcția imbricată.

### 1.4.3. Argumente funcție

În multe probleme, cum ar fi integrarea numerică, rezolvarea unor ecuații operatoriale, minimizarea unei funcții, este nevoie ca o funcție să fie transmisă ca argument unei alte funcții. Aceasta se poate realiza în mai multe feluri, depinzând de modul în care funcția apelată a fost scrisă. Vom ilustra aceasta cu funcția `ezplot`, care reprezintă grafic funcția  $f(x)$  peste domeniul implicit  $[-2\pi, 2\pi]$ . Un prim mod este transmiterea funcției printr-o construcție numită *function handle*. Acesta este un tip de date MATLAB care conține toate informațiile necesare pentru a evalua o funcție. Un function handle poate fi creat punând caracterul @ în fața numelui de funcție. Astfel, dacă `fun` este un fișier M de tip funcție de forma cerută de `ezplot`, atunci putem tasta

```
ezplot(@fun)
```

`fun` poate fi numele unei funcții predefinite:

```
ezplot(@sin)
```

Numele unei funcții poate fi transmis ca un sir de caractere:

```
ezplot('exp')
```

Function handle a fost introdus începând cu MATLAB 6 și este de preferat utilizării sirurilor, fiind mai eficient și mai versatil. Totuși, ocazional se pot întâlni funcții care să accepte argumente de tip funcție sub formă de sir, dar nu sub formă de function handle. Conversia dintr-o formă în alta se poate face cu `func2str` și `str2func` (vezi `help function_handle`). Mai există două moduri de a transmite o funcție lui `ezplot`: ca expresie între apostrofuri,

---

**Sursa MATLAB 1.3 Funcția fd\_deriv**

---

```
function y = fd_deriv(f,x,h)
%FD_DERIV Aproximarea derivatei cu diferența divizată.
%           FD_DERIV(F,X,H) este diferența divizată a lui F cu
%           nodurile X și X+ H. H implicit: SQRT(EPS).

if nargin < 3, h = sqrt(eps); end
y = (feval(f,x+h) - feval(f,x))/h;
```

---

ezplot('x^2-1'), ezplot('1/(1+x^2)')

sau ca obiect inline

ezplot(inline('exp(x)-1'))

Un obiect inline este o funcție definită printr-un sir și care poate fi atribuită unei variabile și apoi evaluată:

```
>> f=inline('exp(x)-1'), f(2)
f =
    Inline function:
    f(x) = exp(x)-1
ans =
    6.3891
```

MATLAB determină și ordenează argumentele unei funcții inline. Dacă acest lucru nu este satisfăcător, argumentele se pot defini și ordona explicit, transmițând lui `inline` parametrii suplimentari:

```
>> f = inline('log(a*x)/(1+y^2)')
f =
    Inline function:
    f(a,x,y) = log(a*x)/(1+y^2)
>> f = inline('log(a*x)/(1+y^2)', 'x', 'y', 'a')
f =
    Inline function:
    f(x,y,a) = log(a*x)/(1+y^2)
```

Începând cu versiunea 7, MATLAB permite funcții anonime. Ele pot fi definite în linii de comandă, fișiere M de tip funcție sau script și nu necesită un fișier M. Sintaxa pentru crearea unei funcții anonime este

`f = @(listaarg) expresie`

Instrucțiunea de mai jos crează o funcție anonimă care ridică argumentul ei la patrat:

`sqr = @(x) x.^2;`

Dăm și un exemplu de utilizare

```
a = sqr(5)
a =
    25
```

Pentru a evalua în corpul unei funcții o funcție transmisă ca parametru se utilizează funcția `feval`. Sintaxa ei este `feval(fun,x1,x2,...,xn)`, unde `fun` este funcția, iar `x1,x2,...,xn` sunt argumentele sale. Să considerăm funcția `fd_deriv` din sursa 1.3. Această funcție aproximează derivata funcției dată ca prim argument cu ajutorul diferenței divizate

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

Când se tastează

```
>> fd_deriv(@sqrt,0.1)
ans =
    1.5811
```

primul apel la `feval` din `fd_deriv` este equivalent cu `sqrt(x+h)`. Putem utiliza funcția `sqrtn` (sursa MATLAB 1.2) în locul funcție predefinirea `sqrt`:

```
>> fd_deriv(@sqrtn,0.1)
k          x_k          er. relativă
1:  5.5000000745058064e-001  8.18e-001
2:  3.6590910694939033e-001  5.03e-001
% Restul ieșirii lui sqrtn se omite
ans =
    1.5811
```

Putem transmite lui `fd_deriv` un obiect inline, dar o expresie de tip sir nu funcționează:

```
>> f = inline('exp(-x)/(1+x^2)');
>> fd_deriv(f,pi)
ans =
    -0.0063
>> fd_deriv('exp(-x)/(1+x^2)',pi)
??Error using ==> feval Invalid function name 'exp(-x)/(1+x^2)'
Error in ==> C:\radu\sane2\FD_DERIV.M
On line 8 ==> y = (feval(f,x+h) - feval(f,x))/h;
```

Pentru a face `fd_deriv` să accepte expresii de tip sir vom insera

```
f=fcnchk(f);
la începutul funcției (în acest mod lucrează ezplot și alte funcții MATLAB, vezi [46] pentru exemple).
```

Este uneori necesar să se „vectorizeze” un obiect inline sau o expresie de tip sir, adică să se convertească înmulțirile, ridicările la putere și împărțirile în operații în sens tablou, astfel ca să se poată utiliza argumente vectori și matrice. Acest lucru se poate realiza cu funcția `vectorize`:

```
>> f = inline('log(a*x)/(1+y^2)');
>> f = vectorize(f)
f =
    Inline function:
    f(a,x,y) = log(a.*x)./(1+y.^2)
```

Dacă `fcnchk` se apelează cu un argument suplimentar '`vectorized`', ca în `fcnchk(f, 'vectorized')`, atunci ea vectorizează sirul `f`.

MATLAB 7 a simplificat modul de apel al argumentelor de tip funcție. Se pot apela funcții referite cu function handle prin interfața de apel standard în loc de `feval`. Astfel, un function handle va fi tratat ca și un nume de funcție

```
fhandle(arg1, arg2, ..., argn)
```

Dacă funcția nu are nici un argument apelul ei are forma

```
fhandle()
```

Linia a doua a funcției `fd_deriv` (sursa 1.3) ar fi putut fi scrisă în MATLAB 7 sub forma

```
y = (f(x+h) - f(x))/h;
```

#### 1.4.4. Număr variabil de argumente

În anumite situații o funcție trebuie să accepte sau să returneze un număr variabil, posibil nelimitat, de argumente. Aceasta se poate realiza utilizând funcțiile `varargin` și `varargout`. Să presupunem că dorim să scriem o funcție `companb` ce construiește matricea companion pe blocuri, de dimensiune  $mn \times mn$ , a matricelelor  $n \times n$   $A_1, A_2, \dots, A_m$ :

$$C = \begin{bmatrix} -A_1 & -A_2 & \dots & \dots & -A_m \\ I & 0 & & & 0 \\ & I & \ddots & & \vdots \\ & & \ddots & & \vdots \\ & & & I & 0 \end{bmatrix}.$$

Soluția este de a utiliza `varargin` aşa cum se arată în sursa MATLAB 1.4. Când

---

#### Sursa MATLAB 1.4 Funcția `companb`

---

```
function C = companb(varargin)
%COMPANB    Matrice companion pe blocuri.
%
%          C = COMPANB(A_1,A_2,...,A_m) este matricea
%          companion pe blocuri corespunzatoare
%          matricelor n-pe-n A_1,A_2,...,A_m.

m = nargin;
n = length(varargin{1});
C = diag(ones(n*(m-1),1),-n);
for j = 1:m
    Aj = varargin{j};
    C(1:n, (j-1)*n+1:j*n) = -Aj;
end
```

---

`varargin` apare în lista de argumente, argumentele furnizate sunt copiate într-un tablou de celule numit `varargin`. Tablourile de celule (cell arrays) sunt structuri de date de tip tablou, în care fiecare element poate păstra date de tipuri și dimensiuni diferite. Elementele unui tablou de celule pot fi selectate utilizând acolade. Considerăm apelul

```
>> X = ones(2); C =companb(X, 2*X, 3*X)
C =
    -1     -1     -2     -2     -3     -3
    -1     -1     -2     -2     -3     -3
     1      0      0      0      0      0
     0      1      0      0      0      0
     0      0      1      0      0      0
     0      0      0      1      0      0
```

Dacă inserăm o linie ce conține doar varargin la începutul lui companb apelul de mai sus produce

```
varargin =
[2x2 double] [2x2 double] [2x2 double]
```

Deci, varargin este un tablou de celule  $1 \times 3$  ale cărui elemente sunt matrice  $2 \times 2$  transmise lui companb ca argumente, iar varargin{j} este a  $j$ -a matrice de intrare,  $A_j$ . Nu este necesar ca varargin să fie singurul argument de intrare, dar dacă apare el trebuie să fie ultimul.

Analogul lui varargin pentru argumente de ieșire este varargout. În sursa MATLAB 1.5 este dat un exemplu care calculează momentele unui vector, până la un ordin dorit. Numărul de argumente de ieșire se determină cu nargout și apoi se crează tabloul de celule varargout ce conține ieșirea dorită. Ilustrăm cu apelurile funcției momente din sursa 1.5:

```
>> m1 = momente(1:4)
m1 =
    2.5000
>> [m1,m2,m3] = momente(1:4)
m1 =
    2.5000
m2 =
    7.5000
m3 =
    25
```

### Sursa MATLAB 1.5 Funcția momente

```
function varargout = momente(x)
%MOMENTE Momentele unui vector.
%           [m1,m2,...,m_k] = MOMENTE(X) returneaza momentele de
%           ordin 1, 2, ..., k ale vectorului X, unde momentul
%           de ordin j este SUM(X.^j)/LENGTH(X).

for j=1:nargout, varargout(j) = sum(x.^j)/length(x); end
```

## 1.4.5. Variabile globale

Variabilele din interiorul unei funcții sunt locale spațiului de lucru al acelei funcții. Uneori este convenabil să creăm variabile care există în mai multe spații de lucru, eventual chiar

cel principal. Aceasta se poate realiza cu ajutorul instrucțiunii `global`. Ca exemplu dăm codurile pentru funcțiile `tic` și `toc` (cu unele comentarii prescurtate). Aceste funcții pot contoriza timpul, gestionând un cronometru. Variabila globală `TICTOC` este vizibilă în ambele funcții, dar este invizibilă în spațiul de lucru de bază (nivel linie de comandă sau script) sau în orice altă funcție care nu o declară cu `global`.

```
function tic
%     TIC Start a stopwatch timer.
%         TIC; any stuff; TOC
%     prints the time required.
%     See also: TOC, CLOCK.
global TICTOC
TICTOC = clock;
function t = toc
%     TOC Read the stopwatch timer.
%     TOC prints the elapsed time since TIC was used.
%     t = TOC; saves elapsed time in t, does not print.
%     See also: TIC, ETIME.
global TICTOC
if nargout < 1
    elapsed_time = etime(clock,TICTOC)
else
    t = etime(clock,TICTOC);
end
```

În interiorul unei funcții, variabilele globale vor apărea înaintea primei apariții a unei variabile locale, ideal la începutul fișierului. Se convine ca numele de variabile globale să fie scrise cu litere mari, să fie lungi și sugestive.

#### 1.4.6. Recursivitate

Funcțiile pot fi recursive, adică ele se pot autoapela, direct sau indirect. Recursivitatea este un instrument puternic, deși nu toate calculele descrise în manieră recursivă pot fi implementate eficient în mod recursiv.

Funcția `koch` din sursa MATLAB 1.6 utilizează recursivitatea pentru a desena o curbă Koch și este inspirată din [32]. Construcția de bază este înlocuirea unui segment de dreaptă prin patru segmente mai scurte. Partea din stânga sus a figurii 1.1 arată rezultatul aplicării acestei construcții unei linii orizontale. Imaginea din dreapta jos ne arată ce se întâmplă când fiecare din aceste linii este prelucrată. Imaginea din stânga și dreapta jos ne arată următoarele două niveluri de recursivitate.

Funcția `koch` are trei argumente de intrare. Primele două, `pl` și `pr` dau coordonatele  $(x, y)$  ale capetelor segmentului curent și al treilea, `level`, indică nivelul de recursivitate cerut. Dacă `level = 0` se desenează un segment; altfel `koch` se autoapelează de patru ori cu `level` decrementat cu 1 și cu puncte care definesc capetele celor patru segmente mai scurte.

Figura 1.1 a fost obținută cu următorul cod:

```
pl=[0;0]; %left endpoint
pr=[1;0]; %right endpoint
```

**Sursa MATLAB 1.6 Funcția koch**


---

```

function koch(pl,pr,level)
%KOCH    Curba Koch generata recursiv.
%          Apel KOCH(PL, PR, LEVEL) unde punctele PL și PR
%          sunt extremitatea stângă și dreaptă
%          LEVEL este nivelul de recursivitate.

if level == 0
    plot([pl(1),pr(1)], [pl(2),pr(2)]); % Uneste pl si pr.
    hold on
else
    A = (sqrt(3)/6)*[0 1; -1 0];        % matrice rot./scal.

    pmidl = (2*pl + pr)/3;
    koch(pl,pmidl,level-1)                % ramura stanga

    ptop = (pl + pr)/2 + A*(pl-pr);
    koch(ptop,pmidl,level-1)              % ramura stanga-mijloc

    pmidr = (pl + 2*pr)/3;
    koch(ptop,pmidr,level-1)              % ramura mijloc

    koch(pmidr,pr,level-1)                % ramura dreapta
end

```

---

```

for k = 1:4
    subplot(2,2,k)
    koch(pl,pr,k)
    axis('equal')
    title(['Koch curve: level = ',num2str(k)],'FontSize',16)
end
hold off

```

Apelând `koch` cu perechi de puncte echidistante situate pe cercul unitate se obține o curbă numită fulgul de zăpadă al lui Koch (Koch's snowflake). Codul este dat în sursa 1.7. Funcția `snowflake` acceptă la intrare numărul de laturi `edges` și nivelul de recursivitate `level`. Valorile implicate ale acestor parametrii sunt 7 și respectiv 4. În figura 1.2 se dau două exemple.

Pentru alte exemple de recursivitate, a se vedea funcțiile MATLAB `quad` și `quadl` și sursele din secțiunile 7.4 și 7.6, utilizate pentru integrarea numerică.

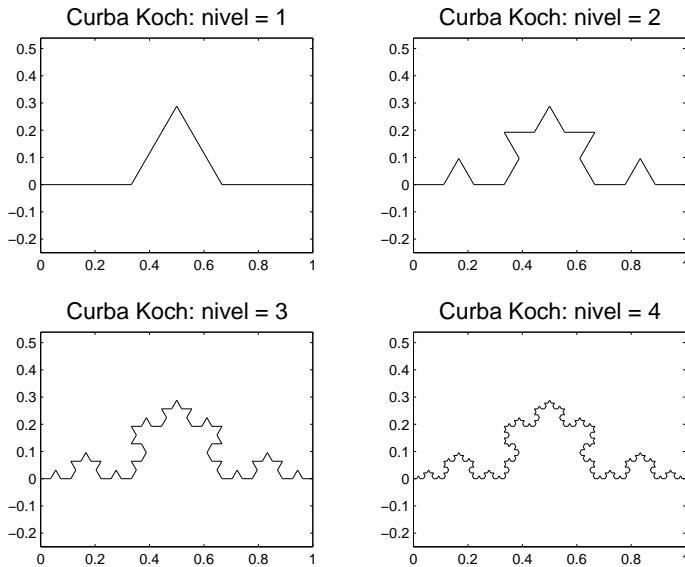


Figura 1.1: Curbe Koch create cu funcția koch.

---

### Sursa MATLAB 1.7 Fulgul lui Koch

---

```

function snowflake(edges,level)
if nargin<2, level=4; end
if nargin<1, edges=7; end

clf
for k = 1:edges
    pl = [cos(2*k*pi/edges); sin(2*k*pi/edges)];
    pr = [cos(2*(k+1)*pi/edges); sin(2*(k+1)*pi/edges)];
    koch(pl,pr,level);
end
axis('equal')
s=sprintf('Koch snowflake, level=%d, edges=%d',level, edges);
title(s,'FontSize',16,'FontAngle','italic')
hold off

```

---

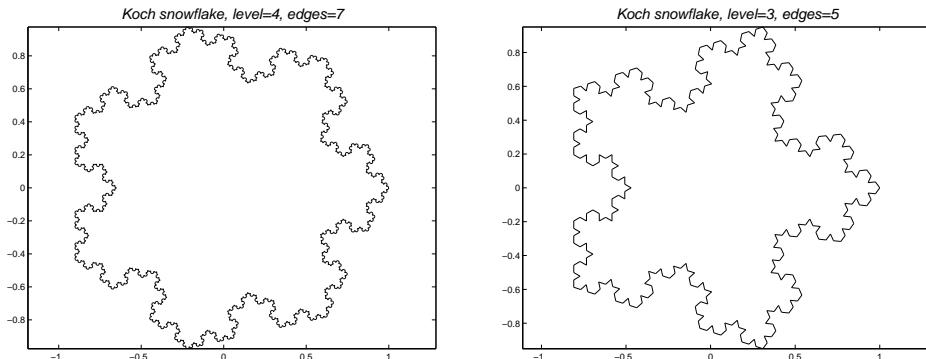


Figura 1.2: Fulgi Koch obținuți cu snowflakes

### 1.4.7. Alte tipuri numerice

Tipul de date implicit în MATLAB este tipul de date `double`. Pe lângă acesta, MATLAB furnizează și alte tipuri de date, având scopul în principal de a realiza economie de memorie. Acestea sunt

- `int8` și `uint8` – întregi pe 8 biți cu semn și fără semn;
- `int16` și `uint16` – întregi pe 16 biți cu semn și fără semn;
- `int32` și `uint32` – întregi pe 32 biți cu semn și fără semn;
- `single` – numere în virgulă flotantă simplă precizie (pe 32 de biți).

Funcțiile `eye`, `ones`, `zeros` pot returna date de ieșire de tipuri întregi sau `single`. De exemplu,

```
>> ones(2,2,'int8')
returnează o matrice  $2 \times 2$  cu elemente de tipul int8
```

```
ans =
1     1
1     1
```

Funcțiile care definesc tipuri de date întregi au același nume ca și tipul. De exemplu

```
x = int8(5);
```

atribuie lui `x` valoarea 5 reprezentată sub forma unui întreg pe 8 biți. Funcția `class` permite verificarea tipului unui rezultat.

```
>> class(x)
ans =
int8
```

Conversia unui număr de tip `double` la un tip întreg se face prin routunjire la cel mai apropiat întreg:

```
>> int8(2.7)
ans =
    3
>> int8(-2.5)
ans =
   -3
```

Funcțiile `intmax` și `intmin`, având ca argument un nume de tip întreg, returnează cea mai mare și respectiv cea mai mică valoare de acel tip:

```
>> intmax('int16')
ans =
    32767
>> intmin('int16')
ans =
   -32768
```

Dacă se încearcă conversia unui număr mai mare decât valoarea maximă a unui întreg de un anumit tip la acel tip, MATLAB returnează valoarea maximă (saturation on overflow).

```
>> int8(300)
ans =
    127
```

Analog, pentru valori mai mici decât valoarea minimă, se returnează valoarea minimă de acel tip.

Dacă se realizează operații aritmetice între întregi de același tip rezultatul este un întreg de acel tip. De exemplu

```
>> x=int16(5)+int16(9)
x =
    14
>> class(x)
ans =
int16
```

Dacă rezultatul este mai mare decât valoarea maximă de acel tip, MATLAB returnează valoarea maximă de acel tip. Analog, pentru un rezultat mai mic decât valoarea minimă, se returnează valoarea minimă de acel tip. În ambele situații se dă un mesaj de avertisment care se poate inhiba (sau reactiva) cu funcția `intwarning`.

Dacă A și B sunt tablouri de tip întreg, împărțirile în sens tablou,  $A ./ B$  și  $A .\ B$ , se realizează în aritmetică în dublă precizie, iar rezultatul se convertește la tipul întreg original, ca în exemplul

```
>> int8(4)./int8(3)
ans =
    1
```

Se pot combina în expresii scalari de tip double cu scalari sau tablouri de tip întreg, rezultatul fiind de tip întreg:

```
class(5*int8(3))
ans =
int8
```

Nu se pot combina scalari întregi sau tablouri de tip întreg cu scalari sau tablouri de un tip întreg diferit sau de tip `single`.

Pentru toate operațiile binare în care un operand este un tablou de tip întreg iar celălat este un scalar de tip `double`, MATLAB realizează operația element cu element în dublă precizie și convertește rezultatul în tipul întreg originar. De exemplu,

```
>> int8([1,2,3,4,5])*0.8
ans =
    1     2     2     3     4
```

De notat că: MATLAB calculează  $[1, 2, 3, 4, 5] * 0.8$  în dublă precizie și apoi convertește rezultatul în `int8`; al doilea și al treilea element din tablou după înmulțirea în dublă precizie sunt 1.6 și 2.4, care sunt routunjite la cel mai apropiat întreg, 2.

Tipul `single` va fi descris în §3.6.

Pentru detalii asupra tipurilor `nondouble` vezi [43, 44].

### 1.4.8. Controlul erorilor

Instrucțiunea `try` permite să se testeze dacă o anumită comandă a generat o eroare. Forma generală a instrucțiunii `try-catch` este

```
try
    instructiune
    ...
    instructiune
catch
    instructiune
    ...
    instructiune
end
```

Se execută instrucțiunile dintre `try` și `catch`. Dacă apare o eroare se execută instrucțiunile dintre `catch` și `end`. Această parte trebuie să trateze eroare într-un anumit mod. Blocurile `try-catch` se pot imbrica.

În MATLAB 7 funcția `error` se poate apela cu un mesaj de eroare cu format (ca în `sprintf`), după cum ne arată exemplul următor:

```
error('File %s not found\n', filename)
```

sau cu un identificator de mesaj

```
error('MATLAB:noSuchFile', 'File %s not found\n',...
filename)
```

Sursa ultimei erori se poate identifica cu `lasterr`, care returnează ultimul mesaj de eroare sau cu `lasterror`, care returnează o structură ce conține mesajul de eroare și identificatorul acestuia.

Exemplul următor determină cauza erorii la înmulțirea a două matrice utilizând `try-catch` și `lasterr`:

```
function matrixMultiply2(A, B)
try
    A * B
catch
    errmsg = lasterr;
    if(strfind(errmsg, 'Inner matrix dimensions'))
        disp('** Wrong dimensions for matrix multiply')
    else
        if(strfind(errmsg, 'not defined for values of class'))
            disp('** Both arguments must be double matrices')
        end
    end
end
```

Dacă dimensiunea matricelor este ilegală, se afișază primul mesaj:

```
>> A = [1 2 3; 6 7 2; 0 1 5];
>> B = [9 5 6; 0 4 9];
>> matrixMultiply2(A, B)
** Wrong dimensions for matrix multiply
```

iar dacă funcția este apelată cu un argument tablou de celule, al doilea mesaj:

```
>> C = {9 5 6; 0 4 9};
>> matrixMultiply2(A, C)
** Both arguments must be double matrices
```

Pentru detalii a se vedea `doc try` și [44].

## 1.5. Toolbox-urile Symbolic

Toolbox-urile Symbolic Math încorporează facilități de calcul simbolic în mediul numeric al MATLAB. Toolbox-urile se bazează pe nucleul Maple®. Există două toolbox-uri:

- Symbolic Math Toolbox care asigură accesul la nucleul Maple și la pachetul de algebră liniară al Maple utilizând o sintaxă și un stil care sunt extensii naturale ale limbajului MATLAB.
- Extended Symbolic Math Toolbox extinde facilitățile amintite mai sus pentru a asigura acces la facilitățile pachetelor negrafice Maple, facilitățile de programare și proceduri definite de utilizator.

Toolboxul Symbolic Math definește un nou tip de date MATLAB numit obiect simbolic sau `sym`. Intern, un obiect simbolic este o structură de date care memorează o reprezentare sub formă de sir a simbolului. Toolbox-ul Symbolic Math utilizează obiectele simbolice pentru a reprezenta variabile simbolice, expresii și matrice. Aritmetică cu care se operează asupra obiectelor simbolice este implicit cea rațională.

Obiectele simbolice se construiesc cu ajutorul declarației `sym`. De exemplu, instrucțiunea  
`x = sym('x');`  
produce o variabilă simbolică numită `x`. Se pot combina mai multe declarații de acest tip folosind forma `syms`:

```
syms a b c x y f g
```

Exemplul din această secțiune presupune că s-a executat această comandă.

**Derivare.** O expresie simbolică se poate deriva cu `diff`. Să creăm o expresie simbolică:

```
>> f=exp(a*x)*sin(x);
```

Derivata ei în raport cu `x` se obține astfel

```
>> diff_f=diff(f,x)
diff_f =
a*exp(a*x)*sin(x)+exp(a*x)*cos(x)
```

Dacă `n` este un întreg, `diff(f, x, n)` calculează derivata de ordinul `n` a lui `f`. De exemplu, pentru derivata de ordinul al doilea

```
>> diff(f,x,2)
ans = a^2*exp(a*x)*sin(x)+2*a*exp(a*x)*cos(x)-exp(a*x)*sin(x)
```

Pentru detalii suplimentare a se vedea `help sym/diff` sau `doc sym/diff`.

**Integrare.** Pentru a calcula primitiva unei expresii simbolice `f` se poate folosi `int(f, x)`.

Ca exemplu, să calculăm primitiva lui  $g(x) = e^{-ax} \sin cx$ :

```
>> g = exp(-a*x)*sin(c*x);
>> int_g=int(g,x)
int_g =
-c/(a^2+c^2)*exp(-a*x)*cos(c*x)-a/(a^2+c^2)*exp(-a*x)*sin(c*x)
```

Dând comanda `diff(int_g, x)` nu se obține `g` ci o expresie echivalentă, dar după simplificare cu comanda `simple(diff(int_g, x))` se obține un sir de mesaje care informează utilizatorul asupra regulilor utilizate și în final

```
ans = exp(-a*x)*sin(c*x)
```

Calculul  $\int_{-\pi}^{\pi} g dx$  se poate realiza cu comanda `int(g, x, -pi, pi)`. Rezultatul nu este foarte elegant. Vom calcula acum integrala  $\int_0^{\pi} x \sin x dx$ :

```
>> int('x*sin(x)',x,0,pi)
ans = pi
```

Dacă Maple nu poate găsi integrala sub forma analitică, ca în exemplul

```
>> int(exp(sin(x)),x,0,1)
Warning: Explicit integral could not be found.
ans =
int(exp(sin(x)),x = 0 .. 1)
```

se poate încerca o aproximare numerică:

```
>> quad('exp(sin(x))', 0, 1)
ans =
    1.6319
```

Pentru detalii suplimentare a se vedea `help sym/int` sau `doc sym/int`.

**Substituții și simplificări.** Înlocuirea unui parametru cu o valoare sau cu un alt parametru se realizează cu `subs`. De exemplu, să calculăm integrala definită a lui  $g$  de mai sus pentru  $a=2$  și  $c=4$ :

```
>> int_sub=subs(int_def_g, {a,c}, {2,4})
int_sub =
    107.0980
```

A se vedea `help sym/subs` sau `doc sym/subs`.

Funcția `simplify` este o funcție puternică care aplică diverse tipuri de identități pentru a aduce o expresie la o formă mai „simplă”. Exemplu:

```
>> syms h
>> h=(1-x^2)/(1-x);
>> simplify(h)
ans = x+1
```

Funcția `simple` este o funcție neortodoxă care are drept scop obținerea unei expresii echivalente care să aibă cel mai mic număr de caractere. Am dat mai sus un exemplu, dar mai considerăm unul:

```
>> [jj,how]=simple(cos(x)^2+sin(x)^2)
jj =
1
how =
simplify
```

Cel de-al doilea parametru de ieșire are rolul de a inhiba mesajele lungi referitoare la procesul de simplificare.

A se vedea `help sym/simplify` și `help sym/simple` sau `doc sym/simplify` și `doc sym/simple`.

**Serii Taylor.** Comanda `taylor` este utilă pentru a genera dezvoltări Taylor simbolice în jurul unei valori date a argumentului. Ca exemplu, să calculăm dezvoltarea de ordinul 5 a lui  $e^x$  în jurul lui  $x = 0$ :

```
>> clear, syms x, Tay_expx=taylor(exp(x),5,x,0)
Tay_expx =
1+x+1/2*x^2+1/6*x^3+1/24*x^4
```

Comanda `pretty` scrie o expresie simbolică într-un format apropiat de cel din matematică:

```
>> pretty(Tay_expx)
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4$$

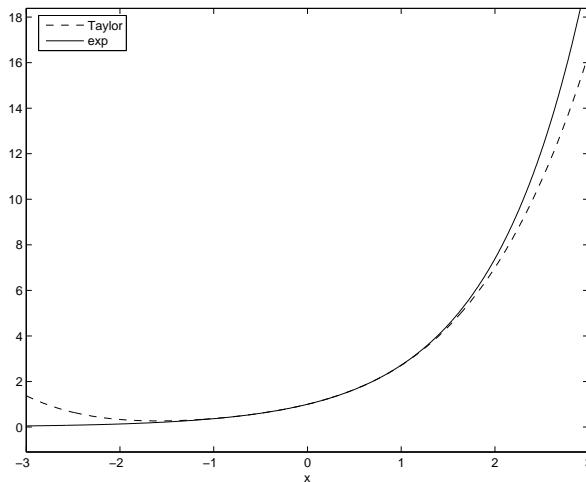


Figura 1.3: Comparație între exponențială și dezvoltarea sa Taylor

Să comparăm acum aproximanta pentru  $x = 2$  cu valoarea exactă:

```
>> approx=subs(Tay_expx,x,2), exact=exp(2)
approx =
    7
exact =
    7.3891
>> frac_err=abs(1-approx/exact)
frac_err =
    0.0527
```

Putem compara și grafic cele două aproximante cu `ezplot` (a se vedea capitolul 2 pentru o descriere a facilităților grafice din MATLAB):

```
>> ezplot(Tay_expx, [-3,3]), hold on
>> ezplot(exp(x), [-3,3]), hold off
>> legend('Taylor','exp','Location','Best')
```

Graficul apare în figura 1.3. Funcția `ezplot` este o modalitate convenabilă de a reprezenta grafic expresii simbolice.

Pentru detalii suplimentare a se vedea `help sym/taylor` sau `doc sym/taylor`.

**Limite.** Pentru sintaxa și modul de utilizare a comenzi `limit` a se vedea `help sym/limit` sau `doc sym/limit`. Ne vom limita la a da două exemple simple. Primul calculează  $\lim_{x \rightarrow 0} \frac{\sin x}{x}$ :

```
>> L=limit(sin(x)/x,x,0)
L =
    1
```

Al doilea exemplu calculează limitele laterale ale funcției tangentă în  $\frac{\pi}{2}$ .

```
>> LS=limit(tan(x),x,pi/2,'left')
LS =
Inf
>> LD=limit(tan(x),x,pi/2,'right')
LD =
-Inf
```

**Rezolvarea ecuațiilor.** Toolbox-ul Symbolic Math poate rezolva ecuații și sisteme de ecuații, inclusiv neliniare. A se vedea `help sym/solve` sau `doc sym/solve`. Vom da câteva exemple. Înainte de rezolvarea unei ecuații vom șterge memoria, vom defini simbolurile și ecuațiile. (Este o bună practică de a șterge memoria înainte de rezolvarea unei noi probleme pentru a evita efectele provocate de valorile precedente ale unor variabile.)

Vom începe cu rezolvarea ecuației de gradul al doilea  $ax^2 + bx + c = 0$ .

```
>> clear, syms x a b c
>> eq='a*x^2+b*x+c=0';
>> x=solve(eq,x)
x =
1/2/a*(-b+(b^2-4*a*c)^(1/2))
1/2/a*(-b-(b^2-4*a*c)^(1/2))
```

Este posibil ca la rezolvarea unei ecuații să se obțină mai multe soluții (în exemplul de mai sus s-au obținut două). Se poate selecta una concretă prin indexare, de exemplu `x(1)`.

Să rezolvăm acum sistemul liniar  $2x - 3y + 4z = 5$ ,  $y + 4z + x = 10$ ,  $-2z + 3x + 4y = 0$ .

```
>> clear, syms x y z
>> eq1='2*x-3*y+4*z=5';
>> eq2='y+4*z+x=10';
>> eq3=' -2*z+3*x+4*y=0';
>> [x,y,z]=solve(eq1,eq2,eq3,x,y,z)
x =
-5/37
y =
45/37
z =
165/74
```

De notat că ordinea în care se dau variabilele în lista de parametrii de intrare nu este importantă, pe când ordinea în care se dau variabilele de ieșire este importantă.

Exemplul următor rezolvă sistemul neliniar  $y = 2e^x$  și  $y = 3 - x^2$ .

```
>> clear, syms x y
>> eq1='y=2*exp(x)'; eq2='y=3-x^2';
>> [x,y]=solve(eq1,eq2,x,y)
x =
.36104234240225080888501262630700
y =
2.8696484269926958876157155521484
```

Ultimul exemplu rezolvă ecuația trigonometrică  $\sin x = \frac{1}{2}$ . Aceasta are o infinitate de soluții. Secvența de comenzi

```
>> clear, syms x, eq='sin(x)=1/2';
>> x=solve(eq,x)
```

dă doar soluția

```
x =
1/6*pi
```

Pentru soluțiile dintr-un anumit interval, de exemplu [2,3] se poate folosi comanda simbolică fsolve:

```
>> clear, x=maple('fsolve(sin(x)=1/2,x,2..3)')
x =
2.6179938779914943653855361527329
```

Rezultatul este un sir de caractere, care poate fi convertit în double cu str2double:

```
>> z=str2double(x), whos
z =
2.6180
Name      Size            Bytes  Class
ans       1x33           264  double array
x         1x33            66  char array
y         1x1              8   double array
z         1x1              8   double array
Grand total is 68 elements using 346 bytes
```

Funcția maple trimite comenzi Maple nucleului Maple. A se consulta help-urile corespunzătoare și documentația.

**Aritmetică cu precizie variabilă (vpa).** Există trei tipuri de operații aritmetice în toolbox:

- numeric – operațiile MATLAB în virgulă flotantă;
- rațional – aritmetică simbolică exactă Maple;
- VPA – aritmetică cu precizie variabilă Maple (variable precision arithmetic).

Aritmetică de precizie variabilă se realizează cu ajutorul funcției vpa. Numărul de cifre este controlat de variabila Maple Digits. Funcția digits afișează valoarea lui Digits, iar digits(n), unde n este întreg setează Digits la n cifre. Comanda vpa (E) evaluează E cu precizia Digits, iar vpa (E, n) evaluează E cu n cifre. Rezultatul este de tip sym.

De exemplu, instrucțiunile MATLAB

```
>> clear
>> format long
1/2+1/3
```

folosesc modul de calcul numeric pentru a produce

```
ans =
0.833333333333333
```

Cu toolbox-ul Symbolic Math, instrucțiunea

```
>> sym(1/2)+1/3
```

va produce, folosind calculul simbolic

```
ans =
5/6
```

Tot în toolbox, cu aritmetică cu precizie variabilă, instrucțiunile

```
>> digits(25)
>> vpa('1/2+1/3')
```

au ca rezultat

```
>> ans =
.833333333333333333333333
```

Pentru a converti un număr în precizie variabilă într-unul de tip `double` se poate folosi funcția `double`.

În exemplul următor

```
>> digits(32)
>> clear, phil=vpa((1+sqrt(5))/2)
phil =
1.6180339887498949025257388711907
>> phi2=vpa('(1+sqrt(5))/2'), diff=phil-phi2
phi2 =
1.6180339887498948482045868343656
diff =
.543211520368251e-16
```

discrepanța dintre `phil` și `phi2` se explică prin aceea că prima atribuire face calculele în dublă precizie și convertește rezultatul în `vpa`, iar a doua utilizează sirul și face toate calculele în `vpa`.

Pentru detalii suplimentare asupra Symbolic Math Toolbox trimitem cititorul la [42].

## Probleme

**Problema 1.1.** Calculați eficient suma

$$S_n = \sum_{k=1}^n \frac{1}{k^2},$$

pentru  $k = \overline{20, 200}$ . Cât de bine aproximează  $S_n$  suma seriei

$$S = \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}?$$

**Problema 1.2.** Scrieți un fișier M de tip funcție care evaluează dezvoltarea MacLaurin a funcției  $\ln(x + 1)$ :

$$\ln(x + 1) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + (-1)^{n+1} \frac{x^n}{n} + \cdots$$

Convergența are loc pentru  $x \in [-1, 1]$ . Testați funcția MATLAB pentru valori ale lui  $x$  din  $[-0.5, 0.5]$  și verificați ce se întâmplă când  $x$  se apropie de -1 sau 1.

**Problema 1.3.** Scrieți un script MATLAB care citește un întreg și determină scrierea sa cu cifre romane.

**Problema 1.4.** Implementați algoritmul lui Euclid în MATLAB.

**Problema 1.5.** Implementați în MATLAB căutarea binară într-un tablou ordonat.

**Problema 1.6.** Scrieți cod MATLAB care crează, pentru o valoare dată a lui  $n$ , matricea tridiagonală

$$B_n = \begin{bmatrix} 1 & n & & & \\ -2 & 2 & n-1 & & \\ & -3 & 3 & n-2 & \\ & & \ddots & \ddots & \ddots \\ & & & -n+1 & n-1 & 2 \\ & & & & -n & n \end{bmatrix}.$$

**Problema 1.7.** Care este cea mai mare valoare a lui  $n$  cu proprietatea că

$$S_n = \sum_{k=1}^n k^2 < L,$$

unde  $L$  este dat? Rezolvați prin însumare și utilizând formula care dă pe  $S_n$ .

**Problema 1.8.** Generați matricea  $H_n = (h_{ij})$ , unde

$$h_{ij} = \frac{1}{i+j-1}, \quad i, j = \overline{1, n},$$

folosind toolbox-ul Symbolic.

**Problema 1.9.** Să se genereze matricea triunghiulară a coeficienților binomiali, pentru puteri mergând de la 1 la un  $n \in \mathbb{N}$  dat.

**Problema 1.10.** Scrieți o funcție MATLAB care primește la intrare coordonatele vârfurilor unui triunghi și subdivizează recursiv triunghiul în patru triunghiuri, după mijloacele laturilor. Subdivizarea continuă când se atinge un nivel dat.

**Problema 1.11.** Scrieți o funcție MATLAB care să extragă dintr-o matrice dată  $A$  o parte care să fie diagonală pe blocuri, unde blocurile sunt de dimensiune dată, iar colțul din stânga sus al fiecărui bloc este situat pe diagonala principală.



# CAPITOLUL 2

---

## Grafică în MATLAB

---

MATLAB are facilități grafice puternice și versatile. Se pot genera grafice și figuri relativ ușor, iar atributele lor se pot modifica cu ușurință. Nu ne propunem să fim exhaustivi, ci dorim doar să introducем cititorul în facilitățile grafice MATLAB care vor fi necesare în continuare. Figurile existente pot fi modificate ușor cu utilitarul Plot Editor. Pentru utilizarea sa a se vedea `help plotedit` și meniul Tools sau bara din ferestrele figurilor. Celor care doresc detalii sau să aprofundeze subiectul le recomandăm [32, 40, 45, 48].

### 2.1. Grafice bidimensionale

#### 2.1.1. Grafice de bază

Funcția MATLAB `plot` realizează grafice bidimensionale simple unind punctele vecine. Tastând

```
>>x=[1.5,2.2,3.1,4.6,5.7,6.3,9.4];  
>>y=[2.3,3.9,4.3,7.2,4.5,3.8,1.1];  
>>plot(x,y)
```

se obține imaginea din stânga figurii 2.1(a), în care punctele  $x(i)$ ,  $y(i)$  sunt unite în secvență. MATLAB deschide o fereastră pentru figură (dacă una nu a fost deja deschisă ca rezultat al unei comenzi precedente) în care desenează imaginea. În acest exemplu se utilizează valori implicate ale unor facilități cum ar fi domeniul pentru axele  $x$  și  $y$ , spațiile dintre diviziunile de pe axe, culoarea și tipul liniei.

Mai general, în loc de `plot(x,y)`, putem utiliza `plot(x,y,sir)`, unde  $sir$  este un sir de caractere ce controlează culoarea, marcajul și stilul de linie. De exemplu, `plot(x,y,'r*--')` ne spune că în fiecare punct  $x(i)$ ,  $y(i)$  se va plasa un asterisc

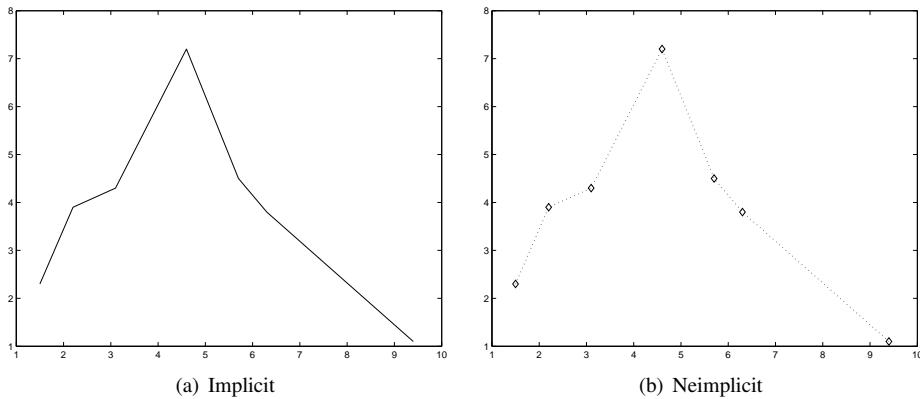


Figura 2.1: Grafice  $x$ - $y$  simple

roșu, punctele fiind unite cu o linie roșie întreruptă. `plot(x, y, ' +y')` marchează punctele cu un plus galben, fără a le uni cu nici o linie. Tabela 2.1 dă toate opțiunile disponibile. Imaginea din partea dreaptă a figurii 2.1 s-a obținut cu `plot(x, y, 'kd:')`, care desenează o linie punctată neagră marcată cu romburi. Cele trei elemente din *șir* pot apărea în orice ordine; de exemplu, `plot(x, y, 'ms--')` și `plot(x, y, 's--m')` sunt echivalente. De notat că `plot` acceptă mai multe seturi de date. De exemplu,

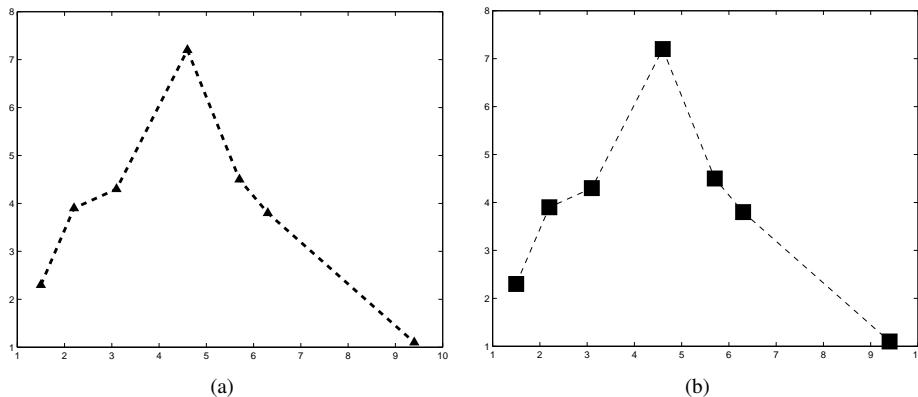
```
plot(x,y,'g-',b,c,'r--')
```

desenează în aceeași figură graficele pentru  $x(i)$ ,  $y(i)$  și  $b(i)$ ,  $c(i)$  cu linie continuă verde și respectiv cu linie întretreruptă roșie.

Culoare	Marcaj	Stil de linie
r	cerc	- continuă (implicit)
g	asteric	-- întreruptă
b	punct	:
+	plus	-.
x	ori	
s	pătrat	
d	romb	
^	triunghi în sus	
v	triunghi în jos	
>	triunghi dreapta	
<	triunghi stânga	
p	pentagramă (stea cu 5 colțuri)	
h	hexagramă (stea cu 6 colțuri)	

Tabela 2.1: Opțiuni pentru comanda plot

Comanda `plot` acceptă și argumente matriciale. Dacă  $x$  este un vector de dimensiune  $m$  și  $Y$  este o matrice  $m \times n$ , `plot(x, Y)` suprapune graficele obținute din  $x$  și fiecare coloană

Figura 2.2: Două grafice  $x$ - $y$  neimplicite

a lui  $\mathbb{Y}$ . Similar, dacă  $X$  și  $\mathbb{Y}$  sunt matrice de aceeași dimensiune,  $\text{plot}(X, \mathbb{Y})$  suprapune graficele obținute din coloanele corespunzătoare ale lui  $X$  și  $\mathbb{Y}$ . Dacă argumentele lui  $\text{plot}$  nu sunt reale, atunci părțile imaginare sunt în general ignorate. Singura excepție este atunci când  $\text{plot}$  este apelat cu un singur argument. Dacă  $\mathbb{Y}$  este complex,  $\text{plot}(\mathbb{Y})$  este echivalent cu  $\text{plot}(\text{real}(\mathbb{Y}), \text{imag}(\mathbb{Y}))$ . În cazul când  $\mathbb{Y}$  este real,  $\text{plot}(\mathbb{Y})$  desenează graficul obținut luând pe abscisă indicii punctelor și pe ordonată  $\mathbb{Y}$ .

Atributele se pot controla furnizând argumente suplimentare lui  $\text{plot}$ . Proprietățile `LineWidth` (implicit 0.5 puncte) și `MarkerSize` (implicit 6 puncte) pot fi specificate în puncte, unde un punct este 1/72 inch. De exemplu, comanda

```
>>plot(x,y,'m--^','LineWidth',3,'MarkerSize',5)
```

produce un grafic cu linie cu lățimea 3 puncte și marcaje cu dimensiunea 5 puncte. Culoarea laturilor marcajului și a interiorului marcajului se poate seta pe una din culorile din tabela 2.1 cu proprietățile `MarkerEdgeColor` și `MarkerFaceColor`. Astfel, de exemplu

```
plot(x,y,'o','MarkerEdgeColor','m')
```

colorează cercurile (nu și interiorul) în magenta. Graficul din stânga figurii 2.2 s-a obținut cu

```
plot(x,y,'m--^','LineWidth',3,'MarkerSize',5)
```

iar cel din dreapta cu comanda

```
plot(x,y,'--rs','MarkerSize',20,'MarkerFaceColor','g')
```

Funcția `loglog`, spre deosebire de `plot`, scalează axele logarithmic. Această facilitate este utilă pentru a reprezenta relații de tip putere sub forma unei drepte. În continuare vom reprezenta graficul restului Taylor de ordinul al doilea  $|1 + h + h^2/2 - \exp(h)|$  al lui  $\exp(h)$  pentru  $h = 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$ . Când  $h$  este mic, această cantitate se comportă ca un multiplu al lui  $h^3$  și deci pe o scară log-log valorile vor fi situate pe o dreaptă cu pantă 3. Vom verifica aceasta reprezentând restul și dreapta de referință cu pantă prevăzută cu linie punctată. Graficul apare în figura 2.3

```

h=10.^[0:-1:-4];
taylerr=abs((1+h+h.^2/2)-exp(h));
loglog(h,taylerr,'-',h,h.^3,'--')
 xlabel('h'), ylabel('|eroare|')
title('Eroarea in aproximarea Taylor de grad 2 a lui exp(h)')
box off

```

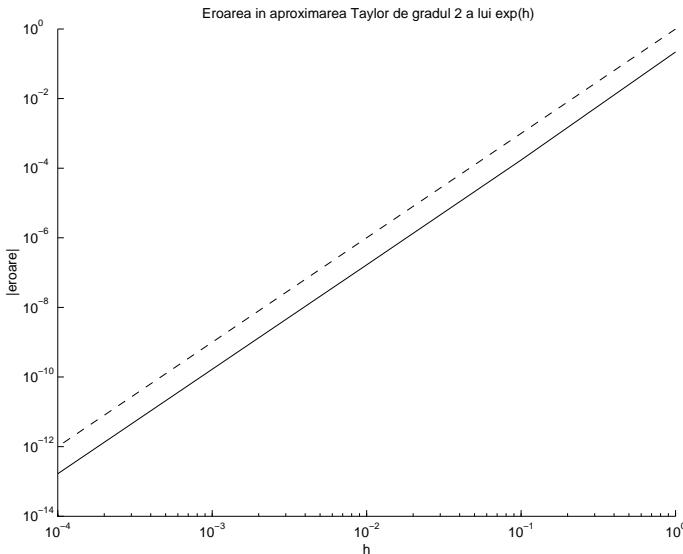


Figura 2.3: Exemplu cu `loglog`

În acest exemplu s-au utilizat comenziile `title`, `xlabel` și `ylabel`. Aceste funcții afișează sirul parametru de intrare deasupra imaginii, axei  $x$  și respectiv axei  $y$ . Comanda `box off` elimină caseta de pe marginea graficului curent, lăsând doar axe de coordonate. Dacă `loglog` primește și valori nepozitive, MATLAB va da un avertisment și va afișa doar datele pozitive. Funcțiile înrudite `semilogx` și `semilogy`, scalează doar una din axe.

Dacă o comandă de afișare este urmată de alta, atunci noua imagine o va înlocui pe cea veche sau se va suprapune peste ea, depinzând de starea `hold` curentă. Comanda `hold on` face ca toate imaginile care urmează să se suprapună peste cea curentă, în timp ce `hold off` ne spune că fiecare imagine nouă o va înlocui pe cea precedentă. Starea implicită corespunde lui `hold off`.

Se pot reprezenta curbe în coordonate polare cu ajutorul comenzi `polar(t, r)`, unde  $t$  este unghiul polar, iar  $r$  este raza polară. Se poate folosi și un parametru suplimentar  $s$ , cu aceeași semnificație ca la `plot`. Graficul unei curbe în coordonate polare, numită cardioidă, și care are ecuația

$$r = a(1 + \cos t), \quad t \in [0, 2\pi],$$

unde  $a$  este o constantă reală dată, apare în figura 2.4 și se obține cu secvența:

```
t=0:pi/50:2*pi;
```

```
a=2; r=a*(1+cos(t));
polar(t,r)
title('cardioida')
```

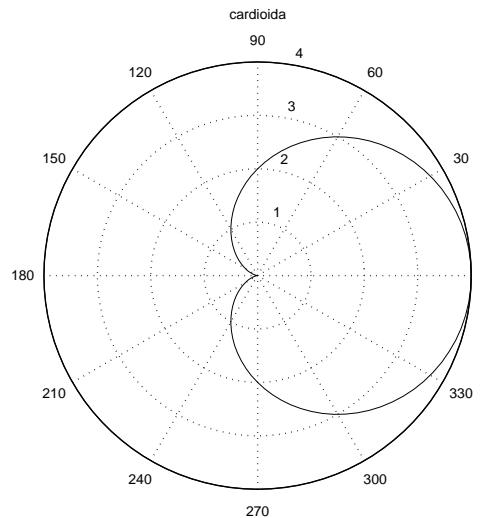


Figura 2.4: Grafic în coordonate polare — cardioidă

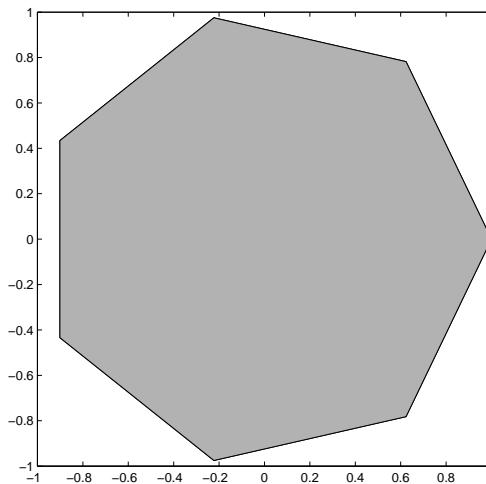
Funcția `fill` lucrează la fel ca `plot`. Comanda `fill(x, y, c)` reprezintă poligonul cu vârfurile  $x(i)$ ,  $y(i)$  în culoarea  $c$ . Punctele se iau în ordine și ultimul se unește cu primul. Culoarea  $c$  se poate da și sub forma unui triplet RGB,  $[r \ g \ b]$ . Elementele  $r$ ,  $g$  și  $b$ , care trebuie să fie scalari din  $[0, 1]$ , determină nivelul de roșu, verde și albastru din culoare. Astfel, `fill(x, y, [0 1 0])` umple poligonul cu culoarea verde, iar `fill(x, y, [1 0 1])` cu magenta. Dând proporții egale de roșu, verde și albastru se obțin nuanțe de gri care variază de la negru ( $[0 0 0]$ ) la alb ( $[1 1 1]$ ). Exemplul următor desenează un heptagon regulat în gri:

```
n=7;
t=2*(0:n-1)*pi/n;
fill(cos(t),sin(t),[0.7,0.7,0.7])
axis square
```

Rezultatul apare în figura 2.5.

Comanda `clf` șterge figura curentă, iar `close` o închide. Este posibil să avem mai multe ferestre figuri pe ecran. Cel mai simplu mod de a crea o nouă figură este comanda `figure`. A n-a fereastră figură (unde  $n$  apare în bara de titlu) poate fi făcută figură curentă cu comanda `figure(n)`. Comanda `close all` va închide toate ferestrele figuri.

De notat că multe atrbute ale unei figuri pot fi modificate interactiv, după afișarea figurii, utilizând meniul Tool al ferestrei sau bara de instrumente (toolbar). În particular, este posibil să se facă zoom pe o regiune particulară cu ajutorul mouse-ului (vezi `help zoom`).

Figura 2.5: Exemplu de utilizare `fill`

<code>axis([xmin xmax ymin ymax])</code>	Setează limitele axelor $x$ și $y$
<code>axis auto</code>	Returnează limitele implicate
<code>axis equal</code>	Egalează unitățile pe axe de coordonate
<code>axis off</code>	Elimină axe
<code>axis square</code>	Face caseta axelor pătrată (cubică)
<code>axis tight</code>	Setează limitele axelor egale cu limitele datelor
<code>xlim([xmin xmax])</code>	Setează limitele pe axa $x$
<code>ylim([ymin,ymax])</code>	Setează limitele pe axa $y$

Tabela 2.2: Unele comenzi pentru controlul axelor

## 2.1.2. Axe și adnotarea

Diversele aspecte ale unui grafic pot fi controlate cu comanda `axis`. Unele opțiuni se dau în tabela 2.2. Axele pot fi eliminate cu `axis off`. Raportul dintre unitatea pe  $x$  și cea pe  $y$  (aspect ratio) poate fi făcut egal cu unu, astfel ca cercurile să nu pară elipse, cu `axis equal`. Comanda `axis square` face caseta axelor pătrată.

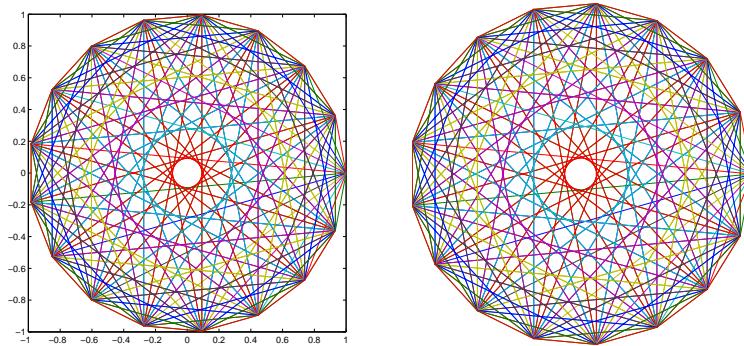
Graficul din stânga figurii 2.6 a fost obținut cu

```
plot(fft(eye(17))), axis equal, axis square
```

Deoarece figura este situată în interiorul cercului unitate, axele sunt foarte necesare. Graficul din dreapta figurii 2.6 a fost generat cu

```
plot(fft(eye(17))), axis equal, axis off
```

Comanda `axis([xmin xmax ymin ymax])` setează limitele pentru axa  $x$  și respectiv  $y$ . Pentru a reveni la setările implicate, pe care MATLAB le alege automat în funcție

Figura 2.6: Utilizarea `axis off`

de datele care urmează a fi reprezentate, se utilizează `axis auto`. Dacă se dorește ca una dintre limite să fie aleasă automat de către MATLAB, ea se ia `-inf` sau `inf`; de exemplu, `axis([-1,1,-inf,0])`. Limitele pe axa  $x$  sau  $y$  se pot seta individual cu `xlim([xmin xmax])` și `ylim([ymin ymax])`.

Exemplul următor reprezintă funcția  $1/(x-1)^2 + 3/(x-2)^2$  pe intervalul  $[0,3]$ :

```
x = linspace(0, 3, 500);
plot(x, 1./ (x-1) .^ 2+3 ./ (x-2) .^ 2)
grid on
```

Comanda `grid on` produce o grilă de linii orizontale și verticale care pornesc de la diviziunile axelor. Rezultatul se poate vedea în figura 2.7(a). Datorită singularităților din  $x = 1, 2$  graficul nu dă prea multă informație. Totuși, executând comanda

```
ylim([0,50])
```

se obține figura 2.7(b), care se focalizează asupra părții interesante a primului grafic.

Exemplul următor reprezintă epicicloida

$$\left. \begin{array}{l} x(t) = (a+b) \cos(t) - b \cos((a/b+1)t) \\ y(t) = (a+b) \sin(t) - b \sin((a/b+1)t) \end{array} \right\} \quad 0 \leq t \leq 10\pi,$$

pentru  $a = 12$  și  $b = 5$ .

```
a = 12; b=5;
t=0:0.05:10*pi;
x = (a+b)*cos(t)-b*cos((a/b+1)*t);
y =(a+b)*sin(t)-b*sin((a/b+1)*t);
plot(x,y)
axis equal
axis([-25 25 -25 25])
grid on
title('epicicloida: a=12, b=5')
xlabel('x(t)'),
ylabel('y(t)')
```

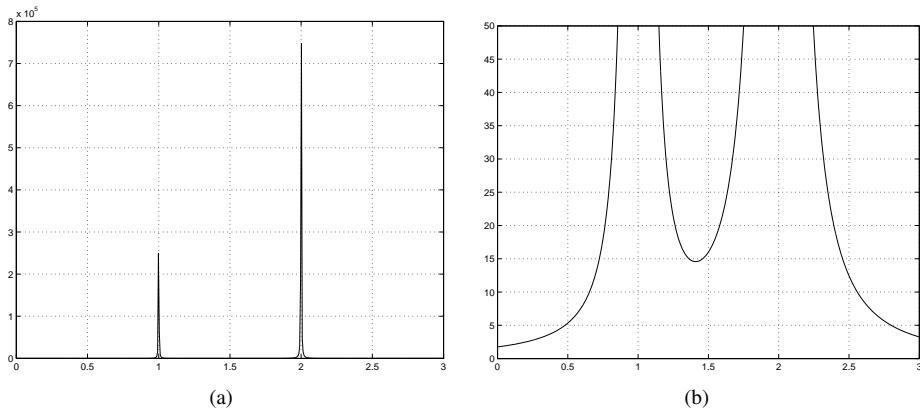


Figura 2.7: Utilizarea lui `ylim` (dreapta) pentru a schimba limitele automate pe axa  $y$  (stânga).

Rezultatul apare în figura 2.8. Limitele din axis au fost alese astfel ca să rămână un oarecare spațiu în jurul epicicloidei.

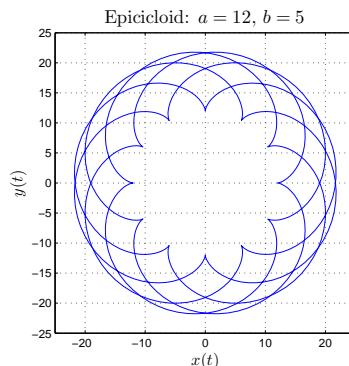


Figura 2.8: Epicicloidă

Comanda `legend('string1','string2',...,'stringn',pp)` va ataşa unui grafic o legendă care pune 'stringi' după informația culoare/marcaj/stil pentru graficul corespunzător. Parametrul optional `pp` indică poziția legendei (vezi `help legend`). Exemplul care urmează adaugă o legendă unui grafic al sinusului și cosinusului (figura 2.9):

```

x = -pi:pi/20:pi;
plot(x,cos(x),'-ro',x,sin(x),'-.b')
h = legend('cos','sin',2);

```

Textele se pot include în grafice cu ajutorul comenzi `text(x, y, s)`, unde `x` și `y` sunt coordonatele textului, iar `s` este un sir de caractere sau o variabilă de tip sir. Începând cu ver-

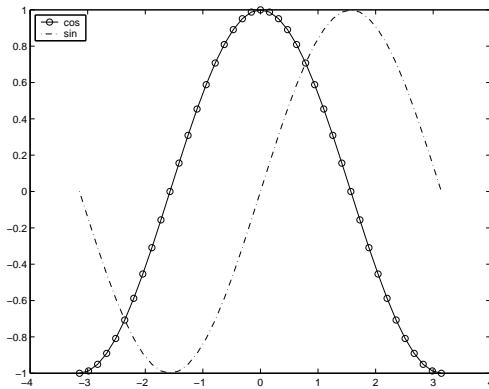


Figura 2.9: Grafic cu legendă

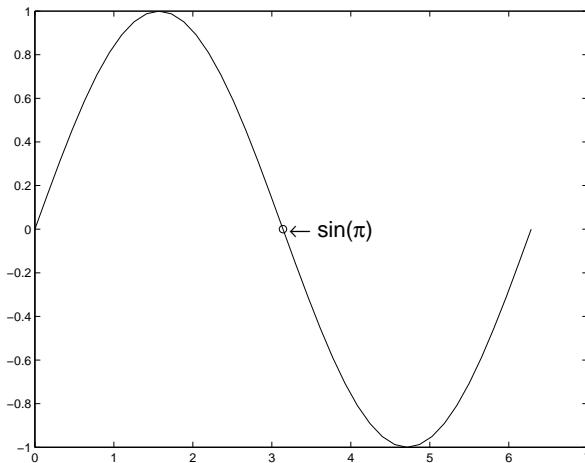


Figura 2.10: Exemplu de utilizare text

siunea 5, MATLAB permite introducerea în interiorul parametrului `s` a unor construcții  $\text{\TeX}$ , de exemplu – pentru indice,  $^{\wedge}$  pentru exponent, sau litere grecești (`\alpha`, `\beta`, `\gamma`, etc.). De asemenea, anumite atrbute ale textului, cum ar fi tipul font-ului, dimensiunea și altele sunt selectabile începând cu versiunea 4. Comenzile

```
plot(0:pi/20:2*pi,sin(0:pi/20:2*pi),pi,0,'o')
text(pi,0,' \leftarrow sin(\pi)','FontSize',18)
```

adnotează punctul de coordonate  $(\pi, 0)$  cu sirul  $\sin(\pi)$ . Rezultatul apare în figura 2.10. Aceste facilități se pot utiliza și în titluri, legende sau etichete ale axelor, care sunt obiecte de tip text. Începând cu MATLAB 7 primitivele text suportă un subset puternic  $\text{\LaTeX}$ . Proprietatea corespunzătoare se numește `Interpreter` și poate avea valorile `TeX`, `LaTeX` sau `none`. Pentru un exemplu de utilizare a macrourilor  $\text{\LaTeX}$  a se vedea script-ul `graphLegendre.m`,

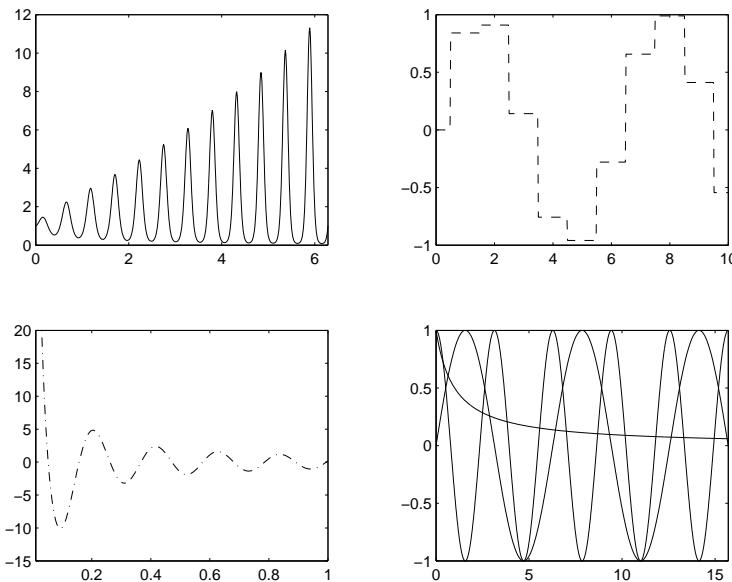


Figura 2.11: Exemple cu subplot și fplot

pagina 167.

### 2.1.3. Mai multe grafice pe aceeași figură

Funcția MATLAB `subplot` permite plasarea mai multor imagini pe o grilă în aceeași figură. Utilizând `subplot(mnp)`, sau echivalent, `subplot(m, n, p)`, fereastra figurii se împarte într-un tablou  $m \times n$  de regiuni, fiecare având propriile ei axe. Comanda de desenare curentă se va aplica celei de-a  $p$ -a dintre aceste regiuni, unde contorul variază de-a lungul primei linii, apoi de-a lungul celei de-a două și a.m.d. De exemplu, `subplot(425)` împarte fereastra figurii într-o matrice  $4 \times 2$  de regiuni și ne spune că toate comenziile de desenare se vor aplica celei de-a cincea regiuni, adică primei regiuni din al treilea rând. Dacă se execută mai târziu `subplot(427)`, atunci poziția (4,1) devine activă. Vom da în continuare mai multe exemple.

Pentru cei nefamiliarizați cu grafica MATLAB, comanda de reprezentare grafică a unei funcții, `fplot`, este foarte utilă. Ea alege în mod adaptiv un număr suficient de puncte pentru a produce un grafic suficient de precis. Exemplul următor generează graficele din figura 2.11.

```
subplot(221), fplot('exp(sqrt(x))*sin(12*x)', [0 2*pi])
subplot(222), fplot('sin(round(x))',[0,10], '--')
subplot(223), fplot('cos(30*x)/x',[0.01 1 -15 20], '-.')
subplot(224)
fplot(['sin(x),cos(2*x),1/(1+x)'], [0 5*pi -1.5 -1.5 1.5])
```

În acest exemplu primul apel al lui `fplot` produce graficul funcției  $\exp(\sqrt{x} \sin 12x)$  pe intervalul  $0 \leq x \leq 2\pi$ . În al doilea apel se selectează stilul de linie întreruptă cu `'--'`, în

locul celei implicate, continue. Argumentul `[0.01 1 -15 20]` din cel de-al treilea apel setează limitele pe axele  $x$  și  $y$  la  $0.01 \leq x \leq 1$  și respectiv  $-15 \leq y \leq 20$ , iar `'-'` utilizează stilul de linie linie-punct. Ultimul exemplu arată cum se pot reprezenta mai multe funcții la un singur apel.

Sintaxa generală a lui `fplot` este

```
fplot(fun,lims,tol,N,'LineSpec',p1,p2,...)
```

cu semnificația:

- `fun` este funcția de reprezentat (function handle, obiect inline sau expresie);
- `lims` dă limitele pe axele  $x$  și/sau  $y$ ;
- `tol` este eroarea relativă (implicit  $2 \times 10^{-3}$ );
- se folosesc cel puțin  $N+1$  puncte la generarea graficului;
- `LineSpec` determină tipul de linie;
- `p1, p2, ...` sunt parametrii adiționali transmiși lui `fun`, care trebuie să aibă parametrii de intrare `x, p1, p2, ...`.

Este posibil să se obțină grile neregulate de imagini apelând `subplot` cu şablonane de grile diferite. Figura 2.12 a fost generată cu:

```
x = linspace(0,15,100);
subplot(2,2,1), plot(x,sin(x))
subplot(2,2,2), plot(x,round(x))
subplot(2,1,2), plot(x,sin(round(x)))
```

Al treilea argument al lui `subplot` poate fi un vector ce specifică mai multe regiuni; ultima linie se poate înlocui cu

```
subplot(2,2,3:4), plot(x,sin(round(x)))
```

În încheierea acestei secțiuni, tabela 2.3 dă cele mai populare funcții MATLAB pentru grafice 2D. Funcțiile cu numele de forma `ezfun` sunt variante ușor de utilizat (easy) ale funcțiilor `fun` corespunzătoare.

## 2.2. Grafice tridimensionale

Funcția `plot3` este un analog tridimensional al lui `plot`. Figura 2.13 a fost obținută cu:

```
t = -5:0.005:5;
x = (1+t.^2).*sin(20*t);
y = (1+t.^2).*cos(20*t);
z=t;
plot3(x,y,z)
grid on
xlabel('x(t)'), ylabel('y(t)'), zlabel('z(t)')
title('{\it Exemplu} plot3','FontSize',14)
```

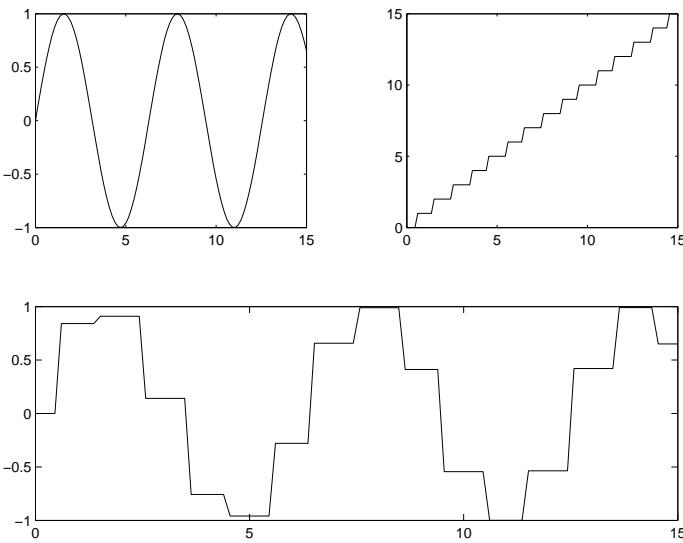


Figura 2.12: Grile neregulate produse cu subplot

<code>plot</code>	grafic $x$ - $y$ simplu
<code>loglog</code>	grafic cu scară logaritmică pe ambele axe
<code>semilogx</code>	grafic cu scară logaritmică pe axa $x$
<code>semilogy</code>	grafic cu scară logaritmică pe axa $y$
<code>plotyy</code>	grafic $x$ - $y$ cu axe $y$ și al stânga și la dreapta
<code>polar</code>	grafic polar
<code>fplot</code>	reprezentare grafică automată a unei funcții
<code>ezplot</code>	versiune ușor de utilizat (easy-to-use) a lui <code>fplot</code>
<code>ezpolar</code>	versiune ușor de utilizat (easy-to-use) a lui <code>polar</code>
<code>fill</code>	umplere poligon
<code>area</code>	grafic de tip arie plină
<code>bar</code>	grafic de tip bară
<code>barh</code>	grafic de tip bară orizontală
<code>hist</code>	histogramă
<code>pie</code>	grafic cu sectoare de cerc
<code>comet</code>	grafic $x$ - $y$ animat
<code>errorbar</code>	grafic cu bare de eroare
<code>quiver</code>	câmp de vectori bidimensional
<code>scatter</code>	Grafic dispersat (nor de puncte)

Tabela 2.3: Funcții pentru grafice 2D

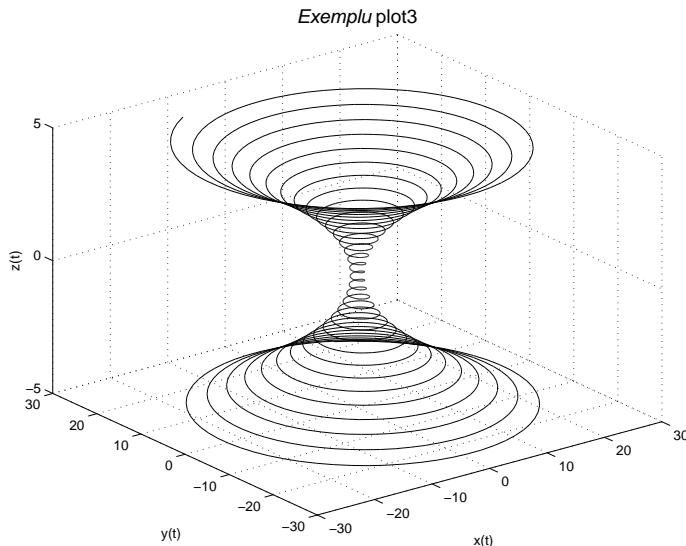


Figura 2.13: Grafic 3D creat cu plot3

Acest exemplu utilizează funcțiile `xlabel`, `ylabel` și `title`, precum și analogul lor `zlabel`. De notat și utilizarea notației  $\text{\TeX}$  în titlu, pentru a produce text italic. Culoarea, marcajul și stilul de linie pentru `plot3` se controlează la fel ca pentru `plot`. Limitele de axe în spațiu tridimensional se determină automat, dar ele pot fi schimbatе cu

`axis([xmin, xmax, ymin, ymax, zmin, zmax])`

În afară de `xlim` și `ylim`, există și `zlim`, prin care se pot schimba limitele pe axa  $z$ .

O facilitate ușor de utilizat de desenare a contururilor este oferită de `ezcontour`. Apelul lui `ezcontour` în exemplul următor produce contururi pentru funcția  $\sin(3y - x^2 + 1) + \cos(2y^2 - 2x)$  pe domeniul dat de  $-2 \leq x \leq 2$  și  $-1 \leq y \leq 1$ ; rezultatul se poate vedea în jumătatea de sus a figurii 2.14.

```
subplot(211)
ezcontour('sin(3*y-x.^2+1)+cos(2*y.^2-2*x)', [-2, 2, -1, 1]);
%
x=-2:.01:2; y=-1:0.01:1;
[X, Y] = meshgrid(x, y);
Z = sin(3*Y-X.^2+1)+cos(2*Y.^2-2*X);
subplot(212)
contour(x, y, Z, 20)
```

De notat că nivelurile de contur au fost alese automat. Pentru jumătatea de jos a figurii 2.14 s-a utilizat funcția `contour`. Întâi se fac inițializările  $x = -2 : .01 : 2$  și  $y = -1 : .01 : 1$  pentru a obține puncte mai apropiate în domeniul respectiv. Apoi se execută  $[X, Y] = \text{meshgrid}(x, y)$ , care obține matricele  $X$  și  $Y$  astfel încât fiecare linie a lui  $X$  să fie o copie a lui  $x$  și fiecare coloană a lui  $Y$  să fie o copie a vectorului  $y$ . (Funcția `meshgrid` este foarte utilă la pregătirea datelor pentru multe funcții MATLAB de grafică 3D.) Matricea

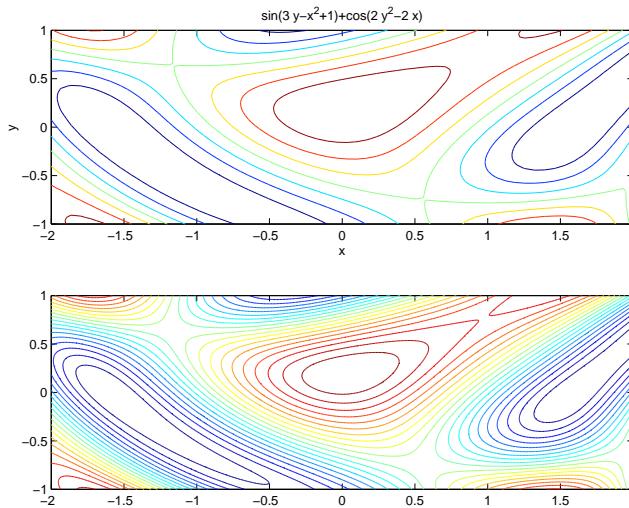


Figura 2.14: Contururi obținute cu ezcontour (sus) și contour (jos).

---

### Sursa MATLAB 2.1 Reprezentarea grafică a unei funcții implice

---

```

xm=-3:0.2:3; ym=-2:0.2:1;
[x,y]=meshgrid(xm,ym);
f=y.^3+exp(y)-tanh(x);
contour(x,y,f,[0,0],'k-')
xlabel('x'); ylabel('y');
title('y^3+e^y=tanh(x)','FontSize',14)

```

---

$Z$  este apoi generată prin operații de tip tablou din  $X$  și  $Y$ ;  $Z(i, j)$  memorează valoarea funcției corespunzând lui  $x(j)$  și  $y(i)$ . Aceasta este forma cerută de `contour`. Apelul `contour(x, y, Z, 20)` spune MATLAB să privească  $Z$  ca fiind formată din cote deasupra planului  $xOy$  cu spațierea dată de  $x$  și  $y$ . Ultimul argument de intrare spune că se vor utiliza 20 de niveluri de contur; dacă acest argument este omis, MATLAB va alege automat numărul de niveluri de contur.

Funcția `contour` se poate utiliza și la reprezentarea funcțiilor implice cum ar fi

$$y^3 + \exp(y) = \tanh(x).$$

Pentru a o reprezenta grafic, rescriem ecuația sub forma

$$f(x, y) = y^3 + \exp(y) - \tanh(x)$$

și desenăm conturul pentru  $f(x, y) = 0$  (vezi script-ul 2.1 și figura 2.15).

Pentru aplicații în mecanică ale funcției `contour` vezi [38].

Funcția `mesh` acceptă date în aceeași formă ca și `contour` și produce o reprezentare de suprafață de tip cadru de sârmă (wire-frame). Funcția `meshc` se deosebește de `mesh`

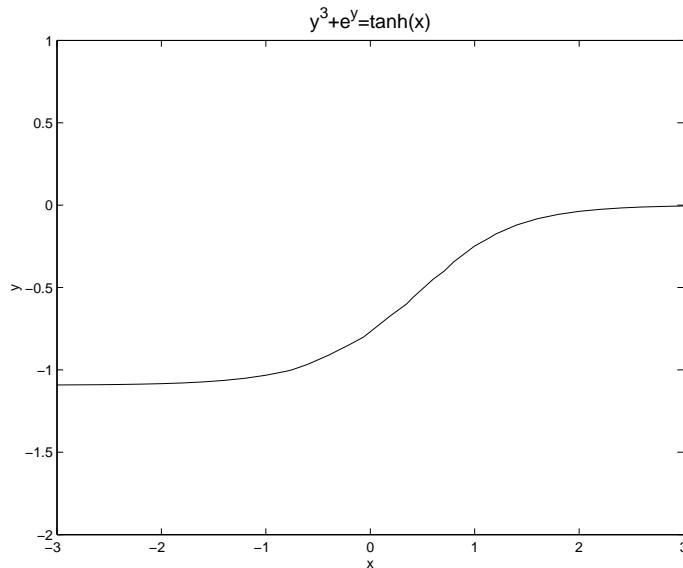


Figura 2.15: Curbă dată printr-o funcție implicită

prin aceea că adaugă un grafic de tip contur dedesubtul suprafeței. Exemplul de mai jos, care produce figura 2.16, lucrează cu suprafața definită de  $\sin(y^2 + x) - \cos(y - x^2)$  pentru  $0 \leq x, y \leq \pi$ . Primul grafic este generat cu subplot și mesh (Z). Deoarece nu se dă nici o informație pentru abscisă și ordonată, mesh utilizează în locul lor indicii de linie și de coloană. A doua imagine arată efectul lui meshc (Z). Pentru cea de-a treia, s-a utilizat mesh (x, y, Z), și deci gradațiile de pe axele x și y corespund valorilor x și y. Limitele pe axe s-au specificat cu axis ([0 pi 0 pi -5 5]). Pentru ultima imagine s-a utilizat din nou mesh (Z), urmată de hidden off, care inhibă afișarea liniilor ascunse.

```
x = 0:0.1:pi; y=0:0.1:pi;
[X, Y]=meshgrid(x,y);
Z=sin(Y.^2+X)-cos(Y-X.^2);
subplot(221)
mesh(Z)
subplot(222)
meshc(Z)
subplot(223)
mesh(x,y,Z)
axis([0 pi 0 pi -5 5])
subplot(2,2,4)
mesh(Z)
hidden off
```

Funcția surf diferă de mesh prin aceea că produce un grafic de suprafață cu celulele umplute (colorate), iar surfc adaugă contururi dedesubt. În exemplul următor am folosit

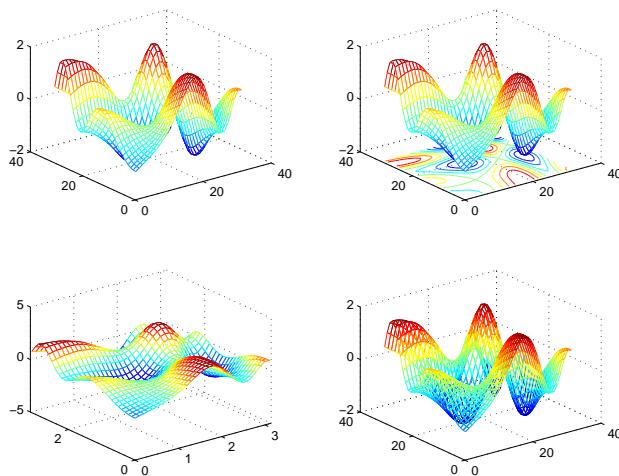


Figura 2.16: Grafice obținute cu mesh și meshc

funcția MATLAB membrane, care returnează funcția proprie a unei membrane în formă de L. Imaginele de pe prima linie a figurii 2.17 arată efectele lui surf și surfc. În ele apare o scară de culori, realizată utilizând colormap. Harta de culori curentă se poate seta cu colormap; vezi doc colormap. A treia imagine (poziția (2,1)) utilizează funcția shading cu opțiunea flat pentru a elimina liniile grilei de pe suprafață; o altă opțiune este interp, care selectează culorile prin interpolare. A patra imagine (poziția (2,2)) utilizează funcția înrudită waterfall, care este similară lui mesh, dar fără cadrul de sărmă pe direcția coloanelor.

```
Z=membrane; FS = 'FontSize';
subplot(2,2,1), surf(Z), title('\bf{surf}',FS,14), colorbar
subplot(2,2,2), surfc(Z), title('\bf{surfc}',FS,14), colorbar
subplot(2,2,3), surf(Z), shading flat
    title('\bf{surf} shading flat',FS,14)
subplot(2,2,4), waterfall(Z), title('\bf{waterfall}',FS,14)
```

Graficele tridimensionale din figurile 2.13, 2.16 și 2.17 utilizează unghiurile de vizualizare implicate ale MATLAB. Acestea pot fi modificate cu view. Apelul view(a,b) alege unghiul de rotație în sens invers acelor de ceasornic în jurul axei  $z$  (azimutul) de  $a$  grade și unghiul față de planul  $xOy$  (elevația) de  $b$  grade. Implicit este view(-37.5, 30). Instrumentul rotate 3D de pe bara de instrumente a figurii fereastră permite utilizarea mouse-ului pentru schimbarea unghiurilor de vedere.

Este posibil să vedem un grafic 2D ca pe unul 3D, utilizând comanda view pentru a da unghiurile de vedere, sau mai simplu utilizând view(3). Figura 2.18 a fost obținută tastând

```
plot(fft(eye(17))); view(3); grid
```

Tabela 2.4 dă un rezumat al celor mai populare funcții pentru grafice 3D. Așa cum indică tabela, unele funcții au și variante “easy to use” al căror nume începe cu ez.

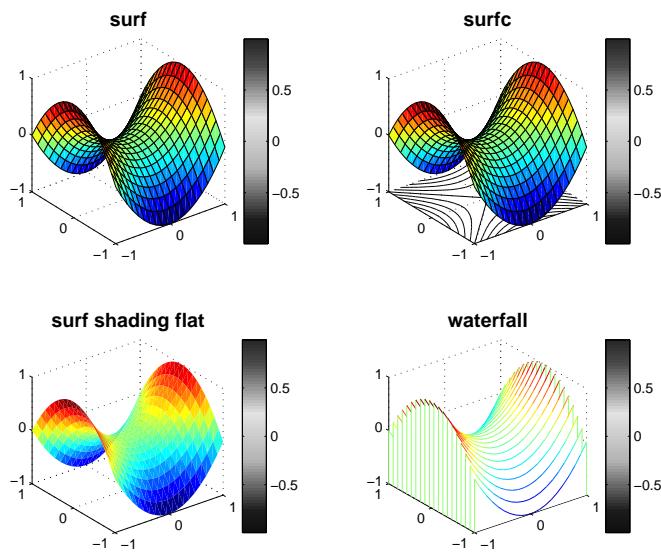


Figura 2.17: Suprafete desenate cu `surf`, `surfc` și `waterfall`

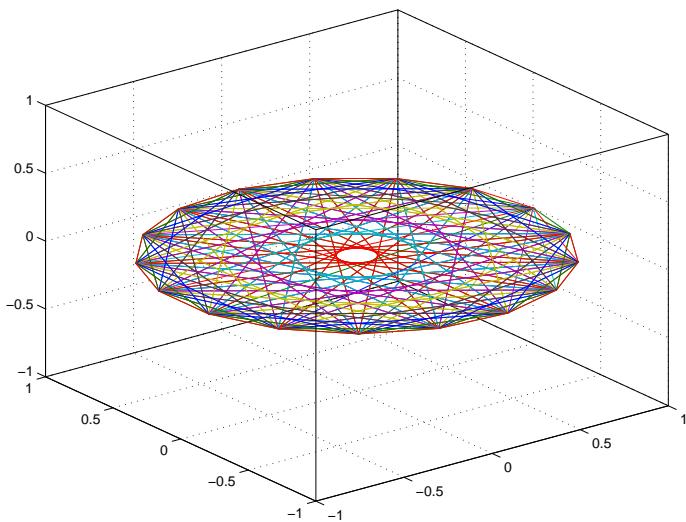


Figura 2.18: Vedere 3D a unei imagini 2D.

plot3*	grafic simplu $x-y-z$
contour*	contur
contourf*	contur plin
contour3	contur 3D
mesh*	reprezentare wire-frame
meshc*	reprezentare wire-frame plus contururi
meshz	suprafață wire-frame cu cortină
surf*	suprafață plină
surfC*	suprafață plină plus contururi
waterfall	wire-frame unidirecțional
bar3	bare 3D
bar3h	bare 3D orizontale
pie3	grafice sector 3D
fill3	poligon umplut tridimensional
comet3	curbă 3D animatedă
scatter3	nor de puncte 3D

Aceste funcții fun au corespondent e z fun

Tabela 2.4: Funcții grafice 3D

O trăsătură comună tuturor funcțiilor grafice este aceea că valorile NaN sunt interpretate ca „date lipsă” și nu sunt reprezentate. De exemplu,

`plot([1 2 NaN 3 4])`  
desenează două linii disjuncte și nu unește punctele 2 și 3, în timp ce

`A=peaks(80); A(28:52,28:52)=NaN; surfC(A)`

produce graficul `surfC` cu gaură din figura 2.19. (Funcția `peaks` din MATLAB are expresia

```
z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
- 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
- 1/3*exp(-(x+1).^2 - y.^2)
```

și generează o matrice de cote utilă pentru a testa și demonstra facilitățile grafice 3D.)

## 2.3. Salvarea și imprimarea graficelor

Comanda `print` permite listarea unui grafic la imprimantă sau salvarea lui pe disc într-un format grafic sau sub formă de fișier M. Formatul ei este:

`print -dperiferic -optiuni numefisier`  
Ea are mai multe opțiuni, care pot fi vizualizate cu `help print`. Dintre tipurile de periferice admise amintim:

- `dps` - Postscript pentru imprimante alb-negru;
- `dpsc` - Postscript pentru imprimante color;

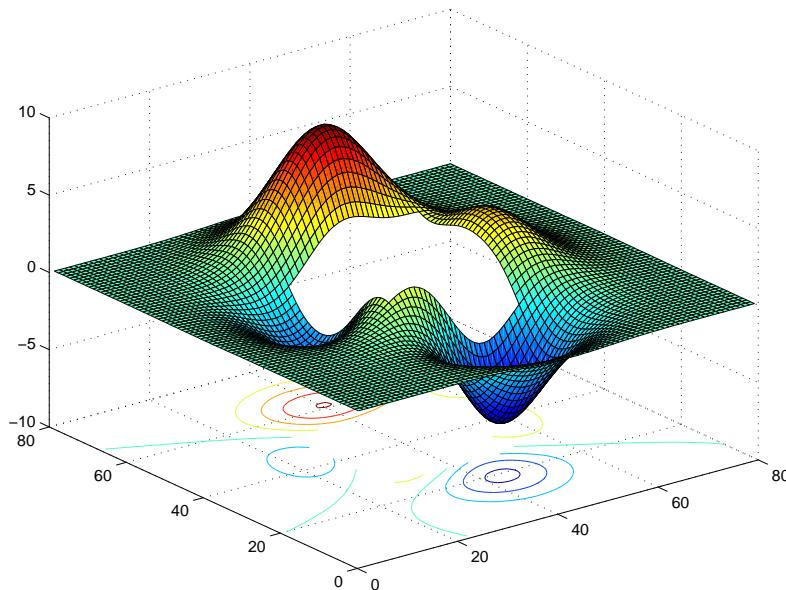


Figura 2.19: Grafic `surf` al unei matrice ce conține NaN-uri.

- `dps2` - Postscript nivelul 2 pentru imprimante alb-negru;
- `dpsc2` - PostScript nivelul 2 pentru imprimante color;
- `deps` - Encapsulated PostScript pentru imprimante alb-negru;
- `depsc` - Encapsulated PostScript pentru imprimante color;
- `deps2` - Encapsulated PostScript nivelul 2 pentru imprimante alb-negru;
- `depsc2` - Encapsulated PostScript nivelul 2 pentru imprimante color;
- `djpeg` - <nn> - imagine JPEG la nivel de calitate nn (implicit nn=75).

Dacă imprimanta dumneavoastră este setată corespunzător, comanda `print` va trimite conținutul figurii curente spre ea. Comanda

```
print -deps2 myfig.eps
```

crează un fișier Postscript încapsulat alb și negru, nivelul 2, numit `myfig.eps`, care poate fi listat pe o imprimantă PostScript sau inclus într-un document. Acest fișier poate fi încorporat într-un document L<sup>A</sup>T<sub>E</sub>X, aşa cum se schițează mai jos:

```
\documentclass{article}
\usepackage[dvips]{graphics}

...
\begin{document}
```

```

...
\begin{figure}
\begin{center}
\includegraphics[width=8cm]{myfig.eps}
\end{center}
\caption{...}
\end{figure}
...
\end{document}

```

Comanda `print` se poate utiliza și în formă funcțională (vezi secțiunea 1.4.2, pagina 29 pentru dualitatea comenzi/funcții). Pentru a ilustra utilitatea formei funcționale, exemplul următor generează o secvență de cinci figuri și le salvează în fișierele `fig1.eps`, ..., `fig5.eps`:

```

x = linspace(0,2*pi,50);
for i=1:5
    plot(x,sin(i*x))
    print('-deps2',['fig',int2str(i),'.eps'])
end

```

Al doilea argument al comenzi `print` este format prin concatenare, utilizând funcția `int2str`, care convertește un întreg în sir. Astfel, de exemplu, pentru `i=1`, instrucțiunea `print` este echivalentă cu `print('-deps2', 'fig1.eps')`.

Comanda `saveas` salvează o figură într-un fișier care apoi poate fi încărcat de către MATLAB. De exemplu,

```
saveas(gcf,'myfig','fig')
```

salvează figura curentă în format binar FIG, care poate fi încărcat în MATLAB cu comanda `open('myfig.fig')`.

Se pot salva și imprima figuri din meniul `File` al ferestrei figurii.

## 2.4. Facilități grafice noi în MATLAB 7

În MATLAB 7 s-au introdus instrumente grafice noi și s-au perfecționat unele facilități:

- noi interfețe grafice interactive pentru crearea și modificarea graficelor fără a introduce cod M;
- generare de cod M dintr-o figură, permitând reutilizarea graficelor din program;
- facilități îmbunătățite de adnotare a graficelor, inclusiv desenarea de figuri geometrice, aliniere, legare a adnotărilor de puncte;
- Instrumente de explorare a datelor;
- posibilitatea de a aplica transformări cum ar fi rotații, translații și scalări asupra grupurilor de obiecte grafice;

- panouri de interfețe utilizator și controale ActiveX accesibile din GUIDE;
- suport Handle Graphics® îmbunătățit și suport pentru afișarea unor ecuații cu interfață completă T<sub>E</sub>X sau L<sub>A</sub>T<sub>E</sub>X.

## Probleme

**Problema 2.1 (Curba Lissajous (Bodwitch)).** Să se reprezinte curba parametrică:

$$\alpha(t) = (a \sin(nt + c), b \sin t), \quad t \in [0, 2\pi],$$

pentru (a)  $a = 2, b = 3, c = 1, n = 2$ , (b)  $a = 5, b = 7, c = 9, n = 4$ , (c)  $a = b = c = 1, n = 10$ .

**Problema 2.2 (Curba fluture).** Să se reprezinte curba polară

$$r(t) = e^{\cos(t)} - a \cos(bt) + \sin^5(ct),$$

pentru valorile (a)  $a = 2, b = 4, c = 1/12$ , (b)  $a = 1, b = 2, c = 1/4$ , (c)  $a = 3, b = 1, c = 1/2$ . Experimentați pentru  $t$  în intervale de forma  $[0, 2k\pi]$ ,  $k \in \mathbb{N}$ .

**Problema 2.3 (Rodoneea sferică).** Să se reprezinte grafic curba tridimensională definită prin

$$\alpha(t) = a(\sin(nt) \cos t, \sin(nt) \sin t, \cos(nu)),$$

pentru valorile (a)  $a = n = 2$ , (b)  $a = 1/2, n = 1$ , (c)  $a = 3, n = 1/4$ , (d)  $a = 1/3, n = 5$ .

**Problema 2.4 (Sfera răsucită (Corkscrew)).** Să se reprezinte grafic suprafața parametrică dată prin

$$\chi(u, v) = (a \cos u \cos v, a \sin u \cos v, a \sin v + bu)$$

unde  $(u, v) \in [0, 2\pi] \times [-\pi, \pi]$ , pentru (a)  $a = b = 1$ , (b)  $a = 3, b = 1$ , (c)  $a = 1, b = 0$ , (d)  $a = 1, b = -3/2$ .

**Problema 2.5.** Să se reprezinte curba dată implicit

$$b^2 y^2 = x^3(a - x),$$

pentru (a)  $a = 1, b = 2$ , (b)  $a = b = 1$ , (c)  $a = 2, b = 1$ .

**Problema 2.6 (Elicoid).** Să se reprezinte grafic suprafața parametrică dată prin

$$\chi(u, v) = (av \cos u, bv \sin u, cu + ev)$$

unde  $(u, v) \in [0, 2\pi] \times [-d, d]$ , pentru (a)  $a = 2, b = c = 1, e = 0$ , (b)  $a = 3, b = 1, c = 2, e = 1$ .



## CAPITOLUL 3

---

### Elemente de Teoria erorilor și aritmetică în virgulă flotantă

---

Aprecierea preciziei rezultatelor calculelor este un obiectiv important în Analiza numerică. Se disting mai multe tipuri de erori care pot limita această precizie:

1. erori în datele de intrare;
2. erori de rotunjire;
3. erori de aproximare.

Erorile în datele de intrare sunt în afara (dincolo de) controlului calculelor. Ele se pot datora, de exemplu, imperfecțiunilor inerente ale măsurătorilor fizice.

Erorile de rotunjire apar dacă se fac calcule cu numere a căror reprezentare se restrâng la un număr finit de cifre, aşa cum se întâmplă de obicei.

Pentru al treilea tip de erori, multe metode nu dau soluția exactă a problemei date  $P$ , chiar dacă calculele se fac fără rotunjire, ci mai degrabă soluția unei alte probleme mai simple  $\tilde{P}$ , care aproximează  $P$ . De exemplu, problema  $P$  a însumării unei serii infinite:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

poate fi înlocuită cu problema mai simplă  $\tilde{P}$  de a însuma numai un număr finit de termeni ai seriei. Eroarea de aproximare astfel obținută se numește *eroare de trunchiere* (totuși, acest termen este de asemenea utilizat pentru erorile de rotunjire comise prin ștergerea ultimelor cifre ale reprezentării). Multe probleme de aproximare se obțin prin „discretizarea“ problemei originale  $P$ : integralele definite se aproximează prin sume finite, derivatele prin diferențe, etc. În astfel de cazuri, eroarea de aproximare se numește *eroare de discretizare*. Unii autori extind termenul de „eroare de trunchiere“ pentru a acoperi și eroarea de discretizare.

În acest capitol vom examina efectul general al erorilor de intrare și de rotunjire asupra rezultatelor unui calcul. Erorile de aproximare vor fi discutate în capitolele următoare când vom trata metodele numerice individuale.

### 3.1. Probleme numerice

Combinarea dintre o problemă matematică (PM), (de natură constructivă) și specificațiile de precizie ale rezultatului (SP) se numește *problemă numerică*.

**Exemplul 3.1.1.** Fie  $f : \mathbb{R} \rightarrow \mathbb{R}$  și  $x \in \mathbb{R}$ . Dorim să calculăm  $y = f(x)$ . În general  $x$  nu este reprezentabil în calculator; din acest motiv vom lucra cu o aproximare  $x^*$  a sa,  $x^* \approx x$ . De asemenea este posibil ca  $f$  să nu poată fi calculată exact; vom înlocui  $f$  cu o aproximantă a sa  $f_A$ . Valoarea calculată în calculator va fi  $f_A(x^*)$ . Deci problema numerică este următoarea:

**PM.** dându-se  $x$  și  $f$ , să se calculeze  $f(x)$ ;

**SP.**  $|f(x) - f_A(x^*)| < \varepsilon$ ,  $\varepsilon$  dat. ◊

Problemele numerice aparțin uneia din următoarele categorii:

**Evaluarea unei funcționale:**  $l : \mathcal{F} \rightarrow \mathbb{R}$ , cum ar fi, de exemplu, calcularea valorii unei funcții  $f(x)$ , a derivatelor  $f'(x), f''(x), \dots$  (derivare numerică), a integralelor definite  $\int_a^b f(t)dt$  (integrare numerică) și a normelor  $\|f\|_p$ , etc.

**Rezolvarea ecuațiilor algebrice:** determinarea valorilor unor necunoscute aflate în relații algebrice prin rezolvarea unor sisteme de ecuații liniare sau neliniare.

**Rezolvarea unor ecuații analitice:** determinarea funcțiilor (sau valorilor de funcții) soluției ale unei ecuații operatoriale, cum ar fi ecuațiile diferențiale ordinare sau cu derivate parțiale, ecuațiile integrale, ecuații funcționale, etc.

**Probleme de optimizare:** determinarea unor valori numerice particulare ale unor funcții, care optimizează (minimizează sau maximizează) o funcție obiectiv, cu restricții sau fără restricții.

Problemele matematice constructive, din care provin problemele numerice, pot fi privite ca o aplicație abstractă  $F : X \rightarrow Y$  între două spații liniare normate.

În funcție de care dintre cantitățile  $y, x$  sau  $F$  este necunoscută în ecuația

$$Fx = y,$$

avem de-a face cu o *problemă directă*, o *problemă inversă* sau o *problemă de identificare*:

	$F$	$x$	$y$
problemă directă	dată	dat	<b>dorit</b>
problemă inversă	dată	<b>dorit</b>	dat
problemă de identificare	<b>dorită</b>	dat	dat

**Exemplul 3.1.2 (Integrare).** Evaluarea unei integrale definite

$$Fx = \int_a^b x(t)dt$$

este o *problemă directă*. În această problemă  $F : \mathcal{F} \rightarrow \mathbb{R}$  este o funcțională care aplică, de exemplu  $\mathcal{F} = C[a, b]$  pe  $\mathbb{R}$ . Integrandul – în acest caz o funcție  $x$  definită pe  $[a, b]$  – este dat, iar integrala  $y = Fx$  trebuie determinată.  $\diamond$

**Exemplul 3.1.3 (Sistem de ecuații liniare).** Rezolvarea sistemului de ecuații liniare

$$Ax = b, \quad A \in \mathbb{R}^{n \times n}, \quad x, b \in \mathbb{R}^n$$

este o *problemă inversă* tipică. Aplicația liniară  $F$  este caracterizată de  $n^2$  coeficienți (elementele lui  $A$ ). Vectorul  $b$  (îmaginea) este dat, iar vectorul  $x$  (care este transformat în  $b$ ) trebuie determinat.  $\diamond$

**Exemplul 3.1.4 (Analiza unei mixturi).** Trebuie determinată compoziția unei mixturi de substanțe care se evaporă. Presupunând că în unitatea de timp se evaporă o cantitate fixată din fiecare substanță, se poate utiliza următorul model dependent de timp al mixturii

$$y(t) = \sum_{i=1}^m y_{i_0} e^{-k_i t}. \quad (3.1.1)$$

În afară de cantitățile inițiale  $y_{i_0}$ ,  $i = 1, 2, \dots, m$  și de ratele de evaporare  $k_1, \dots, k_m$  ale substanțelor necunoscute, numărul  $m$  de substanțe este de asemenea necunoscut. Dacă la momentele  $t_1, \dots, t_n$  se măsoară cantitățile  $y_1, \dots, y_n$ , trebuie determinați parametrii  $y_{i_0}$ ,  $k_i$ ,  $i = \overline{1, m}$  și  $m$  ai aplicației

$$F : (t_1, \dots, t_n) \mapsto (y_1, \dots, y_n) = \left( \sum_{i=1}^m y_{i_0} e^{-k_i t_1}, \dots, \sum_{i=1}^m y_{i_0} e^{-k_i t_n} \right);$$

deci avem o *problemă de identificare*. Pentru orice  $m \in \{1, 2, \dots, n/2\}$  (cel mult  $2 \cdot n/2 = n$  necunoscute în acest caz) se poate încerca minimizarea distanței între funcția model (3.1.1) și valorile date, care poate fi definită prin

$$r_m = \sum_{j=1}^n \left( y_j - \sum_{i=1}^m y_{i_0} e^{-k_i t_j} \right)^2. \quad (3.1.2)$$

Aceasta este o problemă inversă extrem de dificilă (problemă de estimație neliniară). Minimul trebuie determinat folosind tehnici de optimizare cu restricții. Valoarea reziduului minim  $r_m^*$  face posibilă determinarea valorii lui  $m$  care poate fi utilizată pentru cea mai bună adaptare a modelului la datele de intrare. Totuși, aceasta nu înseamnă minimizarea lui  $r_m$  fără restricții asupra lui  $m$ , ci mai degrabă a nu alege  $m$  mai mare decât este necesar, în scopul de a obține modelul cel mai simplu posibil.  $\diamond$

## 3.2. Măsuri ale erorii

**Definiția 3.2.1.** Fie  $X$  un spațiu liniar normat,  $A \subseteq X$  și  $x \in X$ . Un element  $x^* \in A$  se numește aproximantă a lui  $x$  din  $A$  (notație  $x^* \approx x$ ).

**Definiția 3.2.2.** Dacă  $x^*$  este o aproximantă a lui  $x$  diferența  $\Delta x = x - x^*$  se numește eroare, iar

$$\|\Delta x\| = \|x^* - x\| \quad (3.2.1)$$

se numește eroare absolută.

**Definiția 3.2.3.** Expresia

$$\delta x = \frac{\|\Delta x\|}{\|x\|}, \quad x \neq 0 \quad (3.2.2)$$

se numește eroare relativă.

**Observația 3.2.4.**

1. Deoarece în practică  $x$  este necunoscut, se folosește aproximarea  $\delta x = \frac{\|\Delta x\|}{\|x^*\|}$ . Dacă  $\|\Delta x\|$  este mic comparativ cu  $x^*$ , atunci aproximanta este bună.
2. Dacă  $X = \mathbb{R}$ , se lucrează cu  $\delta x = \frac{\Delta x}{x}$  și  $\Delta x = x^* - x$ . ◊

### 3.3. Eroarea propagată

Fie  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $x = (x_1, \dots, x_n)$  și  $x^* = (x_1^*, \dots, x_n^*)$ . Dorim să evaluăm eroarea absolută și relativă  $\Delta f$  și respectiv  $\delta f$  când se aproximează  $f(x)$  prin  $f(x^*)$ . Aceste erori se numesc *erori propagate*, deoarece ne spun cum se propagă eroarea inițială (absolută sau relativă) pe parcursul calculării lui  $f$ . Să presupunem că  $x = x^* + \Delta x$ , unde  $\Delta x = (\Delta x_1, \dots, \Delta x_n)$ . Pentru eroarea absolută avem (folosind formula lui Taylor)

$$\begin{aligned} \Delta f &= f(x_1^* + \Delta x_1, \dots, x_n^* + \Delta x_n) - f(x_1^*, \dots, x_n^*) = \\ &= \sum_{i=1}^n \Delta x_i \frac{\partial f}{\partial x_i^*}(x_1^*, \dots, x_n^*) + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \Delta x_i \Delta x_j \frac{\partial^2 f}{\partial x_i^* \partial x_j^*}(\theta), \end{aligned}$$

unde  $\theta \in [(x_1^*, \dots, x_n^*), (x_1^* + \Delta x_1, \dots, x_n^* + \Delta x_n)]$ .

Dacă  $\Delta x_i$  sunt suficient de mici, atunci  $\Delta x_i \Delta x_j$  sunt neglijabile comparativ cu  $\Delta x_i$  și obținem

$$\Delta f \approx \sum_{i=1}^n \Delta x_i \frac{\partial f}{\partial x_i^*}(x_1^*, \dots, x_n^*). \quad (3.3.1)$$

Pentru eroarea relativă avem

$$\begin{aligned} \delta f &= \frac{\Delta f}{f} \approx \sum_{i=1}^n \Delta x_i \frac{\frac{\partial f}{\partial x_i^*}(x^*)}{f(x^*)} = \sum_{i=1}^n \Delta x_i \frac{\partial}{\partial x_i^*} \ln f(x^*) = \\ &= \sum_{i=1}^n x_i^* \delta x_i \frac{\partial}{\partial x_i^*} \ln f(x^*). \end{aligned}$$

Deci

$$\delta f = \sum_{i=1}^n x_i^* \frac{\partial}{\partial x_i^*} \ln f(x^*) \delta x_i. \quad (3.3.2)$$

De o mare importanță practică este și problema inversă: cu ce precizie trebuie aproximată datele pentru ca rezultatul să aibă o precizie dată? Adică, dându-se  $\varepsilon > 0$ , cât trebuie să fie  $\Delta x_i$  sau  $\delta x_i$ ,  $i = \overline{1, n}$  astfel încât  $\Delta f$  sau  $\delta f < \varepsilon$ ? O metodă de rezolvare se bazează pe *principiul efectelor egale*: se presupune că toți termenii care intervin în (3.3.1) sau (3.3.2) au același efect, adică

$$\frac{\partial f}{\partial x_1^*}(x^*) \Delta x_1 = \dots = \frac{\partial f}{\partial x_n^*}(x^*) \Delta x_n.$$

Din (3.3.1) se obține

$$\Delta x_i \approx \frac{\Delta f}{n \left| \frac{\partial f}{\partial x_i^*}(x^*) \right|}. \quad (3.3.3)$$

Analog,

$$\delta x_i = \frac{\delta f}{n \left| x_i^* \frac{\partial}{\partial x_i^*} \ln f(x^*) \right|}. \quad (3.3.4)$$

## 3.4. Reprezentarea în virgulă flotantă

### 3.4.1. Parametrii reprezentării

Deși au fost propuse mai multe reprezentări pentru numerele reale, reprezentarea în virgulă flotantă este cea mai răspândită. Parametrii reprezentării în virgulă flotantă sunt baza  $\beta$  (întotdeauna pară), precizia  $p$ , exponentul maxim  $e_{\max}$  și cel minim  $e_{\min}$ , toate numere naturale. În general, un număr în virgulă flotantă se reprezintă sub forma

$$x = \pm d_0.d_1d_2 \dots d_{p-1} \times \beta^e, \quad 0 \leq d_i < \beta \quad (3.4.1)$$

unde  $d_0.d_1d_2 \dots d_{p-1}$  se numește *semnificant* sau *mantisă*, iar  $e$  *exponent*. Valoarea lui  $x$  este

$$\pm (d_0 + d_1\beta^{-1} + d_2\beta^{-2} + \dots + d_{p-1}\beta^{-(p-1)})\beta^e. \quad (3.4.2)$$

Pentru ca reprezentarea în virgulă flotantă să fie unică, numerele flotante se *normalizează*, adică se modifică reprezentarea (nu valoarea) astfel încât  $d_0 \neq 0$ . Zero se reprezintă ca  $1.0 \times \beta^{e_{\min}-1}$ . În acest mod ordinea numerică ușoară a numerelor reale nenegative corespunde ordinii lexicografice a reprezentării lor flotante (cu exponentul în stânga semnificantului).

Termenul de *număr în virgulă flotantă* este utilizat pentru a desemna un număr real care poate fi reprezentat exact în virgulă flotantă.

Distribuția numerelor în virgulă flotantă pe axa reală apare în figura 3.1.

Fiecare interval de forma  $[\beta^e, \beta^{e+1})$  din  $\mathbb{R}$  conține  $\beta^p$  numere în virgulă flotantă (numărul posibil de semnificanți). Intervalul  $(0, \beta^{e_{\min}})$  este gol și din acest motiv se introduc *numerele denormalizate*, adică numere cu semnificantul de forma  $0.d_1d_2 \dots d_{p-1}$  și exponentul

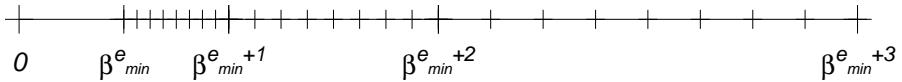


Figura 3.1: Distribuția numerelor în virgulă flotantă pe axa reală (fără denormalizare)

$\beta^{e_{\min}-1}$ . Faptul că se admit numere denormalizate sau nu se consideră a fi un parametru suplimentar al reprezentării. Mulțimea numerelor în virgulă flotantă pentru un set de parametri dați ai reprezentării se va nota cu

$$\mathbb{F}(\beta, p, e_{\min}, e_{\max}, \text{denorm}), \quad \text{denorm} \in \{\text{true}, \text{false}\}.$$

Distribuția numerelor în virgulă flotantă când se admite denormalizarea apare în figura 3.2.

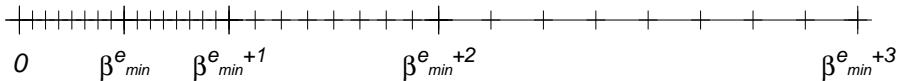


Figura 3.2: Distribuția numerelor în virgulă flotantă pe axa reală (cu denormalizare)

Această mulțime nu coincide cu  $\mathbb{R}$  din următoarele motive:

1. este o submulțime finită a lui  $\mathbb{Q}$ ;
2. pentru  $x \in \mathbb{R}$  putem avea  $|x| > \beta \times \beta^{e_{\max}}$  (depășire superioară) sau  $|x| < 1.0 \times \beta^{e_{\min}}$  (depășire inferioară).

Operațiile aritmetice uzuale pe  $\mathbb{F}(\beta, p, e_{\min}, e_{\max}, \text{denorm})$  se notează cu  $\oplus, \ominus, \otimes, \oslash$ , iar funcțiile uzuale cu SIN, COS, EXP, LN, SQRT și.a.m.d.  $(\mathbb{F}, \oplus, \otimes)$  nu este corp deoarece

$$(x \oplus y) \oplus z \neq x \oplus (y \oplus z) \quad (x \otimes y) \otimes z \neq x \otimes (y \otimes z) \\ (x \oplus y) \otimes z \neq x \otimes z \oplus y \otimes z.$$

Pentru măsurarea erorii de reprezentare, în afară de eroarea relativă, se folosește *ulps* – units in the last place (unități în ultima poziție). Dacă numărul  $z$  se reprezintă prin  $d_0.d_1d_2 \dots d_{p-1} \times \beta^e$ , atunci eroarea este

$$|d_0.d_1d_2 \dots d_{p-1} - z/\beta^e| \beta^{p-1} \text{ulps}.$$

Eroarea relativă ce corespunde la  $\frac{1}{2}$  ulps este

$$\frac{1}{2}\beta^{-p} \leq \frac{1}{2}\text{ulps} \leq \frac{\beta}{2}\beta^{-p},$$

căci eroarea absolută este  $\underbrace{0.0\dots0}_{p}\beta' \times \beta^e$ , cu  $\beta' = \frac{\beta}{2}$ . Valoarea  $\text{eps} = \frac{\beta}{2}\beta^{-p}$  se numește *epsilon-ul mașinii*.

Rotunjirea implicită se face după regula cifrei pare: dacă  $x = d_0.d_1\dots d_{p-1}d_p\dots$  și  $d_p > \frac{\beta}{2}$  rotunjirea se face în sus, dacă  $d_p < \frac{\beta}{2}$  rotunjirea se face în jos, iar dacă  $d_p = \frac{\beta}{2}$  și printre cifrele eliminate există una nenulă rotunjirea se face în sus, iar în caz contrar ultima cifră păstrată este pară. Dacă notăm cu  $\text{fl}$  operația de rotunjire, operațiile aritmetice din  $\mathbb{F}$  se pot defini prin

$$x \odot y = \text{fl}(x \circ y). \quad (3.4.3)$$

Se pot alege și alte variante de rotunjire – cât mai depărtat de 0, spre  $-\infty$ , spre  $+\infty$ , spre 0 (trunchiere). În raționamentele asupra operațiilor în virgulă flotantă vom folosi următorul model

$$\forall x, y \in \mathbb{F}, \exists \delta \text{ cu } |\delta| < \text{eps} \text{ astfel încât } x \odot y = (x \circ y)(1 + \delta). \quad (3.4.4)$$

În cuvinte, orice operație în aritmetică în virgulă flotantă este exactă până la o eroare relativă de cel mult  $\text{eps}$ .

Formula (3.4.4) se numește *axioma fundamentală a aritmeticii în virgulă flotantă*.

### 3.4.2. Anularea

Din formulele pentru eroarea relativă (3.3.2), dacă  $x \approx x(1 + \delta_x)$  și  $y \approx y(1 + \delta_y)$ , avem următoarele expresii pentru erorile relative ale operațiilor în virgulă flotantă:

$$\delta_{xy} = \delta_x + \delta_y, \quad (3.4.5)$$

$$\delta_{x/y} = \delta_x - \delta_y, \quad (3.4.6)$$

$$\delta_{x+y} = \frac{x}{x+y}\delta_x + \frac{y}{x+y}\delta_y. \quad (3.4.7)$$

Singura operație critică din punct de vedere al erorii este scăderea a două cantități apropiate  $x \approx y$ , caz în care  $\delta_{x-y} \rightarrow \infty$ . Acest fenomen se numește anulare și este reprezentat grafic în figura 3.3. Aici  $b, b', b''$  sunt cifre binare acceptabile, iar  $g$ -urile reprezintă cifre binare contaminate de eroare (gunoaie – garbage digits). De notat că, gunoi - gunoi = gunoi, dar mai important, normalizarea mută prima cifră contaminată de pe poziția a 12-a pe poziția a treia.

Anularea este de două tipuri: *benignă*, când se scad două cantități exacte și *catastrofală*, când se scad două cantități deja rotunjite. Programatorul trebuie să fie conștient de posibilitatea apariției anulării și să înceerce să o evite. Expresiile în care apare anularea trebuie rescrise, iar o anulare catastrofală trebuie întotdeauna transformată în una benignă. Vom da în continuare câteva exemple de anulări catastrofale și modul de transformare a lor.

**Exemplul 3.4.1.** Dacă  $a \approx b$ , atunci expresia  $a^2 - b^2$  se transformă în  $(a - b)(a + b)$ . Forma inițială este de preferat în cazul când  $a \gg b$  sau  $b \gg a$ .  $\diamond$

x	=	1	0	1	1	0	0	1	0	1	b	b	g	g	g	g	e
y	=	1	0	1	1	0	0	1	0	1	b'	b'	g	g	g	g	e
x-y	=	0	0	0	0	0	0	0	0	0	b''	b''	g	g	g	g	e
	=	b''	b''	g	g	g	g	?	?	?	?	?	?	?	?	?	e-9

Figura 3.3: Anularea

**Exemplul 3.4.2.** Dacă anularea apare într-o expresie cu radicali, se amplifică cu conjugată:

$$\sqrt{x+\delta} - \sqrt{x} = \frac{\delta}{\sqrt{x+\delta} + \sqrt{x}}, \quad \delta \approx 0. \quad \diamond$$

**Exemplul 3.4.3.** Diferența valorilor unei funcții pentru argumente apropiate se transformă folosind formula lui Taylor:

$$f(x+\delta) - f(x) = \delta f'(x) + \frac{\delta^2}{2} f''(x) + \dots \quad f \in C^n[a, b]. \quad \diamond$$

**Exemplul 3.4.4.** La ecuația de gradul al doilea  $ax^2 + bx + c = 0$ , anularea poate să apară dacă  $b^2 \gg 4ac$ . Formulele uzuale

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad (3.4.8)$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad (3.4.9)$$

pot să conducă la anulare astfel: pentru  $b > 0$  anularea apare la calculul lui  $x_1$ , iar pentru  $b < 0$  anularea apare la calculul lui  $x_2$ . Remediul este să amplificăm cu conjugată

$$x_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}} \quad (3.4.10)$$

$$x_2 = \frac{2c}{-b + \sqrt{b^2 - 4ac}} \quad (3.4.11)$$

și să utilizăm în primul caz formulele (3.4.10) și (3.4.9), iar în al doilea caz (3.4.8) și (3.4.11).

Să considerăm acum ecuația de gradul al doilea cu  $a = 1$ ,  $b = 100000000$  și  $c = 1$ . Aplicând formulele 3.4.8 și 3.4.9 obținem:

```
>>a=1; c=1; b=-100000000;
>>x1=(-b+sqrt(b^2-4*a*c))/(2*a)
```

```
x1 =
100000000
```

```
>>x2=(-b-sqrt(b^2-4*a*c))/(2*a)
```

```
x2 =
7.45058059692383e-009
```

Dacă se amplifică cu conjugata pentru a calcula  $x_2$  avem:

```
>>x1=(-b+sqrt(b^2-4*a*c)) / (2*a)
```

```
x1 =
100000000
```

```
>> x2a=2*c/(-b+sqrt(b^2-4*a*c))
```

```
x2a =
1e-008
```

Aceleași rezultate se obțin și cu funcția `root`.  $\diamond$

## 3.5. Standardizarea reprezentării în virgulă flotantă

### 3.5.1. Parametrii standardului

Există două standarde diferite pentru calculul în virgulă flotantă: IEEE 754 care prevede  $\beta = 2$  și IEEE 854 care permite  $\beta = 2$  sau  $\beta = 10$ , dar lasă o mai mare libertate de reprezentare. Ne vom ocupa numai de primul standard.

Parametrii standardului se dau în tabela 3.1.

	Precizia			
	Simplă	Simplă extinsă	Dublă	Dublă extinsă
p	24	$\geq 32$	53	$\geq 64$
$e_{\max}$	+127	$\geq +1023$	+1023	$\geq +16383$
$e_{\min}$	-126	$\leq -1022$	-1022	$\leq -16382$
dim. exponent	8	$\geq 11$	11	$\geq 15$
dim. număr	32	$\geq 43$	64	$\geq 79$

Tabela 3.1: Parametrii reprezentării flotante

Motivele pentru formatele extinse sunt:

1. o mai bună precizie;
2. pentru conversia din binar în zecimal și invers este nevoie de 9 cifre în simplă precizie și de 17 cifre în dublă precizie.

Motivul pentru care  $|e_{\min}| < e_{\max}$  este acela că  $1/2^{e_{\min}}$  nu trebuie să dea depășire.

Operațiile  $\oplus, \ominus, \otimes, \oslash$  trebuie să fie exact rotunjite. Precizia aceasta se asigură cu două cifre de gardă și un bit suplimentar.

Reprezentarea exponentului se numește *reprezentare cu exponent deplasat*, adică în loc de  $e$  se reprezintă  $e + D$ , unde  $D$  este fixat la alegerea reprezentării.

În cazul IEEE 754,  $D = 127$ .

### 3.5.2. Cantități speciale

În standardul IEEE 754 există următoarele cantități speciale:

Exponent	Semnificant	Ce reprezintă
$e = e_{min} - 1$	$f = 0$	$\pm 0$
$e = e_{min} - 1$	$f \neq 0$	$0.f \times 2^{e_{min}}$
$e_{min} \leq e \leq e_{max}$		$1.f \times 2^e$
$e = e_{max} + 1$	$f = 0$	$\pm \infty$
$e = e_{max} + 1$	$f \neq 0$	NaN

**NaN.** Avem de fapt o familie de valori NaN, operațiile ilegale sau nedeterminate conduc la NaN:  $\infty + (-\infty)$ ,  $0 \times \infty$ ,  $0/0$ ,  $\infty/\infty$ ,  $x \text{ REM } 0$ ,  $\infty \text{ REM } y$ ,  $\sqrt{x}$  pentru  $x < 0$ . Dacă un operand este NaN rezultatul va fi tot NaN.

**Infinit.**  $1/0 = \infty$ ,  $-1/0 = -\infty$ . Valorile infinite dă posibilitatea continuării calculului, lucru mai sigur decât abortarea sau returnarea celui mai mare număr reprezentabil.

$\frac{x}{1+x^2}$  pentru  $x = \infty$  dă rezultatul 0.

**Zero cu semn.** Avem doi de 0:  $+0, -0$ ; relațiile  $+0 = -0$  și  $-0 < +\infty$  sunt adevărate. Avantaje: tratarea simplă a depășirilor inferioare și discontinuităților. Se face distincție între  $\log 0 = -\infty$  și  $\log x = \text{NaN}$  pentru  $x < 0$ . Fără 0 cu semn nu s-ar putea face distincție la logaritmul între un număr negativ care să depășească superioară și 0.

## 3.6. Numere în virgulă flotantă în MATLAB

MATLAB utilizează numere în virgulă flotantă dublă precizie, conform standardului IEEE. Nu se face distincție între numerele întregi sau reale. Comanda `format hex` este utilă la vizualizarea reprezentării în virgulă flotantă. De exemplu, reprezentările lui 1, -1, 0.1 și ale secțiunii de aur,  $\phi = (1 + \sqrt{5})/2$ , se obțin cu:

```
>> format hex
>> 1,-1
ans =
3ff0000000000000
ans =
bff0000000000000
>> 0.1
ans =
3fb999999999999a
>> phi=(1+sqrt(5))/2
phi =
3ff9e3779b97f4a8
```

Se consideră că fracția  $f$  satisfacă  $0 \leq f < 1$ , iar exponentul  $-1022 \leq e \leq 1023$ . Sistemul de numere în virgulă flotantă al MATLAB poate fi caracterizat de trei constante: `realmin`, `realmax` și `eps`. Constanta `realmin` reprezintă cel mai mic număr normalizat în virgulă flotantă. Orice cantitate mai mică decât ea este fie un număr denormalizat, fie să depășească inferioară. Cel mai mare număr reprezentabil în virgulă flotantă se numește `realmax`. Orice

cantitate mai mare decât el dă depășire superioară. Epsilon-ul mașinii este desemnat prin `eps`. Valorile acestor constante sunt

	binar	zecimal	hexazecimal
<code>eps</code>	$2^{-52}$	2.220446049250313e-016	3cb0000000000000
<code>realmin</code>	$2^{-1022}$	2.225073858507201e-308	0010000000000000
<code>realmax</code>	$(2-\text{eps}) \times 2^{1023}$	1.797693134862316e+308	7feffffffffffffff

Funcțiile din sursele 3.1 și 3.2 calculează `eps`. Este instructiv să se tasteze individual, la prompterul MATLAB, fiecare linie din a doua variantă.

---

### Sursa MATLAB 3.1 Calculul lui `eps` - varianta 1

---

```
function eps=myeps1
eps = 1;
while (1+eps) > 1
    eps = eps/2;
end
eps = eps*2;
```

---



---

### Sursa MATLAB 3.2 Calculul lui `eps` - varianta 2

---

```
function z=myeps2
x=4/3-1;
y=3*x;
z=1-y;
```

---

Interesant că:

```
>> format long
>> 2*realmax
ans =
      Inf
>> realmin*eps
ans =
    4.940656458412465e-324
>> realmin*eps/2
ans =
      0
```

sau în hexazecimal:

```
>> format hex
>> 2*realmax
ans =
7ff0000000000000
>> realmin*eps
```

```
ans =
    00000000000000001
>> realmin*eps/2
ans =
    0000000000000000
```

Numerele denormalizate sunt situate în intervalul dintre  $\text{eps} * \text{realmin}$  și  $\text{realmin}$ . Numerele denormalizate sunt reprezentate luând  $e = -1023$ . Deplasamentul este  $D = 1023$ . Infinitul,  $\text{Inf}$ , se reprezintă luând  $e = 1024$  și  $f = 0$ , iar  $\text{NaN}$  cu  $e = 1024$  și  $f \neq 0$ . De exemplu

```
>> format short  
>> Inf-Inf  
ans =  
    NaN  
>> Inf/Inf  
ans =  
    NaN
```

Două funcții care descompun în componente și recompun din componente numerele în virgulă flotantă sunt `log2` și `pow2`. Expresia `[F, E]=log2(X)` returnează un tablou F de numere reale, de aceeași dimensiune ca și X, care verifică  $0.5 \leq \text{abs}(F) < 1$  și un tablou E de întregi, de aceeași dimensiune ca și X, astfel încât  $X = F \cdot 2^E$ . Dacă un element al lui X este zero, pe poziția respectivă vom avea  $F=0$  și  $E=0$ . Această funcție corespunde funcției ANSI C `frexp()` și funcției `logb()` din standardul IEEE. Comanda `X=pow2(F, E)`, unde F este un tablou de reale și E un tablou de întregi calculează  $X=F \cdot (2.^E)$ . Intern, se adună E la exponenții flotați ai elementelor lui F. Această funcție corespunde funcției ANSI C `ldexp()` și funcției `scalbn()` din standardul IEEE. În aritmetică IEEE, comanda `[F, E] = log2(X)` conduce la:

F	E	X
$1/2$	1	1
$\pi/4$	2	$\pi$
$-3/4$	2	-3
$1/2$	-51	eps
$1-\text{eps}/2$	1024	realmax
$1/2$	-1021	realmin

iar X = pow2(F, E) reconstituie pe X.

Am văzut în § 1.4.7 că în MATLAB 7 există și numere în virgulă flotantă în simplă precizie. De exemplu,

```
>> a = single(5);
```

atribuie lui a valoarea 5 reprezentată în virgulă flotantă în simplă precizie. Putem compara această valoare cu corespondentul ei în dublă precizie:

```
>> b=5;
>> whos
```

Name	Size	Bytes	Class
b	1x1	4	double

a	1x1	4 single array
b	1x1	8 double array

```
Grand total is 2 elements using 12 bytes
```

```
>> format hex
>> a,b
a =
    40a00000
b =
    4014000000000000
```

La conversia din double în single, datorită rotunjirii la cel mai apropiat număr în simplă precizie, valoarea poate fi afectată:

```
>> format long
>> single(3.14)
ans =
    3.1400001
```

În operații binare cu operanți de tip single rezultatul este tot de tip single. Rezultatul unei operații binare între un single și un double este de tip single, ca în exemplele:

```
>> x = single(2)*single(3)
x =
    6
>> class(x)
ans =
    single
>> x = single(8)+3
x =
    11
>> class(x)
ans =
    single
```

Funcțiile `eps`, `realmin` și `realmax` pot fi apelate și pentru simplă precizie:

```
>> eps('single')
ans =
    1.1921e-007
>> realmin('single'), realmax('single')
ans =
    1.1755e-038
ans =
    3.4028e+038
```

În MATLAB 7 funcția `eps` dă distanța între numerele în virgulă flotantă. Comanda `d=eps(x)` unde `x` este un număr în virgulă floatantă `single` sau `double`, dă distanța de la `abs(x)` la cel mai apropiat număr în virgulă flotantă mai mare decât `x` și de aceeași precizie cu `x`. Exemplu:

```
>> format long
>> eps
ans =
2.220446049250313e-016
>> eps(5)
ans =
8.881784197001252e-016
```

`eps('double')` este echivalent cu `eps` sau cu `eps(1.0)`, iar `eps('single')` este echivalent cu `eps(single(1.0))`.

Evident, distanța dintre numerele în virgulă flotantă simplă precizie este mai mare decât cea între numerele în virgulă flotantă dublă precizie. De exemplu,

```
>> x = single(5)
>> eps(x)
```

returnează

```
ans =
4.7683716e-007
```

care este mai mare decât `eps(5)`.

### 3.7. Condiționarea unei probleme

Putem gândi o problemă ca o aplicație

$$f : \mathbb{R}^m \rightarrow \mathbb{R}^n, \quad y = f(x). \quad (3.7.1)$$

Ne interesează sensibilitatea aplicației într-un punct dat  $x$  la mici perturbații ale argumentului, adică cât de mare sau cât de mică este perturbația lui  $y$  comparativ cu perturbația lui  $x$ . În particular, dorim să măsurăm gradul de sensibilitate printr-un singur număr, numărul de condiționare al aplicației  $f$  în punctul  $x$ . Vom presupune că  $f$  este calculată exact, cu precizie infinită. *Condiționarea lui  $f$  este deci o proprietate inherentă a funcției  $f$  și nu depinde de nici o considerație algoritmică legată de implementarea sa.*

Aceasta nu înseamnă că determinarea condiționării unei probleme este nerelevantă pentru orice soluție algoritmică a problemei. Din contră. Motivul este acela că soluția calculată cu (3.7.1),  $y^*$  (utilizând un algoritm specific și aritmetică în virgulă flotantă) este (și acest lucru se poate demonstra) soluția unei probleme „apropiate“

$$y^* = f(x^*) \quad (3.7.2)$$

cu

$$x^* = x + \delta \quad (3.7.3)$$

și, mai mult, distanța  $\|\delta\| = \|x^* - x\|$  poate fi estimată în termeni de precizie a mașinii. Deci, dacă știm cât de tare sau cât de slab reacționează aplicația la mici perturbații, cum ar fi  $\delta$  în (3.7.3), putem spune ceva despre eroarea  $y^* - y$  a soluției cauzată de această perturbație.

Se poate considera și condiționarea între aplicații mai generale, dar pentru implementări practice este suficient să ne limităm la cazul finit dimensional.

Fie

$$x = [x_1, \dots, x_m]^T \in \mathbb{R}^m, \quad y = [y_1, \dots, y_n]^T \in \mathbb{R}^n,$$

$$y_\nu = f_\nu(x_1, \dots, x_m), \quad \nu = \overline{1, n}.$$

$y_\nu$  va fi privit ca o funcție de o singură variabilă  $x_\mu$

$$\gamma_{\nu\mu} = (\text{cond}_{\nu\mu} f)(x) = \left| \frac{x_\mu \frac{\partial f_\nu}{\partial x_\mu}}{f_\nu(x)} \right|. \quad (3.7.4)$$

Aceasta ne dă o matrice de numere de condiționare

$$\Gamma(x) = \begin{pmatrix} \frac{x_1 \frac{\partial f_1}{\partial x_1}}{f_1(x)} & \dots & \frac{x_m \frac{\partial f_1}{\partial x_m}}{f_1(x)} \\ \vdots & \ddots & \vdots \\ \frac{x_1 \frac{\partial f_n}{\partial x_1}}{f_n(x)} & \dots & \frac{x_m \frac{\partial f_n}{\partial x_m}}{f_n(x)} \end{pmatrix} = [\gamma_{\nu\mu}(x)] \quad (3.7.5)$$

și vom lua ca *număr de condiționare*

$$(\text{cond } f)(x) = \|\Gamma(x)\|. \quad (3.7.6)$$

**Altă abordare.** Considerăm norma  $\|\cdot\|_\infty$

$$\Delta y_\nu \approx \sum_{\mu=1}^m \frac{\partial f_\nu}{\partial x_\mu} \Delta x_\mu (= f_\nu(x + \Delta x) - f_\nu(x))$$

$$|\Delta y_\nu| \leq \sum_{\mu=1}^n \left| \frac{\partial f_\nu}{\partial x_\mu} \right| \Delta x_\mu \leq \max_\mu |\Delta x_\mu| \sum_{\mu=1}^m \left| \frac{\partial f_\nu}{\partial x_\mu} \right| \leq$$

$$\leq \max_\mu |\Delta x_\mu| \max_\nu \sum_{\mu=1}^m \left| \frac{\partial f_\nu}{\partial x_\mu} \right|$$

Am obținut

$$\|\Delta y\|_\infty \leq \|\Delta x\|_\infty \left\| \frac{\partial f}{\partial x} \right\|_\infty \quad (3.7.7)$$

unde

$$J(x) = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \in \mathbb{R}^n \times \mathbb{R}^m \quad (3.7.8)$$

este matricea jacobiană a lui  $f$

$$\frac{\|\Delta y\|_\infty}{\|y\|_\infty} \leq \frac{\|x\|_\infty \left\| \frac{\partial f}{\partial x} \right\|_\infty}{\|f(x)\|_\infty} \cdot \frac{\|\Delta x\|}{\|x\|_\infty}. \quad (3.7.9)$$

Estimarea (3.7.9) este mai grosieră decât (3.7.4).

Dacă  $m = n = 1$  în ambele abordări se obține

$$(\text{cond } f)(x) = \left| \frac{xf'(x)}{f(x)} \right|,$$

pentru  $x \neq 0, y \neq 0$ .

Dacă  $x = 0 \wedge y \neq 0$  se consideră eroarea absolută pentru  $x$  și eroarea relativă pentru  $y$

$$(\text{cond } f)(x) = \left| \frac{f'(x)}{f(x)} \right|;$$

pentru  $y = 0 \wedge x \neq 0$  se ia eroarea absolută pentru  $y$  și eroarea relativă pentru  $x$ , iar pentru  $x = y = 0$

$$(\text{cond } f)(x) = f'(x).$$

**Exemplul 3.7.1 (Sisteme de ecuații liniare algebrice).** Dându-se matricea  $A \in \mathbb{R}^{m \times n}$  și vectorul  $b \in \mathbb{R}^n$  să se rezolve sistemul

$$Ax = b. \quad (3.7.10)$$

Aici datele de intrare sunt elementele lui  $A$  și  $b$ , iar rezultatul este vectorul  $x$ . Pentru a simplifica lucrurile să presupunem că  $A$  este o matrice fixată care nu se schimbă și că  $b$  ar putea fi perturbat. Avem aplicația  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  dată de

$$x = f(b) := A^{-1}b,$$

care este liniară. Deci  $\frac{\partial f}{\partial b} = A^{-1}$  și utilizând (3.7.9) obținem

$$\begin{aligned} (\text{cond } f)(b) &= \frac{\|b\| \|A^{-1}\|}{\|A^{-1}b\|} = \frac{\|Ax\| \|A^{-1}\|}{\|A^{-1}b\|}, \\ \max_{\substack{b \in \mathbb{R}^n \\ b \neq 0}} (\text{cond } f)(b) &= \max_{\substack{x \in \mathbb{R}^n \\ b \neq 0}} \frac{\|Ax\|}{\|x\|} \|A^{-1}\| = \|A\| \|A^{-1}\|. \end{aligned} \quad (3.7.11) \quad \diamond$$

Numărul  $\|A\| \|A^{-1}\|$  se numește număr de condiționare al matricei  $A$  și se notează  $\text{cond } A$ .

$$\text{cond } A = \|A\| \|A^{-1}\|.$$

Vom reveni asupra acestei probleme în capitolul 4, consacrat rezolvării sistemelor de ecuații algebrice liniare.

## 3.8. Condiționarea unui algoritm

Fie problema

$$f : \mathbb{R}^m \rightarrow \mathbb{R}^n, \quad y = f(x). \quad (3.8.1)$$

Împreună cu  $f$  se dă un algoritm  $A$  care rezolvă problema. Adică dându-se un vector  $x \in \mathbb{F}(\beta, p, e_{min}, e_{max}, \text{denorm})$ , algoritmul  $A$  produce un vector  $y_A$  (în aritmetică în virgulă flotantă), despre care se presupune că aproximează  $y = f(x)$ . Astfel avem o aplicație  $f_A$  ce descrie modul în care problema  $f$  este rezolvată de algoritmul  $A$

$$f_A : \mathbb{F}^m(\dots) \rightarrow \mathbb{F}^n(\dots), \quad y_A = f_A(x).$$

Pentru a putea analiza  $f_A$  facem următoarea ipoteză de bază

$$\forall x \in \mathbb{F}^m \exists x_A \in \mathbb{F}^m : \quad f_A(x) = f(x_A). \quad (3.8.2)$$

Adică, soluția calculată corespunzând unei anumite intrări  $x$  este soluția exactă pentru o altă intrare, diferită de prima,  $x_A$  (nu neapărat număr în virgulă flotantă, și nici unic determinată) despre care sperăm că este apropiată de  $x$ . Cu cât găsim un  $x_A$  mai apropiat de  $x$ , cu atât avem mai mare încredere în algoritm.

Definim *condiționarea lui A în x* comparând eroarea relativă la intrare cu eps:

$$(\text{cond } A)(x) = \inf_{x_A} \frac{\|x_A - x\|}{\|x\|} / \text{eps}.$$

Justificare:

$$\delta_y = \frac{f_A(x) - f(x)}{f(x)} = \frac{(x_A - x)f'(x)}{f(x)} \approx \frac{x_A - x}{x} \cdot \frac{1}{\text{eps}} \frac{xf'(x)}{f(x)} \text{eps}.$$

Infimumul se ia după toți  $x_A$  ce satisfac  $y_A = f(x_A)$ . În practică se poate lua orice valoare  $x_A$  și se obține o margine superioară a numărului de condiționare

$$(\text{cond } A)(x) \leq \frac{\frac{\|x_A - x\|}{\|x\|}}{\text{eps}}. \quad (3.8.3)$$

## 3.9. Eroarea globală

Considerăm din nou problema:

$$f : \mathbb{R}^m \rightarrow \mathbb{R}^n, \quad y = f(x). \quad (3.9.1)$$

Aceasta este problema (matematică) idealizată, în care datele sunt numere reale, iar soluția este soluția matematică exactă. Când o rezolvăm în aritmetică în virgulă flotantă, cu precizia eps, utilizând un algoritm  $A$ , rotunjim la început toate datele și acestora nu le aplicăm  $f$ , ci  $f_A$ .

$$x^* = x \text{ rotunjit}, \quad \frac{\|x^* - x\|}{\|x\|} = \varepsilon, \quad y_A^* = f_A(x^*).$$

Aici  $\varepsilon$  este eroarea de rotunjire. Ea poate proveni și din alte surse (măsurători). Eroarea totală este

$$\frac{\|y_A^* - y\|}{\|y\|}.$$

Pe baza ipotezei (3.8.2) alegând  $x_A$  optimal avem

$$\begin{aligned} f_A(x^*) &= f(x_A^*), \\ \frac{\|x_A^* - x^*\|}{\|x^*\|} &= (\text{cond } A)(x^*) \text{ eps}. \end{aligned} \quad (3.9.2)$$

Fie  $y^* = f(x^*)$ . Avem utilizând inegalitatea triunghiului

$$\frac{\|y_A^* - y\|}{\|y\|} \leq \frac{\|y_A^* - y^*\|}{\|y\|} + \frac{\|y^* - y\|}{\|y\|} \approx \frac{\|y_A^* - y^*\|}{\|y^*\|} + \frac{\|y^* - y\|}{\|y^*\|}.$$

Am presupus că  $\|y\| \approx \|y^*\|$ . Din (3.9.2) rezultă pentru primul termen

$$\begin{aligned} \frac{\|y_A^* - y^*\|}{\|y^*\|} &= \frac{\|f_A(x^*) - f(x^*)\|}{\|f(x^*)\|} = \frac{\|f(x_A^*) - f(x^*)\|}{\|f(x^*)\|} \leq \\ &\leq (\text{cond } f)(x^*) \frac{\|x_A^* - x\|}{\|x^*\|} = (\text{cond } f)(x^*)(\text{cond } A)(x^*) \text{ eps}, \end{aligned}$$

iar pentru al doilea

$$\frac{\|y^* - y\|}{\|y\|} = \frac{\|f(x^*) - f(x)\|}{\|f(x)\|} \leq (\text{cond } f)(x) \frac{\|x^* - x\|}{\|x\|} = (\text{cond } f)(x)\varepsilon.$$

Presupunând că  $(\text{cond } f)(x^*) \approx (\text{cond } f)(x)$  obținem

$$\frac{\|y_A^* - y\|}{\|y\|} \leq (\text{cond } f)(x)[\varepsilon + (\text{cond } A)(x^*) \text{ eps}]. \quad (3.9.3)$$

Interpretare: erorile în date și eps contribuie împreună la eroarea totală. Ambele sunt amplificate de condiționarea problemei, dar ultima este amplificată și de condiționarea algoritmului.

### 3.10. Probleme prost condiționate și probleme incorect puse

Dacă numărul de condiționare al unei probleme este mare ( $(\text{cond } f)(x) \gg 1$ ), atunci chiar pentru erori (relative) mici trebuie să ne așteptăm la erori foarte mari în datele de ieșire. Astfel de probleme se numesc *probleme prost condiționate*. Nu este posibil să tragem o linie clară de demarcare între problemele bine condiționate și cele prost condiționate. O categorizare a unei probleme în prost condiționată sau bine condiționată depinde de specificațiile de precizie. Dacă dorim ca

$$\frac{\|y^* - y_A^*\|}{\|y\|} < \tau$$

și în (3.9.3) ( $\text{cond } f(x)\varepsilon \geq \tau$ , atunci problema este sigur prost condiționată).

Este important să se aleagă o limită rezonabilă pentru eroare, căci în caz contrar, chiar dacă creștem numărul de iterații, nu vom putea crește precizia.

Dacă rezultatul unei probleme matematice depinde discontinuu de date ce variază continuu, atunci este imposibil de dat o soluție numerică a problemei în vecinătatea discontinuității. În astfel de cazuri rezultatul poate fi perturbat substanțial, chiar dacă datele de intrare sunt precise și utilizăm aritmetică de precizie multiplă. Astfel de probleme se numesc *probleme incorect puse*. O problemă incorect pusă poate să apară, de exemplu, dacă un rezultat întreg este calculat din date reale (adică date care variază continuu), de exemplu numărul de rădăcini reale ale unei funcții sau rangul unei matrice.

**Exemplul 3.10.1 (Numărul de zerouri reale ale unui polinom).** Ecuația

$$P_3(x, c_0) = c_0 + x - 2x^2 + x^3$$

are una, două sau trei rădăcini reale, după cum  $c_0$  este strict pozitiv, zero sau strict negativ (vezi figura 3.4). Deci, pentru valori ale lui  $c_0$  apropiate de zero, numărul de zerouri reale ale lui  $P_3$  este o problemă incorect pusă.  $\diamond$

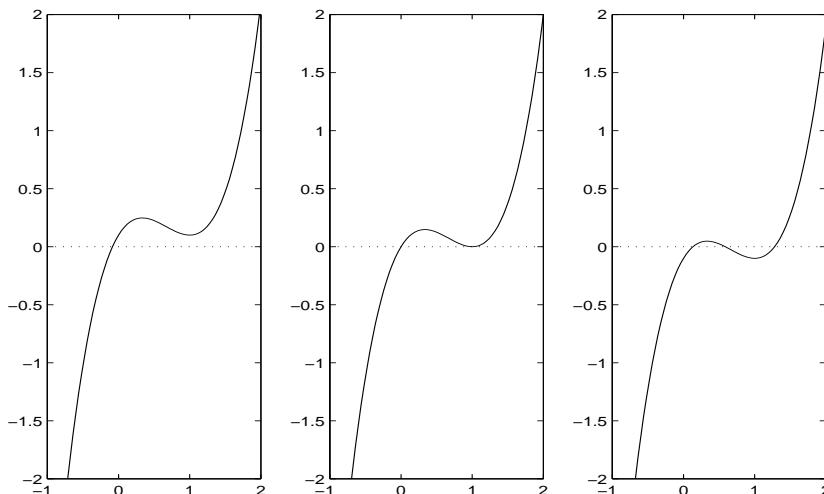


Figura 3.4: Problemă incorect pusă

Unii autori (vezi, de exemplu, [15]) numesc probleme incorect puse problemele al căror număr de condiționare este infinit.

**Exemplul 3.10.2.** Un exemplu clasic este problema evaluării unui polinom cu schema lui Horner (a se vedea [15, secțiunea 1.6] și [46]). Vom compara valoarea obținută prin evaluarea lui  $p(x) = (x - 1)^7$  prin expandare și utilizarea schemei lui Horner și prin calcul direct, folosind formula care îl dă pe  $p$ . Figura 3.5(a) s-a obținut cu secvență:

```
x = 0.988:1e-4:1.012;
p=[1,-7,21,-35,35,-21,7,-1];
y2=polyval(p,x);
plot(x,y2)
```

iar 3.5(b) cu

```
y1=(x-1).^7;
plot(x,y1)
```

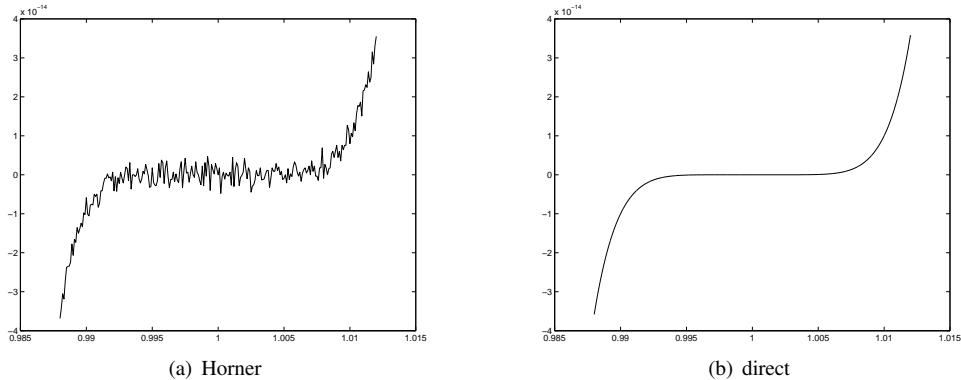


Figura 3.5: Evaluarea lui  $p(x) = (x - 1)^7$  cu schema lui Horner (stânga) și direct

## 3.11. Stabilitatea

### 3.11.1. Notații asimptotice

Această subsecțiune introduce notațiile asimptotice de bază și câteva abuzuri comune. Pentru o funcție dată  $g(n)$  vom nota cu  $\Theta(g(n))$  mulțimea de funcții

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \quad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \leq n_0\}.$$

Deși  $\Theta(g(n))$  este o mulțime; scriem  $f(n) = \Theta(g(n))$  pentru a indica faptul că  $f(n) \in \Theta(g(n))$ . Acest abuz de egalitate pentru a nota apartenența la o mulțime poate părea la început confuz, dar vom vedea că are anumite avantaje. Vom spune că  $g(n)$  este o *margine asimptotică strânsă* (*assymptotically tight bound*) pentru  $f(n)$ .

Definiția mulțimii  $\Theta(g(n))$  necesită ca fiecare membru al ei să fie *asimptotic nenegativ*, adică  $f(n) \geq 0$  când  $n$  este suficient de mare. În consecință  $g(n)$  trebuie să fie și ea asimptotic negativă, căci altfel  $\Theta(g(n))$  este vidă. Din acest motiv vom presupune că fiecare funcție utilizată în interiorul notației  $\Theta$  este asimptotic nenegativă. Această presupunere are loc și pentru celelalte notații asimptotice care vor fi definite în acest capitol.

Pentru o funcție dată  $g(n)$  vom nota cu  $O(g(n))$  mulțimea de funcții

$$O(g(n)) = \{f(n) : \exists c, n_0 \quad 0 \leq f(n) \leq cg(n), \quad \forall n \geq n_0\}.$$

Pentru a indica faptul că  $f(n)$  este un membru al lui  $O(g(n))$  scriem  $f(n) = O(g(n))$ . Observăm că  $f(n) = \Theta(g(n)) \implies f(n) = O(g(n))$ , deoarece notația  $\Theta$  este mai tare decât notația  $O$ . Utilizând notațiile din teoria mulțimilor avem  $\Theta(g(n)) \subseteq O(g(n))$ . Una dintre proprietățile ciudate ale notației este aceea că  $n = O(n^2)$ .

Pentru o funcție dată  $g(n)$  vom nota prin  $\Omega(g(n))$  mulțimea de funcții

$$\Omega(g(n)) = \{f(n) : \exists c, n_0 \ 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}.$$

Această notație furnizează o *margine asimptotică inferioară*. Din definițiile notațiilor asimptotice se obține imediat:

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n)).$$

Spunem că funcțiile  $f$  și  $g : \mathbb{N} \rightarrow \mathbb{R}$  sunt *asimptotic echivalente*, notație  $\sim$  dacă

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1.$$

Extinderea notațiilor asimptotice la mulțimea numerelor reale este naturală. De exemplu  $f(t) = O(g(t))$  înseamnă că există o constantă pozitivă  $C$  astfel încât pentru orice  $t$  suficient de apropiat de o limită subînțeleasă (de exemplu  $t \rightarrow \infty$  sau  $t \rightarrow 0$ ) avem

$$|f(t)| \leq Cg(t). \quad (3.11.1)$$

### 3.11.2. Precizie și stabilitate

Vom considera o *problemă* ca fiind o aplicație  $f : X \rightarrow Y$ , unde  $X$  și  $Y$  sunt spații liniare normate (pentru scopurile noastre ne vom limita la cazul finit dimensional). Ne va interesa comportarea problemei într-un punct particular  $x \in X$  (comportarea poate diferi de la un punct la altul). Combinarea unei probleme  $f$  cu niște date prescrise  $x$  se va numi *instanță a problemei*, dar se obișnuiește să se utilizeze termenul de *problemă* pentru ambele noțiuni.

Numerele complexe sunt reprezentate printr-o pereche de numere în virgulă flotantă și operațiile elementare se realizează pe această reprezentare. Rezultatul este acela că axioma (3.4.4) este valabilă și pentru numere complexe, exceptând faptul că la operațiile  $\otimes$  și  $\oslash$  eps trebuie mărit cu un factor de  $2^{3/2}$  și respectiv  $2^{5/2}$ .

Un algoritm va fi privit ca o aplicație  $f_A : X \rightarrow Y$ , unde  $X$  și  $Y$  sunt ca mai sus. Fie  $f$  o problemă, un calculator al cărui sistem de numere în virgulă flotantă satisfacă (3.4.4), dar nu neapărat (3.4.3), un algoritm  $f_A$  pentru  $f$  și o implementare a acestui algoritm sub formă de program pe calculator,  $A$ , toate fixate. Dându-se o dată  $x \in X$ , o vom rotunji la un număr în virgulă flotantă și apoi o vom furniza programului. Rezultatul este o colecție de numere în virgulă flotantă care aparțin spațiului  $Y$  (deoarece algoritmul a fost conceput să rezolve  $f$ ). Vom numi acest rezultat calculat  $f_A(x)$ .

Exceptând cazul trivial,  $f_A$  nu poate fi continuă. Vom spune că un algoritm  $f_A$  pentru problema  $f$  este *precis*, dacă pentru orice  $x \in X$ , eroarea sa relativă verifică

$$\frac{\|f_A(x) - f(x)\|}{\|f(x)\|} = O(\text{eps}). \quad (3.11.2)$$

Dacă problema  $f$  este prost condiționată, obiectivul preciziei, aşa cum este definit de (3.11.2), este nerezonabil de ambicioz. Erorile de rotunjire în datele de intrare sunt inevitabile pe calculatoare numerice și chiar dacă toate calculele următoare se realizează perfect, această perturbație ne conduce la o modificare semnificativă a rezultatului. În loc să urmărim precizia în toate cazurile, este mai rezonabil să urmărim stabilitatea. Spunem că algoritmul  $f_A$  pentru problema  $f$  este *stabil* dacă pentru orice  $x \in X$  există un  $\tilde{x}$  cu

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\text{eps}) \quad (3.11.3)$$

astfel încât

$$\frac{\|f_A(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = O(\text{eps}). \quad (3.11.4)$$

În cuvinte, *un algoritm stabil ne dă un răspuns aproape corect la o problemă aproape exactă*.

Mulți dintre algoritmii din algebra liniară numerică satisfac o condiție care este mai puternică și mai simplă decât stabilitatea. Spunem că un algoritm  $f_A$  pentru problema  $f$  este *regresiv stabil* (backward stable) dacă

$$\forall x \in X \exists \tilde{x} \text{ cu } \frac{\|\tilde{x} - x\|}{\|x\|} = O(\text{eps}) \text{ astfel încât } f_A(x) = f(\tilde{x}). \quad (3.11.5)$$

Aceasta este o întărire a definiției stabilității în sensul că  $O(\text{eps})$  din (3.11.4) a fost înlocuit cu 0. În cuvinte, *un algoritm este regresiv stabil dacă dă răspunsul corect la o problemă aproape exactă*.

### **Observația 3.11.1.** Semnificația notăției

$$\|\text{cantitate calculată}\| = O(\text{eps}) \quad (3.11.6)$$

este următoarea:

- $\|\text{cantitate calculată}\|$  reprezintă norma unui număr sau a unei colecții de numere determinate de algoritmul  $f_A$  pentru o problemă  $f$ , depinzând atât de datele de intrare  $x \in X$  ale lui  $f$  cât și de  $\text{eps}$ . Un exemplu este eroarea relativă.
- Procesul implicit de trecere la limită este  $\text{eps} \rightarrow 0$  (adică  $\text{eps}$  corespunde lui  $t$  din (3.11.1)).
- $O$  se aplică uniform tuturor datelor  $x \in X$ . La formularea rezultatelor de stabilitate această uniformitate va fi implicită.
- Pentru orice aritmetică pe calculator particulară  $\text{eps}$  este o cantitate fixată. Când vorbim despre limita  $\text{eps} \rightarrow 0$ , considerăm o idealizare a unui calculator sau a unei familii de calculatoare. Ecuația (3.11.6) înseamnă că dacă rulăm algoritmul în cauză pe calculatoare ce satisfac (3.4.3) și (3.4.4) pentru un sir de  $\text{eps}$ -uri descrescător ce tinde către zero, se garantează că  $\|\text{cantitate calculată}\|$  descrește proporțional cu  $\text{eps}$  sau mai repede. Aceste calculatoare ideale li se cere să satisfacă doar (3.4.3) și (3.4.4) și nimic altceva.

- Constanta implicită din notația  $O$  poate depinde și de dimensiunile argumentelor (de exemplu pentru rezolvarea unui sistem  $Ax = b$  de dimensiunile lui  $A$  și  $b$ ). În general, în probleme practice, creșterea erorii datorată dimensiunii este lentă, dar pot exista situații în care să apară factori cum ar fi  $2^m$ , care fac astfel de margini inutile în practică.
- ◊

Datorită echivalenței normelor pe spații finit dimensionale, pentru problemele  $f$  și algoritmi  $f_A$  definite pe astfel de spații, proprietățile de precizie, stabilitate și stabilitate regresivă au loc sau nu independent de alegerea normelor.

### 3.11.3. Analiza regresivă a erorilor

Stabilitatea regresivă implică precizia în sens relativ.

**Teorema 3.11.2.** *Presupunem că se aplică un algoritm regresiv stabil  $f_A$  unei probleme  $f : X \rightarrow Y$  cu numărul de condiționare  $(\text{cond } f)(x)$  pe un calculator ce satisface (3.4.3) și (3.4.4). Atunci eroarea relativă satisface*

$$\frac{\|f_A(x) - f(x)\|}{\|f(x)\|} = O((\text{cond } f)(x) \text{eps}). \quad (3.11.7)$$

*Demonstrație.* Din definiția (3.11.5) a stabilității regresive, există un  $\tilde{x} \in X$  ce satisface

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\text{eps}),$$

astfel încât  $f_A(x) = f(\tilde{x})$ . Din definiția (3.7.5) și (3.7.6) a lui  $(\text{cond } f)(x)$  aceasta implică

$$\frac{\|f_A(x) - f(x)\|}{\|f(x)\|} \leq ((\text{cond } f)(x) + o(1)) \frac{\|\tilde{x} - x\|}{\|x\|}, \quad (3.11.8)$$

unde  $o(1)$  desemnează o cantitate ce converge către zero când  $\text{eps} \rightarrow 0$ . Combinând aceste delimitări se obține (3.11.7). □

Procesul urmat în demonstrația teoremei 3.11.2 este cunoscut sub numele de *analiza regresivă a erorilor* (backward error analysis). Se obține o estimare a preciziei în doi pași. Primul pas este investigarea condiționării problemei. Celălalt este investigarea stabilității propriu-zise a algoritmului. Conform teoremei 3.11.2, dacă algoritmul este regresiv stabil, atunci precizia finală reflectă acel număr de condiționare.

În afară de analiza regresivă a erorilor, există și o analiză directă sau *progresivă*. Aici se estimează erorile de rotunjire la fiecare pas și apoi modul cum se compun și în final un total (secțiunea 3.3).

Experiența a arătat că pentru cei mai mulți algoritmi ai algebrei liniare numerice analiza progresivă a erorilor este mai greu de realizat decât cea regresivă. Cei mai buni algoritmi pentru cele mai multe probleme ale algebrei liniare numerice nu fac altceva mai bun decât să calculeze soluția exactă pentru niște date ușor perturbate. Analiza regresivă este o metodă de raționament bine adaptată acestei situații.

## Probleme

**Problema 3.1.** Scrieți funcții MATLAB pentru calculul lui  $\sin x$  și  $\cos x$  folosind formula lui Taylor:

$$\begin{aligned}\sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \dots \\ \cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} + \dots\end{aligned}$$

Știm de la cursul de Analiză matematică urmatoarele:

- modulul erorii este mai mic decat modulul primului termen neglijat;
- raza de convergență este  $R = \infty$ .

Ce se întâmplă pentru  $x = 10\pi$  (și în general pentru  $x = 2k\pi$ ,  $k$  mare)? Explicați fenomenul și propuneți un remediu.

**Problema 3.2.** Fie

$$E_n = \int_0^1 x^n e^{x-1} dx.$$

Se observă că  $E_1 = 1/e$  și  $E_n = 1 - nE_{n-1}$ ,  $n = 2, 3, \dots$ . Se poate arăta că

$$0 < E_n < \frac{1}{n+1}$$

și dacă  $E_1 = c$  avem

$$\lim_{n \rightarrow \infty} E_n = \begin{cases} 0, & \text{pentru } c = 1/e \\ \infty & \text{altfel.} \end{cases}$$

Explicați fenomenul, găsiți un remediu și calculați  $e$  cu precizia eps.

**Problema 3.3.** Să se studieze teoretic și experimental condiționarea problemei determinării rădăcinilor ecuației polinomiale

$$x^n + a_1 x^{n-1} + a_2 x^{n-2} + \dots + a_n = 0 \quad (3.11.9)$$

cunoscându-se coeficienții. Se va scrie o rutină pentru calculul numărului de condiționare al fiecărei rădăcini și se va studia grafic efectul perturbării fiecărui coeficient cu o variabilă aleatoare normală cu media 0 și dispersia  $1e-10$ .

Aplicație pentru ecuațiile

$$(x-1)(x-2)\dots(x-n) = 0$$

și (3.11.9) pentru  $a_k = 2^{-k}$ . Se va lua ca exemplu practic pentru testare  $n = 20$ . Ce se întâmplă dacă perturbația urmează legea uniformă?

**Problema 3.4.** Care este indicele celui mai mare număr Fibonacci care poate fi reprezentat *exact* în MATLAB în dublă precizie? Care este indicele celui mai mare număr Fibonacci care poate fi reprezentat în MATLAB în dublă precizie fără a da depășire flotantă superioară?

**Problema 3.5.** Fie  $F$  mulțimea tuturor numerelor în virgulă flotantă IEEE, exceptând NaN și Inf, care au exponentul deplasat 7ff (în hexazecimal) și numerele denormalizate, care au exponentul deplasat 000 (în hexazecimal).

- (a) Câte elemente are  $F$ ?
- (b) Ce proporție a elementelor lui  $F$  este în intervalul  $[1, 2)$ ?
- (c) Ce proporție a elementelor lui  $F$  este în intervalul  $[1/64, 1/32)$ ?
- (d) Determinați prin selecție aleatoare proporția de elemente din  $F$  care satisfac relația logică MATLAB

$$x * (1/x) == 1$$

**Problema 3.6.** Ce numere reale familiare sunt aproximate prin numerele în virgulă flotantă pentru care cu format hex se afișează următoarele valori:

```
4059000000000000
3f847ae147ae147b
3fe921fb54442d18
```

**Problema 3.7.** Explicați rezultatele afișate de

```
t = 0.1
n = 1:10
e = n/10 - n*t
```

**Problema 3.8.** Ce face fiecare din următoarele programe? Câte linii de ieșire produce fiecare program? Care sunt ultimele două valori ale lui  $x$  afișate?

```
x=1; while 1+x>1, x=x/2, pause (.02), end
x=1; while x+x>x, x=2*x, pause (.02), end
x=1; while x+x>x, x=x/2, pause (.02), end
```

**Problema 3.9.** Scrieți o funcție MATLAB care calculează aria unui triunghi când se cunosc laturile, folosind formula lui Heron:

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

unde  $p$  este semiperimetru. Ce se întâmplă dacă triunghiul este aproape degenerat? Propuneți un remediu și dați o evaluare a erorii (vezi [27]).

**Problema 3.10.** Dispersia de selecție se definește prin

$$s^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2,$$

unde

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i.$$

Ea se mai poate calcula și cu formula

$$s^2 = \frac{1}{N-1} \left[ \sum_{i=1}^N x_i^2 - \frac{1}{N} \left( \sum_{i=1}^N x_i \right)^2 \right].$$

Care formulă este mai precisă din punct de vedere numeric? Dați un exemplu care să justifice răspunsul.

# CAPITOLUL 4

## Rezolvarea numerică a sistemelor de ecuații algebrice liniare

Există două clase de metode de rezolvare a sistemelor algebrice liniare: metode *directe* sau *exacte*, care furnizează soluția într-un număr finit de pași, în ipoteza că toate calculele se fac exact (Cramer, eliminarea gaussiană, Cholesky) și metode *iterative*, care aproximează soluția generând un sir care converge către aceasta (Jacobi, Gauss-Seidel, SOR).

### 4.1. Elemente de Analiză matricială

$p$ -norma unui vector  $x \in \mathbb{K}^n$  se definește prin

$$\|x\|_p = \left( \sum_{i=1}^n |x_i|^p \right)^{1/p} \quad 1 \leq p < \infty.$$

Pentru  $p = \infty$  norma este definită prin

$$\|x\|_\infty = \max_{i=1,n} |x_i|.$$

Norma  $\|\cdot\|_2$  se numește *normă euclidiană*,  $\|\cdot\|_1$  se numește *normă Minkowski*, iar  $\|\cdot\|_\infty$  se numește *normă Cebîșev*.

Funcția MATLAB `norm` calculează  $p$ -norma unui vector. Ea este apelată sub forma `norm(x, p)`, cu valoarea implicită  $p=2$ . În cazul special când  $p=-Inf$ , se calculează cantitatea  $\min_i |x_i|$ . Exemplu:

```
>> x = 1:4;
>> [norm(x, 1), norm(x, 2), norm(x, inf), norm(x, -inf)]
```

```
ans =
10.0000    5.4772    4.0000    1.0000
```

Figura 4.1 dă reprezentările sferelor unitate din  $\mathbb{R}^2$  în diverse  $p$ -norme. Ea a fost obținută cu ajutorul funcției `contour`.

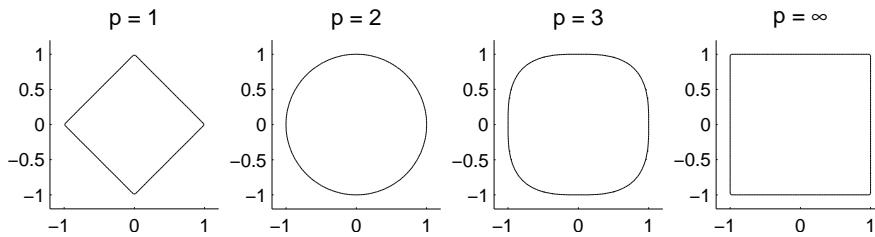


Figura 4.1: Sfera unitate din  $\mathbb{R}^2$  pentru patru  $p$ -norme

Fie  $A \in \mathbb{K}^{n \times n}$ . Polinomul  $p(\lambda) = \det(A - \lambda I)$  se numește *polinomul caracteristic* al lui  $A$ . Rădăcinile lui  $p$  se numesc *valori proprii* ale lui  $A$ , iar dacă  $\lambda$  este o valoare proprie a lui  $A$  vectorul  $x \neq 0$  cu proprietatea că  $(A - \lambda I)x = 0$  se numește *vector propriu* al lui  $A$  corespunzător valorii proprii  $\lambda$ . Multimea valorilor proprii ale lui  $A$  (spectrul lui  $A$ ) se notează cu  $\lambda(A)$ .

Valoarea  $\rho(A) = \max\{|\lambda| \mid \lambda \text{ valoare proprie a lui } A\}$  se numește *rază spectrală* a matricei  $A$ . Vom nota cu  $A^T$  transpusa lui  $A$  și cu  $A^*$  transpusa conjugată a lui  $A$ .

**Definiția 4.1.1.** O matrice  $A$  se numește:

1. normală, dacă  $AA^* = A^*A$ ;
2. unitară, dacă  $AA^* = A^*A = I$ ;
3. hermitiană, dacă  $A = A^*$ ;
4. ortogonală, dacă  $AA^T = A^TA = I$ ,  $A$  reală;
5. simetrică, dacă  $A = A^T$ ,  $A$  reală.

**Definiția 4.1.2.** O normă matricială este o aplicație  $\|\cdot\| : \mathbb{K}^{m \times n} \rightarrow \mathbb{R}$  care pentru orice  $A, B \in \mathbb{K}^{m \times n}$  și  $\alpha \in \mathbb{K}$  verifică următoarele relații

$$(NM1) \quad \|A\| \geq 0, \quad \|A\| = 0 \Leftrightarrow A = O_n;$$

$$(NM2) \quad \|\alpha A\| = |\alpha| \|A\|;$$

$$(NM3) \quad \|A + B\| \leq \|A\| + \|B\|;$$

$$(NM4) \quad \|AB\| \leq \|A\| \|B\|.$$

Primele trei proprietăți ne spun că  $\|\cdot\|$  este o normă pe  $\mathbb{K}^{m \times n}$ , care este și spațiu vectorial de dimensiune  $mn$ , iar (NM4) este specifică normelor matriciale. Un mijloc simplu de construire a normelor matriciale este următorul: fiind dată o normă vectorială  $\|\cdot\|$  pe  $\mathbb{C}^n$ , aplicația  $\|\cdot\| : \mathbb{C}^{n \times n} \rightarrow \mathbb{R}$

$$\|A\| = \sup_{\substack{v \in \mathbb{C}^n \\ v \neq 0}} \frac{\|Av\|}{\|v\|} = \sup_{\substack{v \in \mathbb{C}^n \\ \|v\| \leq 1}} \|Av\| = \sup_{\substack{v \in \mathbb{C}^n \\ \|v\|=1}} \|Av\|$$

este o normă matricială numită *normă matricială subordonată* (normei vectoriale date) sau *normă naturală* (indusă de norma dată).

**Observația 4.1.3.** 1. Aceste norme matriciale subordonate sunt un caz particular al normei unei aplicații liniare  $A : \mathbb{K}^m \rightarrow \mathbb{K}^n$ .

2. O normă subordonată verifică  $\|I\| = 1$ .
3. Dacă matricea  $A$  este reală, marginea superioară a raportului  $\|Av\|/\|v\|$  este atinsă pentru vectori reali (vezi teorema următoare).  $\diamond$

Să calculăm acum normele subordonate normelor vectoriale  $\|\cdot\|_1, \|\cdot\|_2, \|\cdot\|_\infty$ .

**Teorema 4.1.4.** Fie  $A \in \mathbb{K}^{n \times n}(\mathbb{C})$ . Atunci

$$\begin{aligned} \|A\|_1 &:= \sup_{v \in \mathbb{C}^n \setminus \{0\}} \frac{\|Av\|_1}{\|v\|_1} = \max_j \sum_i |a_{ij}|, \\ \|A\|_\infty &:= \sup_{v \in \mathbb{C}^n \setminus \{0\}} \frac{\|Av\|_\infty}{\|v\|_\infty} = \max_i \sum_j |a_{ij}|, \\ \|A\|_2 &:= \sup_{v \in \mathbb{C}^n \setminus \{0\}} \frac{\|Av\|_2}{\|v\|_2} = \sqrt{\rho(A^*A)} = \sqrt{\rho(AA^*)} = \|A^*\|_2. \end{aligned}$$

Norma  $\|\cdot\|_2$  este invariantă la transformările unitare, adică

$$UU^* = I \Rightarrow \|A\|_2 = \|AU\|_2 = \|UA\|_2 = \|U^*AU\|_2.$$

Altfel spus, dacă  $A$  este normală ( $AA^* = A^*A$ ), atunci  $\|A\|_2 = \rho(A)$ .

*Demonstrație.* Pentru orice vector  $v$  avem

$$\begin{aligned} \|Av\|_1 &= \sum_i \left| \sum_j a_{ij} v_j \right| \leq \sum_j |v_j| \sum_i |a_{ij}| \leq \\ &\leq \left( \max_j \sum_i |a_{ij}| \right) \|v\|_1. \end{aligned}$$

Pentru a arăta că  $\max_j \sum_i |a_{ij}|$  este efectiv cel mai mic număr  $\alpha$  pentru care are loc  $\|Av\|_1 \leq \alpha \|v\|_1$ ,  $\forall v \in \mathbb{C}^n$ , să construim un vector  $u$  (care depinde de  $A$ ) astfel încât

$$\|Au\|_1 = \left\{ \max_j \sum_i |a_{ij}| \right\} \|u\|_1.$$

Dacă  $j_0$  este un indice ce verifică

$$\max_j \sum_i |a_{ij}| = \sum_i |a_{ij_0}|,$$

atunci vectorul  $u$  are componentele  $u_i = 0$  pentru  $i \neq j_0$ ,  $u_{j_0} = 1$ .

La fel

$$\|Av\|_\infty = \max_i \left| \sum_j a_{ij} v_j \right| \leq \left( \max_i \sum_j |a_{ij}| \right) \|v\|_\infty.$$

Fie  $i_0$  un indice ce verifică

$$\max_i \sum_j |a_{ij}| = \sum_j |a_{i_0 j}|.$$

Vectorul  $u$  de componente  $u_j = \frac{\overline{a_{i_0 j}}}{|a_{i_0 j}|}$  dacă  $a_{i_0 j} \neq 0$ ,  $u_j = 1$  dacă  $a_{i_0 j} = 0$ , verifică

$$\|Au\|_\infty = \left\{ \max_i \sum_j |a_{ij}| \right\} \|u\|_\infty.$$

Deoarece  $AA^*$  este hermitiană, există o descompunere proprie  $AA^* = Q\Lambda Q^*$ , unde  $Q$  este o matrice unitară (ale cărei coloane sunt vectori proprii) și  $\Lambda$  este matricea diagonală a valorilor proprii, care trebuie să fie toate reale. Dacă ar exista o valoare proprie negativă și  $q$  ar fi vectorul propriu corespunzător, am avea  $0 \leq \|Aq\|_2^2 = q^T A^T A q = q^T \lambda q = \lambda \|q\|_2^2$ . Deci

$$\begin{aligned} \|A\|_2 &= \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{x \neq 0} \frac{(x^* A^* Ax)^{1/2}}{\|x\|_2} = \max_{x \neq 0} \frac{(x^* Q\Lambda Q^* x)^{1/2}}{\|x\|_2} \\ &= \max_{x \neq 0} \frac{((Q^* x)^* \Lambda Q^* x)^{1/2}}{\|Q^* x\|_2} = \max_{y \neq 0} \frac{(y^* \Lambda y)^{1/2}}{\|y\|_2} = \max_{y \neq 0} \sqrt{\frac{\sum \lambda_i y_i^2}{\sum y_i^2}} \\ &\leq \max_{y \neq 0} \sqrt{\lambda_{\max}} \sqrt{\frac{\sum y_i^2}{\sum y_i^2}}, \end{aligned}$$

egalitatea are loc dacă  $y$  este o coloană convenabil aleasă a matricei identitate.

Să arătăm că  $\rho(A^* A) = \rho(AA^*)$ . Dacă  $\rho(A^* A) > 0$ , există  $p$  astfel încât  $p \neq 0$ ,  $A^* Ap = \rho(A^* A)p$  și  $Ap \neq 0$  ( $\rho(A^* A) > 0$ ). Cum  $Ap \neq 0$  și  $AA^*(Ap) = \rho(A^* A)Ap$ , rezultă că  $0 < \rho(A^* A) \leq \rho(AA^*)$  și deci  $\rho(AA^*) = \rho(A^* A)$ , căci  $(A^*)^* = A$ . Dacă  $\rho(A^* A) = 0$ ,

avem  $\rho(AA^*) = 0$ . Deci, în toate cazurile  $\|A\|_2^2 = \rho(A^*A) = \rho(AA^*) = \|A^*\|_2^2$ . Invarianța normei  $\|\cdot\|_2$  la transformări unitare nu este decât traducerea egalităților

$$\rho(A^*A) = \rho(U^*A^*AU) = \rho(A^*U^*UA) = \rho(U^*A^*UU^*AU).$$

În fine, dacă  $A$  este normală, există o matrice  $U$  astfel încât  $U^*AU = \text{diag}(\lambda_i(A)) \stackrel{\text{def}}{=} \Lambda$ . În aceste condiții

$$A^*A = (U\Lambda U^*)^*U\Lambda U = U\Lambda^*\Lambda U^*,$$

ceea ce ne arată că

$$\rho(A^*A) = \rho(\Lambda^*\Lambda) = \max_i |\lambda_i(A)|^2 = (\rho(A))^2.$$

□

**Observația 4.1.5.** 1) Dacă  $A$  este hermitiană sau simetrică (deci normală),

$$\|A\|_2 = \rho(A).$$

2) Dacă  $A$  este unitară sau ortogonală (deci normală),

$$\|A\|_2 = \sqrt{\rho(A^*A)} = \sqrt{\rho(I)} = 1.$$

3) Teorema 4.1.4 ne spune că matricele normale și norma  $\|\cdot\|_2$  verifică

$$\|A\|_2 = \rho(A).$$

(4) Norma  $\|\cdot\|_\infty$  se mai numește norma Cebășev sau m-normă, norma  $\|\cdot\|_1$  norma lui Minkowski sau l-normă, iar norma  $\|\cdot\|_2$  normă euclidiană. ◇

**Teorema 4.1.6.** (1) Fie  $A$  o matrice pătratică oarecare și  $\|\cdot\|$  o normă matricială oarecare (subordonată sau nu). Atunci

$$\rho(A) \leq \|A\|. \quad (4.1.1)$$

(2) Fiind dată o matrice  $A$  și un număr  $\varepsilon > 0$ , există cel puțin o normă matricială subordonată astfel încât

$$\|A\| \leq \rho(A) + \varepsilon. \quad (4.1.2)$$

*Demonstrație.* (1) Fie  $p$  un vector ce verifică  $p \neq 0$ ,  $Ap = \lambda p$ ,  $|\lambda| = \rho(A)$  și  $q$  un vector astfel încât  $pq^T \neq 0$ . Deoarece

$$\rho(A)\|pq^T\| = \|\lambda pq^T\| = \|Apq^T\| \leq \|A\|\|pq^T\|,$$

rezultă (4.1.1).

(2) Fie  $A$  o matrice dată. Există o matrice inversabilă  $U$  astfel încât  $U^{-1}AU$  este triunghiulară superior (de fapt  $U$  este unitară). De exemplu

$$U^{-1}AU = \begin{pmatrix} \lambda_1 & t_{12} & t_{13} & \dots & t_{1,n} \\ & \lambda_2 & t_{23} & \dots & t_{2,n} \\ & & \ddots & & \vdots \\ & & & \lambda_{n-1} & t_{n-1,n} \\ & & & & \lambda_n \end{pmatrix}.$$

scalarii  $\lambda_i$  fiind valorile proprii ale matricei  $A$ . (Pentru demonstrație a se vedea teorema 9.2.6 din capitolul 9.) Fiecare scalar  $\delta \neq 0$  îi asociem matricea

$$D_\delta = \text{diag}(1, \delta, \delta^2, \dots, \delta^{n-1}),$$

astfel ca

$$(UD_\delta)^{-1}A(UD_\delta) = \begin{pmatrix} \lambda_1 & \delta t_{12} & \delta^2 t_{13} & \dots & \delta^{n-1} t_{1n} \\ & \lambda_2 & \delta t_{23} & \dots & \delta^{n-2} t_{2n} \\ & & \ddots & & \vdots \\ & & & \lambda_{n-1} & \delta t_{n-1,n} \\ & & & & \lambda_n \end{pmatrix}.$$

Fiind dat  $\varepsilon > 0$ , fixăm  $\delta$  astfel ca

$$\sum_{j=i+1}^n |\delta^{j-i} t_{ij}| \leq \varepsilon, \quad 1 \leq i \leq n-1.$$

Atunci aplicația

$$\|\cdot\| : B \in \mathbb{K}^{n \times n} \rightarrow \|B\| = \|(UD_\delta)^{-1}B(UD_\delta)\|_\infty,$$

care depinde de  $A$  și de  $\varepsilon$  răspunde problemei. Într-adevăr, avem pe de o parte

$$\|A\| \leq \rho(A) + \varepsilon,$$

conform alegерii lui  $\delta$  și definiției normei  $\|\cdot\|_\infty$  ( $\|c_{ij}\|_\infty = \max_i \sum_j |c_{ij}|$ ) și pe de altă parte ea este o normă matricială subordonată normei vectoriale

$$v \in \mathbb{K}^n \rightarrow \|(UD_\delta)^{-1}v\|_\infty.$$

□

Un exemplu important de normă nesubordonată este dat de teorema următoare. Fie  $tr(X)$  urma matricei  $X$  (suma elementelor de pe diagonală).

**Teorema 4.1.7.** *Aplicația  $\|\cdot\|_E : \mathbb{K}^{n \times n} \rightarrow \mathbb{R}$  definită prin*

$$\|A\|_E = \left\{ \sum_i \sum_j |a_{ij}|^2 \right\}^{1/2} = \{tr(A^* A)\}^{1/2}$$

este o normă matricială nesubordonată, invariantă la transformările unitare

$$UU^* = I \Rightarrow \|A\|_E = \|AU\|_E = \|UA\|_E = \|U^*AU\|_E$$

și care verifică

$$\|A\|_2 \leq \|A\|_E \leq \sqrt{n}\|A\|_2, \forall A \in \mathbb{K}^{n \times n}.$$

*Demonstrație.*  $\|\cdot\|_E$  este norma euclidiană pe  $\mathbb{K}^{n \times n}$  de dimensiune  $n^2$ . Proprietatea (NM4) se demonstrează cu inegalitatea Cauchy-Buniakowski-Schwarz

$$\begin{aligned} \|AB\|_E^2 &= \sum_{i,j} \left| \sum_k a_{jk} b_{kj} \right|^2 \leq \sum_{i,j} \left\{ \sum_k |a_{ik}|^2 \right\} \left\{ \sum_l |b_{lj}|^2 \right\} = \\ &= \left\{ \sum_{i,k} |a_{ik}|^2 \right\} \left\{ \sum_{j,l} |b_{lj}|^2 \right\} = \|A\|_E^2 \|B\|_E^2. \end{aligned}$$

Această normă nu este subordonată, deoarece  $\|I\|_E = \sqrt{n}$ . Dacă  $U$  este o matrice unitară

$$\|A\|_E^2 = \text{tr}(A^* A) = \text{tr}(U^* A^* AU) =$$

$$= \|AU\|_E^2 = \text{tr}(U^* A^* UA) = \|UA\|_E.$$

În fine, inegalitățile din enunț rezultă din inegalitățile

$$\rho(A^* A) \leq \text{tr}(A^* A) \leq n\rho(A^* A).$$

□

Norma  $\|\cdot\|_E$  se numește *normă Frobenius*.

**Theoremă 4.1.8.** Fie  $B$  o matrice pătratică. Următoarele afirmații sunt echivalente:

- (1)  $\lim_{k \rightarrow \infty} B^k = 0$ ;
- (2)  $\lim_{k \rightarrow \infty} B^k v = 0, \forall v \in \mathbb{K}^n$ ;
- (3)  $\rho(B) < 1$ ;
- (4) Există o normă matricială subordonată astfel încât  $\|B\| < 1$ .

*Demonstrație.* (1)  $\Rightarrow$  (2)

$$\|B^k v\| \leq \|B^k\| \|v\| \Rightarrow \lim_{k \rightarrow \infty} B^k v = 0$$

(2)  $\Rightarrow$  (3) Dacă  $\rho(B) \geq 1$  putem găsi  $p$  astfel încât  $p \neq 0, Bp = \lambda p, |\lambda| \geq 1$ . Atunci sirul de vectori  $(B^k p)_{k \in \mathbb{N}}$  ar putea să nu conveargă către 0.

(3)  $\Rightarrow$  (4)  $\rho(B) < 1 \Rightarrow \exists \|\cdot\|$  astfel încât  $\|B\| \leq \rho(B) + \varepsilon, \forall \varepsilon > 0$  deci  $\|B\| < 1$ .

(4)  $\Rightarrow$  (1) Este suficient să aplicăm inegalitatea  $\|B^k\| \leq \|B\|^k$ . □

Funcția `norm` se poate aplica și matricelor. Ea se apelează sub forma `norm(A, p)` unde  $A$  este o matrice, iar  $p=1, 2, \text{inf}$  pentru o  $p$ -normă sau  $p='fro'$  pentru norma Frobenius. Exemplu:

```
>> A=[1:3;4:6;7:9]
A =
    1     2     3
    4     5     6
    7     8     9
>> [norm(A,1) norm(A,2) norm(A,inf) norm(A,'fro')]
ans =
    18.0000    16.8481    24.0000    16.8819
```

## 4.2. Condiționarea unui sistem liniar

Fie sistemul (exemplul este datorat lui Wilson)

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix},$$

cu soluția  $(1, 1, 1, 1)^T$  și considerăm sistemul perturbat, în care membrul drept este foarte puțin modificat, matricea rămânând neschimbată

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} x_1 + \delta x_1 \\ x_2 + \delta x_2 \\ x_3 + \delta x_4 \\ x_4 + \delta x_4 \end{pmatrix} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix},$$

cu soluția  $(9.2, -12.6, 4.5, -1.1)^T$ . Altfel spus, o eroare de  $1/200$  în date (aici componente din membrul drept) atrage o eroare relativă de  $10/1$  asupra rezultatului, deci o mărire a erorii relative de ordin 2000!

Considerăm acum sistemul având de această dată matricea perturbată

$$\begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix} \begin{pmatrix} x_1 + \Delta x_1 \\ x_2 + \Delta x_2 \\ x_3 + \Delta x_4 \\ x_4 + \Delta x_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix},$$

cu soluția  $(-81, 137, -34, 22)^T$ . Din nou, o variație mică în datele de intrare (aici, elementele matricei) modifică complet rezultatul (soluția sistemului liniar). Matricea are un aspect „bun“, ea este simetrică, determinantul ei este 1, iar inversa ei este

$$\begin{pmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{pmatrix},$$

care este de asemenea simptică.

Să analizăm aceste fenomene. În primul caz se dă o matrice *inversabilă*  $A$  și se compară soluțiile *exacte*  $x$  și  $x + \delta x$  ale sistemelor

$$\begin{aligned} Ax &= b \\ A(x + \delta x) &= b + \delta b. \end{aligned}$$

Fie  $\|\cdot\|$  o normă vectorială oarecare și  $\|\cdot\|$  norma matricială subordonată. Din egalitățile  $\delta x = A^{-1}\delta b$  și  $b = Ax$  se deduce

$$\|\Delta x\| \leq \|A^{-1}\| \|\delta b\|, \quad \|b\| \leq \|A\| \|x\|.$$

Eroarea relativă a rezultatului  $\frac{\|\delta x\|}{\|x\|}$  este majorată în funcție de eroarea relativă a datelor prin

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|\delta b\|}{\|b\|}.$$

În al doilea caz, când matricea variază, avem de comparat soluțiile exacte ale sistemelor

$$\begin{aligned} Ax &= b \\ (A + \Delta A)(x + \Delta x) &= b. \end{aligned}$$

Din egalitatea  $\Delta x = -A^{-1}\Delta A(x + \Delta x)$  se deduce

$$\|\Delta x\| \leq \|A^{-1}\| \|\Delta A\| \|x + \Delta x\|$$

care se mai poate scrie

$$\frac{\|\Delta x\|}{\|x + \Delta x\|} \leq \|A\| \|A^{-1}\| \frac{\|\Delta A\|}{\|A\|}.$$

Dacă  $A$  este nesingulară, numărul

$$\text{cond}(A) = \|A\| \|A^{-1}\| \quad (4.2.1)$$

se numește *număr de condiționare* al matricei  $A$ . Dacă  $A$  este singulară convenim să luăm  $\text{cond}(A) = \infty$ .

Se poate da o estimare a numărului de condiționare în care să intervină simultan și perturbațiile lui  $A$  și ale lui  $b$ . Considerăm sistemul parametrizat, cu parametrul  $t$

$$(A + t\Delta A)x(t) = b + t\Delta b, \quad x(0) = x.$$

Matricea  $A$  fiind nesingulară, funcția  $x$  este diferențială în  $t = 0$ :

$$\dot{x}(0) = A^{-1}(\Delta b - \Delta Ax).$$

Dezvoltarea Taylor a lui  $x(t)$  este dată de

$$x(t) = x + t\dot{x}(0) + O(t^2).$$

Rezultă că eroarea absolută poate fi estimată utilizând

$$\begin{aligned}\|\Delta x(t)\| &= \|x(t) - x\| \leq |t| \|x'(0)\| + O(t^2) \\ &\leq |t| \|A^{-1}\| (\|\Delta b\| + \|\Delta A\| \|x\|) + O(t^2)\end{aligned}$$

și datorită lui  $\|b\| \leq \|A\| \|x\|$  obținem pentru eroarea relativă

$$\begin{aligned}\frac{\|\Delta x(t)\|}{\|x\|} &\leq |t| \|A^{-1}\| \left( \frac{\|\Delta b\|}{\|x\|} + \|\Delta A\| \right) + O(t^2) \\ &\leq \|A\| \|A^{-1}\| |t| \left( \frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right) + O(t^2).\end{aligned}\quad (4.2.2)$$

Introducând notațiile

$$\rho_A(t) := |t| \frac{\|\Delta A\|}{\|A\|}, \quad \rho_b(t) := |t| \frac{\|\Delta b\|}{\|b\|}$$

pentru erorile relative în  $A$  și  $b$ , estimarea erorii (4.2.2) se scrie sub forma

$$\frac{\|\Delta x(t)\|}{\|x\|} \leq \text{cond}(A) (\rho_A + \rho_b) + O(t^2).$$

MATLAB are mai multe funcții pentru calculul sau estimarea numărului de condiționare.

- `cond(A, p)`, unde valoarea implicită este  $p=2$ . Pentru  $p=2$  se utilizează `svd`, iar pentru  $p=1$ , `Inf` se utilizează `inv`.
- `condeest(A)` estimează  $\text{cond}_1 A$ . Utilizează `lu` și un algoritm recent al lui Higham și Tisseur [33]. Potrivită pentru matrice mari, rare.
- `rcond(A)` estimează  $1/\text{cond}_1 A$ . Utilizează `lu(A)` și un algoritm utilizat de LINPACK și LAPACK. De interes mai mult istoric.

**Exemplul 4.2.1 (Exemple de matrice prost condiționate).** Matricea lui Hilbert <sup>1</sup>  $H_n = (h_{ij})$ , cu  $h_{ij} = \frac{1}{i+j-1}$ ,  $i, j = \overline{1, n}$  are ordinul de mărime al numărului de condiționare relativ



<sup>1</sup>

David Hilbert (1862-1943) a fost cel mai important reprezentant al școlii matematice din Göttingen. Contribuțiile sale fundamentale în aproape toate domeniile matematicii — algebră, teoria numerelor, geometrie, ecuații integrale, calcul variațional și fundamentele matematicii — și în particular cele 23 de probleme celebre pe care le-a propus în 1900 la un congres internațional al matematicienilor de la Paris, au dat un nou impuls și o nouă direcție matematicii din secolul al XX-lea.

la norma euclidiană dat de (Szegő<sup>2</sup>)

$$\text{cond}_2(H_n) \sim \frac{(\sqrt{2} + 1)^{4n+4}}{2^{14/4}\sqrt{\pi n}}.$$

Pentru diverse valori ale lui  $n$  se obține

$n$	$\text{cond}_2(H_n)$
10	$1.6 \cdot 10^{13}$
20	$2.45 \cdot 10^{28}$
40	$7.65 \cdot 10^{58}$

Un alt exemplu este matricea Vandermonde. Dacă elementele sunt echidistante în  $[-1,1]$ , atunci

$$\text{cond}_{\infty}(V_n) \sim \frac{1}{\pi} e^{-\frac{\pi}{4}} e^{n(\frac{\pi}{4} + \frac{1}{2} \ln 2)},$$

iar pentru  $t_i = \frac{1}{i}$ ,  $i = \overline{1, n}$  avem

$$\text{cond}_{\infty}(V_n) > n^{n+1}. \quad \diamond$$

Să verificăm condiționarea matricelor din exemplul 4.2.1. Pentru matricea Hilbert am folosit secvența (fișierul `testcondhilb.m`):

```
fprintf(' n      cond_2           est. cond       teoretic\n')
for n=[10:15,20,40]
    H=hilb(n);
    et=(sqrt(2)+1)^(4*n+4)/(2^(14/4)*sqrt(pi*n));
    x=[n, norm(H)*norm(invhilb(n)), condest(H), et];
    fprintf('%d %g %g %g\n',x)
end
```

Se obțin următoarele rezultate:

n	cond_2	est. cond	teoretic
10	1.60263e+013	3.53537e+013	1.09635e+015
11	5.23068e+014	1.23037e+015	3.55105e+016
12	1.71323e+016	3.79926e+016	1.15496e+018
13	5.62794e+017	4.25751e+017	3.76953e+019
14	1.85338e+019	7.09955e+018	1.23395e+021
15	6.11657e+020	7.73753e+017	4.04966e+022
20	2.45216e+028	4.95149e+018	1.58658e+030
40	7.65291e+058	7.02056e+019	4.69897e+060

2



Gabor Szegő (1895-1985) Unul dintre cei mai importanți matematicieni maghiari din secolul al XX-lea. Contribuții importante în domeniul problemelor extremale și matricelor Toeplitz.

Verificarea pentru matrice Vandermonde cu elemente echidistante în [-1,1] s-a realizat cu fișierul condvander2.m:

```
warning off
fprintf(' n cond_inf      estimare cond  teoretic\n')
for n=[10,20,40,80]
    t=linspace(-1,1,n);
    V=vander(t);
    et=1/pi*exp(-pi/4)*exp(n*(pi/4+1/2*log(2)));
    x=[n, norm(V,inf)*norm(inv(V),inf), cond(V), et];
    fprintf('%d %e %e %e\n',x)
end
warning on
```

Dăm și rezultatele obținute:

n	cond_inf	estimare cond	teoretic
10	2.056171e+004	1.362524e+004	1.196319e+004
20	1.751063e+009	1.053490e+009	9.861382e+008
40	1.208386e+019	6.926936e+018	6.700689e+018
80	8.059962e+038	8.475473e+038	3.093734e+038

Pentru matricea Vandermonde cu elemente de forma  $1/i$  am folosit secvența (fișierul condvander.m)

```
warning off
fprintf(' n cond_inf      estimare cond  teoretic\n')
for n=10:15
    t=1./(1:n);
    V=vander(t);
    x=[n, norm(V,inf)*norm(inv(V),inf), cond(V), n^(n+1)];
    fprintf('%d %e %e %e\n',x)
end
warning on
```

și am obținut:

n	cond_inf	estimare cond	teoretic
10	5.792417e+011	5.905580e+011	1.000000e+011
11	2.382382e+013	2.278265e+013	3.138428e+012
12	1.060780e+015	9.692982e+014	1.069932e+014
13	5.087470e+016	4.732000e+016	3.937376e+015
14	2.615990e+018	2.419006e+018	1.555681e+017
15	1.436206e+020	1.294190e+020	6.568408e+018

În toate aceste exemple, s-a folosit comanda warning off pentru a inhiba afișarea mesajelor de avertisment de forma:

```
Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = ...
```

## 4.3. Metode exacte

### 4.3.1. Metoda eliminării a lui Gauss

Să considerăm sistemul liniar de  $n$  ecuații cu  $n$  necunoscute

$$Ax = b, \quad (4.3.1)$$

unde  $A \in \mathbb{K}^{n \times n}$ ,  $b \in \mathbb{K}^{n \times 1}$  sunt date, iar  $x \in \mathbb{K}^{n \times 1}$  este necunoscută, sau scris detaliat

$$\left\{ \begin{array}{ll} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 & (E_1) \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 & (E_2) \\ \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n & (E_n) \end{array} \right. \quad (4.3.2)$$

Metoda eliminării a lui Gauss<sup>3</sup> are două etape:

- e1) transformarea sistemului dat într-unul echivalent, triunghiular;
- e2) rezolvarea sistemului triunghiular prin substituție inversă.

La rezolvarea sistemului (4.3.1) sau (4.3.2) sunt permise următoarele operații:

1. Ecuația  $E_i$  poate fi înmulțită cu  $\lambda \in \mathbb{K}^*$ . Această operație se va nota cu  $(\lambda E_i) \rightarrow (E_i)$ .
2. Ecuația  $E_j$  poate fi înmulțită cu  $\lambda \in \mathbb{K}^*$  și adunată la ecuația  $E_i$ , iar rezultatul utilizat în locul lui  $E_i$ , notație  $(E_i + \lambda E_j) \rightarrow (E_i)$ .
3. Ecuațiile  $E_i$  și  $E_j$  pot fi interschimbate, notație  $(E_i) \longleftrightarrow (E_j)$ .

Pentru a exprima convenabil operațiile necesare pentru transformarea sistemului în unul triunghiular vom lucra cu matricea extinsă:

$$\tilde{A} = [A, b] = \left[ \begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & a_{1,n+1} \\ a_{21} & a_{22} & \dots & a_{2n} & a_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & a_{n,n+1} \end{array} \right]$$

Johann Carl Friedrich Gauss (1777-1855) a fost unul dintre cei mai mari matematicieni ai secolului al nouăsprezecelea și probabil ai tuturor timpurilor. A trăit aproape toată viața în Göttingen, unde a fost directorul observatorului astronomic 40 de ani. În timpul studenției la Göttingen, Gauss a descoperit că poligonul cu 17 laturi poate fi construit cu rigla și compasul, rezolvând astfel o problemă deschisă a antichității. În dizertația sa a dat prima demonstrație a teoremei fundamentale a algebrei. A avut contribuții fundamentale în teoria numerelor, geometrie diferențială și neeuclidiană, funcții eliptice și hipergeometrice, mecanică cerească și geodezie și diverse ramuri ale fizicii, în special magnetism și optică. Eforturile sale de calcul în mecanica cerească și geodezie, bazate pe principiul celor mai mici pătrate, au necesitat rezolvarea manuală a unor sisteme de ecuații liniare mari, la care a utilizat metodele cunoscute astăzi sub numele de eliminare gaussiană și metoda relaxării. Lucrările lui Gauss în domeniul cuadraturilor numerice continuă munca predecesorilor săi Newton și Cotes.



cum  $a_{i,n+1} = b_i$ .

Presupunând că  $a_{11} \neq 0$ , vom elimina coeficienții lui  $x_1$  din  $E_j$ , pentru  $j = \overline{2, n}$  prin operația  $(E_j - (a_{j1}/a_{11})E_1) \rightarrow (E_j)$ . Vom proceda apoi la fel cu coeficienții lui  $x_i$ , pentru  $i = \overline{2, n-1}$ ,  $j = \overline{i+1, n}$ . Aceasta este posibil dacă  $a_{ii} \neq 0$ .

Procedura poate fi descrisă astfel: se formează o secvență de matrice extinse  $\tilde{A}^{(1)}, \tilde{A}^{(2)}, \dots, \tilde{A}^{(n)}$ , unde  $\tilde{A}^{(1)} = A$  și  $\tilde{A}^{(k)}$  are elementele  $a_{ij}^{(k)}$  date de

$$\left( E_i - \frac{a_{i,k-1}^{(k-1)}}{a_{k-1,k-1}^{(k-1)}} E_{k-1} \right) \rightarrow (E_i)$$

sau desfășurat

$$a_{i,j}^{(k)} = \begin{cases} a_{ij}^{(k-1)}, & \text{pentru } i = \overline{1, k-1}, j = \overline{1, n+1} \\ 0, & \text{pentru } i = \overline{k, n}, j = \overline{1, k-1} \\ a_{ij}^{(k-1)} - \frac{a_{i,k-1}^{(k-1)}}{a_{k-1,k-1}^{(k-1)}} a_{k-1,j}^{(k-1)}, & \text{pentru } i = \overline{k, n}, j = \overline{k, n+1} \end{cases}$$

**Observația 4.3.1.** Notația  $a_{ij}^{(p)}$  semnifică valoarea elementului  $a_{ij}$  la pasul  $p$ .

◊

Astfel

$$\tilde{A}^{(k)} = \left[ \begin{array}{ccccccccc} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \dots & a_{1,k-1}^{(1)} & a_{1k}^{(1)} & \dots & a_{1n}^{(1)} & a_{1,n+1}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \dots & a_{2,k-1}^{(2)} & a_{2,k}^{(2)} & \dots & a_{2n}^{(2)} & a_{2,n+1}^{(2)} \\ \vdots & \ddots & \dots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & & & a_{k-1,k-1}^{(k-1)} & a_{k-1,k}^{(k-1)} & \dots & a_{k-1,n}^{(k-1)} & a_{k-1,n+1}^{(k-1)} \\ \vdots & & & 0 & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} & a_{k,n+1}^{(k)} \\ \vdots & & & \vdots & \vdots & \dots & \vdots & \vdots & \vdots \\ 0 & \dots & \dots & \dots & 0 & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} & a_{n,n+1}^{(k)} \end{array} \right]$$

reprezintă sistemul liniar echivalent în care variabila  $x_{k-1}$  a fost eliminată din ecuațiile  $E_k, E_{k+1}, \dots, E_n$ . Sistemul corespunzător lui  $\tilde{A}^{(n)}$  este un sistem triunghiular echivalent cu sistemul inițial

$$\left\{ \begin{array}{l} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \dots + a_{1n}^{(1)}x_n = a_{1,n+1}^{(1)} \\ a_{22}^{(2)}x_2 + \dots + a_{2n}^{(2)}x_n = a_{2,n+1}^{(2)} \\ \vdots \\ a_{nn}^{(n)}x_n = a_{n,n+1}^{(n)} \end{array} \right.$$

Se obține

$$x_n = \frac{a_{n,n+1}^{(n)}}{a_{nn}^{(n)}}$$

și în general

$$x_i = \frac{1}{a_{ii}^{(i)}} \left( a_{i,n+1}^{(i)} - \sum_{j=i+1}^n a_{ij}^{(i)} x_j \right), \quad i = \overline{n-1, 1}.$$

Pentru ca procedura să fie aplicabilă trebuie ca  $a_{ii}^{(i)} \neq 0$ ,  $i = \overline{1, n}$ . Elementul  $a_{ii}^{(i)}$  se numește *element pivot*. Dacă în timpul algoritmului de eliminare gaussiană la pasul  $k$  se obține pivotul  $a_{kk}^{(k)} = 0$ , se poate face interschimbarea de linii  $(E_k) \leftrightarrow (E_p)$ , unde  $k + 1 \leq p \leq n$  este cel mai mic întreg cu proprietatea  $a_{pk}^{(k)} \neq 0$ . În practică sunt necesare astfel de operații chiar și în cazul când pivotul este nenul. Aceasta din cauză că un pivot mic în comparație cu elementele care urmează după el în aceeași coloană duce la erori de rotunjire substanțiale. Aceasta se poate remedia alegând ca pivot elementul din aceeași coloană care este situat sub diagonală și care are cea mai mare valoare absolută, adică determinând  $p$  astfel încât

$$|a_{pk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|$$

și apoi făcând  $(E_k) \leftrightarrow (E_p)$ . Această tehnică se numește *pivotare maximală pe coloană* sau *pivotare parțială*.

O altă tehnică care reduce erorile și preîntâmpină anulările în aritmetică flotantă este *pivotarea scalată pe coloană*. La primul pas se definește un factor de scalare pentru fiecare linie

$$s_i = \max_{j=1, n} |a_{ij}| \text{ sau } s_i = \sum_{j=1}^n |a_{ij}|.$$

Dacă există  $i$  astfel încât  $s_i = 0$ , matricea este singulară.

La pașii următori se determină ce interschimbări se vor realiza. La pasul  $i$  se determină cel mai mic întreg  $p$ ,  $i \leq p \leq n$ , pentru care

$$\frac{|a_{pi}|}{s_p} = \max_{1 \leq j \leq n} \frac{|a_{ji}|}{s_j}$$

și apoi  $(E_i) \leftrightarrow (E_p)$ . Efectul scalării este de a ne asigura că elementul cel mai mare din fiecare coloană are mărimea relativă 1 înainte de a realiza comparațiile pentru interschimbarea liniilor. Scalarea se realizează doar în scop de comparație, aşa că împărțirea cu factorul de scalare nu produce erori de rotunjire. O a treia metodă este cea a pivotării totale (sau maximale). La această metodă la pasul  $k$  se determină

$$\max\{|a_{ij}|, i = \overline{k, n}, j = \overline{k, n}\}$$

și se realizează și interschimbări de linii și de coloane.

Pivotarea a fost introdusă de Goldstine și von Neumann în 1947 [28].

Dacă matricea  $A$  este singulară și are rangul  $p - 1$  atunci se obține

$$\tilde{A}^{(p)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1,p-1}^{(1)} & a_{1p}^{(1)} & \dots & a_{1n}^{(1)} & a_{1,n+1}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2,p-1}^{(2)} & a_{2p}^{(2)} & \dots & a_{2n}^{(2)} & a_{2,n+1}^{(2)} \\ \dots & \dots \\ 0 & 0 & \dots & a_{p-1,p-1}^{(p-1)} & a_{p-1,p}^{(p-1)} & \dots & a_{p-1,n}^{(p-1)} & a_{p-1,n+1}^{(p-1)} \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & a_{p,n+1}^{(p)} \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 & a_{n,n+1}^{(n)} \end{bmatrix}.$$

Dacă  $a_{i,n+1}^{(p)} = b_i^{(p)} = 0$ ,  $i = \overline{p, n}$ , atunci sistemul este compatibil nedeterminat, iar dacă există  $q \in \{p, \dots, n\}$  astfel încât  $a_{q,n+1}^{(p)} = b_q^{(p)} \neq 0$  sistemul este incompatibil.

Deci metodele de eliminare de tip Gauss permit atât rezolvarea cât și discuția sistemelor de ecuații liniare.

**Observația 4.3.2.** Dăm câteva sugestii care pot duce la îmbunătățirea timpului de execuție.

1. La metodele de pivotare nu este nevoie să se realizeze fizic interschimbarea de linii și/sau coloane, ci se pot păstra unul sau doi vectori de permutări  $p(q)$  cu semnificația  $p[i](q[i])$  este linia (coloana) care a fost interschimbată cu linia (coloana)  $i$ .
2. Elementele de sub diagonală (care devin 0) pot să nu fie calculate.
3. Matricea  $A$  se poate inversa rezolvând sistemul  $Ax = e_k$ ,  $k = \overline{1, n}$ , unde  $e_k$  sunt vectorii bazei canonice din  $\mathbb{K}^n$ . Metoda se numește *metoda ecuațiilor simultane*. ◇

Să analizăm metoda eliminării a lui Gauss. Descrierea ei apare în sursa MATLAB 4.1. Ca măsură a complexității vom considera numărul de operații aritmetice flotante, desemnată prescurtat prin *flops*.

În corpul ciclului `for` interior avem  $2n - 2i + 3$  flops, deci pentru întreg ciclul  $(n - i)(2n - 2i + 3)$  flops. Pentru ciclul exterior avem un total general de

$$\sum_{i=1}^{n-1} (n - i)(2n - 2i + 3) \sim \frac{2n^3}{3}.$$

Pentru substituția inversă avem  $\Theta(n^2)$  flops.

Total general,  $\Theta(n^3)$ .

## 4.4. Metode bazate pe factorizare

### 4.4.1. Descompunerea LU

**Teorema 4.4.1.** Dacă eliminarea gaussiană pentru sistemul  $Ax = b$  se poate realiza fără interschimbări de linii, atunci  $A$  se poate factoriza în  $A = LU$ , unde  $L$  este triunghiulară inferior, iar  $U$  triunghiulară superior. Perechea  $(L, U)$  se numește descompunerea  $LU$  a matricei  $A$ .

---

**Sursa MATLAB 4.1** Rezolvă sistemul  $Ax = b$  prin metoda eliminării a lui Gauss cu pivot scalat pe coloană

---

```
function x=Gausselim(A,b)
%GAUSSELIM - eliminare gaussiana cu pivot scalat pe coloana
%apel x=Gausselim2(A,b)
%A -matricea, b- vectorul termenilor liberi

[l,n]=size(A);
x=zeros(size(b));
s=sum(abs(A),2);
A=[A,b]; %matricea extinsa
piv=1:n;
%Eliminare
for i=1:n-1
    [u,p]=max(abs(A(i:n,i))./s(i:n)); %pivotare
    p=p+i-1;
    if u==0, error('nu exista solutie unica'), end
    if p~=i %interschimbare linii
        piv([i,p])=piv([p,i]);
    end
    for j=i+1:n
        m=A(piv(j),i)/A(piv(i),i);
        A(piv(j),i+1:n+1)=A(piv(j),i+1:n+1) - ...
            m*A(piv(i),i+1:n+1);
    end
end
%substitutie inversa
x(n)=A(piv(n),n+1)/A(piv(n),n);
for i=n-1:-1:1
    x(i)=(A(piv(i),n+1)-A(piv(i),i+1:n)*x(i+1:n)) ...
        /A(piv(i),i);
end
```

---

**Avantaje.**  $Ax = b \Leftrightarrow LUx = b \Leftrightarrow Ly = b \wedge Ux = y$ .

Dacă avem de rezolvat mai multe sisteme  $Ax = b_i$ ,  $i = \overline{1, m}$ , fiecare rezolvare durează  $\Theta(n^3)$ ; dacă se factorizează la început rezolvarea unui sistem durează  $\Theta(n^2)$ , factorizarea durează  $\Theta(n^3)$ .

**Observația 4.4.2.**  $U$  este matricea triunghiulară superior obținută în urma eliminării gaussiene, iar  $L$  este matricea multiplicatorilor  $m_{ij}$ .  $\diamond$

Dacă eliminarea gaussiană se face cu interschimbări avem de asemenea  $A = LU$ , dar  $L$  nu este triunghiulară inferior.

Metoda obținută se numește factorizare  $LU$ .

Situării când eliminarea gaussiană se face fără interschimbări:

- $A$  este diagonal dominantă pe linii, adică

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|, \quad i = \overline{1, n}$$

- $A$  este pozitiv definită ( $\forall x \neq 0 x^* Ax > 0$ ).

*Demonstrația teoremei 4.4.1.* (schiță) Pentru  $n > 1$  partaționăm  $A$  astfel

$$A = \left[ \begin{array}{c|ccc} a_{11} & a_{12} & \dots & a_{1n} \\ \hline a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{array} \right] = \left[ \begin{array}{cc} a_{11} & w^* \\ v & A' \end{array} \right],$$

unde:  $v$  - vector coloană de dimensiune  $n - 1$ ,  $w^*$  - vector linie de dimensiune  $n - 1$ . Putem factoriza  $A$  prin

$$A = \left[ \begin{array}{cc} a_{11} & w^* \\ v & A' \end{array} \right] = \left[ \begin{array}{cc} 1 & 0 \\ v/a_{11} & I_{n-1} \end{array} \right] \left[ \begin{array}{cc} a_{11} & w^* \\ 0 & A' - vw^*/a_{11} \end{array} \right].$$

Matricea  $A' - vw^*/a_{11}$  se numește *complement Schur* al lui  $A$  în raport cu  $a_{11}$ . Se continuă apoi cu descompunerea recursivă a complementului Schur:

$$A' - vw^*/a_{11} = L'U'.$$

$$\begin{aligned} A &= \left[ \begin{array}{cc} 1 & 0 \\ v/a_{11} & I_{n-1} \end{array} \right] \left[ \begin{array}{cc} a_{11} & w^* \\ 0 & A' - vw^*/a_{11} \end{array} \right] = \\ &= \left[ \begin{array}{cc} 1 & 0 \\ v/a_{11} & I_{n-1} \end{array} \right] \left[ \begin{array}{cc} a_{11} & w^* \\ 0 & L'U' \end{array} \right] = \\ &= \left[ \begin{array}{cc} 1 & 0 \\ v/a_{11} & L' \end{array} \right] \left[ \begin{array}{cc} a_{11} & w^* \\ 0 & U' \end{array} \right]. \end{aligned}$$

$\square$

**Exemplul 4.4.3.** Să se calculeze descompunerea LU a matricei

$$A = \left[ \begin{array}{c|cccc} 2 & 3 & 1 & 5 \\ \hline 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{array} \right]$$

Matricea inițială este

$$\left[ \begin{array}{c|cccc} 2 & 3 & 1 & 5 \\ \hline 3 & 4 & 2 & 4 \\ 1 & 16 & 9 & 18 \\ 2 & 4 & 9 & 21 \end{array} \right]$$

iar primul complement Schur

$$\begin{aligned} A' - vw^*/a_{11} &= \begin{pmatrix} 13 & 5 & 19 \\ 19 & 10 & 23 \\ 10 & 11 & 31 \end{pmatrix} - \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} (3 \ 1 \ 5) = \\ &= \begin{pmatrix} 13 & 5 & 19 \\ 19 & 20 & 23 \\ 10 & 11 & 31 \end{pmatrix} - \begin{pmatrix} 9 & 3 & 15 \\ 3 & 1 & 5 \\ 6 & 2 & 10 \end{pmatrix} = \begin{pmatrix} 4 & 2 & 4 \\ 16 & 9 & 18 \\ 4 & 9 & 21 \end{pmatrix}. \end{aligned}$$

Continuăm cu descompunerea recursivă a complementelor Schur de ordinul 2 și 1:

$$\begin{array}{c} \begin{array}{c|cc} 2 & 3 & 1 & 5 \\ \hline 3 & 4 & 2 & 4 \\ \hline 1 & 4 & 1 & 2 \\ 2 & 1 & 7 & 17 \end{array} \\ \begin{pmatrix} 9 & 18 \\ 9 & 21 \end{pmatrix} - \begin{pmatrix} 4 \\ 1 \end{pmatrix} (2, 4) = \begin{pmatrix} 1 & 2 \\ 7 & 17 \end{pmatrix} \\ \begin{array}{c|cc} 2 & 3 & 1 & 5 \\ \hline 3 & 4 & 2 & 4 \\ \hline 1 & 4 & 1 & 2 \\ 2 & 1 & 7 & 3 \end{array} \end{array}$$

$$A' - vw^*/a_{11} = 17 - 7 \cdot 2 = 3$$

Verificare:

$$\begin{pmatrix} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 2 & 1 & 7 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 1 & 5 \\ 0 & 4 & 2 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 \end{pmatrix} \quad \diamond$$

Avem mai multe posibilități de alegere pentru  $u_{ii}$  și  $l_{ii}$ ,  $i = \overline{1, n}$ . De exemplu dacă  $l_{ii} = 1$  avem *factorizare Doolittle*, dacă  $u_{ii} = 1$  avem *factorizare Crout*. Aici s-a prezentat factorizarea Doolittle.

#### 4.4.2. Descompunere LUP

Ideea din spatele descompunerii  $LUP$  este de a găsi 3 matrice pătratice  $L$ ,  $U$  și  $P$  unde  $L$  - triunghiulară inferior,  $U$  - triunghiulară superior,  $P$  matrice de permutare, astfel încât  $PA = LU$ .

Tripletul  $(L, U, P)$  se va numi *descompunerea LUP* a matricei  $A$ .

Rezolvarea sistemului  $Ax = b$  este echivalentă cu rezolvarea a două sisteme triunghiulare, deoarece

$$Ax = b \Leftrightarrow LUx = Pb \Leftrightarrow Ly = Pb \wedge Ux = y$$

și

$$Ax = P^{-1}LUx = P^{-1}Ly = P^{-1}Pb = b.$$

Vom alege ca pivot în locul lui  $a_{11}$  elementul  $a_{k1}$ . Efectul este înmulțirea cu o matrice de permutare  $Q$ :

$$QA = \begin{bmatrix} a_{k1} & w^* \\ v & A' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{k1} & w^* \\ 0 & A' - vw^*/a_{k1} \end{bmatrix}.$$

Determinăm mai departe descompunerea  $LUP$  a complementului Schur.

$$P'(A' - vw^*/a_{k1}) = L'U'.$$

Definim

$$P = \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} Q,$$

care este tot o matrice de permutare. Avem acum

$$\begin{aligned} PA &= \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} QA = \\ &= \begin{bmatrix} 1 & 0 \\ 0 & P' \end{bmatrix} \begin{bmatrix} 1 & 0 \\ v/a_{k1} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{k1} & w^* \\ 0 & A' - vw^*/a_{k1} \end{bmatrix} = \\ &= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & P' \end{bmatrix} \begin{bmatrix} a_{k1} & w^* \\ 0 & A' - vw^*/a_{k1} \end{bmatrix} = \\ &= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{k1} & w^* \\ 0 & P'(A' - vw^*/a_{k1}) \end{bmatrix} = \\ &= \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & I_{n-1} \end{bmatrix} \begin{bmatrix} a_{k1} & w^* \\ 0 & L'U' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ P'v/a_{k1} & L' \end{bmatrix} \begin{bmatrix} a_{k1} & w^* \\ 0 & U' \end{bmatrix}. \end{aligned}$$

De notat că, în acest raționament, atât vectorul coloană cât și complementul Schur se înmulțesc cu matricea de permutare  $P'$ .

**Exemplul 4.4.4.** Să se calculeze descompunerea LUP a matricei

$$\begin{pmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

Pivoții apar încadrați în casetă. Obținem succesiv

$$\left( \begin{array}{c|ccc} 1 & \boxed{3} & 1 & 6 \\ 2 & 2 & 1 & 3 \\ 3 & 1 & 1 & 1 \end{array} \right) \sim \left( \begin{array}{c|ccc} 1 & 3 & 1 & 6 \\ 2 & \frac{2}{3} & 1 & 3 \\ 3 & \frac{1}{3} & 1 & 1 \end{array} \right) \sim \left( \begin{array}{c|ccc} 1 & 3 & 1 & 6 \\ 2 & \frac{2}{3} & \frac{1}{3} & -1 \\ 3 & \frac{1}{3} & \boxed{\frac{2}{3}} & -1 \end{array} \right),$$

deoarece primul complement Schur este

$$\begin{pmatrix} 1 & 3 \\ 1 & 1 \end{pmatrix} - \begin{pmatrix} \frac{2}{3} \\ \frac{1}{3} \end{pmatrix} \begin{pmatrix} 1 & 6 \end{pmatrix} = \begin{pmatrix} \frac{1}{3} & -1 \\ \frac{2}{3} & -1 \end{pmatrix}.$$

Permutăm liniile 2 și 3 și obținem în continuare:

$$\left( \begin{array}{c|ccc} 1 & 3 & 1 & 6 \\ 3 & \frac{1}{3} & \boxed{\frac{2}{3}} & -1 \\ 2 & \frac{2}{3} & \frac{1}{3} & -1 \end{array} \right) \sim \left( \begin{array}{c|ccc} 1 & 3 & 1 & 6 \\ 3 & \frac{1}{3} & \frac{2}{3} & -1 \\ 2 & \frac{2}{3} & \frac{1}{2} & -1 \end{array} \right) \sim \left( \begin{array}{c|cc} 1 & 3 & 1 & 6 \\ 3 & \frac{1}{3} & \boxed{\frac{2}{3}} & -1 \\ 2 & \frac{2}{3} & \frac{1}{2} & -\frac{1}{2} \end{array} \right).$$

Deci

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{2}{3} & \frac{1}{2} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 3 & 1 & 6 \\ 0 & \frac{2}{3} & -1 \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}.$$

Verificare:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{2}{3} & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 3 & 1 & 6 \\ 0 & \frac{2}{3} & -1 \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}. \quad \diamond$$

Urmează o implementare în MATLAB a descompunerii LUP (funcția 4.2, fișierul `lup.m`). Pentru completare, sursele MATLAB 4.3 și 4.4 dau codurile pentru substituția directă și respectiv inversă.

#### 4.4.3. Factorizarea Cholesky

Matricele hermitiene pozitiv definite pot fi descompuse în factori triunghiulari de două ori mai repede decât matricele generale. Algoritmul standard pentru aceasta, factorizarea Cholesky<sup>4</sup>, este o variantă a eliminării gaussiene ce operează atât la stânga cât și la dreapta matricei, păstrând și exploataând simetria.

Sistemele cu matrice hermitiene pozitiv definite joacă un rol important în algebra liniară numerică și în aplicații. Datorită legilor fundamentale ale fizicii, multe matrice care intervin în probleme practice sunt de acest tip.

<sup>4</sup> Andre-Louis Cholesky (1875-1918) ofițer francez, specialist în topografie și geodezie, a activat în Creta și Africa de nord înaintea începerii primului război mondial. A dezvoltat metoda care îi poartă numele și a aplicat-o la calculul soluțiilor ecuațiilor normale pentru probleme de aproximare în sensul celor mai mici pătrate care apar în geodezie. Lucrarea sa a fost publicată postum în 1924, de către camaradul său Benoît, în *Bulletin Géodesique*. Se pare că a luat parte la misiunea militară franceză din România în timpul primului război mondial.

**Sursa MATLAB 4.2 Descompunere LUP**

---

```

function [L,U,P]=lup(A)
%determina descompunerea LUP a matricei A
%apel [L,U,P]=lup(A)

[m,n]=size(A);
P=zeros(m,n);
piv=(1:m)';
for i=1:m-1
    %pivotare
    [pm,kp]=max(abs(A(i:m,i)));
    kp=kp+i-1;
    %interschimbare linii
    if i~=kp
        A([i,kp],:)=A([kp,i],:);
        piv([i,kp])=piv([kp,i]);
    end
    %complement Schur
    lin=i+1:m;
    A(lin,i)=A(lin,i)/A(i,i);
    A(lin,lin)=A(lin,lin)-A(lin,i)*A(i,lin);
end;
for i=1:m
    P(i,piv(i))=1;
end;
U=triu(A);
L=tril(A,-1)+eye(m);

```

---

**Sursa MATLAB 4.3 Substituție directă**

---

```

function x=forwardsubst(L,b)
%FORWARDSUBST - substitutie directa
%L - matrice triunghiulara inferior
%b - vectorul termenilor liberi

x=zeros(size(b));
n=length(b);
for k=1:n
    x(k)=(b(k)-L(k,1:k-1)*x(1:k-1))/L(k,k);
end

```

---

**Sursa MATLAB 4.4 Substituție inversă**

```

function x=backsubst(U,b)
%BACKSUBST - rezolvare sistem prin substitutie inversa
%U - matrice triunghiulara superior
%b - vectorul termenilor liberi

n=length(b);
x=zeros(size(b));
for k=n:-1:1
    x(k)=(b(k)-U(k,k+1:n)*x(k+1:n))/U(k,k);
end

```

Reamintim câteva proprietăți ale matricelor hermitiene. Dacă  $A$  este o matrice  $m \times m$  hermitiană pozitiv definită și  $X$  este o matrice de tip  $m \times n$  de rang maxim, cu  $m \geq n$ , atunci matricea  $X^*AX$  este de asemenea hermitiană pozitiv definită deoarece  $(X^*AX)^* = X^*A^*X = X^*AX$  și pentru orice vector  $x \neq 0$  avem  $Xx \neq 0$  și astfel  $x^*(X^*AX)x = (Xx)^*A(Xx) > 0$ . Alegând pe post de  $X$  o matrice  $m \times n$  cu un 1 în fiecare coloană și zero în rest, putem scrie orice submatrice principală  $n \times n$  a lui  $A$  sub forma  $X^*AX$ . De aceea, orice submatrice principală a lui  $A$  trebuie să fie pozitiv definită. În particular, orice element diagonal al lui  $A$  este un număr real pozitiv.

Valorile proprii ale unei matrice hermitiene pozitiv definite sunt de asemenea numere reale pozitive, Dacă  $Ax = \lambda x$  pentru  $x \neq 0$ , avem  $x^*Ax = \lambda x^*x > 0$  și deci  $\lambda > 0$  și reciproc, dacă toate valorile proprii sunt pozitive, atunci  $A$  este pozitiv definită. Vectorii proprii ce corespund valorilor proprii distincte ale unei matrice hermitiene sunt ortogonali. Presupunem că  $Ax_1 = \lambda_1 x_1$  și  $Ax_2 = \lambda_2 x_2$  cu  $\lambda_1 \neq \lambda_2$ . Atunci

$$\lambda_2 x_1^* x_2 = x_1^* Ax_2 = \overline{x_2^* Ax_1} = \overline{\lambda_1 x_2^* x_1} = \lambda_1 x_1^* x_2,$$

ășă că  $(\lambda_1 - \lambda_2)x_1^* x_2 = 0$ . Deoarece  $\lambda_1 \neq \lambda_2$ , rezultă că  $x_1^* x_2 = 0$ . Matricele hermitiene sunt de asemenea normale ( $AA^* = A^*A = A^2$ ).

O factorizare Cholesky a unei matrice  $A$  este o descompunere de forma

$$A = R^* R, \quad r_{jj} > 0, \quad (4.4.1)$$

unde  $R$  este o matrice triunghiulară superioră.

**Teorema 4.4.5.** Orice matrice hermitiană pozitiv definită  $A \in \mathbb{C}^{m \times m}$  are o factorizare Cholesky (4.4.1) unică.

*Demonstrație.* (Existența) Deoarece  $A$  este hermitiană și pozitiv definită  $a_{11} > 0$  și putem pune  $\alpha = \sqrt{a_{11}}$ . Observăm că

$$\begin{aligned}
A &= \begin{bmatrix} a_{11} & w^* \\ w & K \end{bmatrix} \\
&= \begin{bmatrix} \alpha & 0 \\ w/\alpha & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - ww^*/a_{11} \end{bmatrix} \begin{bmatrix} \alpha & w^*/\alpha \\ 0 & I \end{bmatrix} = R_1^* A_1 R_1.
\end{aligned} \quad (4.4.2)$$

Acesta este pasul de bază care este repetat în factorizarea Cholesky. Submatricea  $K - ww^*/a_{11}$  fiind o submatrice principală de tip  $(m - 1) \times (m - 1)$  a matricei pozitiv definite  $R_1^*AR_1^{-1}$  este pozitiv definită și deci elementul ei situat în colțul din stânga sus este pozitiv. Se arată prin inducție că toate submatricele care apar în cursul factorizării sunt pozitiv definite și procesul nu poate eşua. Continuăm cu factorizarea lui  $A_1 = R_2^*A_2R_2$  și astfel  $A = R_1^*R_2^*A_2R_2R_1$ ; procesul poate continua până se ajunge la colțul din dreapta jos, obținându-se

$$A = \underbrace{R_1^*R_2^*\dots R_m^*}_{R^*} \underbrace{R_m\dots R_2R_1}_{R},$$

care are chiar forma dorită.

(Unicitatea) De fapt procedeul de mai sus stabilește și unicitatea. La fiecare pas (4.4.2), valoarea  $\alpha = \sqrt{a_{11}}$  este determinată din forma factorizării  $R^*R$  și odată ce  $\alpha$  este determinat, prima linie a lui  $R_1^*$  este de asemenea determinată. Deoarece cantitățile analoage sunt determinate la fiecare pas al reducerii, întreaga factorizare este unică.  $\square$

Când se implementează factorizarea Cholesky, este nevoie să se reprezinte explicit doar jumătate din matricea asupra căreia se operează. Această simplificare permite evitarea a jumătate din operațiile aritmetice. Se dă mai jos una din multele posibilități de prezentare formală a algoritmului (sursa MATLAB 4.5). Matricea  $A$  de la intrare conține diagonala principală și jumătatea de deasupra diagonalei principale a matricei hermitiene și pozitiv definite de tip  $m \times m$  ce urmează a fi factorizată. În implementări practice se pot utiliza scheme de memorare comprimată ce evită irosirea spațiului pentru jumătate din matricea pătratică. Matricea de ieșire reprezintă factorul triunghiular superior din factorizarea  $A = R^*R$ . Fiecare iterație externă corespunde unei singure factorizări elementare: partea triunghiulară superior a submatricei  $A_{k:m,k:m}^*$  reprezintă partea supradiagonală a matricei hermitiene ce trebuie factorizată la pasul  $k$ .

---

#### Sursa MATLAB 4.5 Descompunere Cholesky

---

```
function R=Cholesky(A)
%CHOLESKY - factorizare Cholesky
%apel R=Cholesky(A)
%A - matrice hermitiana
%R - matrice triunghiulara superior

[m,n]=size(A);
for k=1:m
    for j=k+1:m
        A(j, j:m)=A(j, j:m)-A(k, j:m)*A(k, j)/A(k, k);
    end
    A(k, k:m)=A(k, k:m)/sqrt(A(k, k));
end
R=triu(A);
```

---

Operațiile aritmetice în factorizarea Cholesky (sursa MATLAB 4.5) sunt dominate de ciclul interior. O singură execuție a liniei

$A(j, j:m) = A(j, j:m) - A(k, j:m) * A(k, j) / A(k, k)$  ;  
 necesită o împărțire,  $m - j + 1$  înmulțiri și  $m - j + 1$  scăderi, deci un total de  $\sim 2(m - j)$  flops. Acest calcul este repetat pentru fiecare  $j$  de la  $k + 1$  la  $m$  și acest ciclu este repetat pentru fiecare  $k$  de la 1 la  $m$ . Suma se evaluează direct

$$\sum_{k=1}^m \sum_{j=k+1}^m 2(m-j) \sim 2 \sum_{k=1}^m \sum_{j=1}^k j \sim \sum_{k=1}^m k^2 \sim \frac{1}{3} m^3 \text{ flops,}$$

deci jumătate din volumul de calcule necesar pentru eliminarea gaussiană.

#### 4.4.4. Descompunerea QR

**Teorema 4.4.6.** Fie  $A \in \mathbb{R}^{m \times n}$ , cu  $m \geq n$ . Atunci există o matrice ortogonală unică  $Q$  de tip  $m \times n$  și o matrice triunghiulară superior unică, de tip  $n \times n$   $R$  cu diagonala pozitivă ( $r_{ii} > 0$ ) astfel încât  $A = QR$ .

*Demonstrație.* Va rezulta din algoritmul 4.6, care va fi dat în această secțiune.  $\square$

Perechea  $(Q, R)$  din teorema 4.4.6 se numește descompunere QR a lui  $A$ .

Matricele ortogonale și unitare sunt utile în calculele numerice, deoarece ele conservă lungimile, unghiiurile și nu amplifică erorile.

#### Transformări Householder

O transformare Householder<sup>5</sup> (sau reflexie) este o matrice de forma  $P = I - 2uu^T$ , unde  $\|u\|_2 = 1$ . Se verifică ușor că  $P = P^T$  și că  $PP^T = (I - 2uu^T)(I - 2uu^T) = I - 4uu^T + 4uu^Tuu^T = I$ , deci  $P$  este o matrice simetrică și ortogonală. Ea se numește reflexie deoarece  $Px$  este reflexia lui  $x$  față de hiperplanul  $H$  ce trece prin origine și este ortogonal pe  $u$  (figura 4.2).

Dându-se un vector  $x$ , este ușor de găsit reflexia Householder care anulează toate componentele lui  $x$ , exceptând prima:  $Px = [c, 0, \dots, 0]^T = ce_1$ . Vom face aceasta după cum urmează. Scriem  $Px = x - 2u(u^Tx) = ce_1$ , deci  $u = \frac{1}{2(u^Tx)}(x - ce_1)$ , adică  $u$  este o combinație liniară a lui  $x$  și  $e_1$ . Deoarece  $\|x\|_2 = \|Px\|_2 = |c|$ ,  $u$  trebuie să fie paralel cu

---

Alston S. Householder (1904-1993), matematician american. Contribuții importante în biologia matematică și mai ales în algebra liniară numerică. Cartea sa cea mai importantă, „The Theory of Matrices in Numerical Analysis” a avut un impact deosebit asupra dezvoltării analizei numerice și informaticii.



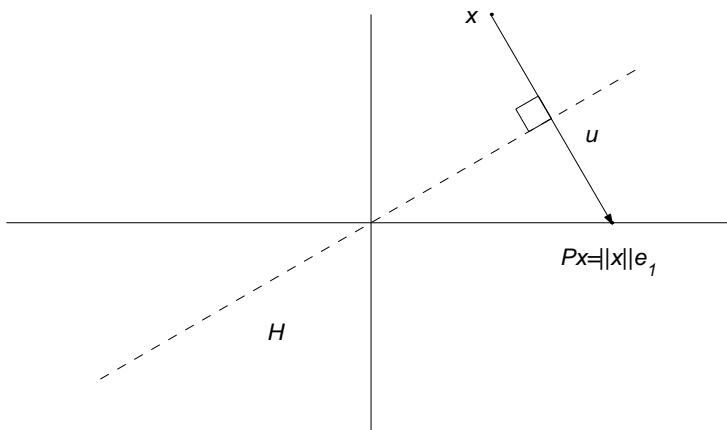


Figura 4.2: Un reflector Householder

vectorul  $\tilde{u} = x \pm \|x\|_2 e_1$ , deci  $u = \tilde{u}/\|\tilde{u}\|_2$ . Se poate verifica că orice alegere de semn ne conduce la un  $u$  ce satisfacă  $Px = ce_1$ , atât timp cât  $\tilde{u} \neq 0$ . Vom utiliza  $\tilde{u} = x + \text{sign}(x_1)\|x\|_2 e_1$ , deoarece astfel nu va apărea nici o anulare flotantă la calculul componentelor lui  $\tilde{u}$ . Dacă  $x_1 = 0$ , vom conveni să luăm  $\text{sign}(x_1) = 1$ . Rezumând, vom lua

$$\tilde{u} = \begin{bmatrix} x_1 + \text{sign}(x_1)\|x\|_2 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{ cu } u = \frac{\tilde{u}}{\|\tilde{u}\|_2}.$$

Vom scrie aceasta sub forma  $u = \text{House}(x)$ . În practică, putem memora  $\tilde{u}$  în locul lui  $u$  pentru a reduce efortul de calcul al lui  $u$  și să utilizăm formula  $P = I - \frac{2}{\|u\|_2^2} \tilde{u} \tilde{u}^T$  în loc de  $P = I - 2uu^T$ .

**Exemplul 4.4.7.** Vom arăta cum se poate calcula descompunerea QR a unei matrice  $5 \times 4$  utilizând transformări Householder. Acest exemplu poate face procedeul mai ușor de înțeles în cazul general. În calculele de mai jos  $P_i$  sunt matrice ortogonale  $5 \times 5$ ,  $x$  este un element generic nenul, iar  $o$  o poziție nulă.

$$1. \text{ Alegem } P_1 \text{ astfel încât } A_1 \equiv P_1 A = \begin{bmatrix} x & x & x & x \\ o & x & x & x \end{bmatrix}.$$

2. Alegem  $P_2 = \begin{bmatrix} I_1 & 0 \\ 0 & P'_2 \end{bmatrix}$  astfel încât  $A_2 \equiv P_2 A_1 = \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & x & x \\ o & o & x & x \end{bmatrix}$ .
3. Alegem  $P_3 = \begin{bmatrix} I_2 & 0 \\ 0 & P'_3 \end{bmatrix}$  astfel încât  $A_3 \equiv P_3 A_2 = \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & o & x \\ o & o & o & x \end{bmatrix}$ .
4. Alegem  $P_4 = \begin{bmatrix} I_3 & 0 \\ 0 & P'_4 \end{bmatrix}$  astfel încât  $A_4 \equiv P_4 A_3 = \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & o & x \\ o & o & o & o \end{bmatrix}$ .

Aici am ales matricele Householder  $P'_i$  pentru a anula elementele subdiagonale din coloana  $i$ ; aceasta nu distrug zerourile deja introduse în coloanele precedente. Să notăm matricea finală  $5 \times 4$  cu  $\tilde{R} = A_4$ . Atunci  $A = P_1^T P_2^T P_3^T P_4^T \tilde{R} = QR$ , unde  $Q$  este formată din primele patru coloane a lui  $P_1^T P_2^T P_3^T P_4^T = P_1 P_2 P_3 P_4$  (deoarece  $P_i$  sunt simetrice), iar  $R$  este formată din primele patru linii ale lui  $R$ .  $\diamond$

Sursa MATLAB 4.6 descrie procesul de calcul pentru descompunerea QR bazată pe transformări Householder.

Pornind de la observația

$$Ax = b \Leftrightarrow QRx = b \Leftrightarrow Rx = Q^T b,$$

putem alege următoarea strategie pentru rezolvarea sistemului de ecuații liniare  $Ax = b$ :

1. Se determină factorizarea  $A = QR$  a lui  $A$ ;
2. Se calculează  $y = Q^T b$ ;
3. Se rezolvă sistemul triunghiular superior  $Rx = y$ .

Strategia de mai sus este implementată în funcția MATLAB 4.7.

Costul calculului descompunerii  $A = QR$  este  $2n^2m - \frac{2}{3}n^3$ , costul pentru calculul lui  $Q^T b$  este  $O(mn)$ , iar costul pentru calculul lui  $Rx$  este de asemenea  $O(mn)$ .

Dacă dorim să calculăm matricea  $Q$  explicit, putem construi  $QI$  ca în sursa 4.6 calculând coloanele sale  $Qe_1, Qe_2, \dots, Qe_m$ .

### Rotății Givens

O rotație Givens

$$R(\theta) := \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

**Sursa MATLAB 4.6 Factorizare QR utilizând reflexii Householder**

---

```

function [R,Q]=HouseQR(A)
%HouseQR - descompunere QR a lui A cu reflexii Householder
%apel [R,Q]=descQR(A)
%A matrice mxn, R triunghiulara superior, Q ortogonală

[m,n]=size(A);
u=zeros(m,n); %vectorii de reflexie
%obtine R
for k=1:n
    x=A(k:m,k);
    x(1)=mysign(x(1))*norm(x)+x(1);
    u(k:m,k)=x/norm(x);
    A(k:m,k:n)=A(k:m,k:n)-2*u(k:m,k)*(u(k:m,k)'*A(k:m,k:n));
end
R=triu(A(1:n,:));
if nargout==2 %se dorește Q
    Q=eye(m,n);
    for j=1:n
        for k=n:-1:1
            Q(k:m,j)=Q(k:m,j)-2*u(k:m,k)*(u(k:m,k)'*Q(k:m,j));
        end
    end
end
%semnul
function y=mysign(x)
if x>=0, y=1;
else, y=-1;
end

```

---

**Sursa MATLAB 4.7 Rezolvarea sistemului  $Ax = b$  prin metoda QR**

---

```

function x=QRSolve(A,b)
%QRSolve - rezolvare sistem prin metoda QR

[m,n]=size(A);
u=zeros(m,n); %vectorii de reflexie
%obtine R si Q^T*b
for k=1:n
    x=A(k:m,k);
    x(1)=mysign(x(1))*norm(x)+x(1);
    u(k:m,k)=x/norm(x);
    A(k:m,k:n)=A(k:m,k:n)-2*u(k:m,k)*(u(k:m,k)'*A(k:m,k:n));
    b(k:m)=b(k:m)-2*u(k:m,k)*(u(k:m,k)'*b(k:m));
end
R=triu(A(1:n,:)); x=R\b(1:n);

```

---

rotește vectorul  $x \in \mathbb{R}^2$  cu un unghi  $\theta$ .

De asemenea, avem nevoie de rotația Givens cu un unghi  $\theta$  în coordonatele  $i$  și  $j$ :

$$R(i, j, \theta) := \begin{pmatrix} & i & & j \\ & 1 & & \\ & & \ddots & \\ i & & & \cos \theta & \sin \theta \\ & & & -\sin \theta & \cos \theta \\ & & & & \ddots \\ j & & & & & 1 \\ & & & & & & 1 \end{pmatrix}.$$

Dându-se  $x$ ,  $i$  și  $j$  putem anula  $x_j$  alegând  $\cos \theta$  și  $\sin \theta$  astfel încât

$$\begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x_i \\ x_j \end{bmatrix} = \begin{bmatrix} \sqrt{x_i^2 + x_j^2} \\ 0 \end{bmatrix},$$

adică

$$\cos \theta = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad \sin \theta = \frac{x_j}{\sqrt{x_i^2 + x_j^2}}.$$

Algoritmul QR bazat pe rotații Givens este analog algoritmului bazat pe reflexii Householder, dar când anulăm coloana  $i$  se anulează un element la un moment dat.

**Exemplul 4.4.8.** Vom ilustra doi pași intermediari din calculul descompunerii QR a unei matrice  $5 \times 4$  utilizând rotații Givens. Pentru a trece de la

$$\begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & x & x \\ o & o & x & x \end{bmatrix} \quad \text{la} \quad \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & o & x \\ o & o & o & x \end{bmatrix}$$

efectuăm înmulțirile

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & c & s \\ & & & -s & c \end{bmatrix} \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & x & x \\ o & o & x & x \end{bmatrix} = \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & x & x \\ o & o & o & x \end{bmatrix}$$

și

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & c' & s' \\ & & -s' & c' \\ & & & 1 \end{bmatrix} \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & x & x \\ o & o & o & x \end{bmatrix} = \begin{bmatrix} x & x & x & x \\ o & x & x & x \\ o & o & x & x \\ o & o & o & x \\ o & o & o & x \end{bmatrix}.$$

◇

Costul descompunerii QR cu rotații Givens este de două ori costul descompunerii cu reflexii Householder. Ele sunt utilizate în alte aplicații (de exemplu valori proprii).

Iată câteva detalii de implementare. Calculul sinusului și cosinusului din matricea Givens este dat în algoritm 4.1. El necesită 5 flops și un radical. Nu necesită funcții trigonometrice

---

**Algoritm 4.1** Dându-se scalarii  $a$  și  $b$ , calculează elementele  $c = \cos(\theta)$  și  $s = \sin(\theta)$  ale unei matrice Givens

---

```

function  $[c, s] = \text{givens}(a, b)$ 
if  $b = 0$  then
     $c := 1;$   $s := 0;$ 
else
    if  $|b| > |a|$  then
         $\tau := -a/b;$   $s := 1/\sqrt{1 + \tau^2};$   $c := s\tau;$ 
    else
         $\tau := -b/a;$   $c := 1/\sqrt{1 + \tau^2};$   $s := c\tau;$ 
    end if
end if

```

---

inverse și nu dă depășire superioară. Descompunerea QR bazată pe rotații Givens este descrisă de algoritm 4.2. Acest algoritm necesită  $3n^2(m - n/3)$  flops. De notat că se pot utiliza și

---

**Algoritm 4.2** Factorizare  $A = QR$  cu ajutorul matricei Givens; rezultatul  $R$  se scrie peste  $A$ .

---

**Intrare:**  $A \in \mathbb{R}^{m \times n}$

**Ieșire:**  $R = Q^T A$ , unde  $R$  este triunghiulară superior.

```

for  $j := 1$  to  $n$  do
    for  $i := m - 1$  downto  $j + 1$  do
         $[c, s] = \text{givens}(A_{i-1,j}, A_{i,j});$ 
         $A_{i-1:i,j:n} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A_{i-1:i,j:n};$ 
    end for
end for

```

---

alte secvențe de rotații; de exemplu primele două **for**-uri din algoritm 4.2 se pot înlocui cu

```

for  $i := m$  downto 2
    for  $j := 1$  to  $\min\{i - 1, n\}$ 

```

Astfel, zerourile vor fi introduse linie cu linie. O altă schimbare posibilă se referă la planele de rotație: în loc să rotim liniile  $i - 1$  și  $i$  ca în algoritm 4.2, am putea roti liniile  $j$  și  $i$  (algoritm 4.3). Deoarece o rotație Givens anulează exact un element, trebuie să memorăm informații despre rotație pe poziția aceluia element. Vom face asta după cum urmează. Fie  $s = \sin \theta$  și  $c = \cos \theta$ . Dacă  $|s| < |c|$  se memorează  $s \cdot \text{sign}(c)$ ; altfel se memorează  $\frac{\text{sign}(s)}{c}$ . Pentru a recupera  $s$  și  $c$  din valoarea memorată (să o numim  $p$ ) vom proceda astfel: dacă  $|p| < 1$ , atunci  $s := p$  și  $c = \sqrt{1 - s^2}$ ; altfel  $c := 1/p$  și  $s := \sqrt{1 - c^2}$ . Motivul pentru care

**Algoritm 4.3** Descompunere QR cu rotații Givens – varianta.

---

```

for  $j := 1$  to  $n$  do
    for  $i := m - 1$  downto  $j + 1$  do
         $[c, s] = \text{givens}(A_{j,j}, A_{i,j});$ 
         $A_{[j \ i], j:n} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T A_{[j \ i], j:n};$ 
    end for
end for

```

---

nu memorăm  $s$  și nu calculăm  $c = \sqrt{1 - s^2}$  este acela că dacă  $s$  este apropiat de 1,  $c$  va fi imprecis (datorită anulării flotante). Se poate recupera fie  $s$  și  $c$ , fie  $-s$  și  $-c$ ; acest lucru este convenabil în practică.

Secvența de rotații Givens se poate aplica în aşa fel încât să fie necesare mai puține flops decât în variantele prezentate aici. Se ajunge astfel la rotații Givens rapide (vezi [29, capitolul 5]).

## 4.5. Rafinarea iterativă

Dacă metoda de rezolvare pentru  $Ax = b$  este nestabilă, atunci  $A\bar{x} \neq b$ , unde  $\bar{x}$  este valoarea calculată. Vom calcula corecția  $\Delta x$  astfel încât

$$A(\bar{x} + \Delta x_1) = b \Rightarrow A\Delta x_1 = b - A\bar{x}$$

Se rezolvă sistemul și se obține un nou  $\bar{x}$ ,  $\bar{x} := x + \Delta x_1$ . Dacă din nou  $Ax \neq b$ , se calculează o nouă corecție până când

$$\|\Delta x_i - \Delta x_{i-1}\| < \varepsilon \quad \text{sau} \quad \|Ax - b\| < \varepsilon.$$

Calculul cantității  $r = b - Ax$ , numită *reziduu*, se va efectua în dublă precizie.

## 4.6. Algoritmul lui Strassen pentru înmulțirea matricelor

Fie  $A, B \in \mathbb{K}^{n \times n}$ . Dorim să calculăm  $C = AB$ . Presupunem că  $n = 2^k$ . Partiționăm  $A$  și  $B$

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \quad C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

În algoritmul clasic avem 8 înmulțiri și 4 adunări pentru un pas, iar timpul de execuție  $T(n) = \Theta(n^3)$ , deoarece  $T(n) = 8T(n/2) + \Theta(n^2)$ .

Interesul este de a reduce numărul de înmulțiri. Volker Strassen a descoperit o metodă de

a reduce numărul de înmulțiri pe pas la 7. Se calculează următoarele cantități:

$$\begin{aligned} p_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ p_2 &= (a_{21} + a_{22})b_{11} \\ p_3 &= a_{11}(b_{12} - b_{22}) \\ p_4 &= a_{22}(b_{21} - b_{11}) \\ p_5 &= (a_{11} + a_{12})b_{22} \\ p_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\ p_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \\ \\ c_{11} &= p_1 + p_4 - p_5 + p_7 \\ c_{12} &= p_3 + p_5 \\ c_{21} &= p_2 + p_4 \\ c_{22} &= p_1 + p_3 - p_2 + p_6 \end{aligned}$$

Procedura este descrisă detaliat de sursa MATLAB 4.8. Presupunem că  $m = 2^q$  și că

---

#### Sursa MATLAB 4.8 Algoritmul lui Strassen pentru înmulțirea a două matrice

---

```
function C=strass(A,B,mmin)
%STRASS - inmultirea a două matrice - algoritmul lui Strassen
%dimensiunea putere a lui 2
%A,B - matrice patratice
%C - produsul
%mmin -dimensiunea minima
[m,n]=size(A); if m<=mmin
    %inmultire clasica
    C=A*B;
else
    %subdivizare si apel recursiv
    n=m/2;
    u=1:n; v=n+1:m;
    P1=strass(A(u,u)+A(v,v),B(u,u)+B(v,v),mmin);
    P2=strass(A(v,u)+A(v,v),B(u,u),mmin);
    P3=strass(A(u,u),B(u,v)-B(v,v),mmin);
    P4=strass(A(v,v),B(v,u)-B(u,u),mmin);
    P5=strass(A(u,u)+A(u,v),B(v,v),mmin);
    P6=strass(A(v,u)-A(u,u),B(u,u)+B(u,v),mmin);
    P7=strass(A(u,v)-A(v,v),B(v,u)+B(v,v),mmin);
    C(u,u)=P1+P4-P5+P7;
    C(u,v)=P3+P5;
    C(v,u)=P2+P4;
    C(v,v)=P1+P3-P2+P6;
end
```

---

$A, B \in \mathbb{R}^{m \times m}$ . Dacă  $m_{\min} = 2^d$  cu  $d \leq q$ , atunci această funcție calculează  $C = AB$  aplicând recursiv procedura lui Strassen de  $q-d$  ori. Deoarece avem 7 înmulțiri și 18 adunări sau scăderi pe pas, timpul de execuție verifică relația de recurență

$$T(n) = 7T(n/2) + \Theta(n^2),$$

cu soluția

$$T(n) = \Theta(n^{\log_2 7}) \sim 28n^{\log_2 7}.$$

Algoritmul se poate generaliza pentru matrice cu dimensiunea  $n = m \cdot 2^k$ .

Dacă  $n$  este impar ultima coloană a lui  $C$  se calculează prin metode standard și se execută algoritmul lui Strassen pentru matrice de dimensiune  $n - 1$ .

$$m \cdot 2^{k+1} \rightarrow m \cdot 2^k$$

$p$ -urile se pot calcula în paralel; la fel și  $c$ -urile.

Accelerarea teoretică obținută pentru înmulțirea matricelor se traduce printr-o accelerare a inversării matricelor, deci și a rezolvării sistemelor de ecuații liniare. Dacă notăm cu  $M(n)$  timpul de înmulțire a două matrice pătratice de ordinul  $n$  și cu  $I(n)$  timpul de inversare a unei matrice de același ordin, atunci  $M(n) = \Theta(I(n))$ . Vom demonstra aceasta în două etape: arătăm că  $M(n) = O(I(n))$  și apoi că  $I(n) = O(M(n))$ .

**Theorem 4.6.1 (Înmulțirea nu este mai grea decât inversarea).** *Dacă putem inversa o matrice  $n \times n$  în timp  $I(n)$ , unde  $I(n) = \Omega(n^2)$  satisfacă condiția de regularitate  $I(3n) = O(I(n))$ , atunci putem înmulți două matrice de ordinul  $n$  în timp  $O(I(n))$ .*

*Demonstrație.* Fie  $A$  și  $B$  două matrice  $n \times n$ . Vrem să calculăm  $C = AB$ . Definim matricea  $D$  de ordinul  $3n \times 3n$  astfel:

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}.$$

Inversa lui  $D$  este

$$D^{-1} = \begin{pmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{pmatrix},$$

și, astfel, putem calcula produsul  $AB$  luând submatricea de ordinul  $n \times n$  din colțul din dreapta sus al matricei  $D^{-1}$ .

Putem calcula matricea  $D$  într-un timp de ordinul  $\Theta(n^2) = O(I(n))$  și conform condiției de regularitate o putem inversa într-un timp  $O(I(3n)) = O(I(n))$ . Deci

$$M(n) = O(I(n)).$$

□

Să observăm că  $I(n)$  satisfacă condiția de regularitate doar dacă  $I(n)$  nu are salturi mari în valoare. De exemplu, dacă  $I(n) = \Theta(n^c \log^d n)$ , pentru orice constante  $c > 0$ ,  $d \geq 0$ , atunci  $I(n)$  satisfacă condiția de regularitate.

Demonstrația că inversarea matricelor nu este mai grea decât înmulțirea lor se bazează pe proprietățile matricelor simetrice, pozitiv definite.

**Teorema 4.6.2 (Inversarea nu este mai grea decât înmulțirea).** *Dacă putem înmulți două matrice reale  $n \times n$  în timp  $M(n)$ , unde  $M(n) = \Omega(n^2)$  și  $M(n)$  satisfac condițiile de regularitate  $M(n) = O(M(n+k))$  pentru orice  $k$ ,  $0 \leq k \leq n$  și  $M(n/2) \leq cM(n)$ , pentru orice constantă  $c < 1/2$ , atunci putem calcula inversa unei matrice reale nesingulare de ordinul  $n \times n$  în timp  $O(M(n))$ .*

*Demonstrație.* Putem presupune că  $n$  este multiplu de 2, deoarece

$$\begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_k \end{pmatrix},$$

pentru orice  $k \in \mathbb{N}^*$ . Așadar, alegând pe  $k$  astfel încât  $n + k$  să fie o putere a lui 2, extindem matricea la o dimensiune care este puterea următoare a lui 2 și obținem răspunsul dorit din răspunsul problemei pentru matricea extinsă în acest mod. Condițiile de regularitate ne asigură că extinderea nu cauzează creșterea timpului decât cu cel mult un factor constant.

Pentru moment, să presupunem că  $A$  este o matrice  $n \times n$  simetrică și pozitiv-definită; o vom partaționa în patru submatrice de ordinul  $n/2 \times n/2$ :

$$A = \begin{pmatrix} B & C^T \\ C & D \end{pmatrix}. \quad (4.6.1)$$

Dacă

$$S = D - CB^{-1}C^T \quad (4.6.2)$$

este complementul Schur al lui  $A$  în raport cu  $B$ , atunci

$$A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^TS^{-1}CB^{-1} & -B^{-1}C^TS^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix}, \quad (4.6.3)$$

relație care se poate verifica prin înmulțire. Matricele  $B^{-1}$  și  $S^{-1}$  există, deoarece  $A$  este simetrică și pozitiv definită, deoarece atât  $B$  cât și  $S$  sunt simetrice și pozitiv definite. În plus,  $B^{-1}C^T = (CB^{-1})^T$  și  $B^{-1}C^TS^{-1} = (S^{-1}CB^{-1})^T$ . Putem folosi ecuațiile (4.6.2) și (4.6.3) pentru a specifica un algoritm recursiv în care intervin patru înmulțiri de matrice de ordinul  $n/2 \times n/2$ :

$$\begin{aligned} & C \cdot B^{-1}, \\ & (C \cdot B^{-1}) \cdot C^T \\ & S^{-1} \cdot (CB^{-1}) \\ & (CB^{-1})^T \cdot (S^{-1}CB^{-1}). \end{aligned}$$

Întrucât matricele de ordinul  $n/2 \times n/2$  se înmulțesc folosind un algoritm pentru matrice de ordinul  $n \times n$ , inversarea matricelor simetrice pozitiv-definite se poate rezolva în timpul

$$I(n) \leq 2I(n/2) + 4M(n) + O(n^2) = 2I(n/2) + O(M(n)) = O(M(n)).$$

Rămâne de studiat cazul când  $A$  este inversabilă, dar nu este simetrică și pozitiv definită. Deoarece  $A^T A$  este simetrică și pozitiv definită, problema inversării lui  $A$  se reduce la problema inversării lui  $A^T A$ . Reducerea se bazează pe observația că, atunci când  $A$  este o matrice nesingulară de ordinul  $n \times n$ , avem

$$A^{-1} = (A^T A)^{-1} A^T,$$

deoarece  $((A^T A)^{-1} A^T) = (A^T A)^{-1} (A^T A) = I_n$ , și inversa unei matrice este unică. Prin urmare, putem calcula  $A^{-1}$ , înmulțind întâi pe  $A^T$  cu  $A$  pentru a obține  $A^T A$ , inversând apoi matricea simetrică și pozitiv definită  $A^T A$  prin algoritmul divide et impera prezentat, și, în final înmulțind rezultatul cu  $A^T$ . Fiecare din acești pași necesită un timp  $O(M(n))$ . Astfel, orice matrice nesingulară cu elemente reale poate fi inversată într-un timp de ordinul  $O(M(n))$ .  $\square$

Demonstrația teoremei 4.6.2 sugerează un mod de a rezolva ecuații de forma  $Ax = b$ , cu  $A$  nesingulară, fără pivotare, rezolvând ecuația echivalentă  $(A^T A)x = b$  prin factorizare Cholesky. Dar în practică, descompunerea LUP funcționează mai bine, iar transformarea propusă mărește numărul de condiționare.

## 4.7. Rezolvarea sistemelor de ecuații liniare în MATLAB

Fie  $m$  numărul de ecuații și  $n$  numărul de necunoscute. Instrumentul fundamental de rezolvare a sistemelor de ecuații liniare este operatorul  $\backslash$  (vezi secțiunea 1.3.3).

El tratează trei tipuri de sisteme de ecuații liniare, pătratice ( $m = n$ ), supradeterminate ( $m > n$ ) și subdeterminate ( $m < n$ ). Vom reveni asupra sistemelor supradeterminate vor fi tratate în capitolul următor. Mai general, operatorul  $\backslash$  poate fi utilizat pentru a rezolva  $AX = B$ , unde  $B$  este o matrice cu  $p$  coloane; în acest caz MATLAB rezolvă sistemele  $AX(:,j) = B(:,j)$  pentru  $j = 1 : p$ . Sistemele de forma  $XA = B$  se pot rezolva cu  $X = B/A$ .

Operatorul  $\backslash$  se bazează pe algoritmi diferenți în funcție de matricea coeficienților. Diversele cazuri, care sunt diagnosticate automat prin examinarea matricei sistemului includ:

- matrice triunghiulare sau permutări de matrice triunghiulare;
- matrice simetrice, pozitiv definite;
- matrice pătratice nesingulară;
- sisteme dreptunghiulare supradeterminate;
- sisteme dreptunghiulare subdeterminate.

### 4.7.1. Sisteme pătratice

Dacă  $A$  este o matrice pătratică nesingulară de ordinul  $n$ , atunci  $A \backslash b$  este soluția sistemului  $Ax=b$ , calculată prin factorizare LU cu pivotare parțială. În timpul rezolvării, MATLAB calculează  $rcond(A)$  și tipărește un mesaj de avertisment dacă rezultatul este mai mic decât  $eps$ :

```
x = hilb(15)\ones(15,1)
Warning: Matrix is close to singular or badly scaled.
          Results may be inaccurate. RCOND = 1.543404e-018.
```

### 4.7.2. Sisteme supradeterminate

Dacă  $m > n$ , în general sistemul  $Ax = b$  nu are nici o soluție. Expresia MATLAB  $A\b$  calculează soluția sistemului în sensul celor mai mici pătrate, adică minimizează norma euclidiană a reziduului (adică  $\text{norm}(A*x-b)$ ) peste toți vectorii  $x$ . Dacă  $A$  are rang maxim  $m$ , atunci soluția este unică. Dacă  $A$  are rangul  $k$  mai mic decât  $m$  (în acest caz spunem că  $A$  este deficentă de rang),  $A\b$  calculează o soluție de bază cu cel mult  $k$  elemente nenule ( $k$  este determinat și  $x$  este calculat utilizând factorizarea QR cu pivotare pe coloană). În ultimul caz MATLAB dă un mesaj de avertisment.

Soluția se mai poate calcula și cu  $x_{\text{min}}=\text{pinv}(A)*b$ , unde  $\text{pinv}(A)$  este pseudo-inversa lui  $A$ . Dacă  $A$  este deficentă de rang,  $x_{\text{min}}$  este soluția unică de normă euclidiană minimală. Vom reveni asupra acestui tip de sisteme în capitolul următor.

Pseudo-inversa Moore-Penrose a lui  $A$ , notată cu  $A^+$  generalizează noțiunea de inversă pentru matrice dreptunghiulare și deficiente de rang. Pseudo-inversa  $A^+$  a lui  $A$  este matricea unică care satisface condițiile

$$AA^+A = A, \quad A^+AA^+ = A^+, \quad (A^+A)^+ = A^+A, \quad (AA^+)^+ = AA^+.$$

Vom ilustra cu următoarele exemple:

```
>> Y=pinv(ones(3))
Y =
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
>> A=[0 0 0 0; 0 1 0 0; 0 0 2 0]
A =
    0     0     0     0
    0     1     0     0
    0     0     2     0
>> pinv(A)
ans =
    0         0         0
    0    1.0000         0
    0         0    0.5000
    0         0         0
```

### 4.7.3. Sisteme subdeterminate

În cazul unui sistem subdeterminat putem fie să nu avem nici o soluție fie să avem o infinitate. În ultimul caz,  $A\b$  produce o soluție de bază cu cel mult  $k$  elemente nenule, unde  $k$  este rangul lui  $A$ . În general această soluție nu are normă euclidiană minimă; soluția cu normă minimă se poate calcula cu  $\text{pinv}(A)*b$ . Dacă sistemul nu are nici o soluție (este

incompatibil), atunci  $A \setminus b$  este o soluție în sensul celor mai mici pătrate. Exemplul următor ilustrează diferența dintre soluția obținută cu \ și pinv:

```
>> A = [1 1 1; 1 1 -1], b=[3; 1]
A =
    1     1     1
    1     1    -1
b =
    3
    1
>> x=A\b; y = pinv(A)*b;
>> [x y]
ans =
    2.0000    1.0000
    0     1.0000
    1.0000    1.0000
>> [norm(x) norm(y)]
ans =
    2.2361    1.7321
```

MATLAB folosește factorizarea QR cu pivotare pe coloană. Fie exemplul

```
>> R=fix(10*rand(2,4))
R =
    9     6     8     4
    2     4     7     0
>> b=fix(10*rand(2,1))
b =
    8
    4
```

Sistemul are 2 ecuații și 4 necunoscute. Deoarece matricea coeficienților conține întregi mici, este recomandabil să afișăm soluția în format rațional. Soluția particulară se obține cu:

```
>> format rat
>> p=R\b
p =
    24/47
    0
    20/47
    0
```

O componentă nenulă este  $p(2)$ , deoarece  $R(:, 2)$  este coloana cu cea mai mare normă. Cealaltă este  $p(4)$ , deoarece  $R(:, 4)$  rămâne dominantă după eliminarea lui  $R(:, 2)$ .

Soluția completă a unui sistem supradeterminat poate fi caracterizată prin adăugarea unui vector arbitrar din spațiul nul al matricei sistemului, care poate fi găsit cu funcția null cu o opțiune care cere o bază rațională

```
>> Z=null(R,'r')
Z =
    5/12      -2/3
```

$$\begin{array}{cc} -47/24 & 1/3 \\ 1 & 0 \\ 0 & 1 \end{array}$$

Se poate verifica că  $R \cdot Z$  este zero și orice vector de forma  $x = p + Z \cdot q$ , unde  $q$  este un vector arbitrar, satisfacă  $R \cdot x = b$ .

#### 4.7.4. Factorizarea LU și Cholesky

Funcția `lu` calculează o factorizare LUP cu pivotare parțială. Apelul `[L, U, P] = lu(A)` returnează factorii triunghiulari și matricea de permutare. Forma `[L, U] = lu(A)` returnează  $L = P^T L$ , deci  $L$  este o matrice triunghiulară cu liniile permute.

```
>> format short g
>> A = gallery('fiedler', 3), [L, U] = lu(A)
A =
    0     1     2
    1     0     1
    2     1     0
L =
    0         1         0
    0.5      -0.5      1
    1         0         0
U =
    2     1     0
    0     1     2
    0     0     2
```

Factorizarea LU este definită și pentru matrice dreptunghiulare. Dacă  $A$  este o matrice  $m \times n$ , apelul `[L, U] = lu(A)` returnează  $L$  de tip  $m \times n$  și  $U$  de tip  $n \times n$  dacă  $m \geq n$  și  $L$  de tip  $m \times m$  și  $U$  de tip  $m \times n$  dacă  $m < n$ .

Rezolvarea sistemului  $Ax = b$  cu  $x = A \backslash b$  cu  $A$  patratică este echivalentă cu secvența MATLAB:

`[L, U] = lu(A); x = U \ (L \ b);`

Determinantul și inversa se calculează de asemenea prin factorizare LU:

```
det(A) = det(L) * det(U) = + - prod(diag(U))
inv(A) = inv(U) * inv(L)
```

Comanda `chol(A)`, unde  $A$  este hermitiană și pozitiv definită calculează  $R$  astfel încât  $A = R^* R$ . Factorizarea Cholesky permite înlocuirea sistemului  $A \cdot x = b$  cu  $R' \cdot R \cdot x = b$ . Deoarece operatorul  $\backslash$  recunoaște sisteme triunghiulare, sistemul se poate rezolva rapid cu  $x = R \backslash (R' \backslash b)$ . Dăm un exemplu de factorizare Cholesky:

```
>> A=pascal(4)
A =
    1     1     1     1
    1     2     3     4
    1     3     6    10
    1     4    10    20
```

```
>> R=chol(A)
R =
    1     1     1     1
    0     1     2     3
    0     0     1     3
    0     0     0     1
```

Funcția `chol` examinează doar partea triunghiulară superior a lui  $A$ . Dacă  $A$  nu este pozitiv definită se dă un mesaj de eroare. În  $[R, p] = \text{chol}(A)$ , dacă  $p=0$  factorizarea s-a terminat cu succes, iar în caz de eșec  $p$  este un număr natural nenul. Pentru detalii a se vedea `help chol` sau `doc chol`.

Funcția `cholupdate` modifică factorizarea Cholesky atunci când matricea originală a fost afectată de o perturbație de rang 1 (adică cu o matrice de forma  $+xx^*$  sau  $-xx^*$ , unde  $x$  este un vector).

#### 4.7.5. Factorizarea QR

În MATLAB există patru variante de factorizare QR – completă sau economică și cu sau fără permutare de coloane.

Factorizarea completă QR a unei matrice  $C$  de dimensiune  $m \times n$ ,  $m > n$  produce o matrice pătratică  $m \times m$   $Q$ , ortogonală și o matrice superior triunghiulară  $R$  de dimensiune  $m \times n$ . Forma de apel este  $[Q, R] = \text{qr}(C)$ . În multe cazuri ultimele  $m - n$  coloane nu sunt necesare, deoarece ele sunt înmulțite cu zerouri în porțiunea de jos a lui  $R$ . De exemplu pentru matricea  $C$  de mai jos:

```
C =
    1     1
    1     2
    1     3
>> [Q, R]=qr(C)
Q =
   -0.5774    0.7071    0.4082
   -0.5774    0.0000   -0.8165
   -0.5774   -0.7071    0.4082
R =
   -1.7321   -3.4641
      0    -1.4142
      0         0
```

Factorizarea economică QR produce o matrice rectangulară  $m \times n$   $Q$  cu coloane ortonormale și o matrice pătratică superior triunghiulară  $R$ , de dimensiune  $n \times n$ . Exemplu

```
>> [Q, R]=qr(C, 0)
Q =
   -0.5774    0.7071
   -0.5774    0.0000
   -0.5774   -0.7071
R =
```

$$\begin{array}{cc} -1.7321 & -3.4641 \\ 0 & -1.4142 \end{array}$$

Pentru matrice dreptunghiulare mari, cu  $m \gg n$ , câștigul de timp și memorie poate fi important.

În contrast cu factorizarea LU, factorizarea QR nu necesită pivotare sau permutări. O factorizare QR cu pivotare pe coloane are forma  $AP = QR$ , unde  $P$  este o matrice de permutare. Strategia de pivotare utilizată produce un factor  $R$  ale cărui elemente diagonale verifică  $|r_{11}| \geq |r_{22}| \geq \dots \geq |r_{nn}|$ . Pivotarea pe coloane este utilă pentru detectarea singularităților sau deficiențelor de rang; detectarea se realizează prin examinarea elementelor diagonale. Dacă  $A$  este apropiată de o matrice de rang  $r < n$ , atunci ultimele  $n - r$  elemente ale lui  $R$  vor avea ordinul `eps*norm(A)`. Pivotarea se indică printr-un al treilea parametru de ieșire, care este o matrice de permutare:

```
>> [Q, R, P] = qr(C)
Q =
-0.2673    0.8729    0.4082
-0.5345    0.2182   -0.8165
-0.8018   -0.4364    0.4082
R =
-3.7417   -1.6036
      0     0.6547
      0       0
P =
  0     1
  1     0
```

Dacă combinăm pivotarea cu forma economică, în locul matricei de permutare se returnează un vector:

```
>> [Q, R, P] = qr(C, 0)
Q =
-0.2673    0.8729
-0.5345    0.2182
-0.8018   -0.4364
R =
-3.7417   -1.6036
      0     0.6547
P =
  2     1
```

Funcțiile `qrdelete`, `qrinsert` și `qrupdate` modifică factorizarea QR când se șterge sau se inserează o coloană din matricea originală sau când matricea este afectată de o perturbație de rang 1.

Să considerăm acum un sistem  $Ax = b$ , pătratic, unde  $A$  este o matrice pătratică de ordinul  $n$  de forma:

$$A = (a_{i,j}), \quad a_{i,j} = \begin{cases} 1, & \text{pentru } i = j \text{ sau } j = n; \\ -1, & \text{pentru } i > j; \\ 0, & \text{în rest.} \end{cases}$$

De exemplu, pentru  $n = 6$ ,

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix}.$$

O astfel de matrice se poate genera, pentru  $n$  dat, cu secvența

```
A=[-tril(ones(n,n-1),-1)+eye(n,n-1),ones(n,1)]
```

Să presupunem că  $b$  s-a inițializat cu  $b=A*ones(n,1)$ . Un astfel de sistem are soluția  $x = [1, 1, \dots, 1]^T$ . Operatorul \ ne dă, pentru  $n=100$

```
>> x=A\b;
>> reshape(x,10,10)
ans =
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
 1     1     1     1     1     1     0     0     0     0
>> norm(b-A*x)/norm(b)
ans =
 0.3191
```

rezultat total eronat, deși  $A$  este bine condiționată

```
>> cond(A)
ans =
 44.8023
```

Dacă rezolvăm folosind metoda QR, se obține

```
>> [Q,R]=qr(A);
>> x2=R\ (Q'*b);
>> x2'
ans =
 Columns 1 through 6
 1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
 ...
 Columns 97 through 100
 1.0000    1.0000    1.0000    1.0000
>> norm(b-A*x2)/norm(b)
ans =
 8.6949e-016
```

Funcția `linsolve` permite rezolvarea mai rapidă a sistemelor de ecuații liniare prin specificarea matricei sistemului. Apelată sub forma

```
x = linsolve(A,b,opts)
```

rezolvă sistemul liniar  $A \cdot x = b$ , selectând un rezolvitor adecvat în funcție de proprietățile matricei  $A$ , descrise de structura `opts`. Nu se face nici un test pentru a verifica dacă  $A$  are astfel de proprietăți. Dăm lista numelor de câmpuri și a valorilor lor posibile.

nume câmp	proprietatea matricei
LT	triunghiulară inferior
UT	triunghiulară superior
UHESS	Hessenberg superioară
SYM	simetrică sau hermitiană
POSDEF	pozitiv definită
RECT	dreptunghiulară
TRANSA	transpusa (conjugată) a lui $A$ – specifică dacă se rezolvă $A \cdot x = b$ sau $A' \cdot x = b$

Dacă `opts` lipsește, sistemul se rezolvă cu factorizare LUP, dacă  $A$  este pătratică și cu factorizare QR cu pivotare în caz contrar. Dacă  $A$  este prost condiționată în cazul pătratic sau deficentă de rang în caz dreptunghiular, se dă un mesaj de avertisment.

```
>> A = triu(rand(5,3)); x = [1 1 1 0 0]';
>> b = A' *x;
>> y1 = (A') \b
y1 =
    1.0000
    1.0000
    1.0000
        0
        0
>> opts.UT = true; opts.TRANSA = true;
>> y2 = linsolve(A,b,opts)
y2 =
    1.0000
    1.0000
    1.0000
        0
        0
```

## 4.8. Rezolvarea iterativă a sistemelor algebrice liniare

Dorim să calculăm soluția sistemului liniar

$$Ax = b, \quad (4.8.1)$$

când  $A$  este inversabilă. Presupunem că am găsit o matrice  $T$  și un vector  $c$  astfel încât  $I - T$  să fie inversabilă și astfel încât punctul fix unic al ecuației

$$x = Tx + c \quad (4.8.2)$$

să fie egal cu soluția sistemului  $Ax = b$ . Fie  $x^*$  soluția lui (4.8.1) sau, echivalent, a lui (4.8.2). Iterațiile: se dă un  $x^{(0)}$  arbitrar; se definește sirul  $(x^{(k)})$  prin

$$x^{(k+1)} = Tx^{(k)} + c, \quad k \in \mathbb{N}. \quad (4.8.3)$$

**Lema 4.8.1.** Dacă  $\rho(X) < 1$ , există  $(I - X)^{-1}$  și

$$(I - X)^{-1} = I + X + X^2 + \cdots + X^k + \dots$$

*Demonstrație.* Fie  $S_k := I + X + \cdots + X^k$ . Observăm că

$$(I - X)S_k = I - X^{k+1},$$

de unde

$$\lim_{k \rightarrow \infty} (I - X)S_k = I \Rightarrow \lim_{k \rightarrow \infty} S_k = (I - X)^{-1},$$

deoarece  $X^{k+1} \rightarrow 0 \Leftrightarrow \rho(X) < 1$ .  $\square$

**Teorema 4.8.2.** Propozițiile următoare sunt echivalente

- (1) metoda (4.8.3) este convergentă;
- (2)  $\rho(T) < 1$ ;
- (3)  $\|T\| < 1$  pentru cel puțin o normă matricială.

*Demonstrație.*

$$\begin{aligned} u^{(k)} &= Tu^{(k-1)} + c = T(Tu^{(k-2)} + c) + c = \cdots = \\ &= T^k x^{(0)} + (I + T + \cdots + T^{n-1})c \end{aligned}$$

(4.8.3) convergentă  $\Leftrightarrow I - T$  inversabilă  $\Leftrightarrow \rho(T) < 1 \Leftrightarrow \exists \|\cdot\|$  astfel încât  $\|B\| < 1$  (din teorema 4.1.8).  $\square$

Aplicând teorema de punct fix a lui Banach se obține:

**Teorema 4.8.3.** Dacă există  $\|\cdot\|$  astfel încât  $\|T\| < 1$ , sirul  $(x^{(k)})$  definit de (4.8.3) este convergent pentru orice  $x^{(0)} \in \mathbb{R}^n$  și are loc

$$\|x^* - x^{(k)}\| \leq \frac{\|T\|^k}{1 - \|T\|} \|x^{(1)} - x^{(0)}\| \leq \frac{\|T\|}{1 - \|T\|} \|x^{(1)} - x^{(0)}\|. \quad (4.8.4)$$

O metodă iterativă de rezolvare a unui sistem algebric liniar  $Ax = b$  pornește de la o aproximatie inițială  $x^{(0)} \in \mathbb{R}^n(\mathbb{C}^n)$  și generează un sir de vectori  $\{x^{(k)}\}$  care converge către soluția  $x^*$  a sistemului. Aceste tehnici transformă sistemul inițial într-un sistem echivalent de forma  $x = Tx + c$ ,  $T \in \mathbb{K}^{n \times n}$ ,  $c \in \mathbb{K}^n$ . Se generează un sir de forma  $x^{(k)} = Tx^{(k-1)} + c$ . Un posibil criteriu de oprire este

$$\|x^{(k)} - x^{(k-1)}\| \leq \frac{1 - \|T\|}{\|T\|} \varepsilon. \quad (4.8.5)$$

El are la bază rezultatul următor:

**Propoziția 4.8.4.** Dacă  $x^*$  este soluția sistemului (4.8.2), cu  $\|T\| < 1$ , atunci

$$\|x^* - x^{(k)}\| \leq \frac{\|T\|}{1 - \|T\|} \|x^{(k)} - x^{(k-1)}\|. \quad (4.8.6)$$

*Demonstrație.* Fie  $p \in \mathbb{N}^*$ . Avem

$$\|x^{(k+p)} - x^{(k)}\| \leq \|x^{(k+1)} - x^{(k)}\| + \cdots + \|x^{(k+p)} - x^{(k+p-1)}\|. \quad (4.8.7)$$

Pe de altă parte, din (4.8.3), rezultă că

$$\|x^{(m+1)} - x^{(m)}\| \leq \|T\| \|x^{(m)} - x^{(m-1)}\|$$

sau, pentru  $k < m$

$$\|x^{(m+1)} - x^{(m)}\| \leq \|T\|^{m-k+1} \|x^{(k)} - x^{(k-1)}\|.$$

Aplicând aceste inegalități, pentru  $m = \overline{k, k+p-1}$  relația (4.8.6) devine

$$\begin{aligned} \|x^{(k+p)} - x^{(k)}\| &\leq (\|T\| + \cdots + \|T\|^p) \|x^{(k)} - x^{(k-1)}\| \\ &\leq (\|T\| + \cdots + \|T\|^p + \dots) \|x^{(k)} - x^{(k-1)}\|. \end{aligned}$$

Deoarece  $\|T\| < 1$ , avem

$$\|x^{(k+p)} - x^{(k)}\| \leq \frac{\|T\|}{1 - \|T\|} \|x^{(k)} - x^{(k-1)}\|,$$

din care prin trecere la limită în raport cu  $p$  se obține (4.8.6).  $\square$

Dacă  $\|T\| \leq 1/2$ , inegalitatea (4.8.6) devine

$$\|x^* - x^{(k)}\| \leq \|x^k - x^{(k-1)}\|,$$

iar criteriul de oprire

$$\|x^{(k)} - x^{(k-1)}\| \leq \varepsilon.$$

Tehnicile iterative sunt rar utilizate pentru a rezolva sisteme de mici dimensiuni, deoarece timpul necesar pentru a obține precizia dorită depășește timpul necesar pentru eliminarea gaussiană. Pentru sisteme rare (a căror matrice are multe zerouri), de mari dimensiuni, metodele iterative sunt eficiente și din punct de vedere al spațiului și din punct de vedere al timpului.

Fie sistemul  $Ax = b$ . Presupunem că putem scrie matricea inversabilă  $A$  sub forma  $A = M - N$ . Dacă  $M$  este ușor de inversat (diagonală, triunghiulară, etc.) este mai convenabil să procedăm astfel:

$$Ax = b \Leftrightarrow Mx = Nx + b \Leftrightarrow x = M^{-1}Nx + M^{-1}b.$$

Ultima ecuație are forma  $x = Tx + c$ , cu  $T = M^{-1}N = I - M^{-1}A$ . Se obține sirul

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b, \quad k \in \mathbb{N},$$

$x^{(0)}$  vector arbitrar.

Prima descompunere pe care o considerăm este  $A = D - L - U$ , unde

$$(D)_{ij} = a_{ij}\delta_{ij}, \quad (-L)_{ij} = \begin{cases} a_{ij}, & i > j \\ 0, & \text{altfel} \end{cases}$$

$$(-U)_{ij} = \begin{cases} a_{ij}, & i < j \\ 0, & \text{altfel} \end{cases}$$

Se ia  $M = D$ ,  $N = L + U$ . Se obține succesiv

$$Ax = b \Leftrightarrow Dx = (L + U)x + b \Leftrightarrow x = D^{-1}(L + U)x + D^{-1}b.$$

Deci  $T = T_J = D^{-1}(L + U)$ ,  $c = c_J = D^{-1}b$ . Metoda se numește *metoda lui Jacobi*<sup>6</sup>. ( $D$  inversabilă, de ce?)

Altă descompunere este  $A = D - L - U$ ,  $M = D - L$ ,  $N = U$ . Se obține  $T_{GS} = (D - L)^{-1}U$ ,  $c_{GS} = (D - L)^{-1}b$  – numită *metoda Gauss-Seidel* ( $D - L$  inversabilă, de ce?)

$$\text{Iterațiile Jacobi se scriu pe componente } x_i^{(k)} = \frac{1}{a_{ii}} \left( b_i - \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij} x_j^{(k-1)} \right).$$

La calculul lui  $x_i^{(k)}$  se utilizează componentele lui  $x^{(k-1)}$  (substituția simultană). Deoarece pentru  $i > 1$ ,  $x_1^{(k)}, \dots, x_{i-1}^{(k)}$  au fost deja calculați și se presupune că sunt aproximări mai bune ale componentelor soluției decât  $x_1^{(k-1)}, \dots, x_{i-1}^{(k-1)}$ , la iterările Gauss-Seidel vom calcula  $x_i^{(k)}$  utilizând valorile cele mai recente, adică

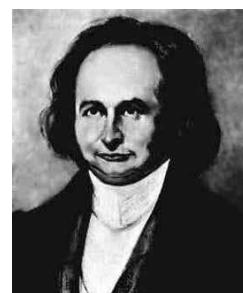
$$x_i^{(k)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^n a_{ij} x_j^{(k-1)} \right).$$

Se pot da condiții necesare și suficiente pentru convergența metodei lui Jacobi și a metodei Gauss-Seidel

$$\rho(T_J) < 1$$

$$\rho(T_{GS}) < 1$$

Carl Gustav Jacob Jacobi (1804-1851) a fost contemporan al lui Gauss, și unul dintre cei mai importanți matematicieni germani din secolul al XIX-lea. Numele său este legat de funcții eliptice, <sup>6</sup>ecuațiile cu derivate partiale ale dinamicii, mecanică cerească. Matricea derivatelor parțiale poartă de asemenea numele său. Este inventatorul metodei iterative de rezolvare a sistemelor algebrice liniare numită metoda lui Jacobi și pe care a aplicat-o în mecanica cerească.



și condiții suficiente: pentru o normă dată avem

$$\|T_J\| < 1$$

$$\|T_{GS}\| < 1.$$

Putem îmbunătăți metoda Gauss-Seidel introducând un parametru  $\omega$  și alegând

$$M = \frac{D}{\omega} - L.$$

Avem

$$A = \left( \frac{D}{\omega} - L \right) - \left( \frac{1-\omega}{\omega} D + U \right),$$

iar iterația obținută este

$$\left( \frac{D}{\omega} - L \right) x^{(k+1)} = \left( \frac{1-\omega}{\omega} D + U \right) x^{(k)} + b.$$

Se obține matricea

$$\begin{aligned} T = T_\omega &= \left( \frac{D}{\omega} - L \right)^{-1} \left( \frac{1-\omega}{\omega} D + U \right) \\ &= (D - \omega L)^{-1} ((1-\omega)D + \omega U). \end{aligned}$$

Metoda se numește *metoda relaxării*. Avem următoarele variante:

- $\omega > 1$  suprarelaxare (SOR - Successive Over Relaxation) ;
- $\omega < 1$  subrelaxare;
- $\omega = 1$  Gauss-Seidel.

Următoarele două teoreme se referă la convergența metodei relaxării. Scriem matricea metodei sub forma

$$T_\omega = (I - \omega \bar{L})^{-1} [(1 - \omega \bar{L})I + \omega \bar{U}],$$

unde  $\bar{L} = D^{-1}L$  și  $\bar{U} = D^{-1}U$ .

**Teorema 4.8.5 (Kahan).** Are loc  $\rho(T_\omega) \geq |\omega - 1|$ . De aici rezultă condiția necesară  $0 < \omega < 2$ .

*Demonstratie.* Scriem polinomul caracteristic al lui  $T_\omega$  sub forma  $p(\lambda) = \det(\lambda I - T_\omega) = \det((I - \omega \bar{L})(\lambda I - T_\omega)) = \det((\lambda + \omega - 1)I - \omega \lambda \bar{L} - \omega \bar{U})$ ; deci avem

$$p(0) = \pm \prod_{i=1}^n \lambda_i(T_\omega) = \pm \det((\omega - 1)I) = \pm (\omega - 1)^n,$$

ceea ce implică  $\max_i |\lambda_i(T_\omega)| \geq |\omega - 1|$ .  $\square$

**Sursa MATLAB 4.9 Jacobi method for linear sytems**

---

```
function [x,ni]=Jacobi(A,b,x0,err,nitmax)
%JACOBI Jacobi method
%call [x,ni]=Jacobi(A,b,x0,err,nitmax)
%parameters
%A - system matrix
%b - right hand side vector
%x0 - starting vector
%err - tolerance (default 1e-3)
%nitmax - maximum number of iterations (default 50)
%x - solution
%ni -number of actual iterations

%parameter check
if nargin < 5, nitmax=50; end
if nargin < 4, err=1e-3; end
if nargin <3, x0=zeros(size(b)); end
[m,n]=size(A);
if (m~=n) | (n~=length(b))
    error('illegal size')
end
%compute T and c (prepare iterations)
M=diag(diag(A));
N=M-A;
T=inv(M)*N;
c=inv(M)*b;
alfa=norm(T,inf);
x=x0(:);
for i=1:nitmax
    x0=x;
    x=T*x0+c;
    if norm(x-x0,inf)<(1-alfa)/alfa*err
        ni=i;
        return
    end
end
error('iteration numeber exceeded')
```

---

**Teorema 4.8.6 (Ostrowski-Reich).** Dacă  $A$  este o matrice pozitiv definită și  $0 < \omega < 2$ , SOR converge pentru orice alegere a aproximăției inițiale  $x^{(0)}$ .

*Demonstrație.* Demonstrația se face în doi pași. Fie  $M = \omega^{-1}(D - \omega L)$ . Atunci:

- (1) definim  $Q = A^{-1}(2M - A)$  și arătăm ca  $\operatorname{Re}\lambda_i(Q) > 0$ , pentru orice  $i$ .
- (2) arătăm că  $T_\omega = (Q - I)(Q + I)$ , de unde rezultă că  $|\lambda_i(T_\omega)| < 1$ , pentru orice  $i$ .

Pentru primul pas, observăm că  $Qx = \lambda x$  implică  $(2M - A)x = \lambda Ax$ . Adunând această ecuație cu transpusa ei conjugată se obține  $x^*(M + M^* - A)x = \operatorname{Re}\lambda(x^*Ax)$ . Astfel  $\operatorname{Re}\lambda = x^*(M + M^* - A)x/x^*Ax = x^*(\frac{2}{\omega} - 1)Dx/x^*Ax > 0$ , căci  $A$  și  $(\frac{2}{\omega} - 1)D$  sunt pozitiv definite.

Pentru a demonstra (2), observăm că

$$(Q - I)(Q + I)^{-1} = (2A^{-1}M - 2I)(2A^{-1}M)^{-1} = I - M^{-1}A = T_\omega,$$

deci

$$|\lambda(T_\omega)| = \left| \frac{\lambda(Q) - 1}{\lambda(Q) + 1} \right| = \left| \frac{(\operatorname{Re}\lambda(Q) - 1)^2 + (\operatorname{Im}\lambda(Q))^2}{(\operatorname{Re}\lambda(Q) + 1)^2 + (\operatorname{Im}\lambda(Q))^2} \right|^{1/2}.$$

□

Teoremele 4.8.5 și 4.8.6 ne dau o condiție necesară și suficientă pentru convergența metodei relaxării dacă matricea sistemului este simetrică și pozitiv definită:  $0 < \omega < 2$ .

**Observația 4.8.7.** Pentru metoda Jacobi (și Gauss-Seidel) avem următoarele condiții suficiente de convergență:

$$\begin{aligned} |a_{ii}| &> \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| \quad (A \text{ diagonal dominantă pe linii}); \\ |a_{ii}| &> \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ji}| \quad (A \text{ diagonal dominantă pe coloane}). \end{aligned} \quad \diamond$$

Valoarea optimală a lui  $\omega$  (valabilă doar pentru anumite tipuri de matrice, de exemplu tridiagonale pe blocuri) este

$$\omega_O = \frac{2}{1 + \sqrt{1 - (\rho(T_J))^2}}.$$

Pentru implementare, vezi sursa MATLAB 4.11. În practică determinarea parametrului optim de relaxare pe baza razei spectrale nu este eficientă. Rutina are scop pur didactic.

**Teorema 4.8.8.** Fie  $B$  o matrice pătratică și  $\|\cdot\|$  o normă matricială oarecare. Atunci

$$\lim_{k \rightarrow \infty} \|B^k\|^{1/k} = \rho(B).$$

**Sursa MATLAB 4.10 Metoda SOR**

---

```
function [x,ni]=relax(A,b,omega,x0,err,nitmax)
%RELAX metoda relaxarii
%apel [z,ni]=relax(A,b,omega,err,nitmax)
%parametri
%A - matricea sistemului
%b - vectorul termenilor liberi
%omega - parametrul relaxarii
%x0 - vector de pornire
%err - toleranta (implicit 1e-3)
%nitmax - numarul maxim de iteratii (implicit 50)
%z - solutia
%ni -numarul de iteratii realizat efectiv

%verificare parametri
if nargin < 6, nitmax=50; end
if nargin < 5, err=1e-3; end
if nargin < 4, x0=zeros(size(b)); end
if (omega<=0) | (omega>=2)
    error('parametrul relaxarii ilegal')
end
[m,n]=size(A);
if (m~=n) | (n~=length(b))
    error('dimensiuni ilegale')
end
%calculul lui T si c (pregatirea iteratiilor)
M=1/omega*diag(diag(A))+tril(A,-1);
N=M-A;
T=M\N;
c=M\b;
alfa=norm(T,inf);
x=x0(:);
for i=1:nitmax
    x0=x;
    x=T*x0+c;
    if norm(x-x0,inf)<(1-alfa)/alfa*err
        ni=i;
        return
    end
end
error('prea multe iteratii')
```

---

**Sursa MATLAB 4.11 Determinarea parametrului optim de relaxare**

```

function omega=relopt(A)
%RELOPT - determina valoarea optima a parametrului
%           relaxarii
M=diag(diag(A)); %determin matricea Jacobi
N=M-A;
T=M \ N;
e=eig(T);
rt=max(abs(e)); %raza spectrala a matricei Jacobi
omega=2/(1+sqrt(1-rt^2));

```

*Demonstrație.* Cum  $\rho(B) \leq \|B\|$  (teorema 4.1.6 și cum  $\rho(B) = (\rho(B^k))^{1/k}$  avem

$$\rho(B) \leq \|B^k\|^{1/k}, \forall k \in \mathbb{N}^*.$$

Se va stabili că, pentru orice  $\varepsilon > 0$  există  $l = l(\varepsilon) \in \mathbb{N}$  astfel încât

$$k \geq l \Rightarrow \|B^k\|^{1/k} \leq \rho(B) + \varepsilon,$$

ceea ce demonstrează relația. Fie deci  $\varepsilon > 0$  dat. Deoarece matricea

$$B_\varepsilon = \frac{B}{\rho(B) + \varepsilon}$$

verifică  $\rho(B_\varepsilon) < 1$  se deduce din teorema 4.8.2 că  $\lim_{k \rightarrow \infty} B_\varepsilon^k = 0$ . Prin urmare, există un întreg  $\ell = \ell_\varepsilon$  astfel încât

$$k \geq \ell \Rightarrow \|B_\varepsilon^k\| = \frac{\|B\|^k}{(\rho(B) + \varepsilon)^k} \leq 1$$

ceea ce este chiar relația căutată.  $\square$

Dăm o teoremă referitoare la matrice de forma  $(I + B)$ .

**Teorema 4.8.9.** (1) Fie  $\|\cdot\|$  o normă matricială subordonată și  $B$  o matrice ce verifică

$$\|B\| < 1.$$

Atunci matricea  $I + B$  este inversabilă și  $\|I + B\|^{-1} \leq \frac{1}{1-\|B\|}$ .

(2) Dacă o matrice de forma  $(I + B)$  este singulară, atunci în mod necesar

$$\|B\| \geq 1$$

pentru orice normă matricială subordonată sau nu.

*Demonstrație.* (1) Deoarece

$$(I + B)u = 0 \Rightarrow \|u\| = \|Bu\|$$

$$\|B\| < 1 \wedge u \neq 0 \Rightarrow \|Bu\| < \|u\|,$$

pentru norma vectorială corespunzătoare, se deduce că

$$(I + B)u = 0 \Rightarrow u = 0.$$

Matricea  $I + B$  fiind inversabilă putem scrie

$$(I + B)^{-1} = I - B(I + B)^{-1}$$

de unde

$$\|(I + B)^{-1}\| \leq 1 + \|B\|\|(I + B)^{-1}\|,$$

ceea ce conduce la inegalitatea căutată.

(2) A spune că matricea  $(I + B)$  este singulară revine la a spune că  $-1$  este valoare proprie a lui  $B$ . În aceste condiții aplicarea teoremei 4.1.6 ne arată că  $\|B\| \geq \rho(B) \geq 1$ .  $\square$

Cum se alege între mai multe metode iterative convergente pentru rezolvarea aceluiasi sistem liniar  $Au = b$ ? Pentru a fixa ideile, presupunem că  $B$  este normală. Atunci

$$\|B^k e_0\|_2 \leq \|B^k\|_2 \|e_0\|_2 = (\rho(B))^k \|e_0\|_2$$

și această inegalitate este cea mai bună posibilă, în sensul că, pentru orice întreg  $k \geq 0$ , există un vector  $e_0(k) \neq 0$  pentru care ea devine egalitate.

În cazul matricelor normale, metoda este cu atât mai rapidă, cu cât  $\rho(B)$  este mai mică, deoarece

$$\sup_{\|e_0\|_2=1} \|B^k e_0\|_2^{1/k} = \rho(B) \text{ pentru orice } k \geq 0.$$

În cazul general (matricea  $B$  oarecare, normă vectorială oarecare), concluzia este identică: asimptotic, vectorul de eroare  $e_k = B^k e_0$  se comportă ca și  $(\rho(B))^k$ , cum precizează rezultatul următor.

**Teorema 4.8.10.** (1) Fie  $\|\cdot\|$  o normă vectorială oarecare și  $u$  astfel încât

$$u = Bu + c.$$

Se consideră metoda iterativă

$$u_{k+1} = Bu_k + c, \quad k \geq 0.$$

Atunci

$$\lim_{k \rightarrow \infty} \left\{ \sup_{\|u_0 - u\|=1} \|u_k - u\|^{1/k} \right\} = \rho(B).$$

(2) Fie  $\|\cdot\|$  o normă vectorială oarecare și fie u astfel încât

$$u = Bu + c = \tilde{B}u + \tilde{c}.$$

Se consideră metodele iterative

$$\tilde{u}_{k+1} = \tilde{B}\tilde{u}_k + \tilde{c}, \quad k \geq 0, \quad u_{k+1} = Bu_k + c, \quad k \geq 0$$

cu  $\rho(B) < \rho(\tilde{B})$ ,  $u_0 = \tilde{u}_0$ . Atunci, oricare ar fi numărul  $\varepsilon > 0$  există un întreg  $\ell(\varepsilon)$  astfel încât

$$k \geq \ell \Rightarrow \sup_{\|u_0 - u\|=1} \left\{ \frac{\|\tilde{u}_k - u\|}{\|u_k - u\|} \right\}^{1/k} \geq \frac{\rho(\tilde{B})}{\rho(B) + \varepsilon}.$$

*Demonstrație.* Fie  $\|\cdot\|$  norma matricială subordonată. Pentru orice  $k$  natural putem scrie

$$(\rho(B))^k = \rho(B^k) \leq \|B^k\| = \sup_{\|e_0\|=1} \|B^k e_0\|$$

astfel ca

$$\rho(B) \leq \sup_{\|e_0\|=1} \|B^k e_0\|^{1/k} = \|B^k\|^{1/k}$$

și aserțiunea (1) decurge din teorema 4.8.8. Conform aceleiasi teoreme, fiind dat un  $\varepsilon > 0$  există un întreg  $\ell(\varepsilon)$  astfel încât

$$k \geq \ell \Rightarrow \sup_{\|e_0\|=1} \|B^k e_0\|^{1/k} \leq (\rho(B) + \varepsilon).$$

Prin urmare, pentru orice  $k \geq \ell$ , există un vector  $e_0 = e_0(k)$  astfel încât

$$\|e_0\| = 1 \quad \text{și} \quad \|\tilde{B}^k e_0\|^{1/k} = \|\tilde{B}^k\|^{1/k} \geq \rho(\tilde{B})$$

și (2) este demonstrată.  $\square$

Deci studiul metodelor iterative răspunde la două probleme.

(1) Fiind dată o metodă iterativă cu matricea  $B$ , să se determine dacă metoda este convergentă, adică dacă  $\rho(B) < 1$ , sau echivalent, dacă există o normă matricială astfel ca  $\|B\| < 1$ .

(2) Fiind date două metode iterative convergente, metoda iterativă cea mai rapidă este cea a cărei matrice are cea mai mică rază spectrală.

Vom încheia secțiunea cu un exemplu care compară cele trei metode iterative staționare. Fie sistemul  $Ax = b$ , unde  $A$  este o matrice tridiagonală care are 5 pe diagonală principală și -1 pe diagonalele -1 și 1, iar  $b$  s-a obținut cu  $b=A*ones(n, 1)$ . Soluția sistemului este  $x=ones(n, 1)$ . Sistemul a fost rezolvat pentru  $n = 500$ , iar timpul de rezolvare a fost contorizat. Se afișeză și numărul de iterații necesare. Secvența de verificare, conținută în fișierul testiter.m, este:

```

o1=ones(n-1,1);
A=5*eye(n)-diag(o1,1)-diag(o1,-1);
b=A*ones(n,1);
err=eps;

```

```
%determin parametrul optim al relaxarii
w=relopt(A);
tic
%Jacobi
[x1,ni1]=Jacobi(A,b,b,err,1000);
%Gauss-Seidel
[x2,ni2]=relax(A,b,1,b,err,1000);
%relaxare
[x3,ni3]=relax(A,b,w,b,err,1000);
toc
ni1,ni2,ni3
```

S-au obținut următoarele rezultate

```
>> testiter
Elapsed time is 1.382000 seconds.
ni1 =
    41
ni2 =
    27
ni3 =
    25
```

Script-ul `testitersparse.m` se obține din `testiter.m`, dacă A este memorată ca matrice rară, adică linia a două se înlocuiește cu

```
A=spdiags([-ones(n,1),5*ones(n,1),-ones(n,1)],-1:1,n,n);  
iar linia w=relopt (A) cu w=relopt (full (A)). La execuția lui se obține
```

```
>> testitersparse
Elapsed time is 0.731000 seconds.
ni1 =
    41
ni2 =
    27
ni3 =
    25
```

Se constată un timp de execuție mult mai scurt, datorită rarității lui A.

## Probleme

**Problema 4.1.** Pentru un sistem având matricea triagonală să se implementeze următoarele metode:

- (a) eliminarea gaussiană, cu și fără pivotare;
- (b) descompunerea LU;
- (c) descompunerea LUP;

- (d) descompunerea Cholesky, dacă matricea este simetrică și pozitiv definită.
- (e) metoda Jacobi;
- (f) metoda Gauss-Seidel;
- (g) metoda SOR.

**Problema 4.2.** Implementați eliminarea gaussiană cu pivotare parțială în două variante: cu permutare de linii logică (prin intermediul unui vector de permutări) și cu permutare efectivă a liniilor. Comparați timpii de execuție pentru diverse dimensiuni ale matricei sistemului. Faceți același lucru și pentru descompunerea LUP.

**Problema 4.3.** Modificați descompunerea LUP astfel ca să returneze și valoarea determinanțului matricei inițiale.

**Problema 4.4.** Se consideră sistemul

$$\begin{aligned} 2x_1 - x_2 &= 1 \\ -x_{j-1} + 2x_j - x_{j+1} &= j, \quad j = \overline{2, n-1} \\ -x_{n-1} + 2x_n &= n \end{aligned}$$

- (a) Să se genereze matricea sistemului folosind `diag`.
- (b) Să se rezolve folosind descompunerea LU.
- (c) Să se rezolve folosind o rutină potrivită din problema 4.1.
- (d) Să se genereze matricea cu `spdiags`, să se rezolve cu \, comparând timpul de rezolvare cu timpul necesar pentru rezolvarea aceluiași sistem cu matrice densă.
- (e) Să se estimeze numărul de condiționare al matricei coeficienților folosind `condest`.

**Problema 4.5.** Modificați eliminarea gaussiană și descompunerea LUP astfel ca să utilizeze pivotarea totală.

**Problema 4.6.** Scrieți o funcție MATLAB care să genereze matrice bandă aleatoare de dimensiune dată și care să fie diagonal dominantă. Testați metoda lui Jacobi și metoda SOR pentru sisteme având astfel de matrice.

**Problema 4.7.** Considerăm ecuația diferențială ordinată

$$y''(x) - p(x)y'(x) - q(x)y(x) = r(x), \quad x \in [a, b]$$

cu condițiile pe frontieră  $y(a) = \alpha, y(b) = \beta$ . Presupunem că  $q(x) \geq q > 0$ . Pentru a rezolva ecuația numeric, o vom discretiza, căutând soluțiile sale pe punctele echidistante  $x_i = a + ih$ ,  $i = 0, \dots, N - 1$ , unde  $h = (b - a)/(N + 1)$ . Definim  $p_i = p(x_i)$ ,  $q_i = q(x_i)$ ,  $r_i = r(x_i)$  și  $y_i \approx y(x_i)$ . Utilizând aproximările

$$y'(x_i) \approx \frac{y_{i+1} - y_i}{2h}$$

și

$$y''(x_i) \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

și ținând cont că  $y_0 = \alpha$  și  $y_{N+1} = \beta$ , se ajunge la un sistem liniar tridiagonal.

- (a) Scrieți sistemul la care se ajunge prin discretizare și studiați proprietăile sale.
- (b) Scrieți o funcție MATLAB care rezolvă numeric ecuația diferențială cu condiții pe frontieră bazată pe ideea de mai sus. Sistemul se va rezolva printr-o metodă directă și una iterativă (dacă este posibil).
- (c) Arătați că sistemul poate fi transformat într-un sistem echivalent a cărui matrice este simetrică și pozitiv definită. Să se implementeze și această variantă.
- (d) Exploatați raritatea matricei sistemului.

Testați pentru problema

$$y'' = -\frac{2}{x}y' + \frac{2}{x^2}y + \frac{\sin(\ln x)}{x^2}, \quad x \in [1, 2], \quad y(1) = 1, \quad y(2) = 2,$$

cu soluția exactă

$$y = c_1 x + \frac{c_2}{x^2} - \frac{3}{10} \sin(\ln x) - \frac{1}{10} \cos(\ln x),$$

unde

$$\begin{aligned} c_2 &= \frac{1}{70}[8 - 12 \sin(\ln 2) - 4 \cos(\ln 2)], \\ c_1 &= \frac{11}{10} - c_2. \end{aligned}$$

**Problema 4.8.** Aplicați ideea din problema precedentă la rezolvarea ecuației lui Poisson unidimensionale

$$-\frac{d^2v(x)}{dx^2} = f, \quad 0 < x < 1,$$

cu condițiile pe frontieră  $v(0) = v(1) = 0$ . Rezolvați sistemul la care se ajunge cu metoda Cholesky și cu metoda SOR.

**Problema 4.9.** Să se determine matricea metodei Gauss-Seidel pentru matricea

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}.$$

**Problema 4.10.** O analiză de tip element finit a sarcinii pe o structură ne conduce la următorul sistem

$$\begin{bmatrix} \alpha & 0 & 0 & 0 & \beta & -\beta \\ 0 & \alpha & 0 & -\beta & 0 & -\beta \\ 0 & 0 & \alpha & \beta & \beta & 0 \\ 0 & -\beta & \beta & \gamma & 0 & 0 \\ \beta & 0 & \beta & 0 & \gamma & 0 \\ -\beta & -\beta & 0 & 0 & 0 & \gamma \end{bmatrix} x = \begin{bmatrix} 15 \\ 0 \\ -15 \\ 0 \\ 25 \\ 0 \end{bmatrix},$$

unde  $\alpha = 482317$ ,  $\beta = 2196.05$  și  $\gamma = 6708.43$ . Aici  $x_1, x_2, x_3$  reprezintă deplasări laterale, iar  $x_4, x_5, x_6$  reprezintă deplasări rotaționale (tridimensionale) corespunzând forței aplicate (membrul drept).

- (a) Determinați  $x$ .
- (b) Cât de precise sunt calculele? Presupunem întâi date exacte, apoi  $\|\Delta A\|/\|A\| = 5 \times 10^{-7}$ .

**Problema 4.11.** Considerăm sistemul

$$\begin{aligned} x_1 + x_2 &= 2 \\ 10x_1 + 10^{18}x_2 &= 10 + 10^{18}. \end{aligned}$$

- (a) Să se rezolve sistemul prin eliminare gaussiană cu pivotare parțială.
- (b) Împărțiți fiecare linie cu maximul în modul din linia respectivă și apoi utilizați eliminarea gaussiană.
- (c) Rezolvați sistemul folosind toolbox-ul Symbolic.

# CAPITOLUL 5

## Aproximarea funcțiilor

Capitolul prezent este legat de aproximarea funcțiilor. Funcțiile în cauză pot să fie definite pe un continuu - de regulă un interval - sau pe o mulțime finită de puncte. Prima situație apare în contextul funcțiilor speciale (elementare sau transcendentale) pe care dorim să le evaluăm ca parte a unei subroutines. Deoarece o astfel de evaluare trebuie să se reducă la un număr finit de operații aritmetice, trebuie în ultimă instanță să aproximăm funcțiile prin intermediul polinoamelor sau funcțiilor raționale. A doua situație este întâlnită în științele fizice, când măsurătorile unor cantități fizice se fac în funcție de alte cantități (cum ar fi timpul). În ambele cazuri dorim să aproximăm o funcție dată, cât mai bine posibil, în termeni de funcții mai simple.

În general o schemă de aproximare poate fi descrisă după cum urmează.

Se dă o funcție  $f \in X$  ce urmează a fi aproximată, împreună cu o clasă  $\Phi$  de aproximante și o normă  $\|\cdot\|$  ce măsoară mărimea funcțiilor. Căutăm o aproximare  $\hat{\varphi} \in \Phi$  a lui  $f$  astfel încât

$$\|f - \hat{\varphi}\| \leq \|f - \varphi\| \text{ pentru orice } \varphi \in \Phi. \quad (5.0.1)$$

Această problemă se numește *problemă de cea mai bună aproximare* a lui  $f$  cu elemente din  $\Phi$ , iar funcția  $\hat{\varphi}$  se numește *(element de) cea mai bună aproximare* a lui  $f$  relativ la norma  $\|\cdot\|$ .

Cunoscându-se o bază  $\{\pi_j\}_{j=1}^n$  a lui  $\Phi$  putem scrie

$$\Phi = \Phi_n = \left\{ \varphi : \varphi(t) = \sum_{j=1}^n c_j \pi_j(t), c_j \in \mathbb{R} \right\}. \quad (5.0.2)$$

$\Phi$  este un spațiu liniar finit dimensional sau o submulțime a acestuia.

**Exemplul 5.0.1.**  $\Phi = \mathbb{P}_m$  - mulțimea polinoamelor de grad cel mult  $m$ . O bază a sa este  $e_j(t) = t^j$ ,  $j = 0, 1, \dots, m$ . Deci  $\dim \mathbb{P}_m = m + 1$ . Polinoamele sunt cele mai utilizate aproximante pentru funcții pe domenii mărginite (intervale sau mulțimi finite). Motivul – teorema lui Weierstrass – orice funcție din  $C[a, b]$  poate fi aproximată oricără de bine prin un polinom de grad suficient de mare.  $\diamond$

**Exemplul 5.0.2.**  $\Phi = \mathbb{S}_m^k(\Delta)$  - spațiul funcțiilor spline polinomiale de grad  $m$  și cu clasa de netezime  $k$  pe diviziunea

$$\Delta : a = t_1 < t_2 < t_3 < \dots < t_{N-1} < t_N = b$$

a intervalului  $[a, b]$ . Acestea sunt funcții polinomiale pe porțiuni de grad  $\leq m$ , racordate în  $t_1, \dots, t_{N-1}$ , astfel încât toate derivatele până la ordinul  $k$  să fie continue pe  $[a, b]$ . Pre-supunem  $0 \leq k < m$ . Pentru  $k = m$  se obține  $\mathbb{P}_m$ . Dacă  $k = -1$  permitem discontinuități în punctele de joncțiune.  $\diamond$

**Exemplul 5.0.3.**  $\Phi = \mathbb{T}_m[0, 2\pi]$  - spațiul polinoamelor trigonometrice de grad cel mult  $m$  pe  $[0, 2\pi]$ . Acestea sunt combinații liniare ale funcțiilor

$$\begin{aligned} \pi_k(t) &= \cos(k-1)t, & k &= \overline{1, m+1}, \\ \pi_{m+1-k}(t) &= \sin kt, & k &= \overline{1, m}. \end{aligned}$$

Dimensiunea spațiului este  $n = 2m + 1$ . Astfel de aproximante sunt alegeri naturale dacă funcția de aproximat este periodică de perioadă  $2\pi$ . (Dacă  $f$  are perioada  $p$  se face schimbarea de variabilă  $t \rightarrow tp/2\pi$ .)  $\diamond$

De notat că mulțimea funcțiilor raționale

$$\Phi = \mathbb{R}_{r,s} = \{\varphi : \varphi = p/q, p \in \mathbb{P}_r, q \in \mathbb{P}_s\},$$

nu este spațiu liniar.

Câteva alegeri posibile ale normei, atât pentru funcții continue, cât și pentru cele discrete apar în tabelul 5.1. Cazul continuu presupune un interval  $[a, b]$  și o funcție pondere  $w(t)$  (posibil și  $w(t) \equiv 1$ ) definită pe intervalul  $[a, b]$  și pozitivă, exceptând zerourile izolate. Cazul discret presupune o mulțime de  $N$  puncte distințe  $t_1, t_2, \dots, t_N$  împreună cu ponderile  $w_1, w_2, \dots, w_N$  (posibil  $w_i = 1$ ,  $i = 1, N$ ). Intervalul  $[a, b]$  poate fi nemărginit, dacă funcția pondere  $w$  este astfel încât integrala pe  $[a, b]$  care definește norma să aibă sens.

Deci, combinând normele din tabelă cu spațiile liniare din exemple se obține o problemă de cea mai bună aproximare (5.0.1) cu sens. În cazul continuu, funcția dată  $f$  și funcțiile  $\varphi$  din clasa  $\Phi$  trebuie definite pe  $[a, b]$  și norma  $\|f - \varphi\|$  trebuie să aibă sens. La fel,  $f$  și  $\varphi$  trebuie definite în punctele  $t_i$  în cazul discret.

De notat că dacă cea mai bună aproximantă  $\widehat{\varphi}$  în cazul discret este astfel încât  $\|f - \widehat{\varphi}\| = 0$ , atunci  $\widehat{\varphi}(t_i) = f(t_i)$ , pentru  $i = 1, 2, \dots, N$ . Spunem că  $\widehat{\varphi}$  interpolează  $f$  în punctele  $t_i$  și numim această problemă de aproximare *problemă de interpolare*.

Cele mai simple probleme de aproximare sunt problema celor mai mici pătrate (vezi secțiunea 5.1) și problema de interpolare, iar spațiul cel mai simplu este cel al polinoamelor.

normă continuă	tip	normă discretă
$\ u\ _\infty = \max_{a \leq t \leq b}  u(t) $	$L^\infty$	$\ u\ _\infty = \max_{1 \leq i \leq N}  u(t_i) $
$\ u\ _1 = \int_a^b  u(t)  dt$	$L^1$	$\ u\ _1 = \sum_{i=1}^N  u(t_i) $
$\ u\ _{1,w} = \int_a^b  u(t)  w(t) dt$	$L_w^1$	$\ u\ _{1,w} = \sum_{i=1}^n w_i  u(t_i) $
$\ u\ _{2,w} = \left( \int_a^b  u(t) ^2 w(t) dt \right)^{1/2}$	$L_w^2$	$\ u\ _{2,w} = \left( \sum_{i=1}^N w_i  u(t_i) ^2 \right)^{1/2}$

Tabela 5.1: Tipuri de aproximare și normele asociate

Înainte de a începe cu problema celor mai mici pătrate, introducem un instrument notațional care ne permite să tratăm cazul continuu și cel discret simultan. Definim în cazul continuu

$$\lambda(t) = \begin{cases} 0, & \text{dacă } t < a \text{ (când } -\infty < a), \\ \int_a^t w(\tau) d\tau, & \text{dacă } a \leq t \leq b, \\ \int_a^b w(\tau) d\tau, & \text{dacă } t > b \text{ (când } b < \infty). \end{cases} \quad (5.0.3)$$

Astfel putem scrie, pentru orice funcție continuă  $u$

$$\int_{\mathbb{R}} u(t) d\lambda(t) = \int_a^b u(t) w(t) dt, \quad (5.0.4)$$

căci  $d\lambda(t) \equiv 0$  în afara lui  $[a, b]$  și  $d\lambda(t) = w(t)dt$  în interiorul lui. Vom numi  $d\lambda$  *măsură* (pozitivă) *continuă*. *Măsura discretă* (numită și „măsura Dirac“) asociată mulțimii de puncte  $\{t_1, t_2, \dots, t_N\}$  este o măsură  $d\lambda$  care este nenuă numai în punctele  $t_i$  și are aici valoarea  $w_i$ . Astfel în acest caz

$$\int_{\mathbb{R}} u(t) d\lambda(t) = \sum_{i=1}^N w_i u(t_i). \quad (5.0.5)$$

O definiție mai precisă se poate da cu ajutorul integralei Stieltjes, dacă definim  $\lambda(t)$  ca fiind o funcție în scără cu saltul în  $t_i$  egal cu  $w_i$ . În particular, definim norma lui  $L^2$  prin

$$\|u\|_{2,d\lambda} = \left( \int_{\mathbb{R}} |u(t)|^2 d\lambda(t) \right)^{\frac{1}{2}} \quad (5.0.6)$$

și obținem norma continuă sau discretă după cum  $\lambda$  este ca în (5.0.3) sau o funcție în scără ca în (5.0.5).

Vom numi *suportul* lui  $d\lambda$  – notat cu  $\text{supp}d\lambda$  – intervalul  $[a, b]$  în cazul continuu și mulțimea  $\{t_1, t_2, \dots, t_N\}$  în cazul discret. Spunem că mulțimea de funcții  $\pi_j$  din (5.0.2) este liniar independentă pe  $\text{supp}d\lambda$  dacă

$$\forall t \in \text{supp}d\lambda \quad \sum_{j=1}^n c_j \pi_j(t) \equiv 0 \Rightarrow c_1 = c_2 = \dots = c_k = 0. \quad (5.0.7)$$

## 5.1. Aproximație prin metoda celor mai mici pătrate

Vom particulariza problema (5.0.1) luând ca normă norma din  $L^2$

$$\|u\|_{2,d_\lambda} = \left( \int_{\mathbb{R}} |u(t)|^2 d\lambda(t) \right)^{\frac{1}{2}}, \quad (5.1.1)$$

unde  $d\lambda$  este fie o măsură continuă (conform (5.0.3)) sau discretă (conform (5.0.5)) și utilizând aproximanta  $\varphi$  dintr-un spațiu liniar  $n$ -dimensional

$$\Phi = \Phi_n = \left\{ \varphi : \varphi(t) = \sum_{j=1}^n c_j \pi_j(t), c_j \in \mathbb{R} \right\}. \quad (5.1.2)$$

Presupunem că funcțiile  $\pi_j$  sunt liniar independente pe  $\text{supp } d\lambda$  și că integrala din (5.1.1) are sens pentru  $u = \pi_j$  sau  $u = f$ .

Problema astfel obținută se numește *problemă de aproximare în sensul celor mai mici pătrate* sau *problemă de aproximare în medie pătratică*. Soluția ei a fost dată la începutul secolului al XIX-lea de către Gauss și Legendre <sup>1</sup>.

Presupunem că funcțiile de bază  $\pi_j$  sunt liniar independente pe  $\text{supp } d\lambda$ . Vom presupune că integrala din (5.1.1) are sens pentru  $u = \pi_j$ ,  $j = 1, \dots, n$  și  $u = f$ .

Soluția problemei de cea mai bună aproximare se exprimă mai ușor cu ajutorul produselor scalare.

### 5.1.1. Produse scalare

Dându-se o măsură continuă sau discretă  $d\lambda$  și două funcții  $u$  și  $v$  având normă finită putem defini produsul scalar

$$(u, v) = \int_{\mathbb{R}} u(t)v(t)d\lambda(t). \quad (5.1.3)$$

Inegalitatea Cauchy-Buniakovski-Schwarz

$$\|(u, v)\| \leq \|u\|_{2,d_\lambda} \|v\|_{2,d_\lambda}$$

ne spune că integrala în (5.1.3) este bine definită.

Produsul scalar real are următoarele proprietăți utile:



Adrien Marie Legendre (1752-1833) matematician francez, bine cunoscut pentru tratatul său asupra integralelor eliptice, dar și pentru lucrările sale de teoria numerelor și geometrie. Este considerat, alături de Gauss, inițiator (în 1805) al metodei celor mai mici pătrate, deși Gauss a utilizat metoda încă din 1794, dar a publicat-o doar în 1809.

- (i) simetria  $(u, v) = (v, u)$ ;
- (ii) omogenitatea  $(\alpha u, v) = \alpha(u, v)$ ,  $\alpha \in \mathbb{R}$ ;
- (iii) aditivitatea  $(u + v, w) = (u, w) + (v, w)$ ;
- (iv) pozitiv definirea  $(u, u) \geq 0$  și  $(u, u) = 0 \Leftrightarrow u \equiv 0$  pe  $\text{supp } d\lambda$ .

Omogenitatea și aditivitatea ne dau liniaritatea

$$(\alpha_1 u_1 + \alpha_2 u_2, v) = \alpha_1(u_1, v) + \alpha_2(u_2, v). \quad (5.1.4)$$

Relația (5.1.4) se poate extinde la combinații liniare finite. De asemenea

$$\|u\|_{2,d\lambda}^2 = (u, u). \quad (5.1.5)$$

Spunem că  $u$  și  $v$  sunt *ortogonale* dacă

$$(u, v) = 0. \quad (5.1.6)$$

Mai general, putem considera sisteme ortogonale  $\{u_k\}_{k=1}^n$ :

$$(u_i, u_j) = 0 \text{ dacă } i \neq j, \quad u_k \neq 0 \text{ pe } \text{supp } d\lambda; \quad i, j = \overline{1, n}, \quad k = \overline{1, n}. \quad (5.1.7)$$

Pentru un astfel de sistem are loc teorema generalizată a lui Pitagora

$$\left\| \sum_{k=1}^n \alpha_k u_k \right\|^2 = \sum_{k=1}^n |\alpha_k|^2 \|u_k\|^2. \quad (5.1.8)$$

O consecință importantă a lui (5.1.8) este aceea că orice sistem ortogonal este liniar independent pe  $\text{supp } d\lambda$ . Într-adevăr, dacă membrul stâng al lui (5.1.8) se anulează, atunci și membrul drept se anulează și deoarece  $\|u_k\|^2 > 0$ , din ipoteză rezultă  $\alpha_1 = \alpha_2 = \dots = \alpha_n = 0$ .

## 5.1.2. Ecuățiile normale

Suntem acum în măsură să rezolvăm problema de aproximare în sensul celor mai mici pătrate. Din (5.1.5) putem scrie pătratul erorii din  $L^2$  sub forma

$$E^2[\varphi] := \|\varphi - f\|^2 = (\varphi - f, \varphi - f) = (\varphi, \varphi) - 2(\varphi, f) + (f, f).$$

Înlocuind pe  $\varphi$  cu expresia sa din (5.1.2) se obține

$$\begin{aligned} E^2[\varphi] &= \int_{\mathbb{R}} \left( \sum_{j=1}^n c_j \pi_j(t) \right)^2 d\lambda(t) - 2 \int_{\mathbb{R}} \left( \sum_{j=1}^n c_j \pi_j(t) \right) f(t) d\lambda(t) + \\ &\quad + \int_{\mathbb{R}} f^2(t) d\lambda(t). \end{aligned} \quad (5.1.9)$$

Pătratul erorii din  $L^2$  este o funcție cuadratică de coeficienții  $c_1, \dots, c_n$  ai lui  $\varphi$ . Problema celei mai bune aproximări în  $L^2$  revine la a minimiza această funcție pătratică; ea se rezolvă anulând derivatele parțiale. Se obține

$$\frac{\partial}{\partial c_i} E^2[\varphi] = 2 \int_{\mathbb{R}} \left( \sum_{j=1}^n c_j \pi_j(t) \right) \pi_i(t) d\lambda(t) - 2 \int_{\mathbb{R}} \pi_i(t) f(t) d\lambda(t) = 0,$$

adică

$$\sum_{j=1}^n (\pi_i, \pi_j) c_j = (\pi_i, f), \quad i = 1, 2, \dots, n. \quad (5.1.10)$$

Acstea ecuații se numesc *ecuații normale* pentru problema celor mai mici pătrate. Ele formează un sistem liniar de forma

$$Ac = b, \quad (5.1.11)$$

unde matricea  $A$  și vectorul  $b$  au elementele

$$A = [a_{ij}], \quad a_{ij} = (\pi_i, \pi_j), \quad b = [b_i], \quad b_i = (\pi_i, f). \quad (5.1.12)$$

Datorită simetriei produsului scalar,  $A$  este o matrice simetrică. Mai mult,  $A$  este pozitiv definită, adică

$$x^T A x = \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j > 0 \text{ dacă } x \neq [0, 0, \dots, 0]^T. \quad (5.1.13)$$

Funcția (5.1.13) se numește *formă pătratică* (deoarece este omogenă de grad 2). Pozitiv definită lui  $A$  ne spune că forma pătratică ai cărei coeficienți sunt elementele lui  $A$  este întotdeauna nenegativă și zero numai dacă variabilele  $x_i$  se anulează.

Pentru a demonstra (5.1.13) să inserăm definiția lui  $a_{ij}$  și să utilizăm proprietățile (i)-(iv) ale produsului scalar

$$x^T A x = \sum_{i=1}^n \sum_{j=1}^n x_i x_j (\pi_i, \pi_j) = \sum_{i=1}^n \sum_{j=1}^n (x_i \pi_i, x_j \pi_j) = \left\| \sum_{i=1}^n x_i \pi_i \right\|^2.$$

Aceasta este evident nenegativă. Ea este zero numai dacă  $\sum_{i=1}^n x_i \pi_i \equiv 0$  pe  $\text{supp} d\lambda$ , care pe baza liniar independenței lui  $\pi_i$  implică  $x_1 = x_2 = \dots = x_n = 0$ .

Este un rezultat cunoscut din algebra liniară că o matrice  $A$  simetrică pozitiv definită este nesingulară. Într-adevăr, determinantul său, precum și minorii principali sunt strict pozitivi. Rezultă că sistemul de ecuații normale (5.1.10) are soluție unică. Corespunde această soluție minimului lui  $E[\varphi]$  în (5.1.9)? Matricea hessiană  $H = [\partial^2 E^2 / \partial c_i \partial c_j]$  trebuie să fie pozitiv definită. Dar  $H = 2A$ , deoarece  $E^2$  este o funcție cuadratică. De aceea,  $H$ , ca și  $A$ , este într-adevăr pozitiv definită și soluția ecuațiilor normale ne dă minimul dorit. Problema de aproximare în sensul celor mai mici pătrate are o soluție unică, dată de

$$\hat{\varphi}(t) = \sum_{j=1}^n \hat{c}_j \pi_j(t) \quad (5.1.14)$$

unde  $\widehat{c} = [\widehat{c}_1, \widehat{c}_2, \dots, \widehat{c}_n]^T$  este vectorul soluție al ecuațiilor normale (5.1.10). Aceasta rezolvă problema de aproximare în sensul celor mai mici pătrate complet în teorie, dar nu și în practică. Referitor la o mulțime generală de funcții de bază liniar independente, pot apărea următoarele dificultăți:

(1) Sistemul de ecuații normale (5.1.10) poate fi prost condiționat. Un exemplu simplu este următorul:  $\text{supp } d\lambda = [0, 1]$ ,  $d\lambda(t) = dt$  pe  $[0, 1]$  și  $\pi_j(t) = t^{j-1}$ ,  $j = 1, 2, \dots, n$ . Atunci

$$(\pi_i, \pi_j) = \int_0^1 t^{i+j-2} dt = \frac{1}{i+j-1}, \quad i, j = 1, 2, \dots, n,$$

adică matricea  $A$  este matricea Hilbert. Prost condiționarea ecuațiilor normale se datorează alegerii neinspirate a funcțiilor de bază. Acestea devin aproape liniar dependente când exponentul crește. O altă sursă de degradare provine din elementele membrului drept  $b_j = \int_0^1 \pi_j(t) f(t) dt$ . Când  $j$  este mare  $\pi_j(t) = t^{j-1}$  se comportă pe  $[0, 1]$  ca o funcție discontinuă. Un polinom care oscilează mai rapid pe  $[0, 1]$  ar fi de preferat, căci ar angaja mai viguros funcția  $f$ .

(2) Al doilea dezavantaj este faptul că toți coeficienții  $\widehat{c}_j$  din (5.1.14) depind de  $n$ , adică  $\widehat{c}_j = \widehat{c}_j^{(n)}$ ,  $j = 1, 2, \dots, n$ . Mărirea lui  $n$  ne dă un nou sistem de ecuații mai mare și cu o soluție complet diferită. Acest fenomen se numește *nepermanența coeficienților*  $\widehat{c}_j$ .

Amândouă neajunsurile (1) și (2) pot fi eliminate (sau măcar atenuate) alegând ca funcții de bază un sistem ortogonal,

$$(\pi_i, \pi_j) = 0 \text{ dacă } i \neq j \quad (\pi_i, \pi_j) = \|\pi_j\|^2 > 0. \quad (5.1.15)$$

Atunci sistemul de ecuații normale devine diagonal și poate fi rezolvat imediat cu formula

$$\widehat{c}_j = \frac{(\pi_j, f)}{(\pi_j, \pi_j)}, \quad j = 1, 2, \dots, n. \quad (5.1.16)$$

Evident, acești coeficienți  $\widehat{c}_j$  sunt independenți de  $n$  și odată calculați rămân la fel pentru orice  $n$  mai mare. Avem acum proprietatea de permanență a coeficienților. De asemenea nu trebuie să rezolvăm sistemul de ecuații normale, ci putem aplica direct (5.1.16). Aceasta nu înseamnă că nu sunt probleme numerice în (5.1.16). Într-adevăr, numitorul  $\|\pi_j\|^2$  crește odată cu  $j$ , în timp ce integrala de la numărător (sau termenii individuali în cazul discret) au același ordin de mărime ca și  $f$ . De aceea ne așteptăm ca valorile  $\widehat{c}_j$  ale coeficienților să descrească rapid. Din acest motiv pot să apară erori de anulare atunci când se calculează produsul scalar de la numărător. Problemele de anulare pot fi ocolite într-o oarecare măsură calculând  $\widehat{c}_j$  în forma alternativă

$$\widehat{c}_j = \frac{1}{(\pi_j, \pi_j)} \left( f - \sum_{k=1}^{j-1} c_k \pi_k, \pi_j \right), \quad j = 1, 2, \dots, n, \quad (5.1.17)$$

unde suma vidă (când  $j = 1$ ) se ia egală cu zero. Evident din ortogonalitatea lui  $\pi_j$ , (5.1.17) este echivalentă cu (5.1.16) din punct de vedere matematic, dar nu neapărat și numeric.

Dăm un algoritm care calculează  $\widehat{c}_j$  cu (5.1.17), dar în același timp și  $\widehat{\varphi}(t)$ :

```

 $s_0 := 0;$ 
for  $j = 1, 2, \dots, n$  do
     $\hat{c}_j := \frac{1}{\|\pi_j\|^2} (f - s_{j-1}, \pi_j);$ 
     $s_j := s_{j-1} + \hat{c}_j \pi_j(t).$ 
end for

```

Orice sistem  $\{\hat{\pi}_j\}$  care este liniar independent pe  $\text{supp} d\lambda$  poate fi ortogonalizat (în raport cu măsura  $d\lambda$ ) prin *procedeul Gram-Schmidt*. Se ia

$$\pi_1 = \hat{\pi}_1$$

și apoi, pentru  $j = 2, 3, \dots$  se calculează recursiv

$$\pi_j = \hat{\pi}_j - \sum_{k=1}^{j-1} c_k \pi_k, \quad c_k = \frac{(\hat{\pi}_j, \pi_k)}{(\pi_k, \pi_k)}, \quad k = \overline{1, j-1}.$$

Atunci fiecare  $\pi_j$  astfel determinat este ortogonal pe toate funcțiile precedente.

### 5.1.3. Eroarea în metoda celor mai mici pătrate. Convergență

Am văzut că dacă  $\Phi = \Phi_n$  constă din  $n$  funcții  $\pi_j$ ,  $j = 1, 2, \dots, n$  care sunt liniar independente pe  $\text{supp} d\lambda$ , atunci problema de aproximare în sensul celor mai mici pătrate pentru  $d\lambda$

$$\min_{\varphi \in \Phi_n} \|f - \varphi\|_{2,d\lambda} = \|f - \hat{\varphi}\|_{2,d\lambda} \quad (5.1.18)$$

are o soluție unică  $\hat{\varphi} = \hat{\varphi}_n$ , dată de (5.1.14). Există multe moduri de a selecta baza  $\{\pi_j\}$  a lui  $\Phi_n$  și de aceea mai multe moduri de a reprezenta soluția, care conduc totuși la aceeași funcție. Eroarea în sensul celor mai mici pătrate – cantitatea din dreapta relației (5.1.18) – este independentă de alegerea funcțiilor de bază (deși calculul soluției, aşa cum s-a menționat anterior, nu este). În studiul acestor erori, putem presupune fără a restrângere generalitatea că baza  $\pi_j$  este un sistem ortogonal (fiecare sistem liniar independent poate fi ortogonalizat prin procedeul Gram-Schmidt). Avem conform (5.1.16)

$$\hat{\varphi}_n(t) = \sum_{j=1}^n \hat{c}_j \pi_j(t), \quad \hat{c}_j = \frac{(\pi_j, f)}{(\pi_j, \pi_j)}. \quad (5.1.19)$$

Observăm întâi că eroarea  $f - \varphi_n$  este ortogonală pe  $\Phi_n$ , adică

$$(f - \hat{\varphi}_n, \varphi) = 0, \quad \forall \varphi \in \Phi_n, \quad (5.1.20)$$

unde produsul scalar este cel din (5.1.3). Deoarece  $\varphi$  este o combinație liniară de  $\pi_k$ , este suficient să arătăm (5.1.20) pentru fiecare  $\varphi = \pi_k$ ,  $k = 1, 2, \dots, n$ . Înlocuind  $\hat{\varphi}_n$  cu expresia sa din (5.1.19) în (5.1.20), găsim

$$(f - \hat{\varphi}_n, \pi_k) = \left( f - \sum_{j=1}^n \hat{c}_j \pi_k, \pi_k \right) = (f, \pi_k) - \hat{c}_k (\pi_k, \pi_k) = 0,$$

ultima ecuație rezultând din formula pentru  $\widehat{c}_k$  din (5.1.19). Rezultatul din (5.1.20) are o interpretare geometrică simplă. Dacă reprezentăm funcțiile ca vectori și spațiul  $\Phi_n$  ca un plan, atunci pentru orice funcție  $f$  care înțeapă planul  $\Phi_n$ , aproximanta în sensul celor mai mici pătrate  $\widehat{\varphi}_n$  este *proiecția ortogonală* a lui  $f$  pe  $\Phi_n$ , vezi figura 5.1.

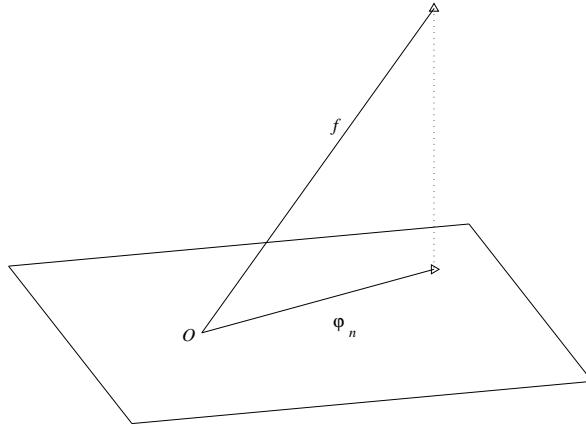


Figura 5.1: Aproximația în sensul celor mai mici pătrate ca proiecție ortogonală

În particular, alegând  $\varphi = \widehat{\varphi}_n$  în (5.1.20) obținem

$$(f - \widehat{\varphi}_n, \widehat{\varphi}_n) = 0$$

și de aceea, deoarece  $f = (f - \widehat{\varphi}) + \widehat{\varphi}$ , conform teoremei lui Pitagora și generalizării sale (5.1.8)

$$\begin{aligned} \|f\|^2 &= \|f - \widehat{\varphi}\|^2 + \|\widehat{\varphi}\|^2 = \|f - \widehat{\varphi}_n\|^2 + \left\| \sum_{j=1}^n \widehat{c}_j \pi_j \right\|^2 \\ &= \|f - \widehat{\varphi}_n\|^2 + \sum_{j=1}^n |\widehat{c}_j|^2 \|\pi_j\|^2. \end{aligned}$$

Exprimând primul termen din dreapta obținem

$$\|f - \widehat{\varphi}_n\| = \left\{ \|f\|^2 - \sum_{j=1}^n |\widehat{c}_j|^2 \|\pi_j\|^2 \right\}^{1/2}, \quad \widehat{c}_j = \frac{(\pi_j, f)}{(\pi_j, \pi_j)}. \quad (5.1.21)$$

De notat că expresia dintre acolade trebuie să fie nenegativă.

Formula (5.1.21) este de interes teoretic, dar de utilitate practică limitată. De notat, într-adevăr, că pe măsură ce eroarea se apropie de nivelul  $\text{eps}$  al preciziei mașinii, calculul erorii din membrul drept al lui (5.1.21) nu poate produce ceva mai mic decât  $\sqrt{\text{eps}}$  datorită erorilor

comise în timpul scăderilor sub radical (astfel se poate obține chiar un rezultat negativ sub radical). Utilizând în loc definiția

$$\|f - \widehat{\varphi}_n\| = \left\{ \int_{\mathbb{R}} [f(t) - \widehat{\varphi}_n(t)]^2 d\lambda(t) \right\}^{1/2},$$

împreună, probabil, cu o regulă de cuadratură (pozitivă) potrivită se garantează că se produce un rezultat nenegativ, potențial la fel de mic ca  $O(\text{eps})$ .

Dacă se dă acum un sir de spații liniare  $\Phi_n, n = 1, 2, 3, \dots$ , avem evident

$$\|f - \widehat{\varphi}_1\| \geq \|f - \widehat{\varphi}_2\| \geq \|f - \widehat{\varphi}_3\| \geq \dots,$$

care rezultă nu numai din (5.1.21), dar mai direct din faptul că

$$\Phi_1 \subset \Phi_2 \subset \Phi_3 \subset \dots$$

Deoarece există o infinitate de astfel de spații, atunci secvența de erori din  $L^2$ , fiind monotonă descrescătoare, trebuie să conveagă la o limită. Este limita 0? Dacă este aşa, spunem că aproximarea prin metoda celor mai mici pătrate converge în medie când  $n \rightarrow \infty$ . Este evident din (5.1.21) că o condiție necesară și suficientă pentru aceasta este

$$\sum_{j=1}^{\infty} |\widehat{c}_j|^2 \|\pi_j\|^2 = \|f\|^2. \quad (5.1.22)$$

Un mod echivalent de a formula convergența este următorul: dându-se  $f$  cu  $\|f\| < \infty$ ,adică  $\forall f \in L^2_{d\lambda}$  și dându-se un  $\varepsilon > 0$  arbitrar de mic, există un întreg  $n = n_\varepsilon$  și o funcție  $\varphi^* \in \Phi_n$  astfel încât  $\|f - \varphi^*\| \leq \varepsilon$ . O clasă de spații  $\Phi_n$  având această proprietate se numește completă în raport cu norma  $\|\cdot\| = \|\cdot\|_{2,d\lambda}$ . Vom numi relația (5.1.22) relația de completitudine. Pentru un interval finit  $[a, b]$  putem defini completitudinea lui  $\{\varphi_n\}$  de asemenea pentru norma uniformă  $\|\cdot\|_\infty$  pe  $[a, b]$ . Se poate presupune că  $f \in C[a, b]$  și  $\pi_j \in C[a, b]$  și numim  $\{\varphi_n\}$  completă în norma  $\|\cdot\|_\infty$  dacă pentru orice  $f \in C[a, b]$  și orice  $\varepsilon > 0$  există  $n = n_\varepsilon$  și un  $\varphi^* \in \Phi_n$  astfel încât  $\|f - \varphi^*\|_\infty \leq \varepsilon$ . Este ușor de văzut că completitudinea lui  $\{\varphi_n\}$  în norma  $\|\cdot\|_\infty$  pe  $(a, b)$  implică completitudinea lui  $\{\varphi_n\}$  în norma din  $L^2$   $\|\cdot\|_{2,d\lambda}$ , unde  $\text{supp}d\lambda = [a, b]$  și deci convergența procesului de aproximare prin metoda celor mai mici pătrate. Într-adevăr, fie  $\varepsilon > 0$  arbitrar și fie  $n$  și  $\varphi^* \in \Phi_n$  astfel încât

$$\|f - \varphi^*\|_\infty \leq \frac{\varepsilon}{\left(\int_{\mathbb{R}} d\lambda(t)\right)^{1/2}}.$$

Conform ipotezei, acest lucru este posibil. Atunci

$$\begin{aligned} \|f - \varphi^*\|_{2,d\lambda} &= \left( \int_{\mathbb{R}} [f(t) - \varphi^*(t)]^2 d\lambda(t) \right)^{1/2} \leq \\ &\leq \|f - \varphi^*\|_\infty \left( \int_{\mathbb{R}} d\lambda(t) \right)^{1/2} \leq \\ &\leq \frac{\varepsilon}{\left(\int_{\mathbb{R}} d\lambda(t)\right)^{1/2}} \left( \int_{\mathbb{R}} d\lambda(t) \right)^{1/2} = \varepsilon, \end{aligned}$$

așa cum s-a afirmat.

**Exemplul 5.1.1.**  $\Phi_n = \mathbb{P}_{n-1}$ . Aici completitudinea lui  $\{\varphi_n\}$  în norma  $\|\cdot\|_\infty$  (pe un interval finit  $[a, b]$ ) este o consecință a teoremei de aproximare a lui Weierstrass. Astfel aproximarea polinomială în sensul celor mai mici pătrate pe un interval finit converge întotdeauna (în medie).  $\diamond$

### 5.1.4. Exemple de sisteme ortogonale

Unul dintre cele mai utilizate sisteme este sistemul trigonometric cunoscut din analiza Fourier. Un alt sistem larg utilizat este cel al polinoamelor ortogonale.

(1) **Sistemul trigonometric** este format din funcțiile:

$$1, \cos t, \cos 2t, \cos 3t, \dots, \sin t, \sin 2t, \sin 3t, \dots$$

El este ortogonal pe  $[0, 2\pi]$  în raport cu măsura

$$d\lambda(t) = \begin{cases} dt & \text{pe } [0, 2\pi] \\ 0 & \text{în rest} \end{cases}$$

Avem

$$\int_0^{2\pi} \sin kt \sin \ell t dt = \begin{cases} 0, & \text{pentru } k \neq \ell \\ \pi, & \text{pentru } k = \ell \end{cases} \quad k, \ell = 1, 2, 3, \dots$$

$$\int_0^{2\pi} \cos kt \cos \ell t dt = \begin{cases} 0, & k \neq \ell \\ 2\pi, & k = \ell = 0 \\ \pi, & k = \ell > 0 \end{cases} \quad k, \ell = 0, 1, 2, \dots$$

$$\int_0^{2\pi} \sin kt \cos \ell t dt = 0, \quad k = 1, 2, 3, \dots, \quad \ell = 0, 1, 2, \dots$$

Aproximarea are forma

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos kt + b_k \sin kt). \quad (5.1.23)$$

Utilizând (5.1.16) obținem

$$\begin{aligned} a_k &= \frac{1}{\pi} \int_0^{2\pi} f(t) \cos kt dt, \quad k = 1, 2, \dots \\ b_k &= \frac{1}{\pi} \int_0^{2\pi} f(t) \sin kt dt, \quad k = 1, 2, \dots \end{aligned} \quad (5.1.24)$$

numiți *coeficienți Fourier* ai lui  $f$ . Ei sunt coeficienții (5.1.16) pentru sistemul trigonometric. Prin extensie coeficienții (5.1.16) pentru orice sistem ortogonal  $(\pi_j)$  se vor numi coeficienții Fourier ai lui  $f$  relativ la acest sistem. În particular, recunoaștem în seria Fourier trunchiată pentru  $k = m$  cea mai bună aproximare a lui  $f$  din clasa polinoamelor trigonometrice de grad  $\leq n$  relativ la norma

$$\|u\|_2 = \left( \int_0^{2\pi} |u(t)|^2 dt \right)^{1/2}.$$

(2) **Polinoame ortogonale.** Dându-se o măsură  $d\lambda$ , știm că orice sistem finit de puteri  $1, t, t^2, \dots$  este liniar independent pe  $[a, b]$ , dacă  $\text{supp}d\lambda = [a, b]$ , iar  $1, t, \dots, t^{n-1}$  este liniar independent pe  $\text{supp}d\lambda = \{t_1, t_2, \dots, t_N\}$ . Deoarece o mulțime de vectori liniar independenti a unui spațiu liniar poate fi ortogonalizată prin procedeul Gram-Schmidt, orice măsură  $d\lambda$  de tipul considerat generează o mulțime unică de polinoame monice  $\pi_j(t, d\lambda)$ ,  $j = 0, 1, 2, \dots$ , ce satisfac

$$\begin{aligned} \text{grad}\pi_j &= j, \quad j = 0, 1, 2, \dots \\ \int_{\mathbb{R}} \pi_k(t)\pi_\ell(t)d\lambda(t) &= 0, \quad \text{dacă } k \neq \ell. \end{aligned} \tag{5.1.25}$$

Acstea polinoame se numesc *polinoame ortogonale* relativ la măsura  $d\lambda$ . Vom permite indiciilor să meargă de la 0. Mulțimea  $\{\pi_j\}$  este infinită dacă  $\text{supp}d\lambda = [a, b]$  și constă din exact  $N$  polinoame  $\pi_0, \pi_1, \dots, \pi_{N-1}$  dacă  $\text{supp}d\lambda = \{t_1, \dots, t_N\}$ . În ultimul caz polinoamele se numesc *polinoame ortogonale discrete*.

Între trei polinoame ortogonale monice<sup>2</sup> consecutive există o relație liniară. Mai exact, există constantele reale  $\alpha_k = \alpha_k(d\lambda)$  și  $\beta_k = \beta_k(d\lambda) > 0$  (depinzând de măsura  $d\lambda$ ) astfel încât

$$\begin{aligned} \pi_{k+1}(t) &= (t - \alpha_k)\pi_k(t) - \beta_k\pi_{k-1}(t), \quad k = 0, 1, 2, \dots \\ \pi_{-1}(t) &= 0, \quad \pi_0(t) = 1. \end{aligned} \tag{5.1.26}$$

(Se subînțelege că (5.1.26) are loc pentru orice  $k \in \mathbb{N}$  dacă  $\text{supp}d\lambda = [a, b]$  și numai pentru  $k = 0, N-2$  dacă  $\text{supp}d\lambda = \{t_1, t_2, \dots, t_N\}$ ).

Pentru a demonstra (5.1.26) și a obține expresiile coeficienților să observăm că

$$\pi_{k+1}(t) - t\pi_k(t)$$

este un polinom de grad  $\leq k$ , și deci poate fi exprimat ca o combinație liniară a lui  $\pi_0, \pi_1, \dots, \pi_k$ . Scriem această combinație sub forma

$$\pi_{k+1} - t\pi_k(t) = -\alpha_k\pi_k(t) - \beta_k\pi_{k-1}(t) + \sum_{j=0}^{k-2} \gamma_{k,j}\pi_j(t) \tag{5.1.27}$$

(sumele vide se consideră nule). Înmulțim scalar ambii membri ai relației anterioare cu  $\pi_k$  și obținem

$$(-t\pi_k, \pi_k) = -\alpha_k(\pi_k, \pi_k),$$

adică

$$\alpha_k = \frac{(t\pi_k, \pi_k)}{(\pi_k, \pi_k)}, \quad k = 0, 1, 2, \dots \tag{5.1.28}$$

La fel, înmulțind scalar cu  $\pi_{k-1}$  obținem

$$(-t\pi_k, \pi_{k-1}) = -\beta_k(\pi_{k-1}, \pi_{k-1}).$$

---

<sup>2</sup>Un polinom se numește *monic* dacă coeficientul său dominant este 1.

Deoarece  $(t\pi_k, \pi_{k-1}) = (\pi_k, t\pi_{k-1})$  și  $t\pi_{k-1}$  diferă de  $\pi_k$  printr-un polinom de grad  $< k$  se obține prin ortogonalitate  $(t\pi_k, \pi_{k-1}) = (\pi_k, \pi_k)$ , deci

$$\beta_k = \frac{(\pi_k, \pi_k)}{(\pi_{k-1}, \pi_{k-1})}, \quad k = 1, 2, \dots \quad (5.1.29)$$

Înmulțind (5.1.27) cu  $\pi_\ell$ ,  $\ell < k - 1$ , se obține

$$\gamma_{k,\ell} = 0, \quad \ell = 0, 1, \dots, k - 1. \quad (5.1.30)$$

Formula de recurență (5.1.26) ne dă o modalitate practică de determinare a polinoamelor ortogonale. Deoarece  $\pi_0 = 1$ , putem calcula  $\alpha_0$  cu (5.1.28) pentru  $k = 0$ . Se continuă apoi cu  $\pi_1$ , utilizând (5.1.26) pentru  $k = 0$ . Mai departe, utilizând alternativ (5.1.28), (5.1.29) și (5.1.26) putem calcula câte polinoame ortogonale dorim. Procedeul – numit procedura lui Stieltjes<sup>3</sup> – este foarte potrivit pentru polinoame ortogonale discrete, căci în acest caz produsul scalar se exprimă prin sume finite. În cazul continuu, calculul produsului scalar necesită calcul de integrale, ceea ce complica lucrurile. Din fericire, pentru multe măsuri speciale importante, coeficienții se cunosc explicit. Cazul special când măsura este simetrică (adică  $d\lambda(t) = w(t)$  cu  $w(-t) = w(t)$  și  $\int_a^b w(t) dt = \int_a^b w(t) t \pi_k^2(t) dt$ ) merită o atenție specială deoarece în acest caz  $\alpha_k = 0$ ,  $\forall k \in \mathbb{N}$ , conform lui (5.1.23) căci

$$(t\pi_k, \pi_k) = \int_{\mathbb{R}} w(t)t\pi_k^2(t)dt = \int_a^b w(t)t\pi_k^2(t)dt = 0,$$

deoarece avem o integrală dintr-o funcție impară pe un domeniu simetric față de origine.

### 5.1.5. Exemple de polinoame ortogonale

#### Polinoamele lui Legendre

Se definesc prin aşa-numita formulă a lui Rodrigues

$$\pi_k(t) = \frac{k!}{(2k)!} \frac{d^k}{dt^k} (t^2 - 1)^k. \quad (5.1.31)$$

Verificăm întâi ortogonalitatea pe  $[-1, 1]$  în raport cu măsura  $d\lambda(t) = dt$ . Pentru orice

---

Thomas Joannes Stieltjes (1856-1894), matematician olandez, a studiat la Institutul Tehnic din Delft, dar nu și-a luat niciodată licență datorită aversiunii pe care o avea pentru examene. A lucrat la Observatorul astronomic din Leyda, în calitate de „calculator asistent pentru calcule astronomice”. Lucrările sale timpurii au atras atenția lui Hermite, care i-a asigurat un post universitar la Toulouse. Prietenia lor a fost exemplară. Stieltjes este cunoscut pentru lucrările sale despre fracții continue și problema momentelor, în care printre altele a introdus integrala care îi poartă numele. A murit de tuberculoză la 38 de ani.



$0 \leq \ell < k$ , prin integrare repetată prin părți se obține:

$$\int_{-1}^1 t^\ell \frac{d^k}{dt^k} (t^2 - 1)^k = \sum_{m=0}^{\ell} (-1)^\ell \ell(\ell-1)\dots(\ell-m+1) t^{\ell-m} \frac{d^{k-m-1}}{dt^{k-m-1}} (t^2 - 1)^k \Big|_{-1}^1 = 0, \quad (5.1.32)$$

ultima relație având loc deoarece  $0 \leq k-m-1 < k$ . Deci,

$$(\pi_k, p) = 0, \quad \forall p \in \mathbb{P}_{k-1},$$

demonstrându-se astfel ortogonalitatea. Datorită simetriei, putem scrie

$$\pi_k(t) = t^k + \mu_k t^{k-2} + \dots, \quad k \geq 2$$

și observând (din nou datorită simetriei) că relația de recurență are forma

$$\pi_{k+1}(t) = t\pi_k(t) - \beta_k \pi_{k-1}(t),$$

obținem

$$\beta_k = \frac{t\pi_k(t) - \pi_{k+1}(t)}{\pi_{k-1}(t)},$$

care este valabilă pentru orice  $t$ . Făcând  $t \rightarrow \infty$ ,

$$\beta_k = \lim_{t \rightarrow \infty} \frac{t\pi_k(t) - \pi_{k+1}(t)}{\pi_{k-1}(t)} = \lim_{t \rightarrow \infty} \frac{(\mu_k - \mu_{k+1})t^{k-1} + \dots}{t^{k-1} + \dots} = \mu_k - \mu_{k+1}.$$

(Dacă  $k = 1$ , punem  $\mu_1 = 0$ .) Din formula lui Rodrigues rezultă

$$\begin{aligned} \pi_k(t) &= \frac{k!}{(2k)!} \frac{d^k}{dt^k} (t^{2k} - kt^{2k-2} + \dots) \\ &= \frac{k!}{(2k)!} (2k(2k-1)\dots(k+1)t^k - k(2k-2)(2k-3)\dots(k-1)t^{k-1} + \dots) \\ &= t^k - \frac{k(k-1)}{2(2k-1)} t^{k-2} + \dots, \end{aligned}$$

așa că

$$\mu_k = \frac{k(k-1)}{2(2k-1)}, \quad k \geq 2.$$

Deci,

$$\beta_k = \mu_k - \mu_{k+1} = \frac{k^2}{(2k-1)(2k+1)}$$

și deoarece  $\mu_1 = 0$ ,

$$\beta_k = \frac{1}{4 - k^{-2}}, \quad k \geq 1. \quad (5.1.33)$$

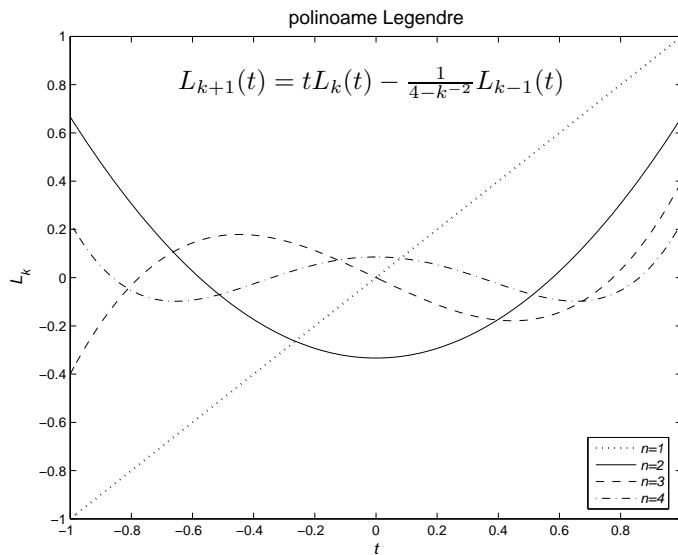


Figura 5.2: Polinoame Legendre

Sursa MATLAB 5.3 calculează aproximanta Legendre de grad  $n$  în sensul celor mai mici pătrate. Se calculează întâi coeficienții aproximantei Legendre cu formula (5.1.16) și în final se evaluatează aproximanta astfel obținută. Funcția `vLegendre` (sursa 5.1) calculează valoarea polinomului Legendre de grad dat pe niște puncte date. Pentru calculul produselor scalare s-a utilizat funcția `quadl`.

În figura 5.2 apar graficele polinoamelor Legendre de grad  $k$ ,  $k = \overline{1,4}$ . Ele au fost obținute cu script-ul `graphLegendre.m`:

```
%grafice polinoame Legendre
n=4; clf
t=(-1:0.01:1)';
s=[];
ls={':','-' ,'--','-' .};
lw=[1.5,0.5,0.5,0.5];
for k=1:n
    y=vLegendre(t,k);
    s=[s;strcat('\itn=',int2str(k))];
    plot(t,y,'LineStyle',ls{k}, 'Linewidth',lw(k), 'Color','k');
    hold on
end
legend(s,4)
xlabel('t','FontSize',12,'FontAngle','italic')
ylabel('L_k','FontSize',12,'FontAngle','italic')
title('polinoame Legendre','Fontsize',14);
text(-0.65,0.8,...
```

```
'$L_{k+1}(t)=tL_k(t)-\frac{1}{4-k^{\{-2\}}}L_{k-1}(t)$',...  
'FontSize',14,'FontAngle','italic','Interpreter','LaTeX')
```

Remarcați folosirea în comanda `text` a comenziilor L<sup>A</sup>T<sub>E</sub>X pentru scrierea mai elegantă a relației de recurență.

---

### Sursa MATLAB 5.1 Calculul polinoamelor Legendre cu relația de recurență

---

```
function vl=vLegendre(x,n)
%VLEGENDRE - valoarea polinomului Legendre
%apel vl=vLegendre(x,n)
%x - punctele
%n - gradul
%vl - valoarea

pnml = ones(size(x));
if n==0, vl=pnml; return; end
pn = x;
if n==1, vl=pn; return; end
for k=2:n
    vl=x.*pn-1/(4-(k-1)^(-2)).*pnml;
    pnml=pn; pn=vl;
end
```

---



---

### Sursa MATLAB 5.2 Calculul coeficienților Legendre

---

```
function c=coeffLegendre(f,n)
%COEFFLEGENDRE - coeficientii aproximantei mcmmp Legendre
%apel c=coefLegendre(f,n)
%f - functia
%n - gradul

n3=2;
for k=0:n
    if k>0, n3=n3*k^2/(2*k-1)/(2*k+1); end
    c(k+1)=quadl(@fleg,-1,1,1e-12,0,f,k)/n3;
end
%subfunctii
function y=fleg(x,f,k)
y=f(x).*vLegendre(x,k);
```

---

**Sursa MATLAB 5.3** Aproximare în sensul celor mai mici pătrate cu polinoame Legendre

---

```
function y=approxLegendre(f,x,n)
%APPROXLEGENDRE - aproximare continua mcmmp Legendre
%apel y=approxLegendre(f,x,n)
%f - functia
%x - punctele
%n - gradul

c=coeffLegendre(f,n);
y=evalaprLegendre(c,x);
```

---

```
function y=evalaprLegendre(c,x)
%EVALAPRLEGENDRE - evaluare aproximanta Legendre mcmmp

y=zeros(size(x));
for k=1:length(c)
    y=y+c(k)*vLegendre(x,k-1);
end
```

---

**Polinoamele Cebîșev de speță I**

Polinoamele lui Cebîșev<sup>4</sup> de speță I se pot defini prin relația

$$T_n(x) = \cos(n \arccos x), \quad n \in \mathbb{N}. \quad (5.1.34)$$

Din identitatea trigonometrică

$$\cos(k+1)\theta + \cos(k-1)\theta = 2 \cos \theta \cos k\theta$$

și din (5.1.34), punând  $\theta = \arccos x$  se obține

$$\begin{aligned} T_{k+1}(x) &= 2xT_k(x) - T_{k-1}(x) \quad k = 1, 2, 3, \dots \\ T_0(x) &= 1, \quad T_1(x) = x. \end{aligned} \quad (5.1.35)$$

---

<sup>4</sup>



Pafnuti Levovici Cebîșev (1821-1894), matematician rus, cel mai important reprezentant al școlii matematice din Sankt Petersburg. A avut contribuții de pionierat în domeniul teoriei numerelor, calculului probabilităților și teoriei aproximării. Este considerat fondatorul teoriei constructive a funcțiilor, dar a lucrat și în domeniul mecaniciei și al balisticiei.

De exemplu

$$\begin{aligned}T_2(x) &= 2x^2 - 1, \\T_3(x) &= 4x^3 - 3x, \\T_4(x) &= 8x^4 - 8x^2 + 1\end{aligned}$$

s.a.m.d.

Din relația (5.1.35) se obține pentru coeficientul dominant al lui  $T_n$  valoarea  $2^{n-1}$  (dacă  $n \geq 1$ ), deci polinomul Cebîșev de speță I monic este

$$\overset{\circ}{T}_n(x) = \frac{1}{2^{n-1}} T_n(x), \quad n \geq 0, \quad \overset{\circ}{T}_0 = T_0. \quad (5.1.36)$$

Din (5.1.34) se pot obține rădăcinile lui  $T_n$

$$x_k^{(n)} = \cos \theta_k^{(n)}, \quad \theta_k^{(n)} = \frac{2k-1}{2n}\pi, \quad k = \overline{1, n}. \quad (5.1.37)$$

Ele sunt proiecțiile pe axa reală ale punctelor de pe cercul unitate de argument  $\theta_k^{(n)}$ ; figura 5.3 ilustrează acest lucru pentru  $n=4$ .

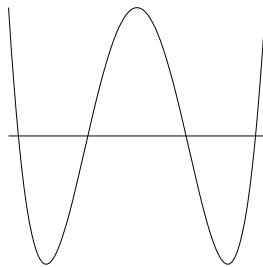
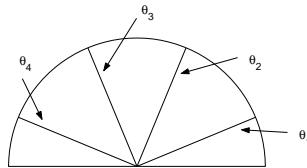
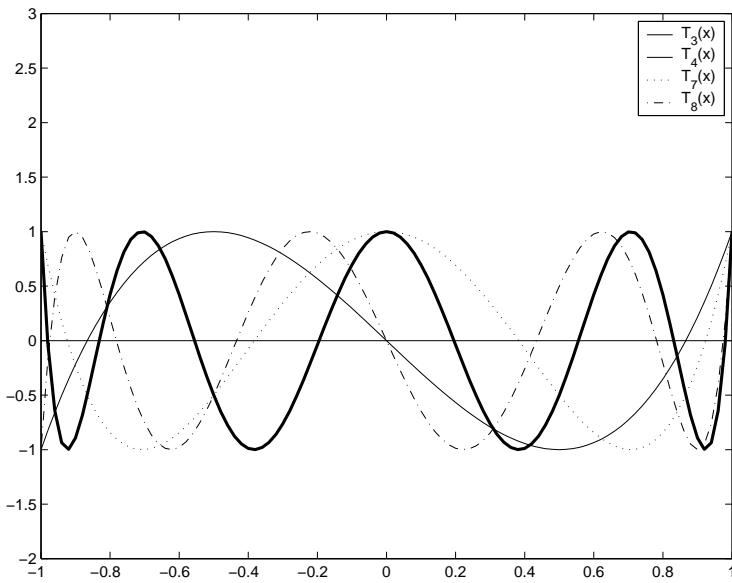


Figura 5.3: Polinomul Cebîșev  $T_4$  și rădăcinile sale

Pe intervalul  $[-1, 1]$   $T_n$  oscilează de la +1 la -1, atingând aceste valori extreme în punctele

$$y_k^{(n)} = \cos \eta_k^{(n)}, \quad \eta_k^{(n)} = \frac{k\pi}{n}, \quad k = \overline{0, n}.$$

În figura 5.4 apar graficele unor polinoame Cebîșev de speță I.

Figura 5.4: Polinoamele Cebîșev  $T_3, T_4, T_7, T_8$  pe  $[-1,1]$ 

Polinoamele Cebîșev de speță I sunt ortogonale în raport cu măsura

$$d\lambda(x) = \frac{dx}{\sqrt{1-x^2}}, \quad \text{pe } [-1, 1].$$

Se verifică ușor din (5.1.34) că

$$\begin{aligned} \int_{-1}^1 T_k(x) T_\ell(x) \frac{dx}{\sqrt{1-x^2}} &= \int_0^\pi T_k(\cos \theta) T_\ell(\cos \theta) d\theta \\ &= \int_0^\pi \cos k\theta \cos \ell\theta d\theta = \begin{cases} 0 & \text{dacă } k \neq \ell \\ \pi & \text{dacă } k = \ell = 0 \\ \pi/2 & \text{dacă } k = \ell \neq 0 \end{cases} \end{aligned} \tag{5.1.38}$$

Dezvoltarea în serie Fourier de polinoame Cebîșev este dată de

$$f(x) = \sum_{j=0}^{\infty}' c_j T_j(x) := \frac{1}{2}c_0 + \sum_{j=1}^{\infty} c_j T_j(x), \tag{5.1.39}$$

unde

$$c_j = \frac{2}{\pi} \int_{-1}^1 f(x) T_j(x) \frac{dx}{\sqrt{1-x^2}}, \quad j \in \mathbb{N}.$$

Păstrând în (5.1.39) numai termenii de grad cel mult  $n$  se obține o aproximare polinomială

utilă de grad  $n$

$$\tau_n(x) = \sum_{j=0}^n c_j T_j(x) := \frac{c_0}{2} + \sum_{j=1}^n c_j T_j(x), \quad (5.1.40)$$

având eroarea

$$f(x) - \tau_n(x) = \sum_{j=n+1}^{\infty} c_j T_j(x) \approx c_{n+1} T_{n+1}(x). \quad (5.1.41)$$

Aproximanta din (5.1.40) este cu atât mai bună cu cât coeficienții din extremitatea dreaptă tind mai repede către zero. Eroarea (5.1.41) oscilează în esență între  $+c_{n+1}$  și  $-c_{n+1}$  și este deci de mărime „uniformă“. Acest lucru contrastează puternic cu dezvoltarea Taylor în jurul lui  $x = 0$ , unde polinomul de grad  $n$  are eroarea proporțională cu  $x^{n+1}$  pe  $[-1, 1]$ .

În raport cu produsul scalar

$$(f, g)_T := \sum_{k=1}^{n+1} f(\xi_k)g(\xi_k), \quad (5.1.42)$$

unde  $\{\xi_1, \dots, \xi_{n+1}\}$  este mulțimea zerourilor lui  $T_{n+1}$ , are loc următoarea proprietate de ortogonalitate discretă

$$(T_i, T_j)_T = \begin{cases} 0, & i \neq j \\ \frac{n+1}{2}, & i = j \neq 0 \\ n+1, & i = j = 0 \end{cases}.$$

Într-adevăr, avem  $\arccos \xi_k = \frac{2k-1}{2n+2}\pi$ ,  $k = \overline{1, n+1}$ . Să calculăm acum produsul scalar:

$$\begin{aligned} (T_i, T_j)_T &= (\cos i \arccos t, \cos j \arccos t)_T = \\ &= \sum_{k=1}^{n+1} \cos(i \arccos \xi_k) \cos(j \arccos \xi_k) = \\ &= \sum_{k=1}^{n+1} \cos\left(i \frac{2k-1}{2(n+1)}\pi\right) \cos\left(j \frac{2k-1}{2(n+1)}\pi\right) = \\ &= \frac{1}{2} \sum_{k=1}^{n+1} \left[ \cos(i+j) \frac{2k-1}{2(n+1)}\pi + \cos(i-j) \frac{2k-1}{2(n+1)}\pi \right] = \\ &= \frac{1}{2} \sum_{k=1}^{n+1} \cos(2k-1) \frac{i+j}{2(n+1)}\pi + \frac{1}{2} \sum_{k=1}^{n+1} \cos(2k-1) \frac{i-j}{2(n+1)}\pi. \end{aligned}$$

Notăm  $\alpha := \frac{i+j}{2(n+1)}\pi$ ,  $\beta := \frac{i-j}{2(n+1)}\pi$  și

$$\begin{aligned} S_1 &:= \frac{1}{2} \sum_{k=1}^{n+1} \cos(2k-1)\alpha, \\ S_2 &:= \frac{1}{2} \sum_{k=1}^{n+1} \cos(2k-1)\beta. \end{aligned}$$

Deoarece

$$\begin{aligned} 2 \sin \alpha S_1 &= \sin 2(n+1)\alpha, \\ 2 \sin \beta S_2 &= \sin 2(n+1)\beta, \end{aligned}$$

se obține  $S_1 = 0$  și  $S_2 = 0$ .

În raport cu produsul scalar

$$\begin{aligned} (f, g)_U &:= \frac{1}{2} f(\eta_0)g(\eta_0) + f(\eta_1)g(\eta_1) + \cdots + f(\eta_{n-1})g(\eta_{n-1}) + \frac{1}{2} f(\eta_n)g(\eta_n) \\ &= \sum_{k=0}^n'' f(\eta_k)g(\eta_k), \end{aligned} \quad (5.1.43)$$

unde  $\{\eta_0, \dots, \eta_n\}$  este mulțimea extremelor lui  $T_n$ , are loc o proprietate similară

$$(T_i, T_j)_U = \begin{cases} 0, & i \neq j \\ \frac{n}{2}, & i = j \neq 0 \\ n, & i = j = 0 \end{cases}.$$

În sursa MATLAB 5.4 se calculează aproximanta continuă în sensul celor mai mici pătrate de grad  $n$  cu polinoame Cebîșev de speță I. Se procedează la fel ca la aproximarea Legendre. Pentru a evita la calculul coeficienților  $c_k = (f, T_k)$  aproximarea unor integrale improprii de forma

$$c_k = \frac{2}{\pi} \int_{-1}^1 \frac{1}{\sqrt{1-x^2}} f(x) \cos(k \arccos x) dx,$$

s-a efectuat schimbarea de variabilă  $u = \arccos x$ . Astfel, formula pentru  $c_k$  devine

$$c_k = \frac{2}{\pi} \int_0^\pi f(\cos u) \cos ku du.$$

Sursa 5.5 calculează valorile polinomului Cebîșev de speță I de grad  $n$  în niște puncte date, iar sursa 5.6 obține coeficienții Fourier ai aproximării în serie Fourier de polinoame Cebîșev.

Funcția pentru calculul aproximantei discrete corespunzătoare produsului scalar (5.1.42) se dă în sursa 5.7. O astfel de aproximare este foarte utilă în probleme practice, iar precizia nu este cu mult mai mică decât cea a aproximării continue; în plus este mult mai simplu de calculat deoarece nu este nevoie de aproximări de integrale (produsele scalare devin sume finite discrete, vezi sursa 5.8).

Dintre toate polinoamele monice de grad  $n$ ,  $\overset{\circ}{T}_n$  are norma uniformă cea mai mică.

**Teorema 5.1.2 (Cebîșev).** *Pentru orice polinom monic  $\overset{\circ}{p}_n$  de grad  $n$  are loc*

$$\max_{-1 \leq x \leq 1} |\overset{\circ}{p}_n(x)| \geq \max_{-1 \leq x \leq 1} \left| \overset{\circ}{T}_n(x) \right| = \frac{1}{2^{n-1}}, \quad n \geq 1, \quad (5.1.44)$$

unde  $\overset{\circ}{T}_n(x)$  este dat de (5.1.36).

**Sursa MATLAB 5.4** Aproximare în sensul celor mai mici pătrate cu polinoame Cebîșev de speță I

```

function y=approxChebyshev(f,x,n)
%APPROXCHEBYSHEV - aproximare continua mcmmpp Cebisev #1
%apel Y=APPROXCHEBYSHEV(F,X)
%F - functia
%X - punctele
%N - gradul
%Y - valorile aproximantei

c=coeffChebyshev(f,n);
y=evalChebyshev(c,x);

function y=evalChebyshev(c,x)
%EVALCHEBYSHEV - evaluare aproximanta Cebisev mcmmpp

y=c(1)/2*ones(size(x));
for k=1:length(c)-1
    y=y+c(k+1)*vChebyshev(x,k);
end

```

---

**Sursa MATLAB 5.5** Calculul polinoamelor Cebîșev de speță I cu relația de recurență

```

function y=vChebyshev(x,n)
%VCHEBYSHEV - calculul valorilor polinomului Cebisev
%apel Y=VCHEBYSHEV(X,N)
%X - punctele
%N - gradul
%Y - valorile polinomului Cebisev

pnml=ones(size(x));
if n==0, y=pnml; return; end
pn=x;
if n==1, y=pn; return; end
for k=2:n
    y=2*x.*pn-pnml;
    pnml=pn;
    pn=y;
end

```

---

---

**Sursa MATLAB 5.6** Aproximare în sensul celor mai mici pătrate cu polinoame Cebîșev de speță I – continuare: calculul coeficienților Fourier
 

---

```
function c=coefceb(f,n,p)
%COEFCEB - coeficienti Cebisev mcmmmp
%apel c=coefceb(f,n,p)
%f - functia
%n - gradul
%p - tabela coeficientilor

if nargin < 3
    p=polceb(n);
end
for k=1:n+1
    c(k)=2/pi*quadl(@fceb,0,pi,1e-10,0,f,k-1);
end
%subfunctie
function y=fceb(x,f,k)
y=cos(k*x).*f(cos(x));
```

---



---

**Sursa MATLAB 5.7** Aproximantă Cebîșev discretă
 

---

```
function y=approxChebyshevdiscr(f,x,n)
%APPROXCHEBYSHEVDISCR - aproximare discreta mcmmmp Cebisev #1
%apel y=approxChebyshevdiscr(f,x,n)
%f - functia
%x - punctele
%n - gradul

c=coeffChebyshevdiscr(f,n);
y=evalChebyshev(c,x);
```

---



---

**Sursa MATLAB 5.8** Coeficienții aproximantei Cebîșev discrete
 

---

```
function c=coeffChebyshevdiscr(f,n)
%COEFFCHEBYSHEVDISCR - coeficienti Cebisev mcmmmp discreti
%apel c=coeffChebyshevdiscr(f,n)
%f - functia
%n - gradul

xi=cos((2*[1:n+1]-1)*pi/(2*n+2));
y=f(xi)';
for k=1:n+1
    c(k)=2/(n+1)*vChebyshev(xi,k-1)*y;
end
```

---

*Demonstrație.* Se face prin reducere la absurd. Presupunem că

$$\max_{-1 \leq x \leq 1} |p_n^\circ(x)| < \frac{1}{2^{n-1}}. \quad (5.1.45)$$

Atunci polinomul  $d_n(x) = \overset{\circ}{T}_n(x) - p_n^\circ(x)$  (de grad  $\leq n-1$ ) satisfacă

$$d_n(y_0^{(n)}) > 0, \quad d_n(y_1^{(n)}) < 0, \quad d_n(y_2^{(n)}) > 0, \dots, (-1)^n d_n(y_n^{(n)}) > 0. \quad (5.1.46)$$

Deoarece  $d_n$  are  $n$  schimbări de semn, el este identic nul; aceasta contrazice (5.1.46) și astfel (5.1.45) nu poate fi adevărată.  $\square$

Rezultatul (5.1.44) se poate interpreta în modul următor: cea mai bună aproximare uniformă din  $\mathbb{P}_{n-1}$  pe  $[-1, 1]$  a lui  $f(x) = x^n$  este dată de  $x^n - \overset{\circ}{T}_n(x)$ , adică, de agregarea termenilor până la gradul  $n-1$  din  $\overset{\circ}{T}_n$  luati cu semnul minus. Din teoria aproximărilor uniforme se știe că cea mai bună aproximare polinomială uniformă este unică. Deci, egalitatea în (5.1.44) poate avea loc numai dacă  $p_n^\circ(x) = \overset{\circ}{T}_n(x)$ .

### Polinoamele lui Cebîșev de speță a doua

Se definesc prin

$$Q_n(t) = \frac{\sin[(n+1)\arccos t]}{\sqrt{1-t^2}}, \quad t \in [-1, 1]$$

Ele sunt ortogonale pe  $[-1, 1]$  în raport cu măsura  $d\lambda(t) = w(t)dt$ ,  $w(t) = \sqrt{1-t^2}$ . Relația de recurență este

$$Q_{n+1}(t) = 2tQ_n(t) - Q_{n-1}(t), \quad Q_0(t) = 1, \quad Q_1(t) = 2t.$$

### Polinoamele lui Laguerre

Polinoamele lui Laguerre<sup>5</sup> sunt ortogonale pe  $[0, \infty)$  în raport cu ponderea  $w(t) = t^\alpha e^{-t}$ ,  $\alpha > 1$ . Se definesc prin

$$l_n^\alpha(t) = \frac{e^t t^{-\alpha}}{n!} \frac{d^n}{dt^n}(t^{n+\alpha} e^{-t}).$$

<sup>5</sup>



Edmond Laguerre (1834-1886) matematician francez, activ în Paris, cu contribuții esențiale în geometrie, algebră și analiză.

Relația de recurență pentru polinoame monice  $\tilde{l}_n^\alpha$  este

$$\tilde{l}_{n+1}^\alpha(t) = (t - \alpha_n)\tilde{l}_n^\alpha(t) - (2n + \alpha + 1)\tilde{l}_{n-1}^\alpha(t),$$

cu  $\alpha_0 = \Gamma(1 + \alpha)$  și  $\alpha_k = k(k + \alpha)$ , pentru  $k > 0$ .

### Polinoamele lui Hermite

Polinoamele lui Hermite sunt ortogonale pe  $(-\infty, \infty)$  în raport cu ponderea  $w(t) = e^{-t^2}$  și se definesc prin

$$H_n(t) = (-1)^n e^{t^2} \frac{d^n}{dt^n}(e^{-t^2}).$$

Relația de recurență pentru polinoamele monice  $\tilde{H}_n(t)$  este

$$\tilde{H}_{n+1}(t) = t\tilde{H}_n(t) - \beta_n \tilde{H}_{n-1}(t),$$

cu  $\beta_0 = \sqrt{\pi}$  și  $\beta_k = k/2$ , pentru  $k > 0$ .

### Polinoamele lui Jacobi

Sunt ortogonale pe  $[-1, 1]$  în raport cu ponderea  $w(t) = (1 - t)^\alpha(1 + t)^\beta$ ,  $\alpha, \beta > -1$ . Pentru relația de recurență vezi observația 7.3.5, pagina 280.

### Un exemplu în MATLAB

Fie funcția  $f : [-1, 1] \rightarrow \mathbb{R}$ ,  $f(x) = x + \sin \pi x^2$ . Vom studia practic următoarele aproximante în sensul celor mai mici pătrate: Legendre, Cebîșev de speță I continuă și Cebîșev de speță I discretă. Pentru început vom încerca să determinăm gradul aproximantei, astfel ca să fie atinsă o anumită eroare. Ideea este următoarea: vom evalua funcția și aproximanta pe un număr mare de puncte și vom vedea dacă norma  $\|\cdot\|_\infty$  a vectorului diferențelor valorilor este mai mică decât eroarea prescrisă. Dacă da, se returnează gradul, iar dacă nu se continuă cu un  $n$  mai mare. Sursa MATLAB 5.9 returnează gradul aproximantei și eroarea efectivă. De exemplu, pentru o eroare de  $10^{-3}$  se obțin următoarele rezultate

```
>> fp=@(x)x+sin(pi*x.^2);
>> [n,er]=excebc(fp,1e-3,@approxLegendre)
n =
     8
er =
    9.5931e-004
>> [n,er]=excebc(fp,1e-3,@approxChebyshev)
n =
     8
er =
    5.9801e-004
>> [n,er]=excebc(fp,1e-3,@approxChebyshevDiscr)
n =
    11
er =
    6.0161e-004
```

**Sursa MATLAB 5.9 Test aproximante mcmmmp**

```

function [n,erref]=excebc(f,err,proc)
%f - functia
%err - eroarea
%proc - procedeul de aproximare (Legendre, Cebisev
%       continuu, Cebisev discret)
x=linspace(-1,1,100); %abscisele
y=f(x); %valorile functiei
n=1;
while 1
    ycc=proc(f,x,n); %valorile aproximantei
    erref=norm(y-ycc,inf); %eroarea
    if norm(y-ycc,inf)<err %succes
        return
    end
    n=n+1;
end

```

Programul următor (exorthpol.m) calculează coeficienții și vizualizează cele trei tipuri de aproximante pentru un grad dat de utilizator:

```

k=input('k=');
fp=inline('x+sin(pi*x.^2)');
x=linspace(-1,1,100);
y=fp(x);
yle=approxLegendre(fp,x,k);
ycc=approxChebyshev(fp,x,k);
ycd=approxChebyshevDiscr(fp,x,k);
plot(x,y,:', x,yle,'--', x,ycc,'.-',x,ycd,'-');
legend('f','Legendre','Cebisev continuu','Cebisev discret',4)
title(['k=',int2str(k)],'FontSize',14);
cl=coeffLegendre(fp,k)
ccc=coeffChebyshev(fp,k)
ccd=coeffChebyshevDiscr(fp,k)

```

Pentru  $k=3$  și  $k=4$  se obțin următoarele valori ale coeficienților

```

k=3 cl =
Columns 1 through 3
0.50485459411369 1.000000000000000 0.56690206826580
Column 4
0.000000000000000
ccc =
Columns 1 through 3
0.94400243153647 1.000000000000114 0.000000000000000
Column 4
-0.000000000000000
ccd =

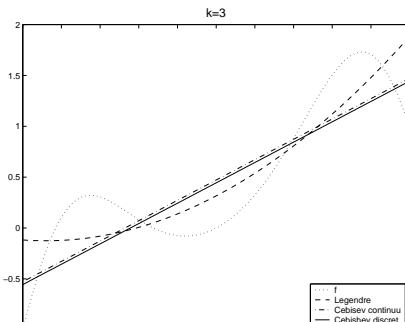
```

```

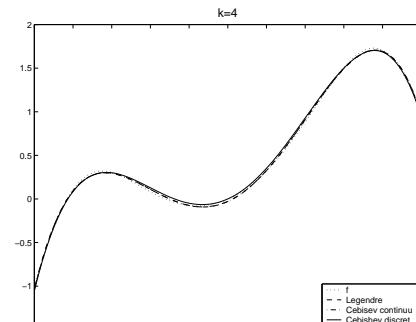
Columns 1 through 3
0.88803168065243    1.000000000000000
Column 4
-0.000000000000000
k=4 cl =
Columns 1 through 3
0.50485459411369    1.000000000000000    0.56690206826580
Columns 4 through 5
0.000000000000000   -4.02634086092250
ccc =
Columns 1 through 3
0.94400243153647    1.000000000000114    0.000000000000000
Columns 4 through 5
-0.000000000000000   -0.49940325827041
ccd =
Columns 1 through 3
0.94400233847188    1.000000000000000   -0.02739538442025
Columns 4 through 5
-0.000000000000000   -0.49939655365619

```

Graficele apar în figura 5.5.



(a)  $k = 3$



(b)  $k = 4$

Figura 5.5: Aproximante în sensul celor mai mici pătrate de grad  $k$  pentru funcția  $f : [-1, 1] \rightarrow \mathbb{R}$ ,  $f(x) = x + \sin \pi x^2$

## 5.2. Polinoame și potrivirea datelor în MATLAB

MATLAB reprezintă un polinom

$$p(x) = p_1x^n + p_2x^{n-1} + p_nx + p_{n+1}$$

printr-un vector linie  $p=[p(1) \ p(2) \ \dots \ p(n+1)]$  al coeficienților, ordonați descrescător după puterile variabilei.

Să considerăm trei probleme legate de polinoame:

- *evaluarea* – dându-se coeficienții să se calculeze valoarea polinomului în unul sau mai multe puncte;
- *determinarea rădăcinilor* – dându-se coeficienții să se determine rădăcinile polinomului;
- *potrivirea datelor (data fitting)* – dându-se o mulțime de date  $(x_i, y_i)_{i=1}^m$  să se determine un polinom (sau o altă combinație de funcții de bază) care „se potrivește” cu aceste date.

Evaluarea se face cu ajutorul schemei lui Horner, implementată în MATLAB prin funcția `polyval`. În comanda `y=polyval(p, x)`  $x$  poate fi o matrice, în acest caz evaluarea făcându-se element cu element (deci, în sens tablou). Evaluarea în sens matricial, adică obținerea matricei

$$p(X) = p_1 X^n + p_2 X^{n-1} + \dots + p_n X + p_{n+1},$$

unde  $X$  este o matrice pătratică se poate face cu comanda `Y = polyvalm(p, X)`.

Rădăcinile (reale și complexe) ale polinomului  $p$  se pot obține cu `z = roots(p)`. Funcția `poly` realizează operația inversă, adică construiește polinomul cunoscând rădăcinile. Ea acceptă ca argument și o matrice pătratică  $A$ , caz în care `p=poly(A)` calculează polinomul caracteristic al lui  $A$ , adică  $\det(xI - A)$ .

Funcția `polyder` calculează coeficienții derivatei unui polinom, fără a o evalua.

Ca exemplu, să considerăm polinomul  $p(x) = x^2 - x - 1$ . Rădăcinile lui le obținem cu

```
>> p = [1 -1 -1]; z = roots(p)
z =
-0.6180
 1.6180
```

Verificăm, în limitele erorilor de rotunjire, că acestea sunt rădăcinile:

```
>> polyval(p, z)
ans =
1.0e-015 *
-0.1110
 0.2220
```

Observăm că  $p$  este polinomul caracteristic al unei anumite matrice  $2 \times 2$

```
>> A = [0 1; 1 1]; cp = poly(A)
cp =
 1.0000   -1.0000   -1.0000
```

Teorema Cayley-Hamilton ne spune că orice matrice satisface polinomul său caracteristic. Aceasta se verifică în limita erorilor de rotunjire:

```
>> polyvalm(cp, A)
ans =
1.0e-015 *
 0.1110      0
      0     0.1110
```

Înmulțirea și împărțirea polinoamelor se realizează cu `conv` și `deconv`. Sintaxa lui `deconv` este  $[q, r] = \text{deconv}(g, h)$ , unde  $g$  este deîmpărțitul,  $h$  împărțitorul,  $q$  câtul și  $r$  restul. În exemplul următor vom împărți  $x^2 - 2x - x + 2$  la  $x - 2$ , obținând câtul  $x^2 - 1$  și restul 0. Polinomul inițial se va obține apoi cu `conv`.

```
>> g = [1 -2 -1 2]; h=[1 -2];
>> [q,r] = deconv(g,h)
q =
    1      0     -1
r =
    0      0      0      0
>> conv(h,q)+r
ans =
    1     -2     -1      2
```

Să tratăm acum problema potrivirii datelor. Să presupunem că avem  $m$  observații ( $y_i$ ) măsurate în valorile specifice ( $t_i$ ):

$$y_i = y(t_i), \quad i = \overline{1, m}.$$

*Modelul* nostru este o combinație de  $n$  funcții de bază ( $\pi_i$ )

$$y(t) \approx c_1\pi_1(t, \alpha) + \cdots + c_n\pi_n(t, \alpha).$$

Matricea de proiectare (design matrix)  $A(\alpha)$  va fi matricea cu elementele

$$a_{i,j} = \pi_j(t_i, \alpha),$$

ale cărei elemente pot depinde de  $\alpha$ . În notație matricială, modelul se poate exprima ca:

$$y \approx A(\alpha)c.$$

*Reziduurile* sunt diferențele dintre valorile observate și cele date de model

$$r_i = y_i - \sum_{j=1}^n c_j \pi_j(t_i, \alpha)$$

sau în notație matricială

$$r = y - A(\alpha)c.$$

Ne propunem să minimizăm o anumită normă a reziduurilor. Cele mai frecvente alegeri sunt

$$\|r\|_2^2 = \sum_{i=1}^m r_i^2$$

sau

$$\|r\|_{2,w}^2 = \sum_{i=1}^m w_i r_i^2.$$

O explicație intuitivă, fizică, a celei de-a doua alegeri ar fi aceea că anumite observații sunt mai importante decât altele și le vom asocia ponderi,  $w_i$ . De exemplu, dacă la observația  $i$  eroarea este aproximativ  $e_i$ , atunci putem alege  $w_i = 1/e_i$ . Deci, avem de a face cu o problemă discretă de aproximare în sensul celor mai mici pătrate. Problema este liniară dacă nu depinde de  $\alpha$  și neliniară în caz contrar.

Orice algoritm de rezolvare a unei probleme de aproximare în sensul celor mai mici pătrate fără ponderi poate fi utilizat la rezolvarea unei probleme cu ponderi prin scalarea observațiilor și a matricei de proiectare. În MATLAB aceasta se poate realiza prin

```
A=diag(w)*A
y=diag(w)*y
```

Dacă problema este liniară și avem mai multe observații decât funcții de bază, suntem conduși la rezolvarea sistemului supradeterminat (vezi secțiunea 4.7.2)

$$Ac \approx y,$$

pe care îl vom rezolva în sensul celor mai mici pătrate

$$c = A \setminus y.$$

Abordarea teoretică se bazează pe rezolvarea ecuațiilor normale

$$A^T Ac = A^T y.$$

Dacă funcțiile de bază sunt liniar independente și deci  $A^T A$  nesingulară, soluția este

$$c = (A^T A)^{-1} A^T y,$$

sau

$$c = A^+ y,$$

unde  $A^+$  este pseudo-inversa lui  $A$ . Ea se poate calcula cu funcția MATLAB `pinv`.

Fie sistemul  $Ax = b$  arbitrar. Dacă  $A$  este o matrice  $m \times n$  cu  $m > n$  și are rangul  $n$ , atunci fiecare din următoarele trei instrucțiuni

```
x=A\b
x=pinv(A)*b
x=inv(A'*A)*A'*b
```

calculează aceeași soluție în sensul celor mai mici pătrate, deși operatorul  $\setminus$  o face cel mai repede.

Totuși, dacă  $A$  nu are rangul complet, soluția în sensul celor mai mici pătrate nu este unică. Există mai mulți vectori care minimizează norma  $\|Ax - b\|_2$ . Soluția calculată cu `x=A\b` este o soluție de bază; ea are cel mult  $r$  componente nenule, unde  $r$  este rangul lui  $A$ . Soluția calculată cu `x=pinv(A)*b` este soluția cu normă minimă (ea minimizează `norm(x)`). Încercarea de a calcula o soluție cu `x=inv(A'*A)*A'*b` eșuează dacă  $A^*A$  este singulară. Iată un exemplu care ilustrează diversele soluții. Matricea

`A=[1, 2, 3; 4, 5, 6; 7, 8, 9; 10, 11, 12];`  
 este deficită de rang. Dacă `b=A(:, 2)`, atunci o soluție evidentă a lui `A*x=b` este `x=[0, 1, 0]'`. Nici una dintre abordările de mai sus nu calculează pe `x`. Operatorul  $\setminus$  ne dă

```
>> x=A\b
Warning: Rank deficient, rank = 2 tol = 1.4594e-014.
x =
0.5000
0
0.5000
```

Această soluție are două componente nenele. Varianta cu pseudoinversă ne dă

```
>> y=pinv(A)*b
y =
0.3333
0.3333
0.3333
```

Se observă ca  $\text{norm}(y) = 0.5774 < \text{norm}(x) = 0.7071$ . A treia variantă eșuează complet:

```
>> z=inv(A'*A)*A'*b
Warning: Matrix is singular to working precision.
z =
Inf
Inf
Inf
```

Abordarea bazată pe ecuații normale are mai multe dezavantaje. Ecuațiile normale sunt întotdeauna mai prost condiționate decât sistemul supradeterminat inițial. Numărul de condiționare se ridică de fapt la pătrat<sup>6</sup>:

$$\text{cond}(A^T A) = \text{cond}(A)^2.$$

În reprezentarea în virgulă flotantă, chiar dacă coloanele lui  $A$  sunt liniar independente,  $(A^T A)^{-1}$  ar putea fi aproape singulară.

MATLAB evită ecuațiile normale. Operatorul \ folosește intern factorizarea QR. Soluția se poate exprima prin  $c=R\backslash(Q'*y)$ .

Dacă baza în care se face aproximarea este  $1, t, \dots, t^n$ , se poate folosi funcția polyfit. Comanda  $p=\text{polyfit}(x, y, n)$  calculează coeficienții polinomului de aproximare discretă de grad  $n$  în sensul celor mai mici pătrate pentru datele  $x$  și  $y$ . Dacă  $n \geq m$ , se returnează coeficienții polinomului de interpolare.

Vom considera două exemple.

O cantitate  $y$  este măsurată în diferite momente de timp,  $t$ , pentru a produce următoarele observații:

$t$	$y$
0.0	0.82
0.3	0.72
0.8	0.63
1.1	0.60
1.6	0.55
2.3	0.50

<sup>6</sup>Pentru o matrice dreptunghiulară  $X$ , numărul de condiționare ar putea fi definit prin  $\text{cond}(X) = \|X\| \|X^+\|$

Acste date pot fi introduse MATLAB prin

```
t=[0,0.3,0.8,1.1,1.6,2.3]';
y=[0.82,0.72,0.63,0.60,0.55,0.50]';
```

Vom încerca să modelăm datele cu ajutorul unei funcții de forma

$$y(t) = c_1 + c_2 e^{-t}.$$

Coefficienții necunoscuți se vor calcula prin metoda celor mai mici pătrate. Avem 6 ecuații și două necunoscute, reprezentate printr-o matrice  $6 \times 2$

```
>> E=[ones(size(t)),exp(-t)]
E =
    1.0000    1.0000
    1.0000    0.7408
    1.0000    0.4493
    1.0000    0.3329
    1.0000    0.2019
    1.0000    0.1003
```

Soluția în sensul celor mai mici pătrate se poate găsi cu ajutorul operatorului \:

```
c=E\y
c =
    0.4760
    0.3413
```

Urmează reprezentarea grafică pe puncte echidistante, completată cu datele originale:

```
T=[0:0.1:2.5]';
Y=[ones(size(T)),exp(-T)]*c;
plot(T,Y,'-',t,y,'o')
xlabel('t'); ylabel('y');
```

Se poate vedea că  $E \cdot c \neq y$ , dar diferența are normă euclidiană minimă (figura 5.6). Dacă matricea  $A$  este deficientă de rang (adică nu are coloane liniar independente), atunci soluția în sensul celor mai mici pătrate a sistemului  $Ax = b$  nu este unică. În acest caz operatorul \ dă un mesaj de avertizare și produce o soluție de bază cu cel mai mic număr posibil de elemente nenule.

Al doilea exemplu are ca date de intrare rezultatele de recensământelor obținute de U. S. Census pentru anii 1900–2000, din zece în zece ani, exprimate în milioane de oameni:

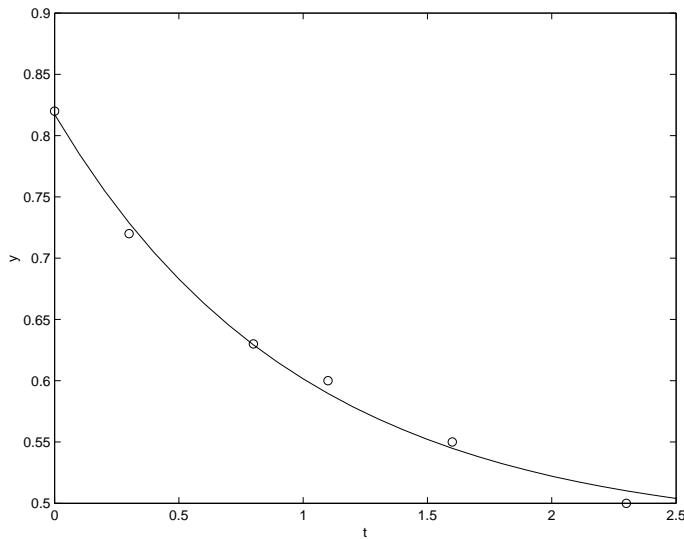


Figura 5.6: Ilustrare a potrivirii datelor

$t$	$y$
1900	75.995
1910	91.972
1920	105.711
1930	123.203
1940	131.669
1950	150.697
1960	179.323
1970	203.212
1980	226.505
1990	249.633
2000	281.422

Se dorește modelarea creșterii populației printr-un polinom de gradul al treilea

$$y(t) = c_1 t^3 + c_2 t^2 + c_3 t + c_4$$

și predicția populației din 2010.

Dacă încercăm să calculăm coeficienții cu  $c=polyfit(t, y, 3)$ , matricea sistemului va fi prost condiționată, coloanele ei vor fi aproape liniar dependente și vom obține mesajul

```
Warning: Polynomial is badly conditioned. Remove repeated
data points or try centering and scaling as
described in HELP POLYFIT.
```

Vom scala datele de intrare:

$$s = (t - \bar{t})/50.$$

Noua variabilă este în intervalul  $[-1, 1]$ , iar sistemul va fi bine condiționat. Script-ul MATLAB 5.10, `census.m`, calculează coeficienții, reprezintă datele și polinomul și estimează populația în 2010. Estimația apare și în clar și marcată cu un asterisc (vezi figura 5.7).

---

#### Sursa MATLAB 5.10 Exemplu de aproximare în sensul celor mai mici pătrate

---

```
%CENSUS - exemplu cu recensamantul
%          potrivire polinomiala

%datele
y = [ 75.995 91.972 105.711 123.203 131.669 150.697 ...
      179.323 203.212 226.505 249.633 281.422]';
t = (1900:10:2000)';
x = (1890:1:2019)';
w = 2010;

s=(t-1950)/50;
xs=(x-1950)/50;
cs=polyfit(s,y,3);
zs=polyval(cs,xs);
est=polyval(cs,(2010-1950)/50);
plot(t,y,'o',x,zs,'-',w,est,'*')
text(1990,est,num2str(est))
title('Populatia SUA', 'FontSize', 14)
xlabel('anul', 'FontSize', 12)
ylabel('milioane', 'FontSize', 12)
```

---

## 5.3. Interpolare polinomială

### 5.3.1. Spațiul $H^n[a, b]$

Pentru  $n \in \mathbb{N}^*$ , definim

$$H^n[a, b] = \{f : [a, b] \rightarrow \mathbb{R} : f \in C^{n-1}[a, b], f^{(n-1)} \text{ absolut continuă pe } [a, b]\}. \quad (5.3.1)$$

Orice funcție  $f \in H^n[a, b]$  admite o reprezentare de tip Taylor cu restul sub formă integrală

$$f(x) = \sum_{k=0}^{n-1} \frac{(x-a)^k}{k!} f^{(k)}(a) + \int_a^x \frac{(x-t)^{n-1}}{(n-1)!} f^{(n)}(t) dt. \quad (5.3.2)$$

$H^n[a, b]$  este un spațiu liniar.

**Observația 5.3.1.** Funcția  $f : I \rightarrow \mathbb{R}$ ,  $I$  interval, se numește absolut continuă pe  $I$  dacă  $\forall \varepsilon > 0 \exists \delta > 0$  astfel încât oricare ar fi un sistem finit de subintervale disjuncte ale lui  $I$

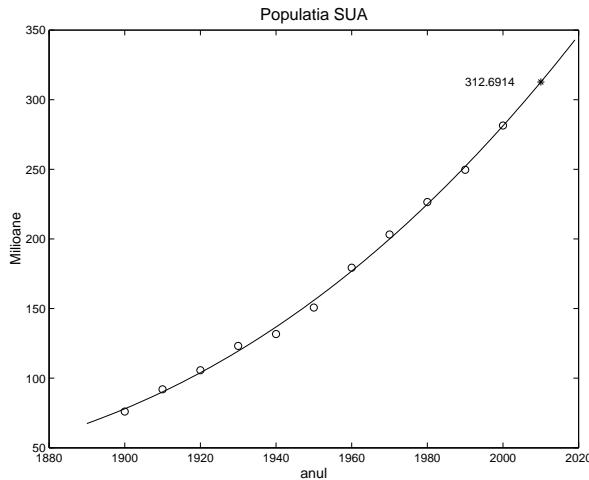


Figura 5.7: Ilustrarea exemplului cu recensământul

$\{(a_k, b_k)\}_{k=1}^n$  cu proprietatea  $\sum_{k=1}^n (b_k - a_k) < \delta$  să avem

$$\sum_{k=1}^n |f(b_k) - f(a_k)| < \varepsilon.$$

◊

Teorema următoare, datorată lui Peano<sup>7</sup>, de o importanță deosebită în analiza numerică, este o teoremă de reprezentare a funcțiilor liniare reale, definite pe  $H^n[a, b]$ .

**Teorema 5.3.2 (Peano).** Fie  $L$  o funcțională reală, continuă, definită pe  $H^n[a, b]$ . Dacă  $\text{Ker } L = \mathbb{P}_{n-1}$  atunci

$$Lf = \int_a^b K(t) f^{(n)}(t) dt, \quad (5.3.3)$$

unde

$$K(t) = \frac{1}{(n-1)!} L[(\cdot - t)_+^{n-1}] \quad (\text{nucleul lui Peano}). \quad (5.3.4)$$

Giuseppe Peano (1858-1932), matematician italian activ la Torino, cu contribuții fundamentale în logica matematică, teoria mulțimilor, și fundamentele matematicii. Teoremele generale de existență din domeniul teoriei ecuațiilor diferențiale îi poartă numele.



**Observația 5.3.3.** Funcția

$$z_+ = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

se numește *parte pozitivă*, iar  $z_+^n$  se numește *putere trunchiată*.  $\diamond$

*Demonstrație.*  $f$  admite o reprezentare de tip Taylor, cu restul în formă integrală

$$f(x) = T_{n-1}(x) + R_{n-1}(x),$$

unde

$$R_{n-1}(x) = \int_a^x \frac{(x-t)^{n-1}}{(n-1)!} f^{(n)}(t) dt = \frac{1}{(n-1)!} \int_a^b (x-t)_+^{n-1} f^{(n)}(t) dt$$

Aplicând  $L$  obținem

$$\begin{aligned} Lf &= \underbrace{LT_{n-1}}_0 + LR_{n-1} \Rightarrow Lf = \frac{1}{(n-1)!} L \left( \int_a^b (\cdot - t)_+^{n-1} f^{(n)}(t) dt \right) = \\ &\stackrel{\text{cont}}{=} \frac{1}{(n-1)!} \int_a^b L(\cdot - t)_+^{n-1} f^{(n)}(t) dt. \end{aligned}$$

$\square$

**Observația 5.3.4.** Concluzia teoremei rămâne valabilă și dacă  $L$  nu este continuă, ci are forma

$$Lf = \sum_{i=0}^{n-1} \int_a^b f^{(i)}(x) d\mu_i(x), \quad \mu_i \in BV[a, b]. \quad \diamond$$

**Corolarul 5.3.5.** Dacă  $K$  păstrează semn constant pe  $[a, b]$  și  $f^{(n)}$  este continuă pe  $[a, b]$ , atunci există  $\xi \in [a, b]$  astfel încât

$$Lf = \frac{1}{n!} f^{(n)}(\xi) L e_n, \quad (5.3.5)$$

unde  $e_k(x) = x^k$ ,  $k \in \mathbb{N}$ .

*Demonstrație.* Deoarece  $K$  păstrează semn constant pe  $[a, b]$  putem aplica în (5.3.3) teorema de medie

$$Lf = f^{(n)}(\xi) \int_a^b K_n(t) dt, \quad \xi \in [a, b].$$

Luând  $f = e_n$  se obține chiar (5.3.5).  $\square$

### 5.3.2. Interpolare Lagrange

Fie intervalul închis  $[a, b] \subset \mathbb{R}$ ,  $f : [a, b] \rightarrow \mathbb{R}$  și o mulțime de  $m + 1$  puncte distincte  $\{x_0, x_1, \dots, x_m\} \subset [a, b]$ .

**Teorema 5.3.6.** Există un polinom și numai unul  $L_m f \in \mathbb{P}_m$  astfel încât

$$\forall i = 0, 1, \dots, m, \quad (L_m f)(x_i) = f(x_i); \quad (5.3.6)$$

acest polinom se scrie sub forma

$$(L_m f)(x) = \sum_{i=0}^m f(x_i) \ell_i(x), \quad (5.3.7)$$

unde

$$\ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^m \frac{x - x_j}{x_i - x_j}. \quad (5.3.8)$$

**Definiția 5.3.7.** Polinomul  $L_m f$  definit astfel se numește polinom de interpolare Lagrange<sup>8</sup> a lui  $f$  relativ la punctele  $x_0, x_1, \dots, x_m$ , iar funcțiile  $\ell_i(x)$ ,  $i = \overline{0, m}$ , se numesc polinoame de bază (fundamentale) Lagrange asociate acelor puncte.

*Demonstrație.* Se verifică imediat că  $\ell_i \in \mathbb{P}_i$  și că  $\ell_i(x_j) = \delta_{ij}$  (simbolul lui Kronecker); rezultă că polinomul  $L_m f$  definit de (5.3.6) este de grad cel mult  $m$  și verifică (5.3.7). Pre-supunem că există un alt polinom  $p_m^* \in \mathbb{P}_m$  care verifică (5.3.7) și punem  $q_m = L_m - p_m^*$ ; avem  $q_m \in \mathbb{P}_m$  și  $\forall i = 0, 1, \dots, m$ ,  $q_m(x_i) = 0$ ; deci  $q_m$  având  $(m + 1)$  rădăcini distincte este identic nul, de unde unicitatea lui  $L_m$ .  $\square$

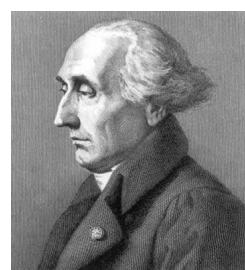
Fișierul `lagr.m` (sursa MATLAB 5.11) conține codul pentru calculul polinomului de interpolare Lagrange folosind formulele (5.3.7) și (5.3.8). Funcția `lagr` poate fi utilizată și pentru variabile simbolice. Exemplu

```
>> a = 1:3; b = (a-2).^3; syms x;
>> P = lagr(a,b,x); pretty(P);
    1/2 (-x + 2) (x - 3) + (1/2 x - 1/2) (x - 2)
```

**Observația 5.3.8.** Polinomul fundamental  $\ell_i$  este deci unicul polinom care verifică

$$\ell_i \in \mathbb{P}_m \text{ și } \forall j = 0, 1, \dots, m, \quad \ell_i(x_j) = \delta_{ij}.$$

Joseph Louis Lagrange (1736-1813), protejat al lui Euler. Clairaut scria despre Tânărul Lagrange: „...un Tânăr nu mai puțin remarcabil prin talent decât prin modestie; temperamentul său este bland și melancolic; nu cunoaște altă plăcere decât studiul.” Lagrange a avut contribuții fundamentale în calculul variațional, teoria numerelor și analiză matematică. Este cunoscut și pentru reprezentarea pe care a dat-o restului din formula lui Taylor. A dat formula de interpolare în 1794. Lucrarea sa *Mécanique Analytique*, publicată în 1788, l-a făcut unul din fondatorii mecanicii analitice.



**Sursa MATLAB 5.11 Interpolare Lagrange**

```

function fi=lagr(x,y,xi)
%LAGR - calculeaza polinomul de interpolare Lagrange
% x, y -coordonatele nodurilor
% xi - punctele in care se evalueaza polinomul

if nargin ~=3
    error('numar ilegal de argumente')
end
[mu,nu]=size(xi);
fi=zeros(mu,nu);
np1=length(y);
for i=1:np1
    z=ones(mu,nu);
    for j=[1:i-1,i+1:np1]
        z=z.*((xi-x(j))/(x(i)-x(j)));
    end;
    fi=fi+z*y(i);
end

```

Punând

$$u(x) = \prod_{j=0}^m (x - x_j),$$

din (5.3.8) se deduce că  $\forall x \neq x_i, \quad \ell_i(x) = \frac{u(x)}{(x-x_i)u'(x_i)}$ . ◊

În figura 5.8 apare reprezentare grafică a polinoamelor fundamentale Lagrange de gradul trei pentru nodurile  $x_k = k$ ,  $k = \overline{0, 3}$ .

Fișierul pf12b.m (sursa MATLAB 5.12) calculează polinoamele fundamentale Lagrange pentru niște noduri date pe o mulțime de puncte date.

Demonstrând teorema 5.3.6 am demonstrat de fapt existența și unicitatea soluției problemei generale de interpolare Lagrange:

(PGIL) Fiind date  $b_0, b_1, \dots, b_m \in \mathbb{R}$ , să se determine

$$p_m \in \mathbb{P}_m \text{ astfel încât } \forall i = 0, 1, \dots, n, \quad p_m(x_i) = b_i. \quad (5.3.9)$$

Problema (5.3.9) conduce la un sistem liniar de  $(m + 1)$  ecuații cu  $(m + 1)$  necunoscute (coeficienții lui  $p_m$ ).

Din teoria sistemelor liniare se știe că

$$\{\text{existența unei soluții } \forall b_0, b_1, \dots, b_m\} \Leftrightarrow \{\text{unicitatea soluției}\} \Leftrightarrow$$

$$\{(b_0 = b_1 = \dots = b_m = 0) \Rightarrow p_m \equiv 0\}$$

Punem  $p_m = a_0 + a_1x + \dots + a_mx^m$

$$a = (a_0, a_1, \dots, a_m)^T, \quad b = (b_0, b_1, \dots, b_m)^T$$

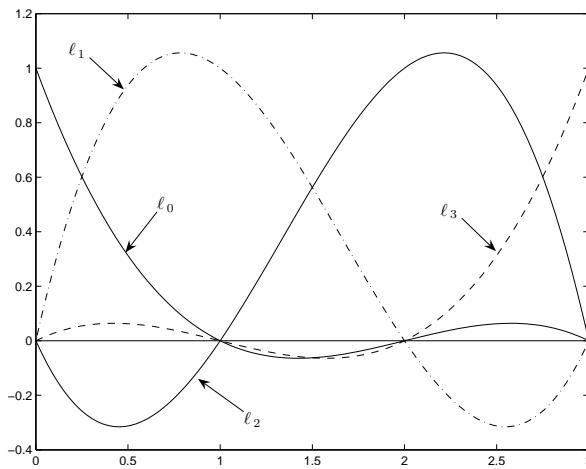


Figura 5.8: Polinoame fundamentale de interpolare pentru nodurile  $x = 0, 1, 2, 3$

**Sursa MATLAB 5.12** Calculul polinoamelor fundamentale Lagrange folosind facilități MATLAB. Rezultatul este returnat într-o matrice: o linie corespunde unui polinom fundamental, iar coloanele corespund punctelor în care se face evaluarea

```
function Z=pfl2b(x,t)
%PFL2b - calculeaza polinoamele fundamentale Lagrange
%apel Z=pfl2(x,t)
%x - nodurile de interpolare
%t - punctele in care se face evaluarea

m=length(x);
n=length(t);
[T,X]=meshgrid(t,x);
TT=T-X;
Z=zeros(m,n);
[U,V]=meshgrid(x,x);
XX=U-V;
for i=1:m
    TX=prod(XX([1:i-1,i+1:m],i));
    Z(i,:)=prod(TT([1:i-1,i+1:m],:))/TX;
end
```

și notăm cu  $V = (v_{ij})$  matricea pătratică de ordin  $m + 1$  cu elementele  $v_{ij} = x_i^j$ . Ecuația (5.3.9) se scrie sub forma

$$Va = b$$

Matricea  $V$  este inversabilă (determinantul ei este Vandermonde); se arată ușor că  $V^{-1} = U^T$  unde  $U = (u_{ij})$  cu  $\ell_i(x) = \sum_{k=0}^m u_{ik}x^k$ ; se obține în acest mod un procedeu puțin costisitor de inversare a matricei Vandermonde și prin urmare și de rezolvare a sistemului (5.3.9).

**Exemplul 5.3.9.** Polinomul de interpolare Lagrange corespunzător unei funcții  $f$  și nodurilor  $x_0$  și  $x_1$  este

$$(L_1 f)(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1),$$

adică dreapta care trece prin punctele  $(x_0, f(x_0))$  și  $(x_1, f(x_1))$ . Analog, polinomul de interpolare Lagrange corespunzător unei funcții  $f$  și nodurilor  $x_0, x_1$  și  $x_2$  este

$$\begin{aligned} (L_2 f)(x) = & \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) + \\ & \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2), \end{aligned}$$

adică parabola care trece prin punctele  $(x_0, f(x_0)), (x_1, f(x_1))$  și  $(x_2, f(x_2))$ . Interpretarea lor geometrică apare în figura 5.9.  $\diamond$

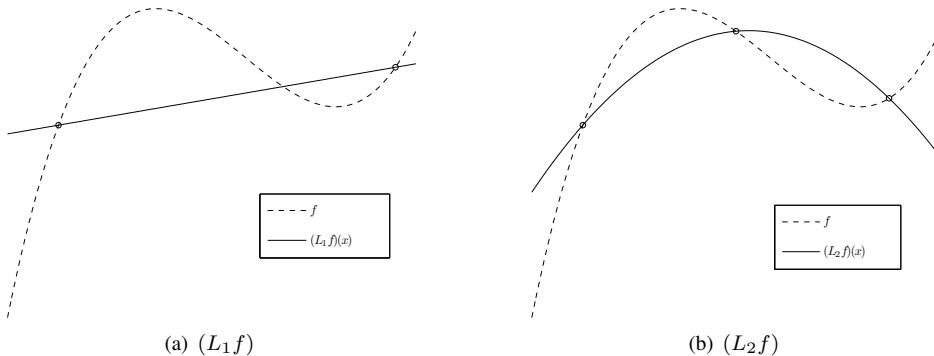


Figura 5.9: Interpretarea geometrică a lui  $L_1 f$  (stânga) și  $L_2 f$

### 5.3.3. Interpolare Hermite

În loc să facem să coincidă  $f$  și polinomul de interpolare în punctele  $x_i$  din  $[a, b]$ , am putea face ca  $f$  și polinomul de interpolare să coincidă împreună cu derivatele lor până la ordinul  $r_i$  în punctele  $x_i$ . Se obține:

**Teorema 5.3.10.** Fiind date  $(m + 1)$  puncte distincte  $x_0, x_1, \dots, x_m$  din  $[a, b]$  și  $(m + 1)$  numere naturale  $r_0, r_1, \dots, r_m$ , punem  $n = m + r_0 + r_1 + \dots + r_m$ . Atunci, fiind dată o funcție  $f$ , definită pe  $[a, b]$  și admitând derivate de ordin  $r_i$  în punctele  $x_i$ , există un singur polinom și numai unul  $H_n f$  de grad  $\leq n$  astfel încât

$$\forall (i, \ell), 0 \leq i \leq m, 0 \leq \ell \leq r_i \quad (H_n f)^{(\ell)}(x_i) = f^{(\ell)}(x_i), \quad (5.3.10)$$

unde  $f^{(\ell)}(x_i)$  este derivata de ordinul  $\ell$  a lui  $f$  în  $x_i$ .

**Definiția 5.3.11.** Polinomul definit în acest mod se numește polinom de interpolare al lui Hermite<sup>9</sup> al funcției  $f$  relativ la punctele  $x_0, x_1, \dots, x_m$  și la întregii  $r_0, r_1, \dots, r_m$ .

*Demonstrație.* Ecuația (5.3.10) conduce la un sistem liniar de  $(n + 1)$  ecuații cu  $(n + 1)$  necunoscute (coeficienții lui  $H_n f$ ), deci este suficient să arătăm că sistemul omogen corespunzător admite doar soluția nulă, adică relațiile

$$H_n f \in \mathbb{P}_n \text{ și } \forall (i, \ell), 0 \leq i \leq k, 0 \leq \ell \leq r_i, (H_n f)^{(\ell)}(x_i) = 0$$

ne asigură că, pentru orice  $i = 0, 1, \dots, m$ ,  $x_i$  este rădăcină de ordinul  $r_i + 1$  a lui  $H_n f$ ; prin urmare  $H_n f$  are forma

$$(H_n f)(x) = q(x) \prod_{i=0}^m (x - x_i)^{r_i+1},$$

unde  $q$  este un polinom. Cum  $\sum_{i=0}^m (\alpha_i + 1) = n + 1$ , acest lucru nu este compatibil cu apartenența lui  $H_n$  la  $\mathbb{P}_n$ , decât dacă  $q \equiv 0$  și deci  $H_n \equiv 0$ .  $\square$

**Observația 5.3.12.** 1) Dându-se numerele reale  $b_{i\ell}$  pentru orice pereche  $(i, \ell)$  astfel încât  $0 \leq i \leq k$  și  $0 \leq \ell \leq r_i$ , am arătat că problema generală de interpolare Hermite

$$\begin{aligned} &\text{să se determine } p_n \in \mathbb{P}_n \text{ a.î. } \forall (i, \ell) \text{ cu } 0 \leq i \leq m \text{ și} \\ &0 \leq \ell \leq r_i, \quad p_n^{(\ell)}(x_i) = b_{i\ell} \end{aligned} \quad (5.3.11)$$

admete o soluție și numai una. În particular, dacă alegem pentru o pereche  $(i, \ell)$  dată  $b_{i\ell} = 1$  și  $b_{jn} = 0$ ,  $\forall (j, m) \neq (i, \ell)$ , se obține un polinom de bază (fundamental) de interpolare Hermite relativ la punctele  $x_0, x_1, \dots, x_m$  și la întregii  $r_0, r_1, \dots, r_m$ . Polinomul de interpolare Hermite definit prin (5.3.10) se obține cu ajutorul polinoamelor de bază (fundamentale) cu formula

$$(H_n f)(x) = \sum_{i=0}^m \sum_{\ell=0}^{r_i} f^{(\ell)}(x) h_{i\ell}(x). \quad (5.3.12)$$

Charles Hermite (1822-1901) matematician francez de frunte, membru al Academiei Franceze, cunoscut pentru lucrările sale în domeniul teoriei numerelor, algebră și analiză. A devenit faimos după ce a dat, în 1873, demonstrația transcendenței numărului  $e$ .



Punând

$$q_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^k \left( \frac{x - x_j}{x_i - x_j} \right)^{r_{j+1}},$$

se verifică că polinoamele de bază  $h_{i\ell}$  sunt definite prin relațiile de recurență

$$h_{ir_i}(x) = \frac{(x - x_i)^{r_i}}{r_i!} q_i(x)$$

și pentru  $\ell = r_i - 1, r_i - 2, \dots, 1, 0$

$$h_{i\ell}(x) = \frac{(x - x_i)^\ell}{\ell!} q_i(x) - \sum_{j=\ell+1}^{r_i} \binom{j}{\ell} q_i^{(j-\ell)}(x_i) h_{ij}(x).$$

- 2) Matricea  $V$  asociată sistemului liniar (5.3.11) se numește matrice Vandermonde generalizată; ea este inversabilă, iar elementele matricei ei inverse sunt coeficienții polinoamelor  $h_{i\ell}$ .
- 3) Interpolarea Lagrange este un caz particular al interpolării Hermite (pentru  $r_i = 0$ ,  $i = 0, 1, \dots, m$ ); polinomul Taylor este un caz particular pentru  $m = 0$  și  $r_0 = n$ .  $\diamond$

Vom prezenta o expresie mai convenabilă a polinoamelor fundamentale Hermite, obținută de Dimitrie D. Stancu în 1957[64]. Ele verifică relațiile

$$\begin{aligned} h_{kj}^{(p)}(x_\nu) &= 0, & \nu \neq k, p = \overline{0, r_\nu} \\ h_{kj}^{(p)}(x_k) &= \delta_{jp}, & p = \overline{0, r_k}, \end{aligned} \tag{5.3.13}$$

pentru  $j = \overline{0, r_k}$  și  $\nu, k = \overline{0, m}$ . Introducând notațiile

$$u(x) = \prod_{k=0}^m (x - x_k)^{r_k+1}$$

și

$$u_k(x) = \frac{u(x)}{(x - x_k)^{r_k+1}},$$

din (5.3.13) rezultă că  $h_{kj}$  are forma

$$h_{kj}(x) = u_k(x)(x - x_k)^j g_{kj}(x), \quad g_{kj} \in \mathbb{P}_{r_k-j}. \tag{5.3.14}$$

Dezvoltând  $g_{kj}$  cu formula lui Taylor, avem

$$g_{kj}(x) = \sum_{\nu=0}^{r_k-j} \frac{(x - x_k)^\nu}{\nu!} g_{kj}^{(\nu)}(x_k); \tag{5.3.15}$$

mai rămân de determinat valorile lui  $g_{kj}^{(\nu)}(x_k)$ ,  $\nu = \overline{0, r_k - j}$ . Scriind (5.3.14) sub forma

$$(x - x_k)^j g_{kj}(x) = h_{kj}(x) \frac{1}{u_k(x)},$$

și aplicând formula lui Lebniz pentru derivata de ordinul  $j + \nu$  a produsului se obține

$$\sum_{s=0}^{j+\nu} \binom{j+\nu}{s} [(x - x_k)^j]^{(j+\nu-s)} g_{kj}^{(s)}(x) = \sum_{s=0}^{j+\nu} \binom{j+\nu}{s} h_{kj}^{(j+\nu-s)}(x) \left[ \frac{1}{u_k(x)} \right]^{(s)}.$$

Lângă  $x = x_k$ , toți termenii din ambii membri se vor anula, cu excepția celor corespunzători lui  $s = \nu$ . Avem deci

$$\binom{j+\nu}{\nu} j! g_{kj}^{(\nu)}(x_k) = \binom{j+\nu}{\nu} \left[ \frac{1}{u_k(x)} \right]_{x=x_k}^{(\nu)}, \quad \nu = \overline{0, r_k - j}.$$

Am obținut

$$g_{kj}^{(\nu)}(x_k) = \frac{1}{j!} \left[ \frac{1}{u_k(x)} \right]_{x=x_k}^{(\nu)},$$

iar din (5.3.15) și (5.3.14) avem în final

$$h_{kj}(x) = \frac{(x - x_k)^j}{j!} u_k(x) \sum_{\nu=0}^{r_k-j} \frac{(x - x_k)^\nu}{\nu!} \left[ \frac{1}{u_k(x)} \right]_{x=x_k}^{(\nu)}.$$

**Propoziția 5.3.13.** *Operatorul  $H_n$  este proiecțor, adică*

- este liniar ( $H_n(\alpha f + \beta g) = \alpha H_n f + \beta H_n g$ );
- este idempotent ( $H_n \circ H_n = H_n$ ).

*Demonstrație.* Liniaritatea rezultă imediat din formula (5.3.12). Datorită unicății polinomului de interpolare Hermite  $H_n(H_n f) - H_n f$  este identic nul, deci  $H_n(H_n f) = H_n f$  și am arătat idempotența.  $\square$

**Exemplul 5.3.14.** Polinomul de interpolare Hermite corespunzător unei funcții  $f$  și nodurilor duble 0 și 1 are expresia

$$(H_3 f)(x) = h_{00}(x)f(0) + h_{10}(x)f(1) + h_{01}(x)f'(0) + h_{11}(x)f'(1),$$

unde

$$\begin{aligned} h_{00}(x) &= (x - 1)^2(2x + 1), \\ h_{01}(x) &= x(x - 1)^2, \\ h_{10}(x) &= x^2(3 - 2x), \\ h_{11}(x) &= x^2(x - 1). \end{aligned}$$

Dacă se adaugă nodul  $x = \frac{1}{2}$ , calitatea aproximării crește (vezi figura 5.10).  $\diamond$

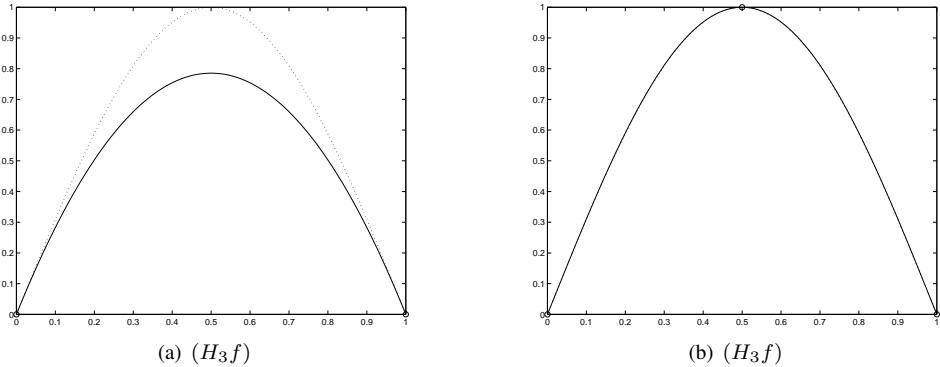


Figura 5.10: Polinoamele de interpolare Hermite  $(H_3f)$  (—) corespunzător funcției  $f : [0, 1] \rightarrow \mathbb{R}$ ,  $f(x) = \sin \pi x$  și nodurilor duble  $x_0 = 0$  și  $x_1 = 1$  (···)(stânga) și  $(H_5f)$  (—) corespunzător funcției  $f : [0, 1] \rightarrow \mathbb{R}$ ,  $f(x) = \sin \pi x$  (···) și nodurilor duble  $x_0 = 0$ ,  $x_1 = \frac{1}{2}$  și  $x_2 = 1$ .

### 5.3.4. Expresia erorii de interpolare

Reamintim că norma unui operator liniar  $P_n$  se poate defini prin

$$\|P_n\| = \max_{f \in C[a,b]} \frac{\|P_n f\|}{\|f\|}, \quad (5.3.16)$$

unde în membrul drept se ia o normă convenabilă pentru funcții. Luând norma  $L^\infty$ , din formula lui Lagrange se obține

$$\begin{aligned} \|(L_m f)(.)\|_\infty &= \max_{a \leq x \leq b} \left| \sum_{i=0}^m f(x_i) \ell_i(x) \right| \\ &\leq \|f\|_\infty \max_{a \leq x \leq b} \sum_{i=0}^m |\ell_i(x)|. \end{aligned} \quad (5.3.17)$$

Fie  $\|\lambda_m\|_\infty = \lambda_m(x_\infty)$ . Egalitatea are loc pentru o funcție  $\varphi \in C[a, b]$ , liniară pe porțiuni și care verifică  $\varphi(x_i) = \text{sgn} \ell_i(x_\infty)$ ,  $i = \overline{0, m}$ . Deci,

$$\|P_n\|_\infty = \Lambda_m, \quad (5.3.18)$$

unde

$$\Lambda_m = \|\lambda_m\|_\infty, \quad \lambda_m(x) = \sum_{i=0}^m |\ell_i(x)|. \quad (5.3.19)$$

Funcția  $\lambda_m(x)$  și maximul său  $\Lambda_m$  se numesc *funcția lui Lebesgue*<sup>10</sup> și respectiv *constanta lui Lebesgue* pentru interpolarea Lagrange. Ele furnizează o primă estimare a erorii de interpolare: fie  $\mathcal{E}_m(f)$  eroarea în cea mai bună aproximare a lui  $f$  prin polinoame de grad  $\leq m$ ,

$$\mathcal{E}_m(f) = \min_{p \in \mathbb{P}_m} \|f - p\|_\infty = \|f - \hat{p}_n\|_\infty, \quad (5.3.20)$$

unde  $\hat{p}_n$  este polinomul de grad  $m$  de la cea mai bună aproximare a lui  $f$ . Utilizând faptul că operatorul  $L_m$  este proiectoare și formulele (5.3.17) și (5.3.19), se găsește

$$\begin{aligned} \|f - L_m f\| &= \|f - \hat{p}_m - L_m(f - \hat{p}_m)\|_\infty \\ &\leq \|f - \hat{p}_m\|_\infty + \Lambda_m \|f - \hat{p}_m\|_\infty; \end{aligned}$$

adică,

$$\|f - L_m f\|_\infty \leq (1 + \Lambda_m) \mathcal{E}_m(f). \quad (5.3.21)$$

Astfel, cu cât  $f$  poate fi aproximată mai bine prin polinoame de grad  $\leq m$ , cu atât este mai mică eroarea de interpolare. Din păcate,  $\Lambda_m$  nu este uniform mărginită: indiferent de cum se aleg nodurile  $x_i = x_i^{(m)}$ ,  $i = \overline{0, m}$ , se poate arăta că  $\Lambda_m > O(\log m)$  când  $m \rightarrow \infty$ . Totuși, nu este posibil să tragem, pe baza teoremei de aproximare a lui Weierstrass (adică din  $\mathcal{E}_m \rightarrow 0$ ,  $m \rightarrow \infty$ ) concluzia că interpolarea Lagrange converge uniform pentru orice funcție  $f$ , nici chiar pentru noduri judicioase alese; se știe că de fapt convergența nu are loc.

Dacă dorim să utilizăm polinomul de interpolare Lagrange sau Hermite pentru a approxima funcția  $f$  într-un punct  $x \in [a, b]$ , distinct de nodurile de interpolare  $(x_0, \dots, x_m)$ , trebuie să estimăm eroarea comisă  $(R_n f)(x) = f(x) - (H_n f)(x)$ . Dacă nu posedăm nici o informație referitoare la  $f$  în afara punctelor  $x_i$ , este clar că nu putem spune nimic despre  $(R_n f)(x)$ ; într-adevăr este posibil să schimbăm  $f$  în afara punctelor  $x_i$  fără a modifica  $(H_n f)(x)$ . Trebuie deci să facem ipoteze suplimentare, care vor fi ipoteze de regularitate asupra lui  $f$ . Să notăm cu  $C^m[a, b]$  spațiul funcțiilor reale de  $m$  ori continuu diferențiabile pe  $[a, b]$ . Avem următoarea teoremă referitoare la estimarea erorii în interpolarea Hermite.

**Teorema 5.3.15.** *Presupunem că  $f \in C^n[\alpha, \beta]$  și există  $f^{(n+1)}$  pe  $(\alpha, \beta)$ , unde  $\alpha = \min\{x, x_0, \dots, x_m\}$  și  $\beta = \max\{x, x_0, \dots, x_m\}$ ; atunci, pentru orice  $x \in [\alpha, \beta]$ , există un  $\xi_x \in (\alpha, \beta)$  astfel încât*

$$(R_n f)(x) = \frac{1}{(n+1)!} u_n(x) f^{(n+1)}(\xi_x), \quad (5.3.22)$$

---

<sup>10</sup>



Henry Lebesgue (1875-1941), matematician francez, cunoscut pentru lucrările sale fundamentale în domeniul teoriei funcțiilor reale, și în special pentru introducerea măsurii și integralei care îi poartă numele.

unde

$$u_n(x) = \prod_{i=0}^m (x - x_i)^{r_{i+1}}.$$

*Demonstrație.* Dacă  $x = x_i$ ,  $(R_n f)(x) = 0$  și (5.3.22) se verifică trivial. Presupunem că  $x$  este distinct de  $x_i$  și considerăm, pentru  $x$  fixat, funcția auxiliară

$$F(z) = \begin{vmatrix} u_n(z) & (R_n f)(z) \\ u_n(x) & (R_n f)(x) \end{vmatrix}.$$

Se observă că  $F \in C^n[\alpha, \beta]$ ,  $\exists F^{(n+1)}$  pe  $(\alpha, \beta)$ ,  $F(x) = 0$  și  $F^{(j)}(x_k) = 0$  pentru  $k = \overline{0, m}$ ,  $j = \overline{0, r_k}$ . Deci,  $F$  are  $(n+2)$  zerouri, luând în considerare și ordinea de multiplicitate. Aplicând succesiv teorema lui Rolle generalizată, rezultă că există cel puțin un  $\xi \in (\alpha, \beta)$  astfel încât  $F^{(n+1)}(\xi) = 0$ , adică

$$F^{(m+1)}(\xi) = \begin{vmatrix} (n+1)! & f^{(n+1)}(\xi) \\ u_n(x) & (R_n f)(x) \end{vmatrix} = 0, \quad (5.3.23)$$

unde s-a ținut cont că  $(R_n f)^{(n+1)} = f^{(n+1)} - (H_n f)^{(n+1)} = f^{(n+1)}$ . Exprimând  $(R_n f)(x)$  din (5.3.23) se obține (5.3.22).  $\square$

**Corolarul 5.3.16.** Punem  $M_{n+1} = \max_{x \in [a, b]} |f^{(n+1)}(x)|$ ; o margine superioară a erorii de interpolare  $(R_n f)(x) = f(x) - (H_n f)(x)$  este dată prin

$$|(R_n f)(x)| \leq \frac{M_{n+1}}{(n+1)!} |u_n(x)|.$$

Deoarece  $H_n$  este proiectoare, rezultă că  $R_n$  este de asemenea proiectoare; în plus  $\text{Ker } R_n = \mathbb{P}_n$ , deoarece  $R_n f = f - H_n f = f - f = 0$ ,  $\forall f \in \mathbb{P}_n$ . Deci, putem aplica lui  $R_n$  teorema lui Peano.

**Teorema 5.3.17.** Dacă  $f \in C^{n+1}[a, b]$ , atunci

$$(R_n f)(x) = \int_a^b K_n(x; t) f^{(n+1)}(t) dt, \quad (5.3.24)$$

unde

$$K_n(x; t) = \frac{1}{n!} \left\{ (x-t)_+^n - \sum_{k=0}^m \sum_{j=0}^{r_k} h_{kj}(x) [(x_k - t)_+^n]^{(j)} \right\}. \quad (5.3.25)$$

*Demonstrație.* Aplicând teorema lui Peano, avem

$$(R_n f)(x) = \int_a^b K_n(x; t) f^{(n+1)}(t) dt$$

și ținând cont că

$$K_n(x; t) = R_n \left[ \frac{(x-t)_+^n}{n!} \right] = \frac{(x-t)_+^n}{n!} - H_n \left[ \frac{(x-t)_+^n}{n!} \right],$$

teorema rezultă imediat.  $\square$

Deoarece interpolarea Lagrange este un caz particular al interpolării Hermite pentru  $r_i = 0$ ,  $i = 0, 1, \dots, m$  din teorema 5.3.15 se obține:

**Corolarul 5.3.18.** Presupunem că  $f \in C^m[\alpha, \beta]$  și există  $f^{(m+1)}$  pe  $(\alpha, \beta)$ , unde  $\alpha = \min\{x, x_0, \dots, x_m\}$  și  $\beta = \max\{x, x_0, \dots, x_m\}$ ; atunci, pentru orice  $x \in [\alpha, \beta]$ , există un  $\xi_x \in (\alpha, \beta)$  astfel încât

$$(R_m f)(x) = \frac{1}{(n+1)!} u_m(x) f^{(m+1)}(\xi_x), \quad (5.3.26)$$

unde

$$u_m(x) = \prod_{i=0}^m (x - x_i).$$

De asemenea, din teorema 5.3.17 avem:

**Corolarul 5.3.19.** Dacă  $f \in C^{m+1}[a, b]$ , atunci

$$(R_m f)(x) = \int_a^b K_m(x; t) f^{(m+1)}(t) dt \quad (5.3.27)$$

unde

$$K_m(x; t) = \frac{1}{m!} \left[ (x-t)_+^m - \sum_{k=0}^m \ell_k(x) (x_k - t)_+^m \right]. \quad (5.3.28)$$

**Exemplul 5.3.20.** Pentru polinoamele de interpolare din exemplul 5.3.9 resturile corespunzătoare sunt

$$(R_1 f)(x) = \frac{(x-x_0)(x-x_1)}{2} f''(\xi)$$

și respectiv

$$(R_2 f)(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{6} f'''(\xi). \quad \diamond$$

**Exemplul 5.3.21.** Restul din formula de interpolare Hermite cu nodurile duble 0 și 1 pentru  $f \in C^4[\alpha, \beta]$  este

$$(R_3 f)(x) = \frac{x^2(x-1)^2}{6!} f^{(4)}(\xi). \quad \diamond$$

**Exemplul 5.3.22.** Luăm  $f(x) = e^x$ . Avem pentru  $x \in [a, b]$ ,  $M_{n+1} = e^b$  și oricum am alege punctele  $x_i$ ,  $|u_n(x)| \leq (b-a)^{n+1}$ , de unde

$$\max_{x \in [a, b]} |(R_n f)(x)| \leq \frac{(b-a)^{n+1}}{(n+1)!} e^b.$$

Se deduce că

$$\lim_{n \rightarrow \infty} \left\{ \max_{x \in [a, b]} |(R_n f)(x)| \right\} = \lim_{n \rightarrow \infty} \|(R_n f)(x)\| = 0,$$

adică  $H_n f$  converge uniform către  $f$  pe  $[a, b]$  când  $n$  tinde la  $\infty$ . De fapt se poate demonstra un rezultat analog pentru orice funcție dezvoltabilă în serie întreagă în jurul punctului  $x = \frac{a+b}{2}$  cu raza de convergență  $r > \frac{3}{2}(b-a)$ .  $\diamond$

### 5.3.5. Convergența interpolării Lagrange

Să definim ce înțelegem prin convergență. Presupunem că se dă un tablou triunghiular de noduri de interpolare  $x_i = x_i^{(m)}$ , având exact  $m + 1$  noduri distincte pentru orice  $m = 0, 1, 2, \dots$

$$\begin{array}{ccccccc} x_0^{(0)} & & & & & & \\ x_0^{(1)} & x_1^{(1)} & & & & & \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & & & & \\ \vdots & \vdots & \vdots & \ddots & & & \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \dots & x_m^{(m)} & & \\ \vdots & \vdots & \vdots & & \vdots & & \end{array} \quad (5.3.29)$$

Presupunem că toate nodurile sunt conținute într-un interval finit  $[a, b]$ . Atunci pentru orice  $m$  definim

$$P_m(x) = L_m(f; x_0^{(m)}, x_1^{(m)}, \dots, x_m^{(m)}; x), \quad x \in [a, b]. \quad (5.3.30)$$

Spunem că interpolarea Lagrange bazată pe tabelul de noduri (5.3.29) converge dacă

$$p_m(x) \rightrightarrows f(x), \text{ când } n \rightarrow \infty \text{ pe } [a, b]. \quad (5.3.31)$$

Convergența depinde evident de comportarea derivatei de ordinul  $k$   $f^{(k)}$  a lui  $f$  când  $k \rightarrow \infty$ . Presupunem că  $f \in C^\infty[a, b]$  și că

$$|f^{(k)}(x)| \leq M_k \text{ pentru } a \leq x \leq b, k = 0, 1, 2, \dots \quad (5.3.32)$$

Deoarece  $|x_i - x_i^{(m)}| \leq b - a$  când  $x \in [a, b]$  și  $x_i^{(n)} \in [a, b]$  avem

$$\left| (x - x_0^{(m)}) \dots (x - x_m^{(m)}) \right| < (b - a)^{m+1}, \quad (5.3.33)$$

deci

$$|f(x) - (L_m f)(x)| \leq (b - a)^{m+1} \frac{M_{m+1}}{(m+1)!}, \quad x \in [a, b]. \quad (5.3.34)$$

Deci avem convergență dacă

$$\lim_{k \rightarrow \infty} \frac{(b - a)^k}{k!} M_k = 0. \quad (5.3.35)$$

Să arătăm că (5.3.35) este adevarată dacă  $f$  este analitică într-o vecinătate suficient de mare din  $\mathbb{C}$  ce conține intervalul  $[a, b]$ . Mai concret fie  $C_r$  discul circular (închis) cu centrul în mijlocul intervalului  $[a, b]$  și de rază  $r$  și presupunem că  $r > \frac{1}{2}(b - a)$ , astfel că  $[a, b] \subset C_r$ . Presupunem că  $f$  este analitică în  $C_r$ . Atunci putem estima derivata în (5.3.32) cu formula lui Cauchy

$$f^{(k)}(x) = \frac{k!}{2\pi i} \int_{C_r} \frac{f(z)}{(z - x)^{k+1}} dz, \quad x \in [a, b]. \quad (5.3.36)$$

Observând că  $|z - x| \geq r - \frac{1}{2}(b - a)$  (vezi figura 5.11) obținem

$$|f^{(k)}(x)| \leq \frac{k!}{2\pi} \frac{\max_{z \in \partial C_r} |f(z)|}{[r - \frac{1}{2}(b - a)]^{k+1}} \cdot 2\pi r;$$

putem lua pentru  $M_k$  în (5.3.32)

$$M_k = \frac{r}{r - \frac{1}{2}(b - a)} \max_{z \in \partial C_r} |f(z)| \frac{k!}{[r - \frac{1}{2}(b - a)]^k} \quad (5.3.37)$$

și (5.3.35) are loc dacă

$$\left( \frac{b - a}{r - \frac{1}{2}(b - a)} \right)^k \rightarrow 0 \text{ când } k \rightarrow \infty,$$

adică, dacă  $b - a < r - \frac{1}{2}(b - a)$  sau echivalent

$$r > \frac{3}{2}(b - a). \quad (5.3.38)$$

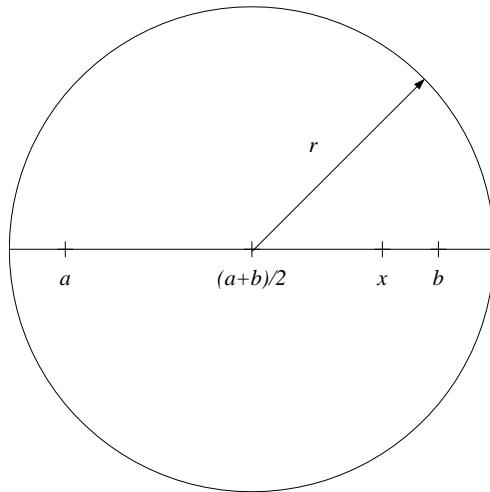


Figura 5.11: Discul circular  $C_r$

Am arătat că interpolarea Lagrange converge (uniform pe  $[a, b]$ ) pentru o mulțime arbitrară de noduri (5.3.29) (toate conținute în  $[a, b]$ ) dacă  $f$  este analitică în discul circular  $C_r$  centrat în  $(a + b)/2$  și având raza suficient de mare astfel ca (5.3.29) să aibă loc.

Deoarece acest rezultat utilizează o estimare grosieră (în particular (5.3.33)), domeniul cerut de analiticitate pentru  $f$  nu este prea îngust. Utilizând metode mai rafinate, se poate arăta următorul lucru. Fie  $d\mu(t)$  distribuția limită a nodurilor de interpolare, adică

$$\int_a^x d\mu(t), \quad a < x \leq b,$$

raportul dintre numărul de noduri  $x_i^{(m)}$  din  $[a, x]$  și numărul total de noduri, asimptotic când  $n \rightarrow \infty$ . (Când nodurile sunt uniform distribuite pe intervalul  $[a, b]$ , atunci  $d\mu(t) = \frac{dt}{b-a}$ ). O curbă cu potențial logaritmic constant este locul geometric al punctelor  $z \in \mathbb{C}$  cu proprietatea

$$u(z) = \int_a^b \ln \frac{1}{|z-t|} d\mu(t) = \gamma,$$

unde  $\gamma$  este o constantă. Pentru  $\gamma$  negativ, foarte mare în modul, aceste curbe arată ca niște cercuri cu raza foarte mare și centrul în  $(a+b)/2$ . Pe măsură ce  $\gamma$  crește, curbele se „comprimă” spre intervalul  $[a, b]$ . Fie  $\Gamma = \sup \gamma$ , unde supremumul se ia pentru toate curbele  $u(z) = \gamma$  ce conțin pe  $[a, b]$  în interior. Domeniul important (ce înlocuiește  $C_r$ ) este domeniul

$$C_\Gamma = \{z \in \mathbb{C} : u(z) \geq \Gamma\},$$

în sensul că dacă  $f$  este analitică în orice domeniu  $C$  ce conține  $C_\Gamma$  în interiorul său (nu contează cât de strâns  $C$  acoperă pe  $C_\Gamma$ ), atunci

$$|f(z) - (L_m f)(z)| \rightarrow 0, \text{ când } n \rightarrow \infty, \quad (5.3.39)$$

uniform pentru  $z \in C_\Gamma$ .

**Exemplul 5.3.23.** Noduri echidistante:  $d\mu(t) = dt/(b-a)$ ,  $a \leq t \leq b$ . În acest caz  $C_\Gamma$  este un domeniu în formă de lentilă, aşa cum se arată în figura 5.12. Astfel, avem convergență uniformă în  $C_\Gamma$  (nu doar pe  $[a, b]$  ca mai sus) cu condiția ca  $f$  să fie analitică într-o regiune puțin mai largă decât  $C_\Gamma$ .

**Exemplul 5.3.24.** Distribuția arcsin pe  $[-1, 1]$ :

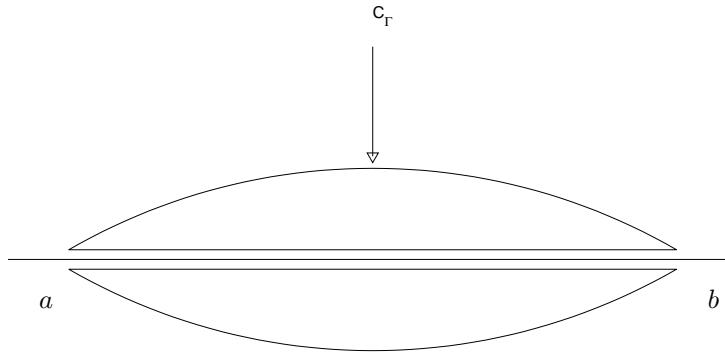
$$d\mu(t) = \frac{1}{\pi} \frac{dt}{\sqrt{1-t^2}}.$$

Nodurile sunt în acest caz rădăcinile polinomului Cebîșev de speță I. Ele sunt mai dens distribuite în apropierea capetelor intervalului  $[-1, 1]$ . În acest caz  $C_\Gamma = [-1, 1]$ , aşa că interpolarea Lagrange converge uniform pe  $[-1, 1]$ , dacă  $f$  este analitică într-o regiune al cărei interior conține intervalul  $[-1, 1]$ .  $\diamond$

Care este semnificația polinoamelor Cebîșev pentru interpolare?

Reamintim că eroarea de interpolare (pe  $[-1, 1]$ , pentru o funcție de clasă  $C^{m+1}[-1, 1]$ ) este dată de

$$f(x) - (L_m f)(x) = \frac{f^{(m+1)}(\xi(x))}{(m+1)!} \prod_{i=0}^m (x - x_i). \quad (5.3.40)$$

Figura 5.12: Domeniul  $C_\Gamma$  pentru noduri uniform distribuite

Primul factor este independent de alegerea nodurilor  $x_i$ . Punctul intermedian  $\xi(x)$  depinde de  $x_i$ , dar de obicei majorăm  $|f^{(m+1)}|$  prin  $\|f^{(m+1)}\|_\infty$ , ceea ce înlătură această dependență. Pe de altă parte, produsul din al doilea factor, inclusiv norma sa

$$\left\| \prod_{i=0}^m (. - x_i) \right\|_\infty, \quad (5.3.41)$$

depinde puternic de  $x_i$ . Are sens, deci, să încercăm să minimizăm (5.3.41) după toți  $x_i \in [-1, 1]$ . Deoarece produsul din (5.3.41) este un polinom monic de grad  $m + 1$ , din teorema 5.1.2 rezultă că nodurile optimale  $x_i = \hat{x}_i^{(m)}$  din (5.3.40) sunt rădăcinile lui  $T_{m+1}$ , adică

$$\hat{x}_i^{(m)} = \cos \frac{2i+1}{2m+2} \pi, \quad i = \overline{0, m}. \quad (5.3.42)$$

Pentru aceste noduri, avem conform lui (5.1.44)

$$\|f(.) - (L_m f)(.)\|_\infty \leq \frac{\|f^{(m+1)}\|_\infty}{(m+1)!} \cdot \frac{1}{2^m}. \quad (5.3.43)$$

Se poate compara acest factor cu marginea mai brută dată în (5.3.34) care în acest caz pe intervalul  $[-1, 1]$  este  $2^{m+1}/(m+1)!$ .

Deoarece, conform (5.3.40), curba de eroare  $y = f - L_m f$  pentru punctele Cebîșev (5.3.42) este în esență echilibrată (modulo variația factorului  $f^{(m+1)}$ ) și astfel liberă de oscilațiile violente pe care le-am văzut pentru puncte echidistante, ne vom aștepta la proprietăți mai favorabile pentru tablouri triunghiulare (5.3.29) formate din noduri Cebîșev. Se poate arăta că dacă  $f \in C^1[-1, 1]$ , atunci

$$(L_m f) \left( x; \hat{x}_0^{(m)}, \hat{x}_1^{(m)}, \dots, \hat{x}_m^{(m)} \right) \Rightarrow f(x), \quad n \rightarrow \infty, \quad (5.3.44)$$

pe  $[-1, 1]$ . Astfel, nu avem nevoie de analiticitatea lui  $f$  pentru ca (5.3.44) să aibă loc.

**Exemplul 5.3.25 (Exemplul lui Runge).** Considerăm funcția

$$f(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5],$$

și nodurile

$$x_k^{(m)} = -5 + 10 \frac{k}{m}, \quad k = \overline{0, m}. \quad (5.3.45)$$

Nodurile sunt echidistante pe  $[-5, 5]$ , deci asimptotic uniform distribuite. Observăm că  $f$  are doi poli în  $z = \pm i$ . Acești poli sunt așezați în interiorul regiunii  $C_\Gamma$  din figura 5.12 pentru intervalul  $[-5, 5]$ , deci  $f$  nu este analitică în  $C_\Gamma$ . Din acest motiv nu ne așteptăm să avem convergență pe întreg intervalul  $[-5, 5]$ . Se poate demonstra că

$$\lim_{m \rightarrow \infty} |f(x) - p_m(f; x)| = \begin{cases} 0 & \text{dacă } |x| < 3.633\dots \\ \infty & \text{dacă } |x| > 3.633\dots \end{cases} \quad (5.3.46)$$

Având în minte figura 5.12 acest rezultat nu este surprinzător. Graficul pentru  $m = 10, 13, 16$  apare în figura 5.13. El a fost generat cu sursa MATLAB 5.13, cu comanda

>> runge3([10, 13, 17], [-5, 5, -2, 2])  
folosind facilitățile de adnotare ale editorului de figuri din MATLAB. ◇

---

### Sursa MATLAB 5.13 Contraexemplul lui Runge

---

```
function runge3(n,w)
%N- vector al gradelor, W- fereastra
clf
xg=-5:0.1:5; yg=1./(1+xg.^2);
plot(xg,yg,'k-','Linewidth',2);
hold on
nl=length(n);
ta=5*[-1:0.001:-0.36,-0.35:0.01:0.35, 0.36:0.001:1]';
ya=zeros(length(ta),nl);
leg=cell(1,nl+1); leg{1}='f';
for l=1:nl
    xn=5*[-1:2/n(l):1]; yn=1./(1+xn.^2);
    ya(:,l)=lagr(xn,yn,ta);
    leg{l+1}=strcat('L_{',int2str(n(l)),'}');
end
plot(ta,ya); axis(w)
legend(leg,-1)
```

---

**Exemplul 5.3.26 (Exemplul lui Bernstein).** Luăm

$$f(x) = |x|, \quad x \in [-1, 1]$$

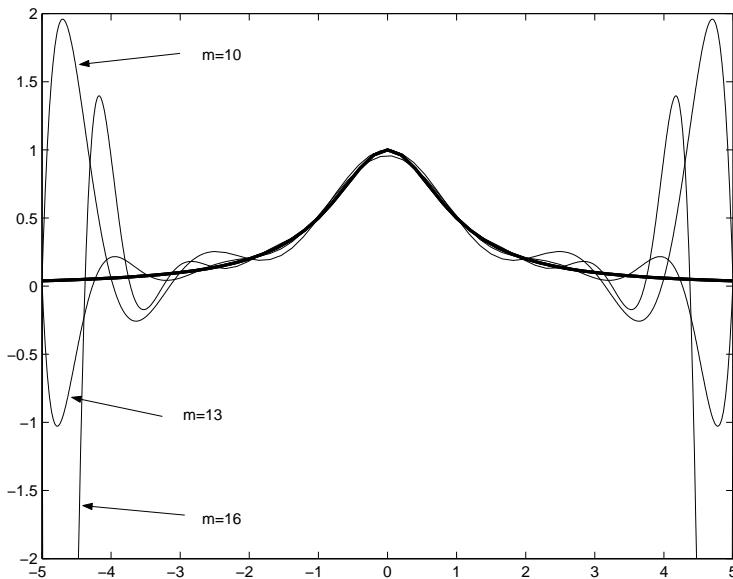


Figura 5.13: O ilustrare grafică a contraexemplului lui Runge

$$x_k^{(m)} = -1 + \frac{2k}{m}, \quad k = 0, 1, 2, \dots, m \quad (5.3.47)$$

Problema analiticității nu se pune, deoarece  $f$  nu este derivabilă în  $x = 0$ . Se obține că

$$\lim_{m \rightarrow \infty} |f(x) - L_m(f; x)| = \infty \quad \forall x \in [-1, 1]$$

exceptând punctele  $x = -1, x = 0$  și  $x = 1$ . Vezi figura 5.14(a), pentru  $m = 20$ . Convergența în  $x = \pm 1$  este trivială deoarece acestea sunt noduri de interpolare și deci eroarea în aceste puncte este 0. Același lucru este adevărat pentru  $x = 0$ , când  $n$  este impar, dar nu și când  $n$  este par. Eșecul convergenței pentru aceste noduri se explică doar parțial prin insuficiența regularității a lui  $f$ . Un alt motiv este distribuția uniformă a nodurilor. Există exemple mai bune de distribuții ale nodurilor, cum ar fi distribuția *arcsin* din exemplul 5.3.24. În figura 5.14(b) se dă graficul pentru  $m = 17$ .  $\diamond$

Problema convergenței a fost rezolvată în cazul general de Faber și Bernstein între 1914 și 1916. Faber a demonstrat că pentru orice tabel triunghiular de noduri din  $[a, b]$ , de tipul 5.3.29, există o funcție  $f \in C[a, b]$  astfel încât sirul polinoamelor de interpolare Lagrange  $L_m f$  pentru nodurile  $x_i^{(m)}$  (pe linii) să nu convergă uniform către  $f$  pe  $[a, b]$ .

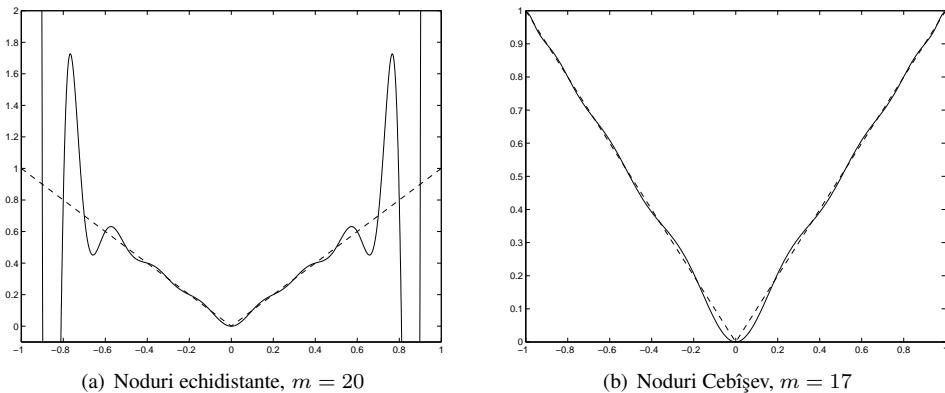


Figura 5.14: Comportarea interpolării Lagrange pentru  $f : [-1, 1] \rightarrow \mathbb{R}$ ,  $f(x) = |x|$ .

Bernstein<sup>11</sup> a demonstrat că pentru orice tablou triunghiular de noduri ca mai sus există o funcție  $f \in C[a, b]$  astfel încât sirul corespunzător  $(L_m f)$  să fie divergent.

Situată se poate remedia în două moduri:

- abordarea locală – intervalul  $[a, b]$  se ia foarte mic – utilizată la rezolvarea numerică a ecuațiilor diferențiale;
- interpolare spline – interpolantul este polinomial pe porțiuni.

## 5.4. Calculul eficient al polinoamelor de interpolare

### 5.4.1. Metode de tip Aitken

În multe situații gradul necesar pentru a atinge precizia dorită în interpolarea polinomială este necunoscut. El se poate determina din expresia restului, dar pentru aceasta este necesar să cunoaștem  $\|f^{(m+1)}\|_\infty$ . Vom nota cu  $P_{m_1, m_2, \dots, m_k}$  polinomul de interpolare Lagrange având nodurile  $x_{m_1}, \dots, x_{m_k}$ .

Sergi Natanovici Bernstein (1880-1968) a adus un aport major în domeniul aproximării polinomiale, continuând tradiția lui Cebîșev.

<sup>11</sup>A dat o demonstrație constructivă a teoremei de aproximare a lui Weierstrass cu ajutorul polinoamelor care îi poartă numele. A avut contribuții importante și în domeniul ecuațiilor diferențiale și al teoriei probabilităților.



**Propoziția 5.4.1.** Dacă  $f$  este definită în  $x_0, \dots, x_k$ ,  $x_j \neq x_i$ ,  $0 \leq i, j \leq k$ , atunci

$$\begin{aligned} P_{0,1,\dots,k} &= \frac{(x - x_j)P_{0,1,\dots,j-1,j+1,\dots,k}(x) - (x - x_i)P_{0,1,\dots,i-1,i+1,\dots,k}(x)}{x_i - x_j} = \\ &= \frac{1}{x_i - x_j} \begin{vmatrix} x - x_j & P_{0,1,\dots,i-1,i+1,\dots,k}(x) \\ x - x_i & P_{0,1,\dots,j-1,j+1,\dots,k}(x) \end{vmatrix} \end{aligned} \quad (5.4.1)$$

*Demonstrație.* Fie  $Q = P_{0,1,\dots,i-1,i+1,\dots,k}$ ,  $\hat{Q} = P_{0,1,\dots,j-1,j+1,k}$  și

$$P(x) = \frac{(x - x_j)\hat{Q}(x) - (x - x_i)Q(x)}{x_i - x_j}.$$

Se observă că, deoarece  $Q(x_r) = \hat{Q}(x_r) = f(x_r)$ , pentru  $r \neq i$  și  $r \neq j$ ,

$$P(x_r) = \frac{(x_r - x_j)\hat{Q}(x_r) - (x_r - x_i)Q(x_r)}{x_i - x_j} = \frac{x_i - x_j}{x_i - x_j}f(x_r) = f(x_r).$$

Dar

$$P(x_i) = \frac{(x_i - x_j)\hat{Q}(x_i) - (x_i - x_j)Q(x_i)}{x_i - x_j} = f(x_i)$$

și

$$P(x_j) = \frac{(x_j - x_i)\hat{Q}(x_j) - (x_j - x_i)Q(x_j)}{x_i - x_j} = f(x_j),$$

deci  $P = P_{0,1,\dots,k}$ .  $\square$

În acest mod am stabilit o relație de recurență între un polinom de interpolare Lagrange de gradul  $k$  și două polinoame de interpolare Lagrange de gradul  $k - 1$ . Calculele pot fi așezate în formă tabelară

$x_0$	$P_0$				
$x_1$	$P_1$	$P_{0,1}$			
$x_2$	$P_2$	$P_{1,2}$	$P_{0,1,2}$		
$x_3$	$P_3$	$P_{2,3}$	$P_{1,2,3}$	$P_{0,1,2,3}$	
$x_4$	$P_4$	$P_{3,4}$	$P_{2,3,4}$	$P_{1,2,3,4}$	$P_{0,1,2,3,4}$

Să presupunem că în acest moment  $P_{0,1,2,3,4}$  nu ne asigură precizia dorită. Se poate selecta un nou nod și adăuga o nouă linie tabelei

$$x_5 \quad P_5 \quad P_{4,5} \quad P_{3,4,5} \quad P_{2,3,4,5} \quad P_{1,2,3,4,5} \quad P_{0,1,2,3,4,5}$$

iar elementele vecine de pe linie, coloană sau diagonală se pot compara pentru a vedea dacă s-a obținut precizia dorită.

Metoda de mai sus se numește *metoda lui Neville*.

Notățile pot fi simplificate

$$Q_{i,j} := P_{i-j, i-j+1, \dots, i-1, i},$$

$$Q_{i,j-1} = P_{i-j+1, \dots, i-1, i},$$

$$Q_{i-1,j-1} := P_{i-j, i-j+1, \dots, i-1}.$$

Din (5.4.1) rezultă

$$Q_{i,j} = \frac{(x - x_{i-j})Q_{i,j-1} - (x - x_i)Q_{i-1,j-1}}{x_i - x_{i-j}},$$

pentru  $j = 1, 2, 3, \dots, i = j + 1, j + 2, \dots$

În plus,  $Q_{i,0} = f(x_i)$ . Obținem tabelul

$x_0$	$Q_{0,0}$				
$x_1$	$Q_{1,0}$	$Q_{1,1}$			
$x_2$	$Q_{2,0}$	$Q_{2,1}$	$Q_{2,2}$		
$x_3$	$Q_{3,0}$	$Q_{3,1}$	$Q_{3,2}$	$Q_{3,3}$	

Dacă procedeul de interpolare converge, atunci sirul  $Q_{i,i}$  converge și el și s-ar putea lua drept criteriu de oprire

$$|Q_{i,i} - Q_{i-1,i-1}| < \varepsilon.$$

Pentru a rapidiza algoritmul nodurilor se vor ordona crescător după valorile  $|x_i - x|$ .

*Metoda lui Aitken* este similară cu metoda lui Neville. Ea construiește tabelul

$x_0$	$P_0$				
$x_1$	$P_1$	$P_{0,1}$			
$x_2$	$P_2$	$P_{0,2}$	$P_{0,1,2}$		
$x_3$	$P_3$	$P_{0,3}$	$P_{0,1,3}$	$P_{0,1,2,3}$	
$x_4$	$P_4$	$P_{0,4}$	$P_{0,1,4}$	$P_{0,1,2,4}$	$P_{0,1,2,3,4}$

Pentru a calcula o nouă valoare se utilizează valoarea din vârful coloanei precedente și valoarea din aceeași linie, coloana precedentă.

## 5.4.2. Metoda diferențelor divizate

Vom nota cu  $L_k f$  polinomul de interpolare Lagrange cu nodurile  $x_0, x_1, \dots, x_k$  pentru  $k = 0, 1, \dots, n$ . Vom construi  $L_m$  prin recurență. Avem

$$(L_0 f)(x) = f(x_0).$$

Pentru  $k \geq 1$  polinomul  $L_k - L_{k-1}$  este de grad  $k$ , se anulează în punctele  $x_0, x_1, \dots, x_{k-1}$  și deci este de forma:

$$(L_k f)(x) - (L_{k-1} f)(x) = f[x_0, x_1, \dots, x_k](x - x_0)(x - x_1) \dots (x - x_{k-1}), \quad (5.4.2)$$

unde  $f[x_0, x_1, \dots, x_k]$  desemnează coeficientul lui  $x^k$  din  $(L_k f)(x)$ . Se deduce expresia polinomului de interpolare  $L_m f$  cu nodurile  $x_0, x_1, \dots, x_n$

$$(L_m f)(x) = f(x_0) + \sum_{k=1}^m f[x_0, x_1, \dots, x_k] (x - x_0)(x - x_1) \dots (x - x_{k-1}), \quad (5.4.3)$$

numită forma Newton<sup>12</sup> a polinomului de interpolare Lagrange.

Formula (5.4.3) reduce calculul prin recurență al lui  $L_m f$  la cel al coeficienților  $f[x_0, x_1, \dots, x_k]$ ,  $k = \overline{0, m}$ .

Are loc

#### **Lema 5.4.2.**

$$\forall k \geq 1 \quad f[x_0, x_1, \dots, x_k] = \frac{f[x_1, x_2, \dots, x_k] - f[x_0, x_1, \dots, x_{k-1}]}{x_k - x_0} \quad (5.4.4)$$

și

$$f[x_i] = f(x_i), \quad i = 0, 1, \dots, k.$$

*Demonstrație.* Notăm, pentru  $k \geq 1$  cu  $L_{k-1}^* f$  polinomul de interpolare pentru  $f$  de grad  $k-1$  și cu nodurile  $x_1, x_2, \dots, x_k$ ; coeficientul lui  $x^{k-1}$  este  $f[x_1, x_2, \dots, x_k]$ . Polinomul  $q_k$  de grad  $k$  definit prin

$$q_k(x) = \frac{(x - x_0)(L_{k-1}^* f)(x) - (x - x_k)(L_{k-1} f)(x)}{x_k - x_0}$$

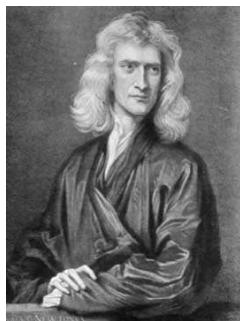
coincide cu  $f$  în punctele  $x_0, x_1, \dots, x_k$  și deci  $q_k(x) \equiv (L_k f)(x)$ . Formula (5.4.4) se obține identificând coeficientul lui  $x^k$  din cei doi membri.  $\square$

**Definiția 5.4.3.** Cantitatea  $f[x_0, x_1, \dots, x_k]$  se numește diferență divizată de ordinul  $k$  a lui  $f$  în punctele  $x_0, x_1, \dots, x_k$ .

Altă notație utilizată este  $[x_0, \dots, x_k; f]$ .

Din definiție rezultă că  $f[x_0, x_1, \dots, x_k]$  este independentă de ordinea punctelor  $x_i$  și ea poate fi calculată în funcție de  $f(x_0), \dots, f(x_m)$ . Într-adevăr PIL de grad  $\leq m$  relativ la punctele  $x_0, \dots, x_m$  se scrie

$$(L_m f)(x) = \sum_{i=0}^m \ell_i f(x_i)$$



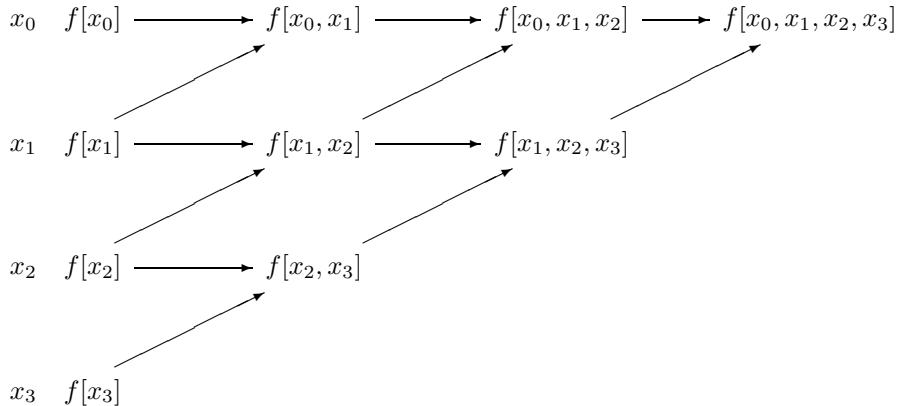
12

Sir Isaac Newton (1643 - 1727) a fost una dintre cele mai remarcabile figuri ale matematicii și fizicii din vremea sa. Nu numai că dat legile fundamentale ale fizicii moderne, dar a fost și unul dintre inventatorii calculului diferențial și integral (alături de Leibniz, cu care a intrat într-o polemică de o viată privind prioritatea). Lucrarea sa care a avut cea mai mare influență a fost *Principia*, care conține ideile sale asupra interpolării și utilizării ei la integrare.

și coeficientul lui  $x^m$  este

$$f[x_0, \dots, x_m] = \sum_{i=0}^m \frac{f(x_i)}{\prod_{\substack{j=0 \\ j \neq i}}^{m-1} (x_i - x_j)}. \quad (5.4.5)$$

Diferențele divizate se pot obține prin algoritm tabelar următor, bazat pe formula (5.4.4), care este mai flexibil și mai puțin costisitor decât aplicarea formulei (5.4.5)



Prima coloană conține valorile lui  $f$ , a doua valorile diferențelor divizate de ordinul I ş.a.m.d; se trece de la o coloană la următoarea utilizând formula (5.4.4): fiecare element este diferență dintre elementul situat în stânga și dedesubtul lui și elementul situat imediat în stânga lui, împărțită la diferența dintre valoarea lui  $x$  determinată mergând pe diagonală în jos și valoarea lui  $x$  situată la stânga pe orizontală. Diferențele divizate care apar în formula lui Newton (5.4.3) sunt cele  $m + 1$  elemente de pe prima linie a tabelei diferențelor divizate. Calculul lor necesită  $n(n+1)$  adunări și  $\frac{1}{2}n(n+1)$  împărțiri. Adăugarea unui nou punct  $(x_{m+1}, f[x_{m+1}])$  necesită generarea diagonalei următoare.  $L_{m+1}f$  poate fi obținut din  $L_m f$  adăugând termenul  $f[x_0, \dots, x_{m+1}](x - x_0) \dots (x - x_{m+1})$ .

Sursa MATLAB 5.14 generează tabela diferențelor divizate, iar sursa 5.15 obține forma Newton a polinomului de interpolare Lagrange.

**Observația 5.4.4.** Eroarea de interpolare este dată de

$$f(x) - (L_m f)(x) = u_m(x) f[x_0, x_1, \dots, x_m, x]. \quad (5.4.6)$$

Într-adevăr, este suficient să observăm că

$$(L_m f)(t) + u_m(t) f[x_0, \dots, x_m; x]$$

este conform lui (5.4.3) polinomul de interpolare (în  $t$ ) al lui  $f$  în punctele  $x_0, x_1, \dots, x_m, x$ . Se deduce din teorema referitoare la restul formulei de interpolare Lagrange (5.3.22) că există  $\xi \in (a, b)$  astfel încât

$$f[x_0, x_1, \dots, x_m] = \frac{1}{m!} f^{(m)}(\xi) \quad (5.4.7)$$

**Sursa MATLAB 5.14** Generarea tabelei diferențelor divizate

---

```

function td=difdiv(x,f);
%DIFDIV - obtine tabela diferențelor divizate
%apel td=difdiv(x,f);
%x - nodurile
%f- valorile functiei
%td - tabela diferențelor divizate

lx=length(x);
td=zeros(lx,1);
td(:,1)=f';
for j=2:lx
    td(1:lx-j+1,j)=diff(td(1:lx-j+2,j-1))./...
        (x(j:lx)-x(1:lx-j+1))';
end

```

---

**Sursa MATLAB 5.15** Calculul formei Newton a polinomului de interpolare Lagrange

---

```

function z=pNewton(td,x,t)
%PNEWTON - calculeaza PIL in forma Newton
%apel z=pNewton(td,x,t)
%td - tabela diferențelor divizate
%x - nodurile de interpolare
%t - punctele in care se calculeaza valoarea
%      polinomului de interpolare
%z - valorile polinomului de interpolare

lt=length(t); lx=length(x);
for j=1:lt
    d=t(j)-x;
    z(j)=[1,cumprod(d(1:lx-1))]*td(1,:)';
end

```

---

(formula de medie pentru diferențe divizate).  $\diamond$

Diferența divizată se poate scrie sub forma unui cât a doi determinanți.

**Teorema 5.4.5.** Are loc

$$f[x_0, \dots, x_m] = \frac{(Wf)(x_0, \dots, x_m)}{V(x_0, \dots, x_m)} \quad (5.4.8)$$

unde

$$(Wf)(x_0, \dots, x_n) = \begin{vmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{m-1} & f(x_0) \\ 1 & x_1 & x_1^2 & \dots & x_1^{m-1} & f(x_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{m-1} & f(x_m) \end{vmatrix}, \quad (5.4.9)$$

iar  $V(x_0, \dots, x_m)$  este determinantul Vandermonde.

*Demonstrație.* Se dezvoltă  $(Wf)(x_0, \dots, x_m)$  după elementele ultimei coloane și înănd cont că fiecare complement algebric este un determinant Vandermonde, se obține

$$\begin{aligned} f[x_0, \dots, x_m] &= \frac{1}{V(x_0, \dots, x_m)} \sum_{i=0}^m V(x_0, \dots, x_{i-1}, x_{i+1}, \dots, x_m) f(x_i) = \\ &= \sum_{i=0}^m (-1)^{m-i} \frac{f(x_i)}{(x_i - x_0) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_n - x_i)}, \end{aligned}$$

din care după schimbarea semnelor ultimilor  $m - i$  termeni rezultă (5.4.5).  $\square$

### 5.4.3. Diferențe finite: formula lui Newton progresivă și regresivă

În cazul când punctele de interpolare  $x_i$  nu sunt echidistante se utilizează algoritmul lui Newton descris anterior; în cazul când punctele sunt echidistante se poate construi un algoritm mai simplu și mai puțin costisitor. Istorici acești algoritmi au avut o mare importanță pentru interpolarea funcțiilor ale căror valori erau tabelate; apariția calculatoarelor moderne a diminuat acest interes, dar noile (co)procesoare flotante i-au readus în actualitate.

Presupunem că funcția  $f$  este cunoscută în punctele  $x_i$  cu pasul  $h$

$$x_i = x_0 + ih, \quad i = 0, 1, \dots$$

Se definesc diferențele progresive prin

$$\Delta^0 f_i = f(x_i) = f_i,$$

și  $\forall k \geq 0$

$$\Delta^{k+1} f_i = \Delta(\Delta^k f_i) = \Delta^k f_{i+1} - \Delta^k f_i.$$

Cu ajutorul formulei (5.4.5) se verifică imediat că

$$\Delta^k f_i = f[x_i, x_{i+1}, \dots, x_{i+k}] k! h^k.$$

Forma polinomului Newton  $L_m f$  de gradul  $m$  a lui  $f$  în punctele  $x_0, x_1, \dots, x_m$  se simplifică. Dacă  $s = (x - x_0)/h$ ,

$$(x-x_0)(x-x_1) \dots (x-x_{k-1})f[x_0, x_1, \dots, x_k] = \frac{s(s-1)\dots(s-k+1)}{k!} \Delta^k f_0$$

şİ deci

$$(L_m f)(x) = f_0 + \frac{s}{1!} \Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 + \dots$$

$$+ \frac{s(s-1)\dots(s-m+1)}{m!} \Delta^m f_0.$$

Utilizând pentru  $s \in \mathbb{R}$ ,  $k \in \mathbb{N}$  notăția

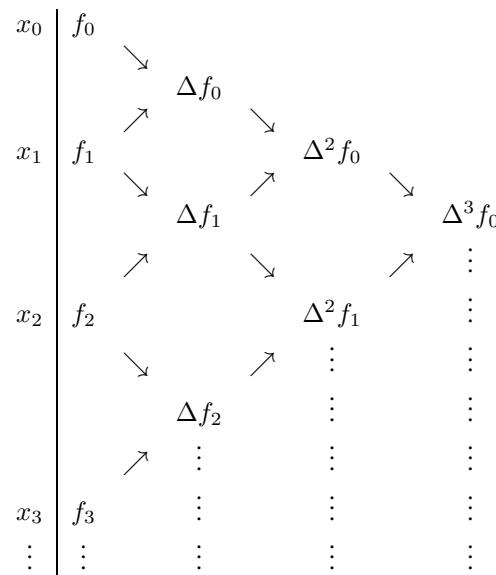
$$\binom{s}{k} = \frac{s(s-1)\dots(s-k+1)}{k!}$$

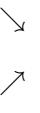
(coefficient binomial generalizat), se obține *formula lui Newton progresivă*

$$(L_m f)(x) = f_0 + \binom{s}{1} \Delta f_0 + \binom{s}{2} \Delta^2 f_0 + \cdots + \binom{s}{n} \Delta^n f_0, \quad (5.4.10)$$

unde  $s = (x - x_0)/h$ .

În practică această formulă servește la calculul lui  $(L_m f)(x)$  în puncte apropiate de începutul tabelei. Diferențele finite succesive se obțin din tabloul triunghiular



unde   $c$  înseamnă  $c = b - a$ .

Eroarea comisă după  $m$  pași, se scrie pentru  $x = x_0 + sh$

$$f(x) - (L_m f)(x) = h^{m+1} \binom{s}{m+1} f^{(m+1)}(\xi_x), \quad (5.4.11)$$

unde  $\xi_x$  aparține celui mai mic interval ce conține  $x_0, x_m$  și  $x$ .

Analog se introduce operatorul de *diferență finită regresivă*  $\nabla$  prin

$$\begin{aligned}\nabla^0 f_i &= f_i \\ \nabla^1 f_i &= f_i - f_{i-1} \\ \nabla^{k+1} f_i &= \nabla(\nabla^k f_i) = \nabla^k f_i - \nabla^k f_{i-1}\end{aligned}$$

Efectuând schimbarea de variabilă  $s = \frac{x - x_n}{m}$  se obține

$$\begin{aligned}(L_m f)(x) &= f_m + \frac{s}{1!} \nabla f_m + \frac{s(s+1)}{2!} \nabla^2 f_m + \dots \\ &\quad + \frac{s(s+1)\dots(s+n-1)}{m!} \nabla^m f_m,\end{aligned}$$

care se mai poate scrie

$$(L_m f)(x) = f_m + \binom{s}{1} \nabla f_m + \binom{s+1}{2} \nabla^2 f_m + \dots + \binom{s+m-1}{m} \nabla^m f_m, \quad (5.4.12)$$

numită *formula lui Newton regresivă*.

Eroarea de interpolare se scrie sub forma

$$f(x) - (L_m f)(x) = h^{m+1} \binom{s+m}{m+1} f^{(m+1)}(\xi_x).$$

unde  $\xi_x$  aparține celui mai mic interval ce conține  $x_0, x_m$  și  $x$ . Formula (5.4.12) servește pentru calculul polinomului de interpolare în valori apropriate de capătul tabelei. Diferențele regresive se pot calcula cu tabela

$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x_{m-3}$	$f_{m-3}$	$\vdots$	$\vdots$	$\vdots$
		$\searrow$	$\vdots$	$\vdots$
			$\nabla f_{m-2}$	$\vdots$
		$\nearrow$	$\searrow$	$\vdots$
$x_{m-2}$	$f_{m-2}$		$\nabla^2 f_{m-1}$	$\vdots$
		$\searrow$	$\nearrow$	$\vdots$
		$\nabla f_{m-1}$	$\nearrow$	$\searrow$
$x_{m-1}$	$f_{m-1}$	$\nearrow$	$\nabla^2 f_m$	$\nearrow$
		$\searrow$	$\nearrow$	$\vdots$
		$\nabla f_m$	$\nearrow$	
$x_m$	$f_m$	$\nearrow$		

unde  $\begin{array}{c} a \\ \searrow \\ c \\ \nearrow \\ b \end{array}$  înseamnă  $c = b - a$ .

#### 5.4.4. Metoda baricentrică

Rescriem (5.3.7), (5.3.8) astfel încât polinomul de interpolare Lagrange să poată fi evaluat și actualizat cu  $O(m)$  operații. Avem

$$\ell_j(x) = \frac{u_m(x)}{\prod_{k \neq j} (x_j - x_k)} \cdot \frac{1}{x - x_j}, \quad (5.4.13)$$

unde

$$u_m(x) = (x - x_0)(x - x_1) \cdots (x - x_m). \quad (5.4.14)$$

Definind ponderile baricentrice prin

$$w_j = \frac{1}{\prod_{k \neq j} (x_j - x_k)}, \quad j = 0, \dots, m, \quad (5.4.15)$$

adică,  $w_j = 1/u'_m(x_j)$ , putem scrie  $\ell_j$  sub forma

$$\ell_j(x) = u_m(x) \frac{w_j}{x - x_j}. \quad (5.4.16)$$

Acum PIL se scrie ( $f_j := f(x_j)$ )

$$(L_m f)(x) = u_m(x) \sum_{j=0}^m \frac{w_j}{x - x_j} f_j. \quad (5.4.17)$$

Formula (5.4.17) se numește *prima formulă baricentrică*.

Interpolând funcția constantă 1 obținem

$$1 = \sum_{j=0}^m \ell_j(x) = u_m(x) \sum_{j=0}^m \frac{w_j}{x - x_j}. \quad (5.4.18)$$

Împărțind (5.4.17) cu expresia de mai sus și simplificând cu  $u_m(x)$ , obținem formula

$$p(x) = \frac{\sum_{j=0}^m \frac{w_j}{x - x_j} f_j}{\sum_{j=0}^m \frac{w_j}{x - x_j}}, \quad (5.4.19)$$

numită *a doua formulă baricentrică* [56].

### Distribuții remarcabile ale nodurilor

În cazul unor noduri particulare se pot da formule explicite pentru ponderile baricentrice  $w_j$ , pornind de la formula

$$w_j = \frac{1}{u'_m(x_j)}, \quad (5.4.20)$$

- Pentru noduri echidistante avem

$$w_j = (-1)^j \binom{m}{j}. \quad (5.4.21)$$

- Familia de *puncte Cebîșev* se poate obține proiectând puncte egal spațiate pe cercul unitate pe intervalul  $[-1, 1]$ .

– *Punctele Cebîșev de speță I* sunt date de

$$x_j = \cos \frac{(2j+1)\pi}{2m+2}, \quad j = 0, \dots, m.$$

Anulând factorii independenți de  $j$  se obține

$$w_j = (-1)^j \sin \frac{(2j+1)\pi}{2m+2}. \quad (5.4.22)$$

– Punctele Cebîșev de speță II sau punctele Gauss-Lobatto pe  $[-1, 1]$  sunt date de

$$x_j = \cos \frac{\pi j}{n}, \quad j = 0, \dots, n \quad (5.4.23)$$

iar ponderile baricentrice au forma

$$w_j = \frac{2^{n-1}}{n} \begin{cases} (-1)^j / 2 & \text{dacă } j = 0 \text{ sau } j = n, \\ (-1)^j & \text{altfel.} \end{cases} \quad (5.4.24)$$

Factorul  $2^{n-1}/n$  se poate elimina deoarece în formula baricentrică (5.4.19) apare și la numărător și la numitor:

$$w_j = (-1)^j \delta_j, \quad \delta_j = \begin{cases} 1/2, & j = 0 \text{ sau } j = m, \\ 1, & \text{altfel.} \end{cases} \quad (5.4.25)$$

- Pentru un interval  $[a, b]$  se face schimbarea de variabilă

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

### Interpolarea în puncte Cebîșev

Dificultățile legate de interpolarea polinomială de grad mare pot fi depășite aglomerând punctele de interpolare la capătul intervalului în loc de a alege puncte echidistante. De exemplu, putem alege ca noduri puncte de interpolare Cebîșev de speță a doua.

Aceste noduri sunt utilizate în pachetul MATLAB `chebfun`, realizat la Universitatea din Oxford, de un colectiv condus de profesorul L. N. Trefethen [18].

Dacă  $(x_j)_{j=0}^n$  sunt puncte Cebîșev, polinomul nodurilor satisface

$$\left| \prod_{j=0}^n (x - x_j) \right| \leq 2^{-n+1}$$

Teorema 5.4.6 da câteva proprietăți ale interpolării în puncte Cebîșev:

**Teorema 5.4.6.** Fie  $f \in C[-1, 1]$ ,  $p_n$  polinomul său de interpolare în puncte Cebîșev (5.4.23) și  $p_n^*$  polinomul său de cea mai bună aproximare în normă  $\|\cdot\|_\infty$ . Atunci

1.  $\|f - p_n\|_\infty \leq (2 + \frac{2}{\pi} \log n) \|f - p_n^*\|_\infty$
2. Dacă  $\exists k \in \mathbb{N}^*$  a.î.  $f^{(k)}$  este cu variație mărginită pe  $[-1, 1]$ , atunci  $\|f - p_n\|_\infty = O(n^{-k})$ , când  $n \rightarrow \infty$ .
3. Dacă  $f$  este analitică într-o vecinătate din planul complex a lui  $[-1, 1]$ , atunci  $\exists C < 1$  a.î.  $\|f - p_n\|_\infty = O(C^n)$ ; în particular dacă  $f$  este analitică în elipsă închisă cu focarele  $\pm 1$  și semiaxele  $M \geq 1$  și  $m \geq 0$ , putem lua  $C = 1/(M + m)$ .

Putem implementa formula de interpolare baricentrică astfel:

**Sursa MATLAB 5.16 Interpolare Lagrange baricentrică**


---

```

function ff=barycentricInterpolation(x,y,xx,c)
%BARYCENTRICINTERPOLATION - barycentric Lagrange interpolation
%call ff=barycentricInterpolation(x,y,xx,c)
%x - nodes
%y - function values
%xx - interpolation points
%c - barycentric weights
%ff - values of interpolation polynomial

numer = zeros(size(xx));
denom = zeros(size(xx));
exact = zeros(size(xx)); %test if xx=nodes
for j=1:n+1
    xdiff = xx-x(j);
    temp = c(j)./xdiff;
    numer = numer+temp*y(j);
    denom = denom+temp;
    exact(xdiff==0) = j;
end
ff = numer ./ denom;
jj = find(exact);
ff(jj) = y(exact(jj));

```

---

- Dacă punctul de evaluare  $x$  este unul dintre noduri, să zicem  $x = x_i$ , atunci se returnează valoarea  $f_i = f(x_i)$ .
- Altfel, se aplică formula (5.4.19).

Dăm o implementare în MATLAB a metodei baricentrice (funcția `barycentricInterpolation`), sursa MATLAB 5.16, inspirată din [5].

Ponderile baricentrice se pot calcula cu funcția `barycentricWeights`, vezi sursa MATLAB 5.17. Pentru interpolarea baricentrică cu noduri Cebîșev de speță I și de speță a două avem funcțiile `ChebLagrangeK1` (sursa MATLAB 5.18) și respectiv `ChebLagrange` (sursa MATLAB 5.19).

Dăm un exemplu. Vom interpola funcția  $f : [-1, 1] \rightarrow \mathbb{R}$ ,  $f(x) = |x| + 1/2x - x^2$  cu  $m = 20$  și  $m = 100$  de noduri Cebîșev de speță a două.

```

m = 20;
fun = @(x) abs(x)+.5*x-x.^2;
x = sort(cos(pi*(0:m)/m));
f = fun(x);
xx = linspace(-1,1,5000);
ff=ChebLagrange(f,xx);
subplot(2,1,1)
plot(x,f,'.',xx,ff,'-')
m = 100;

```

**Sursa MATLAB 5.17** Ponderi baricentrice

---

```
function c = barycentricweights( x )
% BARYCENTRICWEIGHTS - compute barycentric weights (coefficient)
% call c = barycentricweights( x )
%x - nodes
%c - weights

n=length(x)-1;
c=ones(1,n+1);
for j=1:n+1
    c(j)=prod(x(j)-x([1:j-1, j+1:n+1]));
end
c=1./c;
end
```

---

**Sursa MATLAB 5.18** Interpolare Lagrange baricentrică cu noduri Cebîșev de speță I

---

```
function ff=ChebLagrangeK1(y,xx,a,b)
%CHEBLAGRANGEK1 - Lagrange interpolation for Chebyshev #1 points
% - barycentric method
%call ff=ChebLagrangeK1(y,xx,a,b)
%y - function values;
%xx - evaluation points
%a,b - interval
%ff - values of Lagrange interpolation polynomial

n = length(y)-1;
if nargin==2
    a=-1; b=1;
end
c = sin((2*(0:n)' + 1) * pi / (2*n+2)) .* (-1).^(0:n)';
x = sort(cos((2*(0:n)' + 1) * pi / (2*n+2)) * (b-a)/2 + (a+b)/2);
ff=barycentricInterpolation(x,y,xx,c);
end
```

---

**Sursa MATLAB 5.19 Interpolare Lagrange baricentrică cu noduri Cebîșev de speță II**

```

function ff=ChebLagrange(y,xx,a,b)
%CHEBLAGRANGE - Lagrange interpolation for Chebyshev #2 points
%- barycentric
%call ff=ChebLagrange(y,xx,a,b)
%y - function values;
%xx - evaluation points
%a,b - interval
%ff - values of Lagrange interpolation polynomial

n = length(y)-1;
if nargin==2
    a=-1; b=1;
end
c = [1/2; ones(n-1,1); 1/2].*(-1).^(0:n)';
x = sort(cos((0:n)'*pi/n))*(b-a)/2+(a+b)/2;
ff=barycentricInterpolation(x,y,xx,c);
end

```

```

x = sort(cos(pi*(0:m)/m));
f = fun(x);
xx = linspace(-1,1,5000);
ff=ChebLagrange(f,xx);
subplot(2,1,2)
plot(x,f,'.',xx,ff,'-')

```

Graficul apare în figura 5.15.

### 5.4.5. Diferențe divizate cu noduri multiple

Formula (5.4.8) servește ca bază pentru introducerea diferenței divizate cu noduri multiple: dacă  $f \in C^m[a, b]$  și  $\alpha \in [a, b]$ , atunci

$$\lim_{x_0, \dots, x_n \rightarrow \alpha} [x_0, \dots, x_n; f] = \lim_{\xi \rightarrow \alpha} \frac{f^{(m)}(\xi)}{m!} = \frac{f^{(m)}(\alpha)}{m!}$$

Aceasta justifică relația

$$[\underbrace{\alpha, \dots, \alpha}_{m+1}; f] = \frac{1}{m!} f^{(m)}(\alpha).$$

Reprezentând aceasta ca pe un cât de doi determinanți se obține

$$(Wf) \left( \underbrace{\alpha, \dots, \alpha}_{m+1} \right) = \begin{vmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{m-1} & f(\alpha) \\ 0 & 1 & 2\alpha & \dots & (m-1)\alpha^{m-2} & f'(\alpha) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & (m-1)! & f^{(m-1)}(\alpha) \end{vmatrix}$$

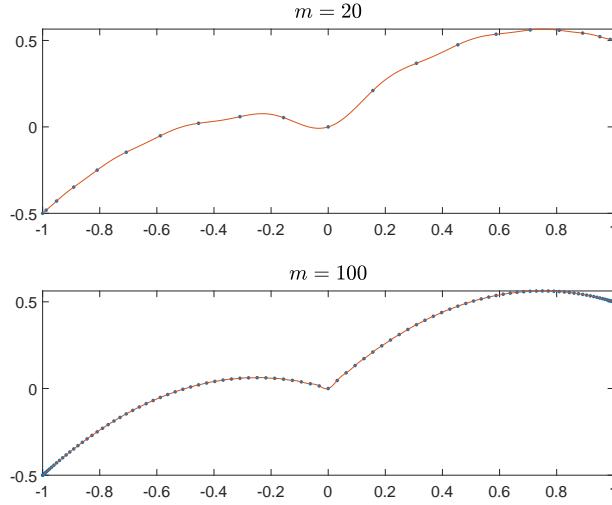


Figura 5.15: Interpolare Lagrange baricentrică

și

$$V \begin{pmatrix} \underbrace{\alpha, \dots, \alpha}_{m+1} \end{pmatrix} = \begin{vmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^m \\ 0 & 1 & 2\alpha & \dots & m\alpha^{m-1} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & m! \end{vmatrix},$$

adică cei doi determinanți sunt constituuiți din linia relativă la nodul  $\alpha$  și derivatele succeseive ale acestuia până la ordinul  $m$  în raport cu  $\alpha$ .

Generalizarea pentru mai multe noduri este următoarea:

**Definiția 5.4.7.** Fie  $r_k \in \mathbb{N}$ ,  $k = \overline{0, m}$ ,  $n = r_0 + \dots + r_m$ . Presupunem că există  $f^{(j)}(x_k)$ ,  $k = \overline{0, m}$ ,  $j = \overline{0, r_k - 1}$ . Mărimea

$$\underbrace{x_0, \dots, x_0}_{r_0}, \underbrace{x_1, \dots, x_1}_{r_1}, \dots, \underbrace{x_m, \dots, x_m}_{r_m}, f = \frac{(Wf)(x_0, \dots, x_0, \dots, x_m, \dots, x_m)}{V(x_0, \dots, x_0, \dots, x_m, \dots, x_m)}$$

unde

$$(Wf)(x_0, \dots, x_0, \dots, x_m, \dots, x_m) =$$

$$= \begin{vmatrix} 1 & x_0 & \dots & x_0^{r_0-1} & \dots & x_0^{n-1} & f(x_0) \\ 0 & 1 & \dots & (r_0-1)x_0^{r_0-2} & \dots & (n-1)x_0^{n-2} & f'(x_0) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & (r_0-1)! & \dots & \prod_{p=1}^{r_0-1} (n-p)x_0^{n-r_0} & f^{(r_0-1)}(x_0) \\ 1 & x_m & \dots & x_m^{r_m-1} & \dots & x_m^{n-1} & f(x_m) \\ 0 & 1 & \dots & (r_m-1)x_m^{r_m-2} & \dots & (n-1)x_m^{n-2} & f'(x_m) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & (r_m-1)! & \dots & \prod_{p=1}^{r_m-1} (n-p)x_m^{n-r_m} & f^{(r_m-1)}(x_n) \end{vmatrix}$$

$$\begin{array}{llll}
z_0 = x_0 & f[z_0] & f[z_0, z_1] = f'(x_0) & f[z_0, z_1, z_2] = \frac{f[z_1, z_2] - f[z_0, z_1]}{z_2 - z_0} \\
z_1 = x_0 & f[z_1] & f[z_1, z_2] = \frac{f(z_2) - f(z_1)}{z_2 - z_1} & f[z_1, z_2, z_3] = \frac{f[z_3, z_2] - f[z_2, z_1]}{z_3 - z_1} \\
z_2 = x_1 & f[z_2] & f[z_2, z_3] = f'(x_1) & f[z_2, z_3, z_4] = \frac{f[z_4, z_3] - f[z_3, z_2]}{z_4 - z_2} \\
z_3 = x_1 & f[z_3] & f[z_3, z_4] = \frac{f(z_4) - f(z_3)}{z_4 - z_3} \\
z_4 = x_2 & f[z_4] & f[z_4, z_5] = f'(x_2) \\
z_5 = x_2 & f[z_5]
\end{array}$$

Tabela 5.2: Tabelă de diferențe divizate pentru noduri duble

iar  $V(x_0, \dots, x_0, \dots, x_m, \dots, x_m)$  este ca mai sus, exceptând ultima coloană care este

$$(x_0^n, nx_0^{n-1}, \dots, \prod_{p=0}^{r_0-2} (n-p)x_0^{n-r_0+1}, \dots, x_m^n, nx_m^{n-1}, \dots, \prod_{p=0}^{r_m-2} x_m^{n-r_m+1})^T$$

se numește diferență divizată cu nodurile multiple  $x_k$ ,  $k = \overline{0, m}$  și ordinele de multiplicitate  $r_k$ ,  $k = \overline{0, m}$ .

Generalizând forma Newton a polinomului de interpolare Lagrange se obține o metodă pentru calculul polinomului de interpolare Hermite bazată pe diferențele divizate cu noduri multiple.

Presupunem că se dau nodurile  $x_i$ ,  $i = \overline{0, m}$  și valorile  $f(x_i)$ ,  $f'(x_i)$ . Definim secvența de noduri  $z_0, z_1, \dots, z_{2n+1}$  prin  $z_{2i} = z_{2i+1} = x_i$ ,  $i = \overline{0, m}$ . Construim acum tabela diferențelor divizate utilizând nodurile  $z_i$ ,  $i = \overline{0, 2m+1}$ . Deoarece  $z_{2i} = z_{2i+1} = x_i$  pentru orice  $i$ ,  $f[z_{2i}, z_{2i+1}]$  este o diferență divizată cu nod dublu și este egală cu  $f'(x_i)$ , deci vom utiliza  $f'(x_0), f'(x_1), \dots, f'(x_m)$  în locul diferențelor divizate de ordinul I

$$f[z_0, z_1], f[z_2, z_3], \dots, f[z_{2m}, z_{2m+1}].$$

Restul diferențelor se obțin în manieră obișnuită, aşa cum se arată în tabelul 5.2. Ideea poate fi extinsă și pentru alte interpolări Hermite. Se pare că metoda este datorată lui Powell.

Sursa MATLAB 5.20 conține o funcție pentru calculul diferențelor divizate cu noduri duble, `dfdivnd`. Ea returnează vectorul nodurilor dublate și tabela de diferențe. Pentru a calcula polinomul de interpolare Hermite se poate utiliza funcția `pNewton` cu tabela și nodurile returnate de `dfdivnd`.

## 5.5. Interpolare spline

Fie  $\Delta$  o diviziune a lui  $[a, b]$

$$\Delta : a = x_1 < x_2 < \dots < x_{n-1} < x_n = b. \quad (5.5.1)$$

Vom utiliza un polinom de grad mic pe subintervalul  $[x_i, x_{i+1}]$ ,  $i = \overline{1, n-1}$ . Motivul este acela că pe intervale suficiente de mici funcțiile pot fi aproximate arbitrar de bine prin polinoame de grad mic, chiar 0 sau 1.

Am introdus în exemplul 5.0.2 spațiul

$$\mathbb{S}_m^k(\Delta) = \{s : s \in C^k[a, b], s|_{[x_i, x_{i+1}]} \in \mathbb{P}_m, i = 1, 2, \dots, n-1\} \quad (5.5.2)$$

$m \geq 0$ ,  $k \in \mathbb{N} \cup \{-1\}$ , numit spațiu funcțiilor spline polinomiale de grad  $m$  și clasă de netezime  $k$ . Dacă  $k = m$ , atunci funcțiile  $s \in \mathbb{S}_m^m(\Delta)$  sunt polinoame.

**Sursa MATLAB 5.20** Generarea tableei de diferențe divizate cu noduri duble

```

function [z,td]=difdivnd(x,f,fd);
%DIFDIVND - tabela diferențelor divizate cu noduri duble
%apel td=difdivnd(x,f,fd)
%x -nodurile
%f - valorile functiei in noduri
%fd - valorile derivatei in noduri
%z - nodurile dublate
%td - tabela de diferențe

z=zeros(1,2*length(x));
lz=length(z);
z(1:2:1z-1)=x;
z(2:2:1z)=x;
td=zeros(lz,lz);
td(1:2:1z-1,1)=f';
td(2:2:1z,1)=f';
td(1:2:1z-1,2)=fd';
td(2:2:1z-2,2)=(diff(f)./diff(x))';
for j=3:1z
    td(1:1z-j+1,j)=diff(td(1:1z-j+2,j-1))./...
        (z(j:1z)-z(1:1z-j+1))';
end

```

**5.5.1. Spline liniare**

Pentru  $m = 1$  și  $k = 0$  se obțin *spline liniare*. Dorim să găsim  $s \in \mathbb{S}_1^0(\Delta)$  astfel încât

$$s(x_i) = f_i, \text{ unde } f_i = f(x_i), \quad i = 1, 2, \dots, n.$$

Soluția este trivială, vezi figura 5.16. Pe intervalul  $[x_i, x_{i+1}]$

$$s(f; x) = f_i + (x - x_i)f[x_i, x_{i+1}], \quad (5.5.3)$$

iar

$$|f(x) - s(f(x))| \leq \frac{(\Delta x_i)^2}{8} \max_{x \in [x_i, x_{i+1}]} |f''(x)|. \quad (5.5.4)$$

Rezultă că

$$\|f(\cdot) - s(f, \cdot)\|_\infty \leq \frac{1}{8} |\Delta|^2 \|f''\|_\infty. \quad (5.5.5)$$

Dimensiunea lui  $\mathbb{S}_1^0(\Delta)$  se calculează astfel: deoarece avem  $n - 1$  porțiuni și pe fiecare 2 coeficienți (2 grade de libertate) și fiecare condiție reduce numărul de grade de libertate cu 1, avem în final

$$\dim \mathbb{S}_1^0(\Delta) = 2n - 2 - (n - 2) = n.$$

O bază a spațiului este dată de așa-numitele funcții *B-spline*. Punem  $x_0 = x_1, x_{n+1} = x_n$ , pentru  $i = \overline{1, n}$

$$B_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}}, & \text{pentru } x_{i-1} \leq x \leq x_i \\ \frac{x_{i+1} - x}{x_{i+1} - x_i}, & \text{pentru } x_i \leq x \leq x_{i+1} \\ 0, & \text{în rest} \end{cases} \quad (5.5.6)$$

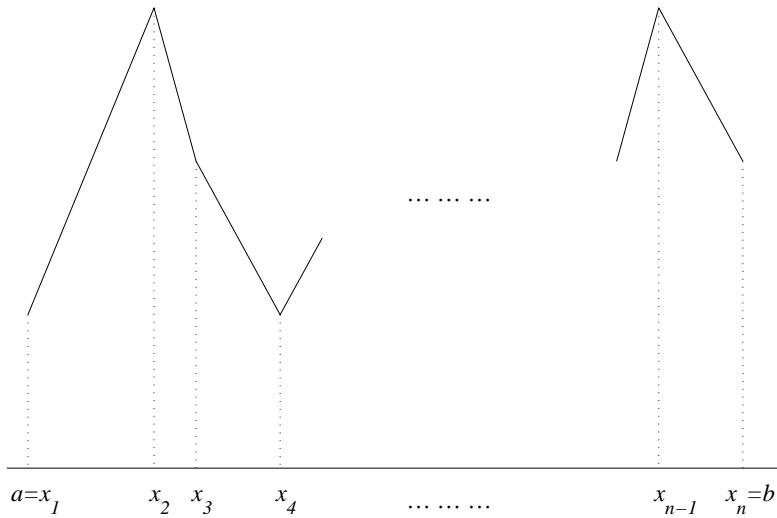


Figura 5.16: Spline liniare

Pentru  $i = 1$  prima și pentru  $i = n$  a doua ecuație se ignoră. Funcția  $B_i$  se numește *pălărie chinezească*. Graficul funcțiilor  $B_i$  apare în figura 5.17. Ele au proprietatea

$$B_i(x_j) = \delta_{ij},$$

sunt liniar independente, deoarece

$$s(x) = \sum_{i=1}^n c_i B_i(x) = 0 \wedge x \neq x_j \Rightarrow c_j = 0.$$

și

$$\langle B_i \rangle_{i=\overline{1,n}} = S_1^0(\Delta),$$

$B_i$  joacă același rol ca polinoamele fundamentale Lagrange  $\ell_i$ .

### 5.5.2. Interpolarea cu spline cubice

Funcțiile spline cubice sunt cele mai utilizate.

Vom discuta întâi problema interpolării pentru  $s \in \mathbb{S}_3^1(\Delta)$ . Continuitatea derivatei de ordinul I pentru  $s_3(f; \cdot)$  se poate realiza impunând valorile primei deriveate în fiecare punct  $x_i$ ,  $i = 1, 2, \dots, n$ . Astfel fie  $m_1, m_2, \dots, m_n$  numere arbitrar date și notăm

$$s_3(f; \cdot)|_{[x_i, x_{i+1}]} = p_i(x), \quad i = 1, 2, \dots, n - 1 \quad (5.5.7)$$

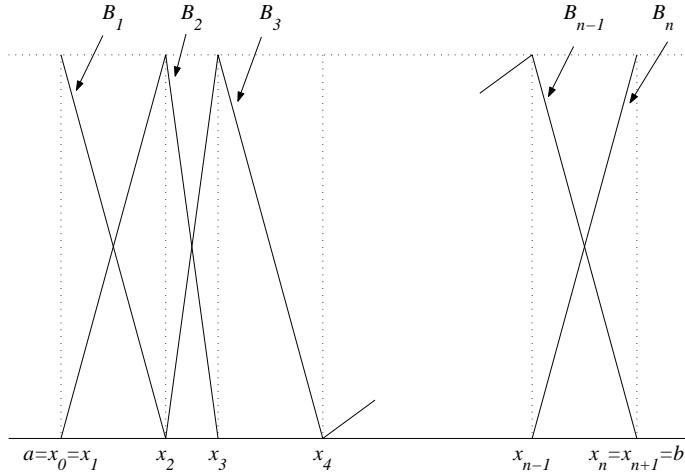


Figura 5.17: Funcții B-spline de grad 1

Realizăm  $s_3(f; x_i) = m_i$ ,  $i = \overline{1, n}$ , luând fiecare bucătă ca soluție unică a problemei de interpolare Hermite, și anume

$$\begin{aligned} p_i(x_i) &= f_i, & p_i(x_{i+1}) &= f_{i+1}, & i &= \overline{1, n-1}, \\ p'_i(x_i) &= m_i, & p'_i(x_{i+1}) &= m_{i+1} \end{aligned} \quad (5.5.8)$$

Vom rezolva problema folosind interpolarea Newton. Diferențele divizate sunt

$$\begin{array}{ccccccccc} x_i & f_i & & m_i & & \frac{f[x_i, x_{i+1}] - m_i}{\Delta x_i} & & \frac{m_{i+1} + m_i - 2f[x_i, x_{i+1}]}{(\Delta x_i)^2} \\ x_i & f_i & f[x_i, x_{i+1}] & & \frac{m_{i+1} - f[x_i, x_{i+1}]}{\Delta x_i} & & & \\ x_{i+1} & f_{i+1} & & m_{i+1} & & & & \\ x_{i+1} & f_{i+1} & & & & & & \end{array}$$

și deci forma Newton a polinomului de interpolare Hermite este

$$\begin{aligned} p_i(x) &= f_i + (x - x_i)m_i + (x - x_i)\frac{f[x_i, x_{i+1}] - m_i}{\Delta x_i} + \\ &\quad + (x - x_i)^2(x - x_{i+1})\frac{m_{i+1} + m_i - 2f[x_i, x_{i+1}]}{(\Delta x_i)^2}. \end{aligned}$$

Forma Taylor a lui  $p_i$  pentru  $x_i \leq x \leq x_{i+1}$  este

$$p_i(x) = c_{i,0} + c_{i,1}(x - x_i) + c_{i,2}(x - x_i)^2 + c_{i,3}(x - x_i)^3 \quad (5.5.9)$$

și deoarece  $x - x_{i+1} = x - x_i - \Delta x_i$ , prin identificare avem

$$\begin{aligned} c_{i,0} &= f_i \\ c_{i,1} &= m_i \\ c_{i,2} &= \frac{f[x_i, x_{i+1}] - m_i}{\Delta x_i} - c_{i,3} \Delta x_i \\ c_{i,3} &= \frac{m_{i+1} + m_i - 2f[x_i, x_{i+1}]}{(\Delta x_i)^2} \end{aligned} \quad (5.5.10)$$

Deci, pentru a calcula  $s_3(f; x)$  într-un punct care nu este nod, trebuie în prealabil să localizăm intervalul  $[x_i, x_{i+1}] \ni x$ , apoi să calculăm coeficienții cu (5.5.10) și să evaluăm spline-ul cu (5.5.9).

Vom discuta câteva alegeri posibile pentru  $m_1, m_2, \dots, m_n$ .

### Interpolare Hermite cubică pe porțiuni

Se alege  $m_i = f'(x_i)$  (presupunând că aceste derive sunt cunoscute). Se ajunge la o schemă strict locală, în care fiecare bucată poate fi determinată independent de cealaltă. Mai mult, eroarea este

$$|f(x) - p_i(x)| \leq \left( \frac{1}{2} \Delta x_i \right)^4 \max_{x \in [x_i, x_{i+1}]} \frac{|f^{(4)}(x)|}{4!}, \quad x_i \leq x \leq x_{i+1}. \quad (5.5.11)$$

Deci

$$\|f(\cdot) - s_3(f; \cdot)\|_\infty \leq \frac{1}{384} |\Delta|^4 \|f^{(4)}\|_\infty. \quad (5.5.12)$$

Pentru puncte echidistante

$$|\Delta| = (b - a)/(n - 1)$$

și deci

$$\|f(\cdot) - s_3(f; \cdot)\|_\infty = O(n^{-4}), \quad n \rightarrow \infty. \quad (5.5.13)$$

### Interpolare cu spline cubice

Cerem ca  $s_3(f; \cdot) \in \mathbb{S}_3^2(\Delta)$ , adică continuitatea derivatelor de ordinul al II-lea. Aceasta înseamnă cu notația (5.5.7)

$$p''_{i-1}(x_i) = p''_i(x_i), \quad i = \overline{2, n-1}, \quad (5.5.14)$$

care convertită în coeficienți Taylor (5.5.9) dă

$$2c_{i-1,2} + 6c_{i-1,3}\Delta x_{i-1} = 2c_{i,2}, \quad i = \overline{2, n-1}.$$

Înlocuind cu valorile explicite (5.5.10) pentru coeficienți, se ajunge la sistemul liniar

$$\Delta x_i m_{i-1} + 2(\Delta x_{i-1} + \Delta x_i)m_i + \Delta x_{i-1}m_{i+1} = b_i, \quad i = \overline{2, n-1}, \quad (5.5.15)$$

unde

$$b_i = 3\{\Delta x_i f[x_{i-1}, x_i] + \Delta x_{i-1} f[x_i, x_{i+1}]\}. \quad (5.5.16)$$

Avem un sistem de  $n - 2$  ecuații liniare cu  $n$  necunoscute  $m_1, m_2, \dots, m_n$ . Odată alese  $m_1$  și  $m_n$ , sistemul devine tridiagonal și se poate rezolva eficient prin eliminare gaussiană, prin factorizare sau cu o metodă iterativă.

Se dau în continuare câteva alegeri posibile pentru  $m_1$  și  $m_n$ .

**Spline complete(racordate, limitate).** Luăm  $m_1 = f'(a)$ ,  $m_n = f'(b)$ . Se știe că pentru acest tip de spline, dacă  $f \in C^4[a, b]$

$$\|f^{(r)}(\cdot) - s^{(r)}(f; \cdot)\|_\infty \leq c_r |\Delta|^{4-r} \|f^{(n)}\|_\infty, \quad r = 0, 1, 2, 3 \quad (5.5.17)$$

unde  $c_0 = \frac{5}{384}$ ,  $c_1 = \frac{1}{24}$ ,  $c_2 = \frac{3}{8}$ , iar  $c_3$  depinde de raportul  $\frac{|\Delta|}{\min_i \Delta x_i}$ .

**Spline care utilizează derivatele secunde.** Impunem condițiile  $s_3''(f; a) = f''(a)$ ;  $s_3''(f; b) = f''(b)$ . Aceste condiții conduc la două ecuații suplimentare

$$\begin{aligned} 2m_1 + m_2 &= 3f[x_1, x_2] - \frac{1}{2}f''(a)\Delta x_1 \\ m_{n-1} + 2m_n &= 3f[x_{n-1}, x_n] + \frac{1}{2}f''(b)\Delta x_{n-1} \end{aligned} \quad (5.5.18)$$

Prima ecuație se pune la începutul sistemului (5.5.15), iar a doua la sfârșitul lui, păstrându-se astfel structura triadiagonală a sistemului.

**Spline cubice naturale.** Impunând  $s''(f; a) = s''(f; b) = 0$ , se obțin două ecuații noi din (5.5.18) luând  $f''(a) = f''(b) = 0$ .

Avantajul – este nevoie numai de valori ale lui  $f$ , nu și ale derivatelor, dar prețul plătit este degradarea preciziei la  $O(|\Delta|^2)$  în vecinătatea capetelor (în afară de cazul când  $f''(a) = f''(b) = 0$ ).

**"Not-a-knot spline".** (C. deBoor). Cerem ca  $p_1(x) \equiv p_2(x)$  și  $p_{n-2}(x) \equiv p_{n-1}(x)$ ; adică primele două părți și respectiv ultimele două trebuie să coincidă. Aceasta înseamnă că primul punct interior  $x_2$  și ultimul  $x_{n-1}$  sunt ambele inactive. Se obțin încă două ecuații suplimentare exprimând continuitatea lui  $s_3'''(f; x)$  în  $x = x_2$  și  $x = x_{n-1}$ . Condiția de continuitate a lui  $s_3(f, \cdot)$  în  $x_2$  și  $x_{n-1}$  revine la egalitatea coeficienților dominanți  $c_{1,3} = c_{2,3}$  și  $c_{n-2,3} = c_{n-1,3}$ . De aici se obțin ecuațiile

$$\begin{aligned} (\Delta x_2)^2 m_1 + [(\Delta x_2)^2 - (\Delta x_1)^2] m_2 - (\Delta x_1)^2 m_3 &= \beta_1 \\ (\Delta x_2)^2 m_{n-2} + [(\Delta x_2)^2 - (\Delta x_1)^2] m_{n-1} - (\Delta x_1)^2 m_n &= \beta_2, \end{aligned}$$

unde

$$\begin{aligned} \beta_1 &= 2\{(\Delta x_2)^2 f[x_1, x_2] - (\Delta x_1)^2 f[x_2, x_3]\} \\ \beta_2 &= 2\{(\Delta x_{n-1})^2 f[x_{n-2}, x_{n-1}] - (\Delta x_{n-2})^2 f[x_{n-1}, x_n]\}. \end{aligned}$$

Prima ecuație se adaugă pe prima poziție iar a două pe ultima poziție a sistemului format din cele  $n - 2$  ecuații date de (5.5.15) și (5.5.16). Sistemul obținut nu mai este triadiagonal, dar el se poate transforma în unul triadiagonal combinând ecuațiile 1 cu 2 și  $n - 1$  cu  $n$ . După aceste transformări prima și ultima ecuație devin

$$\Delta x_2 m_1 + (\Delta x_2 + \Delta x_1) m_2 = \gamma_1 \quad (5.5.19)$$

$$(\Delta x_{n-1} + \Delta x_{n-2}) m_{n-1} + \Delta x_{n-2} m_n = \gamma_2, \quad (5.5.20)$$

unde

$$\begin{aligned} \gamma_1 &= \frac{1}{\Delta x_2 + \Delta x_1} \{f[x_1, x_2] \Delta x_2 [\Delta x_1 + 2(\Delta x_1 + \Delta x_2)] + (\Delta x_1)^2 f[x_2, x_3]\} \\ \gamma_2 &= \frac{1}{\Delta x_{n-1} + \Delta x_{n-2}} \{\Delta x_{n-1}^2 f[x_{n-2}, x_{n-1}] + \\ &\quad [2(\Delta x_{n-1} + \Delta x_{n-2}) + \Delta x_{n-1}] \Delta x_{n-2} f[x_{n-1}, x_n]\}. \end{aligned}$$

Funcția Splinecubic calculează coeficienții celor patru tipuri de spline cubice prezentate. Ea construiește sistemul tridiagonal corespunzător cu matrice rară și îl rezolvă cu operatorul \. Diferențele între tipurile de spline apar la prima și la ultima ecuație, care sunt determinate într-o instrucțiune switch.

```

function [a,b,c,d]=Splinecubic(x,f,tip,der)
%SPLINECUBIC - determina coeficientii spline-ului cubic
%x - abscisele
%f - ordonatele
%tip - 0 complet
%      1 cu derivate secunde
%      2 natural
%      3 not a knot (deBoor)
%der - informatii despre derivate
%      [f'(a),f'(b)] pentru tipul 0
%      [f''(a), f''(b)] pentru tipul 1

if (nargin<4) | (tip==2), der=[0,0]; end

n=length(x);
%sortare noduri
if any(diff(x)<0), [x,ind]=sort(x); else, ind=1:n; end
y=f(ind); x=x(:); y=y(:);
%obtin ecuatiile 2 ... n-1
dx=diff(x); ddiv=diff(y)./dx;
ds=dx(1:end-1); dd=dx(2:end);
dp=2*(ds+dd);
md=3*(dd.*ddiv(1:end-1)+ds.*ddiv(2:end));
%tratare diferențiata tip - ecuatiile 1,n
switch tip
case 0 %complet
    dp1=1; dpn=1; vd1=0; vdn=0;
    md1=der(1); mdn=der(2);
case 1,2 %d2 si natural
    dp1=2; dpn=2; vd1=1; vdn=1;
    md1=3*ddiv(1)-0.5*dx(1)*der(1);
    mdn=3*ddiv(end)+0.5*dx(end)*der(2);
case 3 %deBoor
    x31=x(3)-x(1); xn=x(n)-x(n-2);
    dp1=dx(2); dpn=dx(end-1);
    vd1=x31; vdn=xn;
    md1=((dx(1)+2*x31)*dx(2)*ddiv(1)+dx(1)^2*ddiv(2))/x31;
    mdn=(dx(end)^2*ddiv(end-1)+(2*xn+dx(end))*dx(end-1)*...
        ddiv(end))/xn;
end
%construiesc sistemul rar
dp=[dp1;dp;dpn]; dp1=[0;vd1;dd];
dm1=[ds;vdn;0]; md=[md1;md;mdn];
A=spdiags([dm1,dp,dp1],-1:1,n,n);

```

```

m=A \ md;
d=y(1:end-1);
c=m(1:end-1);
a=[ (m(2:end)+m(1:end-1)-2*ddiv)./(dx.^2)];
b=[ (ddiv-m(1:end-1))./dx-dx.*a];

```

Calculul valorilor funcțiilor spline se poate face cu o singură funcție de evaluare pentru toate tipurile:

```

function z=valspline(x,a,b,c,d,t)
%evaluare spline
%apel z=valspline(x,a,b,c,d,t)
%z - valorile
%t - punctele in care se face evaluare
%x - nodurile
%a,b,c,d - coeficientii
n=length(x);
x=x(:); t=t(:);
k = ones(size(t));
for j = 2:n-1
    k(x(j) <= t) = j;
end
% Evaluare interpolant
s = t - x(k);
z = d(k) + s.* (c(k) + s.* (b(k) + s.* a(k)));

```

### 5.5.3. Proprietăți de minimalitate ale funcțiilor spline cubice

Funcțiile spline cubice complete și naturale au proprietăți interesante de optimalitate. Pentru a le formula, este convenabil să considerăm nu numai subdiviziunea  $\Delta$  ci și

$$\Delta' : a = x_0 = x_1 < x_2 < x_3 < \dots < x_{n-1} < x_n = x_{n+1} = b, \quad (5.5.21)$$

în care capetele sunt noduri duble. Aceasta înseamnă că ori de câte ori interpolăm pe  $\Delta'$ , interpolăm valorile funcției pe punctele interioare, iar la capete valorile funcției și ale derivatei. Prima teoremă se referă la funcții spline cubice complete  $s_{compl}(f; \cdot)$ .

**Teorema 5.5.1.** *Pentru orice funcție  $g \in C^2[a, b]$  care interpolează  $f$  pe  $\Delta'$ , are loc*

$$\int_a^b [g''(x)]^2 dx \geq \int_a^b [s''_{compl}(f; x)]^2 dx, \quad (5.5.22)$$

cu egalitate dacă și numai dacă  $g(\cdot) = s_{compl}(f; \cdot)$ .

**Observația 5.5.2.**  $s_{compl}(f; \cdot)$  din teorema 5.5.1 interpolează  $f$  pe  $\Delta'$  și dintre toți interpolanții de acest tip, derivata sa de ordinul II are normă minimă.  $\diamond$

*Demonstrație.* Folosim notația prescurtată  $s_{compl} = s$ . Teorema rezultă imediat, dacă arătăm că

$$\int_a^b [g''(x)]^2 dx = \int_a^b [g''(x) - s''(x)]^2 dx + \int_a^b [s''(x)]^2 dx. \quad (5.5.23)$$

Aceasta implică imediat (5.5.22) și faptul că egalitatea în (5.5.22) are loc dacă și numai dacă  $g''(x) - s''(x) \equiv 0$ , din care integrând de două ori de la  $a$  la  $x$  și utilizând proprietățile de interpolare ale lui  $s$  și  $g$  în  $x = a$  se obține  $g(x) = s(x)$ . Relația (5.5.23) este echivalentă cu

$$\int_a^b s''(x)[g''(x) - s''(x)]dx = 0. \quad (5.5.24)$$

Integrând prin părți și ținând cont că  $s'(b) = g'(b) = f'(b)$  și  $s'(a) = g'(a) = f'(a)$  se obține

$$\begin{aligned} & \int_a^b s''(x)[g''(x) - s''(x)]dx = \\ & = s''(x)[g'(x) - s'(x)] \Big|_a^b - \int_a^b s'''(x)[g'(x) - s'(x)]dx = \\ & = - \int_a^b s'''(x)[g'(x) - s'(x)]dx. \end{aligned} \quad (5.5.25)$$

Deoarece  $s'''$  este constantă pe porțiuni

$$\begin{aligned} & \int_a^b s'''(x)[g'(x) - s'(x)]dx = \sum_{\nu=1}^{n-1} s'''(x_\nu + 0) \int_{x_\nu}^{x_{\nu+1}} [g'(x) - s'(x)]dx = \\ & = \sum_{\nu=1}^{n-1} s'''(x_{\nu+0})[g(x_{\nu+1}) - s(x_{\nu+1}) - (g(x_\nu) - s(x_\nu))] = 0, \end{aligned}$$

căci atât  $s$  cât și  $g$  interpolează  $f$  pe  $\Delta$ . Aceasta demonstrează (5.5.24) și deci și teorema.  $\square$

Pentru interpolarea pe  $\Delta$  calitatea de a fi optimal revine funcțiilor spline naturale de interpolare  $s_{nat}(f; \cdot)$ .

**Teorema 5.5.3.** Pentru orice funcție  $g \in C^2[a, b]$  ce interpolează  $f$  pe  $\Delta$ , are loc

$$\int_a^b [g''(x)]^2 dx \geq \int_a^b [s''_{nat}(f; x)]^2 dx \quad (5.5.26)$$

cu egalitate dacă și numai dacă  $g(\cdot) = s_{nat}(f; \cdot)$ .

Demonstrația este analoagă cu a teoremei 5.5.1, deoarece (5.5.25) are loc din nou căci  $s''(b) = s''(a) = 0$ .

Punând  $g(\cdot) = s_{compl}(f; \cdot)$  în teorema 5.5.3 se obține

$$\int_a^b [s''_{compl}(f; x)]^2 dx \geq \int_a^b [s''_{nat}(f; x)]^2 dx. \quad (5.5.27)$$

Deci, într-un anumit sens, spline-ul cubic natural este cel mai neted interpolant.

Proprietatea exprimată în teorema 5.5.3 stă la originea numelui de spline. Un spline este o vergea flexibilă folosită pentru a desena curbe. Dacă forma sa este dată de ecuația  $y = g(x)$ ,  $x \in [a, b]$  și dacă spline-ul trebuie să treacă prin punctele  $(x_i, g_i)$ , atunci se presupune că spline-ul are o formă ce minimizează energia potențială

$$\int_a^b \frac{[g''(x)]^2 dx}{(1 + [g'(x)]^2)^3},$$

pentru toate funcțiile  $g$  supuse acelorași restricții. Pentru variații lente ale lui  $g$  ( $\|g'\|_\infty \ll 1$ ) aceasta aproximează bine proprietatea de minim din teorema 5.5.3.

## 5.6. Interpolare în MATLAB

MATLAB are funcții pentru interpolare în una, două sau mai multe dimensiuni. Funcția `polyfit` returnează coeficienții polinomului de interpolare Lagrange dacă gradul  $n$  este egal cu numărul de observații minus 1.

Funcția `interp1` acceptă perechi de date  $x(i)$ ,  $y(i)$  și un vector  $xi$  al punctelor în care se face evaluarea. Ea construiește interpolantul corespunzător datelor  $x$  și  $y$  și returnează valorile interpolantului în punctele din  $xi$ :

```
yi = interp1(x,y,xi,metoda)
```

Elementele vectorului  $x$  trebuie să fie ordonate crescător. Se admit patru tipuri de interpolanți, precizate de parametrul `metoda`, care poate avea una din următoarele valori

- 'nearest' - interpolare bazată pe vecinul cel mai apropiat;
- 'linear' - interpolare liniară pe porțiuni (metoda implicită);
- 'spline' - interpolare cu spline cubice;
- 'cubic' sau 'pchip' - interpolare Hermite cubică pe porțiuni.

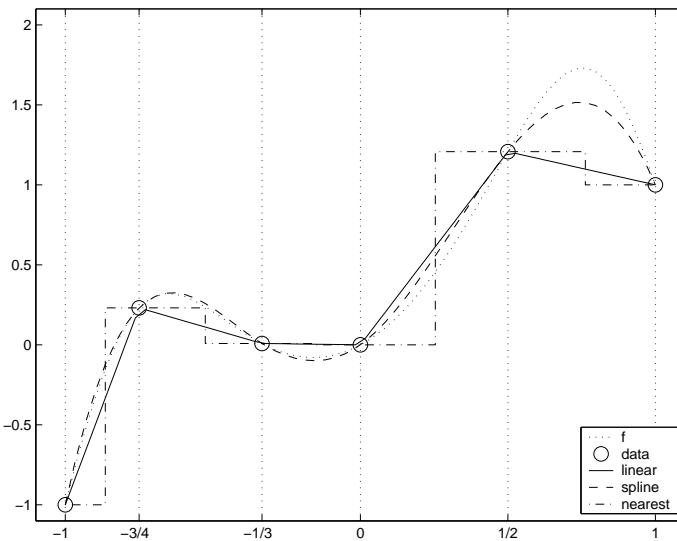
Exemplul de mai jos ilustrează funcționarea lui `interp1` (fișierul `exinterp1.m`).

```
x=[-1,-3/4, -1/3, 0, 1/2, 1]; y=x+sin(pi*x.^2);
xi=linspace(-1,1,60); yi=xi+sin(pi*xi.^2);
yn=interp1(x,y,xi,'nearest');
yl=interp1(x,y,xi,'linear');
ys=interp1(x,y,xi,'spline');
%yc=interp1(x,y,xi,'pchip');
plot(xi,yi,:',x,y,'o','MarkerSize',12); hold on
plot(xi,yl,'--',xi,ys,'-')
stairs(xi,yn,'.-')
set(gca,'XTick',x);
set(gca,'XTickLabel','|-1|-3/4|-1/3|0|1/2|1')
set(gca,'XGrid','on')
axis([-1.1, 1.1, -1.1, 2.1])
legend('f','data','linear', 'spline', 'nearest',4)
hold off
```

Exemplul alege șase puncte de pe graficul lui  $f(x) = x + \sin \pi x^2$  și calculează interpolanții `nearest`, `linear` și `spline`. Nu am inclus în acest exemplu `cubic` deoarece graficul obținut este foarte apropiat de cel obținut cu `spline` și s-ar fi încărcat figura. Graficul apare în figura 5.18.

Interpolarea `spline` este cea mai netedă, dar interpolarea Hermite pe porțiuni păstrează alura. Vom încerca să ilustrăm diferența dintre interpolarea `spline` și interpolarea Hermite pe porțiuni prin exemplul următor (`expсли_cub.m`).

```
x = [-0.99, -0.76, -0.48, -0.18, 0.07, 0.2, ...
       0.46, 0.7, 0.84, 1.09, 1.45];
y = [0.39, 1.1, 0.61, -0.02, -0.33, 0.65, ...
       1.13, 1.46, 1.07, 1.2, 0.3];
plot(x,y,'o'); hold on
xi=linspace(min(x),max(x),100);
ys=interp1(x,y,xi,'spline');
yc=interp1(x,y,xi,'cubic');
```

Figura 5.18: Exemplu de interpolare cu `interp1`

```

h=plot(xi,ys,'-',xi,yc,'-.');
legend(h,'spline','cubic',4)
axis([-1.1,1.6,-0.8,1.6])

```

Figura 5.19 dă graficul astfel obținut.

Interpolarea spline și Hermite cubică pe portiuni se pot realiza și direct, apelând funcția `spline` și respectiv `pchip`.

Dându-se vectorii `x` și `y`, comanda `yy = spline(x,y,xx)` returnează în vectorul `yy` valorile spline-ului în punctele din `xx`. Dacă `y` este o matrice, se consideră că avem de-a face cu valori vectoriale și interpolarea se face după coloanele lui `y`; dimensiunea lui `yy` este `length(xx)` pe `size(y,2)`. Funcția `spline` calculează interpolantul spline de tip deBoor. Dacă `y` conține cu două valori mai multe decât `x`, se calculează interpolantul spline complet, iar prima și ultima valoare din `y` se consideră a fi derivatele în capete. (Pentru terminologia privind tipurile de spline vezi secțiunea 5.5.2).

Exemplul pe care îl dăm în continuare ia șase puncte de pe graficul lui  $y = \sin(x)$ , calculează și reprezintă grafic spline-ul deBoor și cel complet (vezi figura 5.20).

```

x = 0:2:10;
y = sin(x); yc=[cos(0),y,cos(10)];
xx = 0:.01:10;
yy = spline(x,y,xx);
yc = spline(x,yc,xx);
plot(x,y,'o',xx,sin(xx),'-',xx,yy,'--',xx,yc,'-')
axis([-0.5,10.5,-1.3,1.3])
legend('noduri','sin','deBoor','complet',4)

```

Sunt situații în care este convenabil să se lucreze cu coeficienții funcției spline (de exemplu dacă nodurile se păstrează și `xx` se modifică). Comanda `pp=spline(x,y)` memorează coeficienții într-o structură `pp` (piecewise polynomial) care conține forma, nodurile, matricea coeficienților (cu 4 coloane

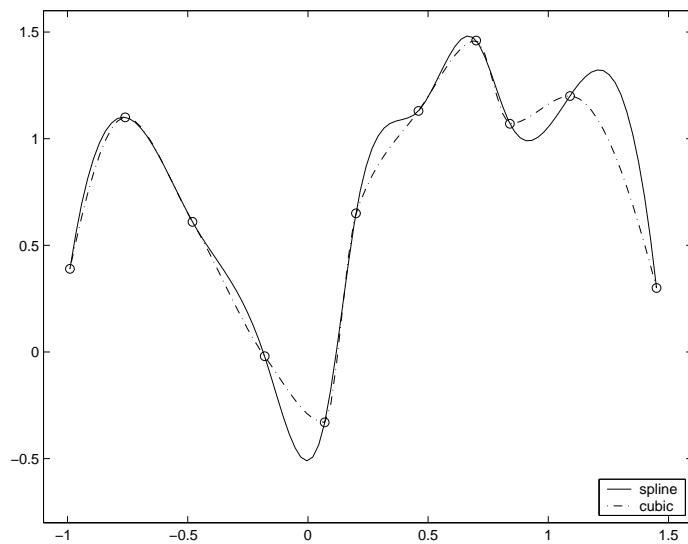


Figura 5.19: Interpolare cubică spline și Hermite pe porțiuni

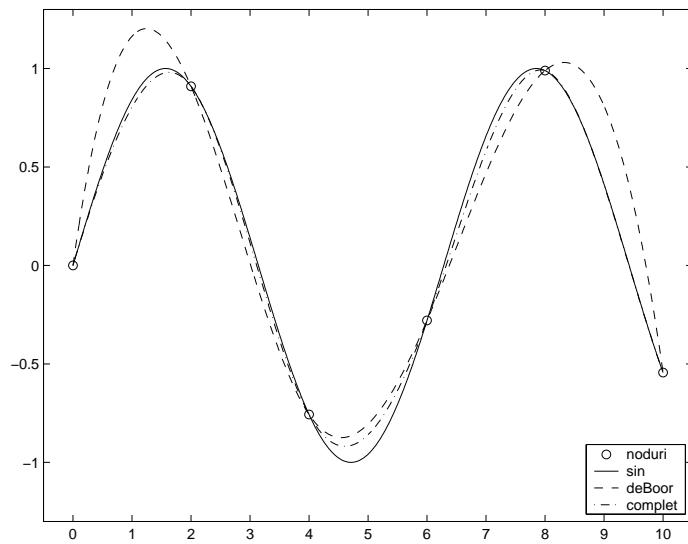


Figura 5.20: Spline deBoor și complet

pentru spline cubice), numărul de subintervale, ordinul (gradul plus 1) și dimensiunea. Funcția `ppval` evaluează spline-ul folosind o astfel de structură. Alte prelucrări de nivel inferior se pot realiza cu `mkpp` (construcția unei structuri `pp`) și `unmkpp` (detalii despre componentele unei structuri `pp`). De exemplu, comanda `yy = spline(x, y, xx)` se poate înlocui cu secvența

```
pp = spline(x, y);
yy = ppval(pp, xx);
```

Vom da acum un exemplu de interpolare spline cu date vectoriale și utilizare `ppval`. Dorim să citim interactiv mai multe puncte de pe ecran și să reprezentăm grafic spline-ul parametric care trece prin aceste puncte, cu două rezoluții diferite (să zicem cu 20 și 150 de puncte intermediare pe curbă). Sursa este conținută în script-ul `splinevect.m` și o dăm în continuare:

```
axis([0,1,0,1]);
hold on
[x,y]=ginput;
data=[x';y'];
t=linspace(0,1,length(x));
tt1=linspace(0,1,20);
tt2=linspace(0,1,150);
pp=spline(t,data);
yy1=ppval(pp,tt1);
yy2=ppval(pp,tt2);
plot(x,y,'o',yy1(1,:),yy1(2,:),yy2(1,:),yy2(2,:));
hold off
```

Citirea punctelor se face interactiv cu `ginput`. Coeficienții spline-ului se calculează o singură dată, iar pentru evaluarea spline-ului se folosește `ppval`. Propunem cititorului să încerce acest exemplu.

Funcția `pchip` se poate apela în una din formele:

```
yi = pchip(x, y, xx)
pp = pchip(x, y)
```

Prima formă returnează valori ale interpolantului în `xx`, iar a doua o structură `pp`. Semnificația parametrilor este aceeași ca în cazul funcției `spline`. Exemplul următor calculează interpolantul spline deBoor și Hermite cubic pe porțiuni pentru un același set de date (script-ul `expchip.m`):

```
x = -3:3;
y = [-1 -1 -1 0 1 1 1];
t = -3:.01:3;
p = pchip(x,y,t);
s = spline(x,y,t);
plot(x,y,'o',t,p,'-',t,s,'-.')
legend({'data','pchip','spline'},4)
```

Graficul apare în figura 5.21. Se observă din nou că interpolantul spline este mai neted, dar interpolantul Hermite cubic pe porțiuni păstrează alura.

## Probleme

**Problema 5.1.** (a) Dându-se o funcție  $f \in C[a, b]$ , să se determine  $\hat{s}_1(f; \cdot) \in \mathbb{S}_1^0(\Delta)$  astfel încât

$$\int_a^b [f(x) - \hat{s}_1(f; x)]^2 dx$$

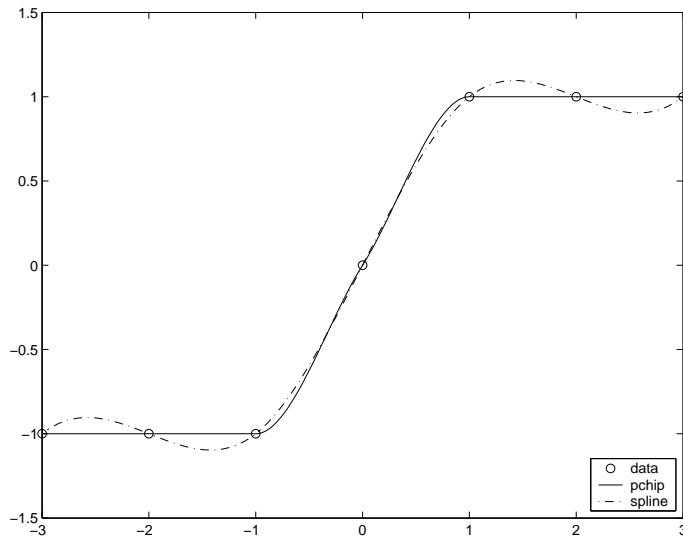


Figura 5.21: Exemplu de utilizare pchip și spline

să fie minimă.

- (b) Scrieți o funcție MATLAB care construiește și rezolvă sistemul de ecuații normale de la punctul (a).
- (c) Testați funcția de mai sus pentru o funcție și o diviziune alese de dumneavoastră.

**Problema 5.2.** Calculați aproximările discrete în sensul celor mai mici pătrate ale funcției  $f(t) = \sin\left(\frac{\pi}{2}t\right)$  pe  $0 \leq t \leq 1$  de forma

$$\varphi_n(t) = t + t(1-t) \sum_{j=1}^n c_j t^{j-1}, \quad n = 1(1)5,$$

utilizând  $N$  abscise  $t_k = k/(N+1)$ ,  $k = \overline{1, N}$ . De notat că  $\varphi_n(0) = 0$  și  $\varphi_n(1) = 1$  sunt valorile exacte ale lui  $f$  în  $t = 0$  și respectiv  $t = 1$ .

(Indicație: Aproximați  $f(t) - t$  printr-o combinație liniară a polinoamelor  $\pi_j(t) = t(1-t)t^{j-1}$ ,  $j = \overline{1, n}$ .) Sistemul de ecuații normale are forma  $Ac = b$ ,  $A = [(\pi_i, \pi_j)]$ ,  $b = [(\pi_i, f - t)]$ ,  $c = [c_j]$ .

Iesire (pentru  $n = 1, 2, \dots, 5$ ) :

- numărul de condiționare al sistemului;
- valorile coeficienților;
- eroarea maximă și minimă;

$$e_{\min} = \min_{1 \leq k \leq N} |\varphi_n(t_k) - f(t_k)|, \quad e_{\max} = \max_{1 \leq k \leq N} |\varphi_n(t_k) - f(t_k)|.$$

Executeați de două ori pentru

- (a)  $N = 5, 10, 20$ ,

(b)  $N = 4$ .

Comentați rezultatele. Reprezentați pe același grafic funcția și aproximantele.

**Problema 5.3.** Să se reprezinte pe același grafic pentru  $[a, b] = [0, 1]$ ,  $n = 11$ , funcția, interpolantul Lagrange și cel Hermite cu noduri duble în cazurile:

$$(a) \quad x_i = \frac{i-1}{n-1}, \quad i = \overline{1, n}, \quad f(x) = e^{-x} \text{ și } f(x) = x^{5/2};$$

$$(b) \quad x_i = \left(\frac{i-1}{n-1}\right)^2, \quad i = \overline{1, n}, \quad f(x) = x^{5/2}.$$

**Problema 5.4.** Aceeași problemă, dar pentru cele patru tipuri de interpolanți spline cubici.

**Problema 5.5.** Alegând diverse valori ale lui  $n$ , pentru fiecare  $n$  astfel ales reprezentați grafic funcția lui Lebesgue pentru  $n$  noduri echidistante și respectiv  $n$  noduri Cebîșev din intervalul  $[0, 1]$ .

**Problema 5.6.** Fie punctele  $P_i \in \mathbb{R}^2$ ,  $i = 0, n$ . Să se scrie:

- (a) o funcție MATLAB care determină o curbă parametrică polinomială de grad  $n$  ce trece prin punctele date;
- (b) o funcție MATLAB care determină o curbă parametrică spline cubic ce trece prin punctele date, folosind funcția pentru spline-ul natural sau cea pentru spline-ul deBoor date în acest capitol.

Testați cele două funcții citind interactiv punctele cu `ginput` și reprezentând apoi grafic punctele și cele două curbe astfel determinate.

**Problema 5.7.** Să se determine o cubică parametrică care trece prin două puncte date și are în acele puncte vectori tangenți dați.

**Problema 5.8.** Scrieți o funcție MATLAB care calculează coeficienții și valoarea splinelor cubice de tip Hermite, adică spline cubice de clasă  $C^1[a, b]$  care verifică

$$s_3(f, x_i) = f(x_i), \quad s'_3(f, x_i) = f'(x_i), \quad i = \overline{1, n}.$$

Reprezentați pe același grafic funcția  $f(x) = e^{-x^2}$  și interpolantul corespunzător pentru 5 noduri echidistante și 5 noduri Cebîșev pe  $[0, 1]$ .

**Problema 5.9.** Implementați o funcție MATLAB care calculează inversa matricei Vandermonde, folosind rezultatele de la paginile 189–192.

**Problema 5.10.** [46] Scrieți o funcție MATLAB pentru calculul coeficienților unui spline periodic de clasă  $C^2[a, b]$ . Aceasta înseamnă că datele trebuie să verifice  $f_n = f_1$  și că interpolantul rezultat trebuie să fie periodic, de perioadă  $x_n - x_1$ . Condițiile de periodicitate de la capete se pot impune mai ușor considerând două puncte suplimentare  $x_0 = x_1 - \Delta x_{n-1}$  și  $x_{n+1} = x_n + \Delta x_1$ , în care funcția să ia valorile  $f_0 = f_{n-1}$  și respectiv  $f_{n+1} = f_2$ .

**Problema 5.11.** Considerăm datele

```
x = -5:5;
y = [0,0,0,1,1,1,0,0,0,0,0];
```

Să se determine coeficienții aproximantei polinomiale de grad 7 în sensul celor mai mici pătrate corespunzătoare și să se reprezinte pe același grafic aproximanta și polinomul de interpolare Lagrange.

**Problema 5.12.** Densitatea sodiului sodiului (în  $\text{kg/m}^3$ ) pentru trei temperaturi (în  $^\circ\text{C}$ ) este dată în tabela

Temperatura	$T_i$	94	205	371
Densitatea	$\rho_i$	929	902	860

- (a) Obțineți polinomul de interpolare Lagrange corespunzător acestor date, folosind toolbox-ul Symbolic.
- (b) Determinați densitatea pentru  $T = 251^\circ$  prin interpolare Lagrange.

**Problema 5.13.** Aproximați

$$y = \frac{1+x}{1+2x+3x^2}$$

pentru  $x \in [0, 5]$  folosind interpolarea Lagrange, Hermite și spline. Alegeți cinci noduri și reprezentați pe același grafic funcția și interpolanții. Reprezentați apoi erorile de aproximare.

**Problema 5.14.** Tabela 5.3 dă valorile pentru o proprietate a titanului ca funcție de temperatură  $T$ . Determinați și reprezentați grafic o funcție spline cubică pentru aceste date utilizând 15 noduri. Cât de

$T$	605	645	685	725	765	795	825
$C(T)$	0.622	0.639	0.655	0.668	0.679	0.694	0.730
$T$	845	855	865	875	885	895	905
$C(T)$	0.812	0.907	1.044	1.336	1.181	2.169	2.075
$T$	915	925	935	955	975	1015	1065
$C(T)$	1.598	1.211	0.916	0.672	0.615	0.603	0.601

Tabela 5.3: O proprietate a titanului în funcție de temperatură

bine aproximează spline-ul datele în celelalte 6 puncte?

**Problema 5.15.** Determinați o aproximare discretă în sensul celor mai mici pătrate de forma

$$y = \alpha \exp(\beta x)$$

pentru datele

$x$	$y$
0.0129	9.5600
0.0247	8.1845
0.0530	5.2616
0.1550	2.7917
0.3010	2.2611
0.4710	1.7340
0.8020	1.2370
1.2700	1.0674
1.4300	1.1171
2.4600	0.7620

Reprezentați grafic punctele și aproximanta.

*Indicație:* logaritmați.

**Problema 5.16.** Implementați în MATLAB aproximanta discretă cu polinoame Cebîșev de speță I bazată pe produsul scalar (5.1.43).

**Problema 5.17.** Determinați o aproximare discretă în sensul celor mai mici pătrate de forma

$$y = c_1 + c_2x + c_3 \sin(\pi x) + c_4 \sin(2\pi x)$$

pentru datele

$i$	$x_i$	$y_i$
1	0.1	0.0000
2	0.2	2.1220
3	0.3	3.0244
4	0.4	3.2568
5	0.5	3.1399
6	0.6	2.8579
7	0.7	2.5140
8	0.8	2.1639
9	0.9	1.8358

Reprezentați grafic datele și aproximanta.

# CAPITOLUL 6

## Aproximare uniformă

Aproximarea uniformă a funcțiilor are două aspecte fundamentale:

- aproximarea unei funcții  $f$  prin funcții ce converg uniform către  $f$  pe domeniul considerat;
- cea mai bună aproximare uniformă a lui  $f$  prin funcții dintr-o mulțime dată.

Vom pune accentul pe studiul primei probleme.

Fie  $\mathcal{B}$  o clasă de funcții definite pe un interval  $[a, b] \subset \mathbb{R}$  și  $\mathcal{A} \subset \mathcal{B}$ . Pentru  $f \in \mathcal{B}$  și  $\varepsilon \in \mathbb{R}_+$  date, se pune problema determinării unei funcții  $g \in \mathcal{A}$  astfel încât

$$|f(x) - g(x)| < \varepsilon, \quad x \in [a, b].$$

Această problemă își are originea în binecunoscuta teoremă de aproximare uniformă a lui Weierstrass<sup>1</sup>.

**Teorema 6.0.1 (Weierstrass 1885).** Fie  $(C[a, b], \|\cdot\|)$ . Subspațiul  $\mathbb{P} \leq C[a, b]$  al polinoamelor de grad arbitrar este dens în  $C[a, b]$  ( $\mathbb{P} = C[a, b]$ ), adică  $\forall f \in C[a, b] \forall \varepsilon \in \mathbb{R}_+ \exists p \in \mathbb{P}$  astfel încât  $|f(x) - p(x)| < \varepsilon, \forall x \in [a, b]$ . (Echivalent  $\exists (p_n) \subset \mathbb{P}$  astfel încât  $p_n \rightrightarrows f$ .)

O importanță deosebită în studiul aproximării uniforme a funcțiilor a avut problema pusă de E. Borel în 1905. Deoarece sirul polinoamelor de interpolare Lagrange  $(L_m f)$  nu converge către  $f$ , el nu poate fi

Karl Theodor Wilhelm Weierstrass (1813-1897), matematician german, considerat a fi unul dintre părinții analizei moderne. A avut contribuții importante în teoria funcțiilor de variabile reale, funcții eliptice, calcul variațional, studiul formelor biliniare și pătratice.



folosit la demonstrația teoremei lui Weierstrass. Din acest motiv Borel a propus căutarea unor procedee de interpolare mai generale, care să permită contruirea unor șiruri de polinoame  $(P_n f)$  ce converg uniform către  $f$ , dacă  $f \in C[a, b]$ . Astfel de procedee ar permite să se dea demonstrații constructive ale teoremei lui Weierstrass. Una dintre primele soluții pentru problema lui Borel a fost dată de Sergiu N. Bernstein în 1912 prin introducerea polinoamelor care încearcă să numească.

## 6.1. Polinoamele lui Bernstein

**Definiția 6.1.1.** Fie  $f : [0, 1] \rightarrow \mathbb{R}$ . Operatorul  $B_m$  definit prin relația

$$(B_m f)(x) = \sum_{k=0}^m p_{mk}(x) f\left(\frac{k}{m}\right), \quad (6.1.1)$$

unde

$$p_{m,k}(x) = \binom{m}{k} x^k (1-x)^{m-k} \quad (6.1.2)$$

se numește operatorul lui Bernstein, iar polinomul  $B_m f$  se numește polinomul lui Bernstein.

În figura 6.1 apar polinoamele de bază Bernstein pentru  $k = 3$ , iar în figura 6.2 polinoamele Bernstein pentru funcția  $f(x) = \sin 2\pi x$  și  $m = 10, 15, 100$ .

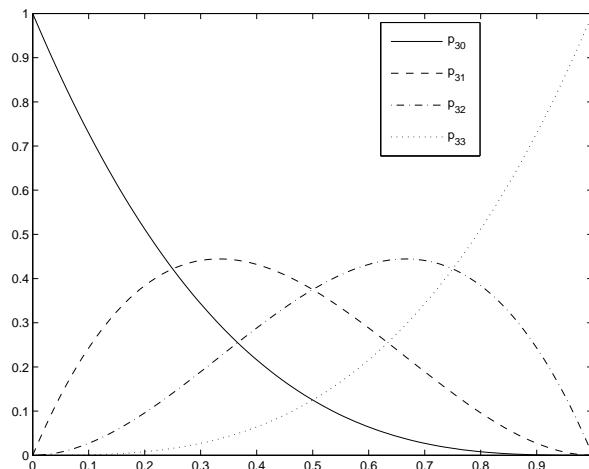


Figura 6.1: Polinoamele de bază Bernstein pentru  $k = 3$

Polinomul lui Bernstein poate fi obținut și pe cale probabilistică astfel: dacă  $f$  este mărginită pe  $[0, 1]$ , se consideră variabila aleatoare  $X$  cu distribuția

$$X : \begin{pmatrix} f\left(\frac{k}{m}\right) \\ p_{m,k}(x) \end{pmatrix},$$

a cărei valoare medie  $M(X) = \sum_{k=0}^m p_{m,k}(x) f\left(\frac{k}{m}\right)$  este chiar  $(B_m f)(x)$ .

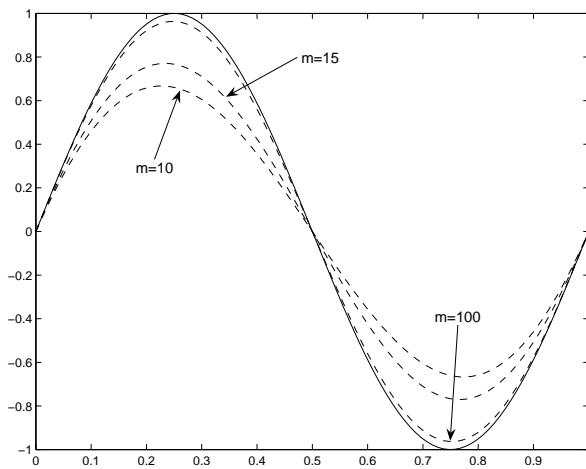


Figura 6.2: Polinoamele Bernstein (linie întreruptă) pentru funcția  $f(x) = \sin 2\pi x$  (linie continuă) și  $m = 10, 15, 100$

## Proprietăți

**Teorema 6.1.2.** 1.  $B_m$  este un operator liniar.

2.  $B_m$  este un operator pozitiv, adică

$$f(x) \geq 0 \Rightarrow (B_m f)(x) \geq 0, \forall x \in [0, 1].$$

3.  $B_m$  reproduce polinoamele până la gradul 1 inclusiv, adică

$$(B_m e_k)(x) = e_k(x), \quad k = 0, 1$$

și în plus

$$(B_m e_2)(x) = e_2(x) + \frac{x(1-x)}{m}.$$

4. Dacă  $m \leq f \leq M$  pe  $[0, 1]$ , atunci  $m \leq B_m f \leq M$  pe  $[0, 1]$ .

5. Dacă  $f \in C[0, 1]$ , atunci  $B_m f \rightrightarrows f$  pe  $[0, 1]$  când  $m \rightarrow \infty$ .

*Demonstratie.* 1. Rezultă imediat din definiție.

2. Imediat, înănd cont că  $\forall x \in [0, 1] p_m(x) \geq 0$ .

3. Se folosește binomul lui Newton

$$(u + v)^m = \sum_{k=0}^m \binom{m}{k} u^k v^{m-k}. \quad (6.1.3)$$

Diferențiem succesiv în raport cu  $u$  egalitatea de mai sus

$$u(u+v)^{m-1} = \sum_{k=0}^m \frac{k}{m} u^k v^{m-k} \binom{m}{k} \quad (6.1.4)$$

$$\left(1 - \frac{1}{m}\right) u^2 (u+v)^{m-2} = \sum_{k=0}^m \left[ \frac{k^2}{m^2} - \frac{k}{m^2} \right] \binom{m}{k} u^k v^{m-k}. \quad (6.1.5)$$

Punând în ultimele trei relații  $u = x$  și  $v = 1 - x$  vom obține

$$\begin{aligned} \sum_{k=0}^m p_{mk}(x) &= 1, & \sum_{k=0}^m \frac{k}{m} p_{mk}(x) &= x, \\ \sum_{k=0}^m \left(\frac{k}{m}\right)^2 p_{mk}(x) &= x^2 + \frac{x(1-x)}{m}. \end{aligned}$$

4. Din  $f - m \geq 0$ , prin pozitivitate, rezultă  $B_m(f - m) \geq 0$ . Analog și cealaltă inegalitate.
5.  $f \in C[0, 1] \Rightarrow f$  uniform continuă pe  $[0, 1]$ , adică  $\forall \varepsilon > 0 \exists \delta = \delta_\varepsilon$  astfel încât  $\forall x, x' \in [0, 1]$  cu  $|x - x'| < \delta$  să avem  $|f(x) - f(x')| < \frac{\varepsilon}{2}$ . Folosind a treia proprietate putem scrie

$$\begin{aligned} f(x) - (B_m f)(x) &= \sum_{k=0}^m p_{mk}(x) \left[ f(x) - f\left(\frac{k}{m}\right) \right] \text{ și} \\ |f(x) - (B_m f)(x)| &\leq \sum_{k=0}^m p_{mk}(x) \left| f(x) - f\left(\frac{k}{m}\right) \right| = \\ &= \sum_{k \in I_m} p_{mk}(x) \left| f(x) - f\left(\frac{k}{m}\right) \right| + \sum_{k \in J_m} p_{mk}(x) \left| f(x) - f\left(\frac{k}{m}\right) \right|, \end{aligned}$$

unde

$$I_m = \left\{ k : \left| \frac{k}{m} - x \right| < \delta \right\} \text{ iar } J_m = \left\{ k : \left| \frac{k}{m} - x \right| \geq \delta \right\}.$$

Dacă  $M = \max_{x \in [0, 1]} |f(x)|$  obținem în continuare

$$\begin{aligned} |f(x) - (B_m f)(x)| &\leq \frac{\varepsilon}{2} \sum_{k=0}^m p_{mk}(x) + 2M \sum_{k \in J_m} p_{mk}(x) \\ &\leq \frac{\varepsilon}{2} + 2M \sum_{k \in J_m} p_{mk}(x). \end{aligned}$$

Deoarece

$$\left| \frac{k}{m} - x \right| \geq \delta \Rightarrow \frac{\left( \frac{k}{m} - x \right)^2}{\delta^2} \geq 1,$$

obținem succesiv, folosind proprietatea 3

$$\begin{aligned} \sum_{k \in J_m} p_{mk}(x) &\leq \frac{1}{\delta^2} \sum_{k \in J_m} \left( \frac{k}{m} - x \right)^2 p_{mk}(x) \leq \frac{1}{\delta^2} \sum_{k=0}^m \left( \frac{k}{m} - x \right)^2 p_{mk}(x) \\ &= \frac{1}{\delta^2} \left( \underbrace{\sum_{k=0}^m \frac{k^2}{m^2} p_{mk}(x)}_{x^2 + \frac{x(1-x)}{m}} - 2x \underbrace{\sum_{k=0}^m \frac{k}{m} p_{mk}(x)}_x + x^2 \underbrace{\sum_{k=0}^m p_{mk}(x)}_1 \right) \\ &= \frac{1}{\delta^2} \frac{x(1-x)}{m} \leq \frac{1}{4m\delta^2}, \quad x \in [0, 1]. \end{aligned}$$

Deci,  $|f(x) - (B_m f)(x)| \leq \frac{\varepsilon}{2} + \frac{M}{2m\delta^2}$ , adică

$$|f(x) - (B_m f)(x)| \leq \varepsilon, \text{ dacă } m > \frac{M}{\varepsilon\delta^2}, \quad x \in [0, 1].$$

Această proprietate este o demonstrație constructivă a teoremei lui Weierstrass.

□

**Definiția 6.1.3.** Formula

$$f = B_m f + R_m f \tag{6.1.6}$$

se numește formula de aproximare a lui Bernstein, iar  $R_m f$  termenul rest.

**Teorema 6.1.4.** Dacă  $f \in C^2[0, 1]$ , atunci

$$(R_m f)(x) = -\frac{x(1-x)}{2m} f''(\xi), \quad \xi \in [0, 1] \tag{6.1.7}$$

și

$$|(R_m f)(x)| \leq \frac{1}{8m} \|f''\|_\infty. \tag{6.1.8}$$

*Demonstrație.* Conform teoremei 6.1.2, proprietățile 1 și 2,  $B_m f$  reproduce polinoamele de grad 1, adică  $B_m f = f$ , pentru orice  $f \in \mathbb{P}_1$ . Aplicând teorema lui Peano se obține

$$(R_m f)(x) = \int_0^1 K(x; t) f''(t) dt,$$

unde

$$K(x; t) = (x - t)_+ - \sum_{k=0}^m p_{mk}(x) \left( \frac{k}{m} - t \right)_+.$$

Cum  $K(x; t) \leq 0$  pentru  $x, t \in [0, 1]$  corolarul la teorema lui Peano ne conduce la

$$(R_m f)(x) = -\frac{x(1-x)}{2m} f''(\xi), \quad \xi \in [0, 1].$$

Inegalitatea (6.1.8) rezultă din faptul că  $x(1-x) \leq \frac{1}{4}$  pe  $[0, 1]$ . □

Importanță teoremei lui Weierstrass a făcut ca ea să fie mereu în atenția matematicienilor. S-au dat numeroase demonstrații constructive și generalizări. Una dintre ele este teorema lui Stone și Weierstrass.

Fie  $D$  un compact,  $C(D)$  spațiul funcțiilor definite pe compactul  $D$  înzestrat cu norma  $\|f\| = \max_D |f(x)|$ .  $C(D)$  este o algebră.

**Teorema 6.1.5 (Stone-Weierstrass, 1948).** *Dacă  $A$  este o subalgebră a lui  $C(D)$  care separă punctele lui  $D$ , atunci închiderea  $\bar{A}$  a lui  $A$  este fie  $C(D)$ , fie mulțimea funcțiilor continue care se anulează într-un punct al lui  $D$ .*

- Observația 6.1.6.**
1. Dacă  $A$  conține funcțiile constante, atunci  $\bar{A} = C(D)$ .
  2. O familie  $S$  de funcții definite pe  $E$  separă punctele lui  $E$  dacă  $\forall u, v \in E, u \neq v, \exists h \in S$  astfel încât  $h(u) \neq h(v)$ .  $\diamond$

Pentru detalii asupra demonstrațiilor teoremei lui Weierstrass și asupra teoremei Stone-Weierstrass a se vedea [67].

## 6.2. B-spline

### 6.2.1. Noțiuni și rezultate de bază

Fie  $m, n \in \mathbb{N}$  și punctele  $t_0, t_1, \dots, t_m$ , cu  $t_i \leq t_{i+1}$ . Fie  $[a, b] \subset \mathbb{R}$  astfel încât  $t_k \leq a$  și  $t_{n-k} \geq b$ . Pentru  $j = \overline{1, m-i}$  punem

$$\omega_{i,j}(x) = \begin{cases} \frac{x - t_i}{t_{i+j} - t_i}, & \text{dacă } t_i < t_{i+j} \\ 0, & \text{altfel.} \end{cases} \quad (6.2.1)$$

**Definiția 6.2.1.** Pentru  $i = \overline{0, m-k-1}$  funcțiile B-spline de rang  $i$  și grad  $k$  se definesc prin recurență astfel

$$B_{i,0}(x) = \begin{cases} 1, & \text{dacă } x \in [t_i, t_{i+1}], \\ 0, & \text{în caz contrar.} \end{cases} \quad (6.2.2)$$

$$B_{i,k}(x) = \omega_{i,k}(x)B_{i,k-1}(x) + (1 - \omega_{i+1,k}(x))B_{i+1,k-1}(x), \quad \text{pentru } k \geq 1. \quad (6.2.3)$$

**Propoziția 6.2.2.** Au loc următoarele proprietăți:

- (a)  $B_{i,k}$  este un polinom de grad  $k$  pe porțiuni;
- (b)  $B_{i,k}(x) = 0$  pentru  $x \notin [t_i, t_{i+k+1}]$ ;
- (c)  $B_{i,k}(x) > 0$  pentru  $x \in (t_i, t_{i+k+1})$ ;  $B_{i,k}(t_i) = 0$ , înafara cazului  $t_i = t_{i+1} = \dots = t_{i+k} < t_{i+k+1}$  (nod de ordin  $k+1$ ) când avem  $B_{i,k}(t_i) = 1$ .
- (d)

$$\sum_{i=0}^{m-k-1} B_{i,k}(x) = 1, \quad \forall x \in [a, b].$$

- (e) Fie  $x \in (t_i, t_{i+k+1})$ ;  $B_{i,k}(x) = 1 \iff t_{i+1} = \dots = t_{i+k} = x$ ;
- (f)  $B_{i,k}$  este continuă (și indefinitely derivabilă) la dreapta  $\forall x \in \mathbb{R}$  (precizăm că  $B_{i,k}(x) = 0$ ,  $\forall x \notin [t_0, t_m]$ ).

*Demonstrație.* (a), (b), (c), (d) și (f) sunt evidente pentru  $k = 0$ ; (a), (b), (c) și (f) pentru  $B_{i,k}$  se deduc prin recurență după  $k$ . Să demonstrăm (d). Fie  $x \in [a, b]$ ; există un  $j$ ,  $k \leq j \leq m - k - 1$  astfel încât  $x \in [t_j, t_{j+1})$ . Dacă  $x = t_j$  și  $B_{j,k}(x) = 1$  proprietatea este evidentă (conform (c)). În celelalte cazuri avem

$$\begin{aligned} \sum_{i=0}^{m-k-1} B_{i,k}(x) &= \sum_{i=j-k}^j B_{i,k}(x) \\ &= \sum_{i=j-k}^j [\omega_{i,k}(x) B_{i,k-1}(x) + (1 - \omega_{i+1,k}(x)) B_{i+1,k-1}(x)] \\ &= \omega_{j-k,k}(x) B_{j-k,k-1}(x) + \sum_{i=j+1-k}^j B_{i,k-1}(x) \\ &\quad + (1 - \omega_{j+1,k}(x)) B_{j+1,k-1}(x). \end{aligned}$$

Dar  $B_{j-k,k-1}(x) = 0$  și  $B_{j+1,k-1}(x) = 0$ , întrucât  $x \in [t_j, t_{j+1})$  (conform lui (b)) și

$$\sum_{i=j+1-k}^j B_{i,k-1}(x) = 1$$

(din ipoteza inducției).

(e) este o consecință imediată a lui (d) și (c).  $\square$

**Observația 6.2.3.** 1. Deoarece B-splinele sunt nenegative ((b) și (c)) și suma lor este 1, spunem că ele formează o *partiție a unității*.

2. Fiecare  $B_{i,k}$  are un suport compact unic.
3. În cazul când  $t_{m-k} = \dots = t_m = b$ , pentru ca (d) și (e) să fie adevărate pe întreg  $[a, b]$  punem  $B_{m-k-1,k}(b) = 1$ .
4. B-splinele se pot defini și pentru un sir infinit de noduri  $t_i$ , în definiția fiecărui  $B_{i,k}$  intervenind doar un număr finit de noduri.
5. Dacă  $t_i$  este un nod de multiplicitate  $\geq k + 2$  ( $t_i = t_{i+k+1}$ ) are loc  $B_{i,k} \equiv 0$ ; reciproca este de asemenea adevărată.  $\diamond$

**Propoziția 6.2.4 (Identitatea lui Marsden).** Fie  $n = m - k$  și

$$\Psi_{i,k}(t) = \begin{cases} (t_{i+1} - t) \dots (t_{i+k} - t), & \text{pentru } k \geq 1; \\ 1, & \text{pentru } k = 0. \end{cases} \quad (6.2.4)$$

Are loc relația

$$(x - t)^k = \sum_{i=0}^{n-1} \Psi_{i,k}(t) B_{i,k}(x). \quad (6.2.5)$$

*Demonstrație.* Se face prin inducție după  $k$ . Cazul  $k = 0$  este trivial (propoziția 6.2.2 (d)). Presupunem că

$$\sum_{i=0}^{n-1} \Psi_{i,k}(t) B_{i,k}(x) = (x - t)^{k-1}$$

este adevărată. Avem  $B_{0,k-1}(x) = 0$  pentru  $x \in [a, b]$ .

$$\begin{aligned}
& \sum_{i=0}^{n-1} \Psi_{i,k}(t) B_{i,k}(x) = \\
&= \sum_{i=0}^{n-1} \Psi_{i,k}(t) [\omega_{i,k}(x) B_{i,k-1}(x) + (1 - \omega_{i+1,k}(x)) B_{i+1,k-1}(x)] \\
&= \Psi_{0,k}(t) \omega_{0,k}(x) B_{0,k-1}(x) \\
&+ \sum_{i=1}^{n-1} B_{i,k-1}(x) [\Psi_{i,k}(t) \omega_{i,k}(x) + \Psi_{i-1,k}(t) (1 - \omega_{i,k}(x))].
\end{aligned}$$

Dacă  $t_i = t_{i+k}$ , atunci  $B_{i,k-1} \equiv 0$  și dacă  $t_i < t_{i+k}$  avem

$$\begin{aligned}
& \Psi_{i,k}(t) \omega_{i,k}(x) + \Psi_{i-1,k}(t) (1 - \omega_{i,k}(x)) = \\
&= \Psi_{i,k-1}(t) [(t_{i+k} - t) \omega_{i,k}(x) + (t_i - t) (1 - \omega_{i,k}(x))] \\
&= \Psi_{i,k-1}(t) (\omega_{i,k}(x) (t_{i+k} - t_i) + t_i - t) = \Psi_{i,k-1}(t) (x - t).
\end{aligned}$$

Deci

$$\sum_{i=0}^{n-1} \Psi_{i,k}(t) B_{i,k}(x) = (x - t) \sum_{i=1}^{n-1} \Psi_{i,k-1}(t) B_{i,k-1}(x) = (x - t)^k,$$

ultima egalitate având loc pe baza ipotezei inducției.  $\square$

**Observația 6.2.5.** Fie  $t = (t_0, \dots, t_m)$  și  $\mathbb{P}_{k,t}([a, b]) = \mathbb{P}_{k,t}$  spațiul funcțiilor polinomiale pe porțiuni pe  $[a, b]$  de grad  $\leq k$  și cu racord de clasă  $C^{k-p_j}$  în  $t_j$  dacă  $t_j$  este un nod de multiplicitate  $p_j$ . Prin convenție, un racord de clasă  $C^{k-p_j}$ , cu  $k - p_j < 0$  nu impune nici un fel de condiții asupra lui  $t_j$ . Dacă toate nodurile sunt de multiplicitate  $\leq k + 1$ , atunci funcțiile  $B_{i,k}$ ,  $i = \overline{0, n-1}$  formează o bază a lui  $\mathbb{P}_{k,t}$  și  $\dim \mathbb{P}_{k,t} = n = m - k$ . Dacă anumite noduri au multiplicitate  $\geq k + 2$ , atunci  $\mathbb{P}_{k,t} = \langle B_{i,k} | i = \overline{0, n-1} \rangle$ , dar  $\{B_{i,k}\}$  nu formează o bază.  $\diamond$

## 6.2.2. Algoritmul de evaluare a unui B-spline

Fie  $x \geq t_k$  și  $S(x) = \sum_{i=0}^{n-1} a_i B_{i,k}(x)$ . Aplicând direct definiția inductivă a B-splinelor se obține:

**Propoziția 6.2.6.** Are loc

$$S(x) = \sum a_i^{(0)}(x) B_{i,k}(x) = \sum a_i^{(1)}(x) B_{i,k-1}(x) = \dots = \sum a_i^{(k)}(x) B_{i,0}(x)$$

cu

$$\begin{aligned}
a_i^{(0)}(x) &= a_i \\
a_i^{(r+1)}(x) &= \omega_{i,k-r}(x) a_i^{(r)}(x) + (1 - \omega_{i,k-r}(x)) a_{i-1}^{(r)}(x).
\end{aligned}$$

**Observația 6.2.7.** 1. Algoritmul este valabil și pentru  $x < t_k$ , considerând că  $B_{i,k} \equiv 0$  pentru  $i \leq 0$ .

2. Dacă evaluăm  $S(x)$  pentru  $x \in [t_j, t_{j+1})$ , avem  $S(x) = a_j^k(x)$ , deoarece  $B_{j,0}(x)$  este funcția caracteristică a intervalului  $[t_j, t_{j+1})$  și pentru a calcula  $a_j^{(k)}(x)$  este suficient să evaluăm  $a_i^{(r)}$

numai pentru  $j - k + r - 1 < i \leq j$ , celelalte  $B_{i,k-r}(x)$  fiind nule pentru  $x \in [t_j, t_{j+1})$ . Avem aşadar

$$S(x) = \sum_{i=j-k}^j a_i^{(0)}(x) B_{i,k}(x) = \sum_{i=j-k+1}^j a_i^{(1)}(x) B_{i,k-1}(x) = \cdots = a_j^{(k)}(x). \quad \diamond$$

Algoritmul se prezintă, aşadar, sub formă triunghiulară, fiecare element obținându-se prin combinarea convexă a două elemente din linia anterioară:

$$\begin{array}{ccccccc} a_{j-k} & a_{j-k+1} & & \dots & a_{j-1} & a_j \\ a_{j-k+1}^{(1)} & & & & a_j^{(1)} & \\ \vdots & & \dots & & \vdots & \\ a_{j-1}^{(k-1)} & & a_{j-1}^{(k-1)} & & & \\ & a_j^{(k)} & & & & \end{array}$$

(Algoritmul Cox-DeBoor)

Nodurile  $t$  formează o diviziune

$$\Delta : t_0 \leq t_1 \leq \cdots \leq t_k \leq a \leq \cdots \leq b \leq t_n \leq t_{n+1} \leq t_{n+k}.$$

Vom presupune că multiplicitatea oricărui nod  $\leq k + 1$ . O alegere frecventă este

$$t_0 = t_1 = \cdots = t_k = a < t_{k+1} \leq \cdots \leq t_{n-1} < b = t_n = \cdots = t_{n+k}.$$

În cazul particular  $t_0 = \cdots = t_k = a < b = t_{k+1} = \cdots = t_{2k+1}$ , funcțiile B-spline se reduc la polinoamele fundamentale Bernstein

$$p_{i,k}(x) = \binom{k}{i} x^i (1-x)^{k-i}.$$

### 6.2.3. Aplicații în grafica pe calculator

#### Curbe B-spline și Bézier

Fie  $P_0, \dots, P_{n-1} \in \mathbb{R}^s$ ,  $s \in \mathbb{N}^*$ .

**Definiția 6.2.8.** (a) Se numește curbă B-spline asociată poligonului de control  $(P_0, \dots, P_{n-1})$  curba parametrică  $(\gamma)$  definită prin

$$(\gamma) \quad S(t) = \sum_{i=0}^{n-1} P_i B_{i,k}(t), \quad a \leq t \leq b.$$

(b) Se numește curbă Bézier<sup>2</sup> asociată poligonului de control  $(P_0, \dots, P_{k-1})$  curba parametrică

$$B(t) = \sum_{i=0}^k P_i p_{i,k}(t), \quad p_{i,k}(x) = \binom{k}{i} x^i (1-x)^{k-i}.$$

**Propoziția 6.2.9.** Curbele B-spline au următoarele proprietăți:

1.  $\gamma$  nu trece în general prin punctele  $P_i$ ; dacă  $t_0 = \dots = t_k$  și  $t_n = \dots = t_{n+k} = b$  avem  $S(a) = P_0$  și  $S(b) = P_{n-1}$ ; în acest caz  $\gamma$  este tangentă în  $P_0$  și  $P_{n-1}$  la laturile poligonului de control.
2.  $\gamma$  este în învelitoarea convexă a punctelor  $P_0, \dots, P_{n-1}$ . Mai exact dacă  $t_i \leq t \leq t_{i+1}$ ,  $S(t)$  este în învelitoarea convexă a punctelor  $P_{i-k}, \dots, P_i$ . Curba Bézier este situată în învelitoarea convexă a punctelor  $P_0, \dots, P_{n-1}$ .
3. În cazul când nodurile  $t_i$  ( $k+1 \leq i \leq n-1$ ) sunt simple,  $\gamma \in C^{k-1}$  și este formată din  $n$  arce parametrice polinomiale, de grad  $\leq k$ .
4.  $\gamma$  este invariantă la deplasările lui  $\mathbb{R}^s$ : dacă  $h$  este o deplasare a lui  $\mathbb{R}^s$ , atunci punctele  $h(P_i)$  sunt punctele de control ale curbei  $h(S(t))$ .
5.  $S(t)$  este regularizantă: dacă  $P$  este poligonul de control al lui  $\gamma$  și dacă  $H$  este un hiperplan, avem  $\text{card}(H \cap \gamma) \leq \text{card}(H \cap P)$ .

**Observația 6.2.10.** Curbele B-spline au un caracter local în sensul că:

- (a) Dacă  $X$  este un punct al curbei corespunzând unei valori  $t_0$  a parametrului, atunci poziția lui  $X$  nu depinde decât de cel mult  $k+1$  puncte  $P_i$ , deoarece dacă  $t_j \leq t_0 < t_{j+1}$  avem

$$X = S(t_0) = \sum_{i=0}^{n-1} P_i B_{i,k}(t_0) = \sum_{i=j-k}^k P_i B_{i,k}(t_0).$$

- (b) Fiecare  $P_j$  nu influențează curba  $S(t)$  decât pentru valorile lui  $t$  pentru care  $B_{j,k}(t) \neq 0$ , adică  $t_j \leq t < t_{j+k+1}$  (în particular dacă modificăm  $P_j$  numai o porțiune a curbei se modifică).
- (c) Caracterul local nu are loc în cazul curbelor Bézier: modificarea unui punct de control atrage modificarea întregii curbe. ◇

### Algoritmi pentru curbe B-spline

Algoritmul 6.1 de evaluare a curbelor B-spline este o simplă traducere a algoritmului de evaluare pentru funcții B-spline. Implementarea MATLAB apare în sursa 6.1.

În cazul particular al unei curbe Bézier se obține un algoritm mult mai simplu, care nu depinde de  $j$  (vezi algoritmul 6.2). El este datorat lui Paul de Casteljau. Implementarea MATLAB apare

Pierre Bezier(1910-1999) Fiul și nepot de ingineri, a studiat ingineria mecanică la Universitatea din Paris, unde în 1977 a obținut doctoratul în Matematică. Timp de 42 de ani (1933-1975) a lucrat la Uzinele Renault. A introdus curbele care îi poartă numele. Este considerat fondator al domeniului CAD/CAM, realizând primul sistem de acest tip (UNISURF - incepând cu 1960). Laureat al mai multor premii prestigioase, printre care „Steven Anson Coons” al ACM (Association for Computing Machinery).



**Algoritmul 6.1** Algoritmul Cox-deBoor pentru evaluarea unei curbe B-spline

**Intrare:** Poligonul de control  $P_i$ ,  $i = \overline{0, n-1}$ , punctul  $t$  în care se face evaluarea și diviziunea  $t_i$

**Ieșire:** Valoarea  $S(t)$

```

1: Fie  $S(t) = \sum_{i=0}^{n-1} P_i B_{i,k}(t)$  și să presupunem că  $t_j \leq t < t_{j+1}$ .
2: for  $i := j - k$  to  $j$  do
3:    $P_i^0(t) := P_i$ ;
4: end for
5: for  $r := 0$  to  $k - 1$  do
6:   for  $i := j - k + r + 1$  to  $j$  do
7:      $P_i^{r+1} := \omega_{i,k-r}(t)P_i^r + (1 - \omega_{i,k-r}(t))P_{i-1}^r$ 
8:      $= \frac{(t - t_i)P_i^r(t) + (t_{i+k-r} - t)P_{i-1}^r}{t_{i+k-r} - t_i};$ 
9:   end for
10: end for
11:  $S_j(t) := P_k^k(t);$ 
```

în sursa 6.2.

Dăm și două posibilități de alegere a punctelor divizionii.

**Varianta 1:**  $t_j = j$ ;

**Varianta 2:**

$$t_j = \begin{cases} 0, & 0 \leq j \leq k; \\ j - k, & k + 1 \leq j \leq n; \\ n - k, & j > n. \end{cases}$$

#### 6.2.4. Exemple

Să desenăm curba cuadratică corespunzând nodurilor  $t_0 = t_1 = t_2 = 1$ ,  $t_3 = 2$ ,  $t_4 = 3$ ,  $t_5 = 4$  și  $t_6 = t_7 = t_8 = 5$  și al cărei poligon de control este dat în figura 6.3. Graficul curbei apare de asemenea în figura 6.3.

Dacă se alege o repartiție uniformă a nodurilor și dorim o curbă de clasă  $C^1$ , putem alege  $t_i = i$ ,  $i \in \mathbb{Z}$ . Avem  $B_i(t) = B_{i+r}(t+r)$ ,  $\forall r \in \mathbb{Z}$  (figura 6.4), adică elementele bazei B-spline se deduc una din alta printr-o translație întreagă. Să considerăm  $S(t) = \sum_{i=-\infty}^{\infty} P_i B_{i,r}(t)$  și să punem  $P_{6+i} = P_i$ ; avem  $S(t-6) = S(t)$ , iar curba este definită pe un interval arbitrar de lungime 6. Se obține în acest mod o curbă închisă (figura 6.5).

În figura 6.6 se dau două curbe Bézier de grad 3 pentru două forme diferite ale poligonului de control.

Urmează acum un exemplu mai detaliat de aplicare a algoritmului Cox-deBoor. Fie nodurile  $a = 0 = t_0 = t_1 = t_2 = t_3$ ,  $t_4 = 1$ ,  $t_5 = 2$ ,  $b = 3 = t_6 = t_7 = t_8 = t_9$ . Gradul curbei va fi  $k = 3$ . Poligonul de control are 6 puncte și este dat în figura 6.7. Avem  $S(t) = \sum_{i=0}^5 P_i B_{i,3}(t)$ ; să calculăm  $S(\tau)$  pentru  $\tau = \frac{3}{2}$ . Deoarece  $t_4 < \tau < t_5$ , calculele se organizează tabelar sub forma

$$\begin{array}{ccccccc} P_1 & & P_2 & & P_3 & & P_4 \\ P_2^1 & & P_3^1 & & P_4^1 & & \\ P_3^2 & & P_4^2 & & & & \\ & & P_4^3 & & & & \end{array}$$

**Sursa MATLAB 6.1 Implementarea algoritmului Cox-deBoor pentru calculul B-splinelor**

```

function q=Cox_deBoor(grad,pc,knots,t)
%algoritmul Cox-DeBoor pentru B-spline
%apel q=Cox_deBoor(grad,pc,knots,t)
%grad- gradul spline-ului
%pc - poligonul de control
%knots - nodurile
%t - abscisele punctelor in care calculam spline-ul
n=length(knots);
knots=knots(:); t=t(:);
lt=length(t);
[lin,n]=size(pc);
%determin pozitie puncte in raport cu nodurile
j = ones(size(t));
for ll = grad+1:n
    j(knots(ll) <= t) = ll;
end
%aplicare efectiva algoritm
for l=1:lt
    qn=pc;
    for r=0:grad-1
        i=j(l)-grad+r+1:j(l);
        w=repmat(omegav(i,grad-r,knots,t(l)),lin,1);
        qn(:,i)=w.*qn(:,i)+(1-w).*qn(:,i-1);
    end
    q(:,l)=qn(:,j(l));
end
%-----
function u=omegav(i,k,knots,t)
u=zeros(size(i));
v1=find(knots(i+k) ~= knots(i));
u(v1)=(t-knots(i))./(knots(i+k)-knots(i));

```

**Algoritm 6.2 Algoritmul de Casteljau pentru evaluarea unei curbe Bézier**

**Intrare:** Poligonul de contrul  $P_i$ ,  $i = \overline{0, k}$  și punctul  $t$  în care se face evaluarea

**Ieșire:** Valoarea  $B(t)$

- 1: Fie  $B(t) = \sum_{i=0}^k P_i p_{i,k}(t)$ .
- 2: **for**  $i := 0$  **to**  $k$  **do**
- 3:    $P_i^0(t) := P_i$ ;
- 4: **end for**
- 5: **for**  $r := 0$  **to**  $k - 1$  **do**
- 6:   **for**  $i := r + 1$  **to**  $k$  **do**
- 7:      $P_i^{r+1} := (1 - t)P_{i-1}^r(t) + tP_i^r(t)$ ;
- 8:   **end for**
- 9: **end for**
- 10:  $B(t) := P_k^k(t)$ ;

**Sursa MATLAB 6.2 Implementarea algoritmului de Casteljau pentru curbe Bézier**

```
function q=deCasteljau(pc,t)
%algoritmul de Casteljau
%apel q=deCasteljau(pc,t)
%pc - punctele de control
%t punctele in care se evalueaza
[mp,k]=size(pc);
lt=length(t);
for l=1:lt
    qn=pc;
    for r=1:k
        i=r+1:k;
        qn(:,i)=(1-t(l))*qn(:,i-1)+t(l)*qn(:,i);
    end
    q(:,l)=qn(:,k);
end;
```

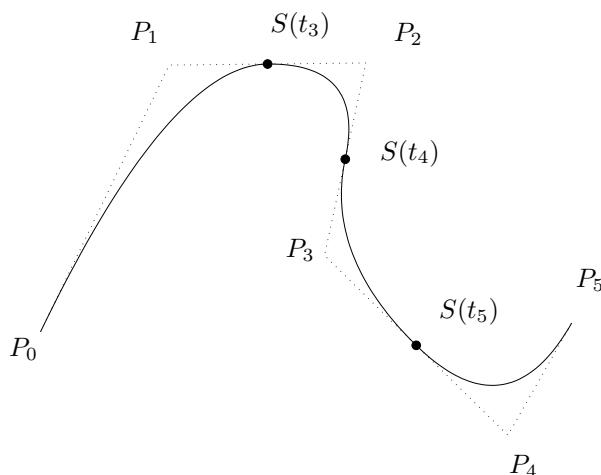


Figura 6.3: Curba B-spline cuadratică și poligonul ei de control

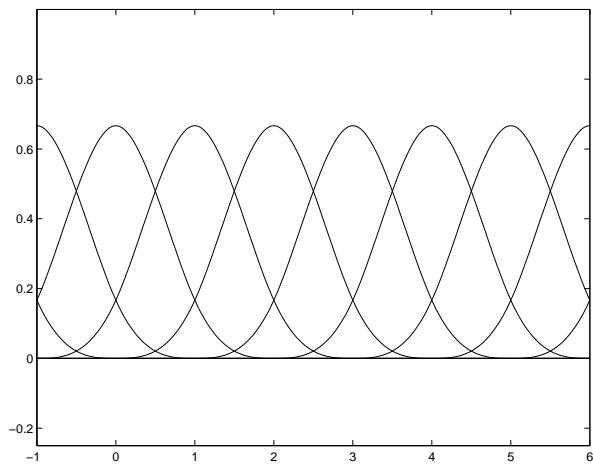
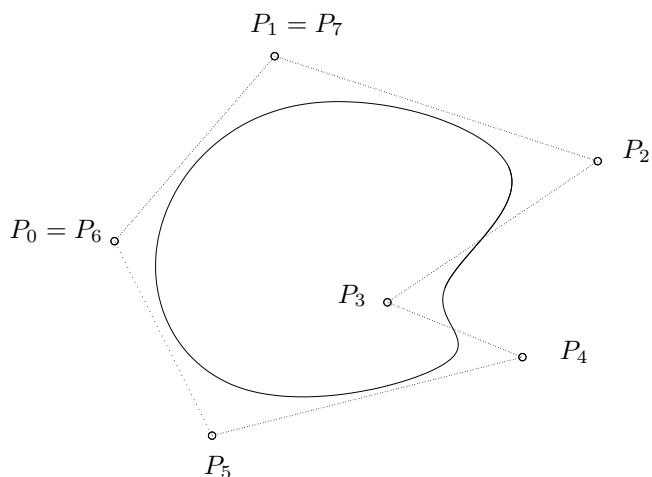
Figura 6.4: Bază pentru noduri din  $\mathbb{Z}$ 

Figura 6.5: Curbă B-spline închisă, periodică

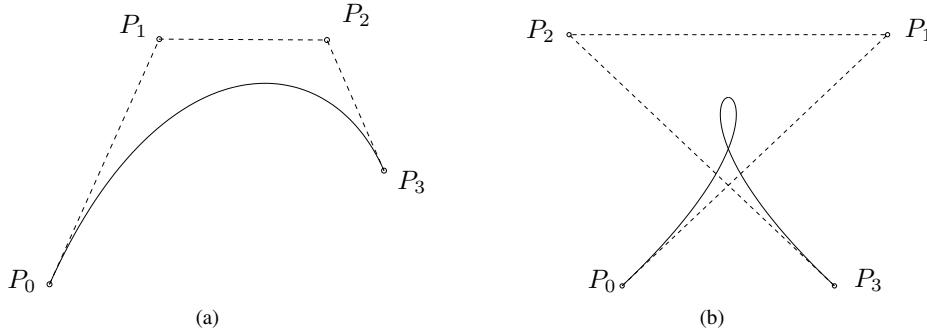


Figura 6.6: Două curbe Bézier de grad 3

unde  $P_4^3 = S\left(\frac{3}{2}\right)$ , iar

$$\begin{aligned} P_2^1 &= \frac{3}{4}P_2 + \frac{1}{4}P_1 & P_3^1 &= \frac{1}{2}P_1 + \frac{1}{2}P_2 & P_4^1 &= \frac{1}{4}P_4 + \frac{3}{4}P_3 \\ P_3^2 &= \frac{3}{4}P_3^1 + \frac{1}{4}P_2^1 & P_4^2 &= \frac{1}{4}P_4^1 + \frac{3}{4}P_3^1 \\ P_4^3 &= \frac{1}{2}P_4^2 + \frac{1}{2}P_3^2 \end{aligned}$$

Graficul curbei apare în figura 6.7. Se observă că  $S(t)$  este tangentă în punctul  $P_j^k(t)$  la segmentul  $P_{j-1}^{k-1}P_j^{k-1}$ . Să ilustrăm acum caracterul local al curbelor B-spline. În figura 6.8 este reprezentată o

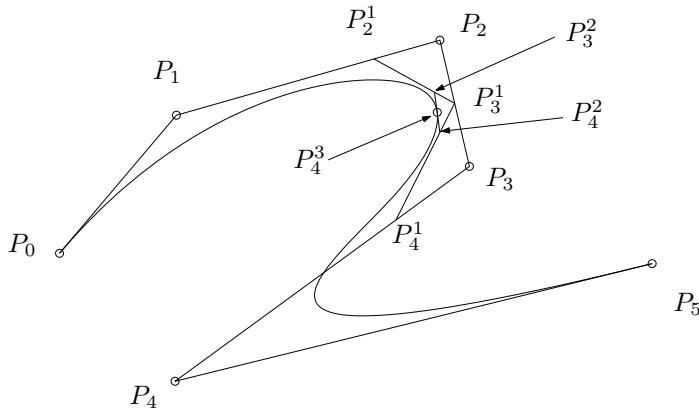


Figura 6.7: Un exemplu de aplicare a algoritmului Cox-deBoor

curbă B-spline de gradul 3 corespunzătoare unui poligon de control cu 9 puncte (linie continuă). Se modifică coordonatele punctului  $P_3$ , iar poligonul de control modificat și B-spline-ul corespunzător apar cu linie întreruptă.

Pentru detalii asupra curbelor și suprafețelor Bézier și B-spline și pentru aplicații ale acestora în CAGD (Computer Aided Geometric Design) a se vedea [19, 30, 51].

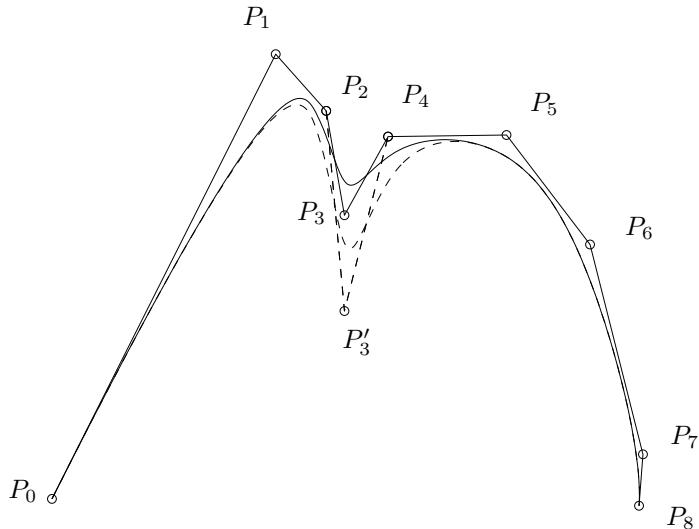


Figura 6.8: Caracterul local al curbelor B-spline. Se ilustrează efectul modificării coordonatelor unui punct ( $P_3$ )

### 6.3. Funcții spline cu variație diminuată

Funcțiile spline cu variație diminuată au fost introduse în 1964 de Isaac J. Schoenberg<sup>3</sup> ca o generalizare a polinoamelor Bernstein.

Fie diviziunea

$$\Delta : t_0 \leq t_1 \leq \dots \leq t_k \leq a \leq \dots \leq b \leq t_n \leq t_{n+1} \leq \dots \leq t_{n+k}.$$

Vom presupune că multiplicitatea oricărui nod nu depășește  $k + 1$ .

**Definiția 6.3.1.** Fie  $f : [t_0, t_{n+k}] \mapsto \mathbb{R}$  și diviziunea  $\Delta = (t_i)_{i=\overline{0,n+k}}$  ca mai sus. Punem

$$\xi_i = \frac{t_{i+1} + \dots + t_{i+k}}{k}, \quad i = \overline{0, n-1} \quad (6.3.1)$$

și definim operatorul  $S_\Delta$

$$(S_\Delta f)(x) = \sum_{i=0}^{n-1} f(\xi_i) B_{i,k}(x). \quad (6.3.2)$$

Isaac J. Schoenberg (1893-1990) - matematician născut la Galați. A studiat la universitățile din Iași, Berlin și Göttingen. În 1926 își susține doctoratul la Universitatea din Iași. Din 1930 a activat în Statele Unite (1941-1966 University of Pennsylvania, 1966-1973 University of Wisconsin). Contribuții în domeniul Teoriei aproximării. Rezultatele sale în domeniul funcțiilor spline l-au făcut celebru (de fapt el a introdus termenul de funcție spline). A fost și un om sensibil, de aleasă cultură.



Acest operator se numește operator spline cu variație diminuată sau operatorul lui Schoenberg, iar  $S_\Delta f$  se numește funcție spline cu variație diminuată.

$S_\Delta f$  este un spline de gradul  $k$ , fiind o combinație liniară de funcții B-spline de grad  $k$ . Pentru punctele  $\xi$  avem

$$a = \xi_1 < \dots < \xi_{n-1} = b.$$

**Propoziția 6.3.2.**  $S_\Delta$  este un operator liniar și pozitiv și  $\forall f \in \mathbb{P}_1$  avem  $S_\Delta f = f$ .

*Demonstrație.* Liniaritatea și pozitivitatea rezultă din definiție folosind proprietățile B-splinelor. Trebuie să arătăm că  $S_\Delta$  reproduce pe 1 și pe  $x$ .  $S_\Delta 1 = 1$  rezultă din partitura unității. Din identitatea lui Marsden (6.2.4) egalând coeficienții lui  $t^{k-1}$  obținem

$$\sum_{i=0}^{n-1} \xi_i B_{i,k}(x) = x,$$

de unde  $S_\Delta x = x$ .  $\square$

**Propoziția 6.3.3.**  $S_\Delta f$  are următoarea proprietate

$$V(S_\Delta f - \ell) \leq V(f - \ell), \text{ pe } [a, b], \forall \ell \in \mathbb{P}_1,$$

unde  $V(g)$  este numărul variațiilor de semn ale lui  $g$ .

Numele de spline cu variație diminuată vine tocmai de la această proprietate.

**Teorema 6.3.4.** Are loc inegalitatea

$$\|g - S_\Delta g\|_\infty \leq \frac{k^2}{2} \|\Delta\|^2 \|g''\|_\infty. \quad (6.3.3)$$

*Demonstrație.* Fie  $x_0 \in (a, b)$ ; trebuie estimată cantitatea  $|g(x_0) - S_\Delta g(x_0)|$ . Presupunem că  $x_0 \in [t_i, t_{i+1})$ , ceea ce implica  $\xi_{i-k} \leq x_0 < \xi_i$  și atrage  $B_{j,k}(x_0) = 0$  pentru  $j \notin [i-k, i]$ . Fie  $P(x) = g(x_0) + g'(x_0)(x - x_0)$  ecuația tangentei la  $g$  în  $x_0$ . Avem  $S_\Delta P = P$  și deci

$$S_\Delta(g - P) = S_\Delta g - S_\Delta P = S_\Delta g - P = (S_\Delta g - g) + g - P$$

și pentru  $x = x_0$

$$g(x_0) - S_\Delta g(x_0) = -S_\Delta(g - P)(x_0) = -\sum_{j=i-k}^i (g - P)(\xi_j) B_{j,k}(x_0),$$

de unde

$$|g(x_0) - S_\Delta g(x_0)| \leq \sup\{|(g - P)(x) : x \in (\xi_{i-k}, \xi_i)\}|.$$

Dar din formula lui Taylor  $(g - P)(x) = \frac{(x-x_0)^2}{2} g''(\xi)$  cu  $\xi \in (x_0, x)$  și  $|x - x_0| \leq k\|\Delta\|$  pentru  $\xi \in (\xi_{i-k}, \xi_i)$ , de unde

$$|g(x_0) - S_\Delta g(x_0)| \leq \frac{k^2}{2} \|\Delta\|^2 \|g''\|_\infty.$$

$\square$

**Corolarul 6.3.5.**

$$\|x^2 - S_\Delta e_2\|_\infty \leq k^2 \|\Delta\|^2.$$

**Definiția 6.3.6.** Fie  $f : [a, b] \rightarrow \mathbb{R}$ . Formula

$$F = S_\Delta f + R_\Delta f \quad (6.3.4)$$

se numește formula de aproximare spline cu variație diminuată, iar  $R_\Delta f$  este termenul rest.

**Teorema 6.3.7.** Dacă  $f \in C^2[a, b]$  atunci

$$(R_\Delta f)(x) = \int_a^b \varphi(x; t) f''(t) dt \quad (6.3.5)$$

unde

$$\varphi(x; t) = (x - t)_+ - \sum_{i=0}^{n-1} B_{i,k}(x)(\xi_i - t)_+$$

respectiv

$$(R_\Delta f)(x) = -\frac{1}{2}[S_\Delta e_2(x) - x^2]f''(\xi).$$

*Demonstrație.* Se aplică teorema lui Peano și se ține cont că  $\varphi(x; t)$  are semn constant pentru  $t, x \in [a, b]$ .  $\square$

**Observația 6.3.8.** În cazul particular când  $t_0 = \dots = t_k = a < b = t_{k+1} = \dots = t_{2k+1}$  se obține operatorul Bernstein ( $S_\Delta f = B_k f$ ).  $\diamond$

În sursa MATLAB 6.3 se dă o funcție care calculează aproximanta spline cu variație diminuată. Deoarece (conform formulei 6.3.2) spline-ul cu variație diminuată este o combinație liniară de B-spline cu coeficienții  $f(\xi_i)$ ,  $i = 0, n - 1$ , s-a folosit algoritmul Cox-deBoor cu punctele de control date de vectorul  $(f(\xi_i))$ .

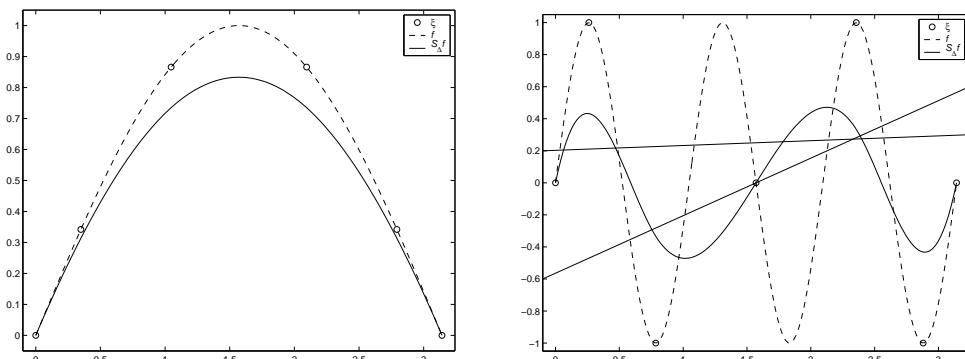


Figura 6.9: Graficul funcției spline cu variație diminuată de gradul 3 pentru funcția  $f(x) = \sin x$  și diviziunea  $\Delta = (0, 0, 0, 0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi, \pi, \pi, \pi)$  (stânga) și ilustrarea proprietății variației diminuate

**Sursa MATLAB 6.3 Calculul funcției spline cu variație diminuată**

```

function [q,xi,t]=spline_vd(grad,d,f)
%SPLINE_VD Spline cu variație diminuată
%apel [t,xi,q]=spline_vd(grad,d,f)
%grad - gradul
%d - diviziunea
%f - funcția
%q - ordonatele
%xi - punctele în care se evaluează f
%t - abscisele

n=length(d)-grad-1;
xi=calcxsi(d,n,grad);
pc=feval(f,xi);
t=min(d):(max(d)-min(d))/150:max(d);
q=Cox_deBoor(grad,pc,d,t);
-----
function xi=calcxsi(d,n,k)
%calculează mediile
for i=0:n-1
    xi(i+1)=sum(d(i+2:i+k+1))/k;
end

```

Figura 6.9(a) dă graficul funcției spline cu variație diminuată de gradul 3 pentru funcția  $f(x) = \sin x$  și diviziunea

$$\Delta = \left( 0, 0, 0, 0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi, \pi, \pi, \pi \right).$$

Ea a fost obținută cu comenziile:

```

kn=[0, 0, 0, 0:3, 3, 3, 3];
d=kn/max(kn)*pi;
[q,xi,t]=spline_vd(3,d,@sin);
plot(xi,sin(xi),'o',t,sin(t),'--',t,q,'-')
legend('\it\xi','\itf','\itS_{\Delta}f')
axis([-0.1,pi+0.1,-0.05,1.05])

```

Figura 6.9(b) ilustrează proprietatea variației diminuate pentru funcția  $f(x) = \sin 6x$  și diviziunea

$$\Delta = \left( 0, 0, 0, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \pi, \pi, \pi \right).$$

## 6.4. Operatori liniari și pozitivi

**Definiția 6.4.1.** Operatorul  $L : X \rightarrow Y$  se numește liniar și pozitiv dacă pentru orice  $f, g \in X$  și orice scalari  $\alpha, \beta$

$$\begin{aligned} L(\alpha f + \beta g) &= \alpha Lf + \beta Lg \\ f \geq 0 \Rightarrow Lf &\geq 0. \end{aligned}$$

**Observația 6.4.2.**  $X$  și  $Y$  sunt spații liniare ordonate. De obicei  $X = \{f : E_1 \subset \mathbb{R} \rightarrow \mathbb{R}\}$ ,  $Y = \{g = Lf : E_2 \subset \mathbb{R} \rightarrow \mathbb{R}\}$ .  $\diamond$

**Teorema 6.4.3.** Dacă  $L : X \rightarrow Y$  este un operator liniar și pozitiv și  $f, g \in X$ , atunci

- (i)  $f \leq g \Rightarrow Lf \leq Lg$  (monotonie);
- (ii)  $|Lf| \leq L|f|$ .

**Observația 6.4.4.** Pentru a pune în evidență faptul că un operator liniar  $L$  se aplică unei funcții  $f$ , ca funcție de variabila independentă  $t$ , iar valoarea funcției obținute se consideră pe punctul  $x$  vom scrie  $L(f(t); x)$  în loc de  $(Lf)(x)$ .  $\diamond$

Teorema care urmează, numită prima teoremă a lui Korovkin sau teorema Bohman-Popoviciu<sup>4</sup>-Korovkin, este un criteriu de convergență uniformă pentru sirurile construite cu ajutorul operatorilor liniari și pozitivi.

**Teorema 6.4.5.** Fie  $(L_m)$ ,  $L_m : C[a, b] \rightarrow C[a, b]$ , un sir de operatori liniari și pozitivi. Dacă

$$\begin{aligned} (L_m e_0)(x) &= 1 + u_m(x); \\ (L_m e_1)(x) &= x + v_m(x); \\ (L_m e_2)(x) &= x^2 + w_m(x) \end{aligned} \quad (6.4.1)$$

și  $\lim_{m \rightarrow \infty} u_m(x) = 0$ ,  $\lim_{m \rightarrow \infty} v_m(x) = 0$  și  $\lim_{m \rightarrow \infty} w_m(x) = 0$ , uniform pe  $[a, b]$ , atunci pentru orice  $f \in C[a, b]$

$$\lim_{m \rightarrow \infty} (L_m f)(x) = f(x),$$

uniform pe  $[a, b]$ .

*Demonstrație.* Fie  $M = \max_{a \leq x \leq b} |f(x)|$  și  $t$  un punct oarecare, dar fix, din intervalul  $[a, b]$ . Funcția  $f$  fiind uniform continuă pe  $[a, b]$ ,  $\forall \varepsilon > 0$ ,  $\exists \delta > 0$ , astfel încât pentru  $x \in [a, b]$  și  $|t - x| < \delta$  să avem  $|f(t) - f(x)| < \frac{\varepsilon}{2}$ . Fie  $J = \{x \in [a, b] : |t - x| \geq \delta\}$ . Pentru  $x \in J$  avem

$$|f(t) - f(x)| \leq 2M \leq \frac{2M}{\delta^2}(t - x)^2;$$

prin urmare

$$|f(t) - f(x)| \leq \frac{\varepsilon}{2} + \frac{2M}{\delta^2}(t - x)^2, \quad x, t \in [a, b]. \quad (6.4.2)$$



<sup>4</sup>

Tiberiu Popoviciu (1906-1975) mare matematician român, membru al Academiei române, fondatorul școlii de Analiză numerică și Teoria aproximării din Cluj-Napoca. Doctorat în Franța, la École Normale Supérieure (1933). Contribuții importante și de pionierat în domeniul Analizei matematice, Teoriei funcțiilor, Teoriei aproximării și Analizei numerice. A contribuit de asemenea la începurile și dezvoltarea informaticii românești și clujene.

Cu notația din observația 6.4.4 ( $L_m(f(t); x)$  în loc de  $(L_m f)(x)$ ) pentru  $t = x$  avem  $L_m(f(x); x) = f(x) \cdot L_m(1; x) = f(x)(L_m e_0)(x)$ . Din enunț (egalitățile (6.4.1)) se observă că putem scrie

$$\begin{aligned}(L_m f)(x) - f(x) &= L_m(f(t); x) - f(x)L_m(1; x) + f(x)u_m(x) \\ &= L_m(f(t) - f(x); x) + f(x)u_m(x).\end{aligned}$$

astfel că avem

$$|(L_m f)(x) - f(x)| \leq |L_m(f(t) - f(x); x)| + |f(x)||u_m(x)|. \quad (6.4.3)$$

Folosind liniaritatea, teorema 6.4.3 și inegalitatea (6.4.2) deducem că

$$\begin{aligned}|L_m(f(t) - f(x); x)| &\leq L_m(|f(t) - f(x)|; x) < L_m\left(\frac{\varepsilon}{2} + \frac{2M}{\delta^2}(t-x)^2; x\right) \\ &= \frac{\varepsilon}{2}L_m(1; x) + \frac{2M}{\delta^2}L_m((t-x)^2; x).\end{aligned}$$

Ultima inegalitate și (6.4.3) ne dau

$$\begin{aligned}|(L_m f)(x) - f(x)| &< \frac{\varepsilon}{2}L_m(1; x) + \frac{2M}{\delta^2}L_m((t-x)^2; x) + |f(x)||u_m(x)| \\ &< \frac{\varepsilon}{2} + \left(|f(x)| + \frac{\varepsilon}{2}\right)|u_m(x)| + \frac{2M}{\delta^2}L_m((t-x)^2; x).\end{aligned} \quad (6.4.4)$$

Din enunț mai rezultă că

$$L_m((t-x)^2; x) = x^2u_m(x) - 2xv_m(x) + w_m(x). \quad (6.4.5)$$

Deoarece  $u_m \rightharpoonup 0$ ,  $v_m \rightharpoonup 0$ ,  $w_m \rightharpoonup 0$ ,

$$\left(|f(x)| + \frac{\varepsilon}{2}\right)|u_m(x)| + \frac{2M}{\delta^2}L_m((t-x)^2; x) < \frac{\varepsilon}{2}, \text{ pentru } m > N_\varepsilon$$

din (6.4.4) se obține

$$|(L_m f)(x) - f(x)| < \frac{\varepsilon}{2}.$$

□

**Corolarul 6.4.6.** Dacă  $(L_m)$ ,  $L_m : C[a, b] \rightarrow C[a, b]$ ,  $m \in \mathbb{N}^*$  este un sir de operatori liniari și pozitivi și

$$\lim_{m \rightarrow \infty} (L_m)(1; x) = 1, \quad \lim_{m \rightarrow \infty} (L_m)((t-x)^2; x) = 0 \quad (6.4.6)$$

pentru orice  $f \in C[a, b]$  sirul  $(L_m f)$  converge uniform la  $f$  pe  $[a, b]$ .

Concluzia rezultă (6.4.4) și (6.4.5).

**Observația 6.4.7.** Funcțiile  $e_0, e_1, e_2$  din teorema 6.4.5 se numesc funcții de probă. Numărul de funcții de probă nu poate fi modificat. ◇

Un rezultat asemănător cu cel din teorema 6.4.5 are loc și pentru funcțiile  $2\pi$ -periodice cu funcțiile de probă  $1, \cos x, \sin x$ , iar corolarul corespunzător se aplică pentru  $1$  și  $\sin^2 \frac{t-x}{2}$ . Fie  $C_{2\pi}$  spațiul funcțiilor continue  $2\pi$ -periodice.

**Teorema 6.4.8 (teorema a doua a lui Korovkin).** Fie  $(L_m)$  un sir de operatori liniari pozitivi  $L_m : C_{2\pi} \rightarrow C_{2\pi}$ . Dacă  $\forall x \in \mathbb{R}$

$$\begin{aligned} L_m(1; x) &= 1 + u_m(x); \\ L_m(\sin t; x) &= \sin x + v_m(x); \\ L_m(\cos t; x) &= \cos x + w_m(x) \end{aligned}$$

și

$$\lim_{m \rightarrow \infty} u_m(x) = \lim_{m \rightarrow \infty} v_m(x) = \lim_{m \rightarrow \infty} w_m(x) = 0$$

uniform pe  $\mathbb{R}_+$ , atunci  $\forall f \in C_{2\pi}$   $L_m f \rightrightarrows f$  pe  $\mathbb{R}$ .

*Demonstrație.* Este analoagă cu a teoremei 6.4.5, analoaga inegalității (6.4.2) fiind

$$|f(t) - f(x)| < \frac{\varepsilon}{2} + 2M \sin^{-2} \frac{\delta}{2} \sin^2 \frac{t-x}{2}.$$

□

## Exemple

**Exemplul 6.4.9 (Operatorul lui Bernstein).** Fie  $(B_m)$  sirul operatorilor Bernstein. Avem

$$B_m(1; x) = 1, \quad B_m(t, x) = x, \quad B_m(t^2; x) = x^2 + \frac{x(1-x)}{m}, \quad x \in [0, 1].$$

Deci  $u_m(x) = 0$ ,  $v_m(x) = 0$ ,  $w_m(x) = \frac{x(1-x)}{m}$ . Deoarece  $\lim_{m \rightarrow \infty} \frac{x(1-x)}{m} = 0$  uniform pe  $[a, b]$ ,  $B_m f \rightrightarrows f$  pe  $[0, 1]$ , rezultat echivalent cu teorema 6.1.2. ◇

**Exemplul 6.4.10 (Operatorul lui Schoenberg).** Operatorul spline cu variație diminuată  $S_\Delta$  este liniar și pozitiv. De asemenea

$$\begin{aligned} S_\Delta(1; x) &= 1; \\ S_\Delta(t; x) &= x; \\ S_\Delta(t^2; x) &= x^2 + E(x). \end{aligned}$$

Deoarece  $E(x) \rightrightarrows 0$  când  $\|\Delta\| \rightarrow 0$ ,  $S_\Delta f \rightrightarrows f$ . ◇

**Exemplul 6.4.11 (Operatorul Hermite-Fejér).** Pornind de la operatorul de interpolare Hermite cu noduri duble rădăcini ale polinomului Cebâșev de speță I,  $T_{m+1}$

$$x_k = \cos \frac{2k+1}{2(m+1)} \pi, \quad k = \overline{0, m},$$

$$(H_{2m+1}f)(x) = \sum_{k=0}^m h_{k0}(x)f(x_k) + \sum_{k=0}^m h_{k1}(x)f(x_k)$$

și omițând a doua sumă, Fejér<sup>5</sup> a obținut operatorul

$$(F_{2m+1}f)(x) = \sum_{k=0}^m h_k(x)f(x_k),$$

unde

$$h_k(x) = h_{k0}(x) = (1 - x_k x) \left[ \frac{T_{m+1}(x)}{(m+1)(x - x_k)} \right]^2.$$

**Propoziția 6.4.12.** *Operatorul lui Fejér are următoarele proprietăți:*

1.  $(F_{2m+1}f)(x_k) = f(x_k)$ ,  $(F_{2m+1}f)'(x_k) = 0$   $\sum_{k=0}^m h_k(x) = 1$ .
2.  $F_{2m+1}$  este un operator liniar și pozitiv.
3. Dacă  $f \in C[-1, 1]$ , atunci  $F_{2m+1}f \Rightarrow f$  pe  $[-1, 1]$ , când  $m \rightarrow \infty$ .

*Demonstrație.* 1. Rezultă din proprietățile polinoamelor de interpolare Hermite.

2. Liniaritatea rezultă din definiție, iar pozitivitatea din

$$1 - x_k x \geq 1 - |x_k| > 0, \quad x \in [-1, 1].$$

3. Din proprietatea 1 rezultă că  $F_{2m+1}(1; x) = 1$ , pentru  $x \in [-1, 1]$ ,

$$\begin{aligned} F_{2m+1}((t-x)^2; x) &= \sum_{k=0}^m (1 - x_k x) \left[ \frac{T_{m+1}(x)}{(m+1)(x - x_k)} \right]^2 (x_k - x)^2 \\ &= \frac{1}{(m+1)^2} T_{m+1}^2(x) \sum_{k=0}^m (1 - x_k x). \end{aligned}$$

Datorită simetriei nodurilor  $x_k$  față de origine,  $\sum_{k=0}^m x_k = 0$  și se obține

$$F_{2m+1}((t-x)^2; x) = \frac{1}{m+1} T_{m+1}^2(x) \leq \frac{1}{m+1};$$

adică

$$\lim_{m \rightarrow \infty} F_{2m+1}((t-x)^2; x) = 0, \text{ uniform pe } [-1, 1].$$

Se aplică apoi corolarul 6.4.6 la teorema 6.4.5.

□

Fiind un operator polinomial, operatorul lui Fejér poate da o demonstrație constructivă a teoremei lui Weierstrass. ◇

5



Leopold Fejér (1880-1959) matematian maghiar, lider al generației sale. A avut contribuții importante în domeniul aproximării și interpoziției în real și complex și teoriei seriilor Fourier. Rezultatul său privind convergența seriilor Fourier a fost obținut când era încă student. A funcționat ca profesor și la Universitatea din Cluj.

## 6.5. Cea mai bună aproximare uniformă

Fie  $f \in C[a, b]$  și presupunem că un polinom de grad cel mult  $n$  minimizează norma  $\|f - p_n\|_\infty$ . Un astfel de polinom se numește *polinomul de cea mai bună aproximare uniformă* a lui  $f$ .

În teoria aproximării uniforme rolul central este jucat de *punctele de alternanță Cebîșev* pentru funcția  $R(x) = f(x) - p_n(x)$ . Punctele de alternanță de grad  $m$  sunt nodurile unei grile (diviziuni)

$$a \leq x_1 \leq \cdots \leq x_m \leq b$$

având următoarele proprietăți:

$$(1) \quad |R(x_i)| = \max_{a \leq x \leq b} |R(x)|, \quad i = \overline{1, m}.$$

$$(2) \quad R(x_i)R(x_{i+1}) < 0, \quad i = \overline{1, m-1}.$$

Vom nota mulțimea tuturor diviziunilor de acest tip pe  $[a, b]$  prin  $\mathcal{A}(m, a, b, R)$ .

**Teorema 6.5.1 (Cebîșev).** Pentru ca un polinom  $p_n$  de grad cel mult  $n$  să fie cea mai bună approximantă uniformă a lui  $f \in C[a, b]$ , este necesar și suficient ca  $\mathcal{A}(n + 2, a, b, R)$  să fie nevidă.

*Demonstrația suficienței.* Presupunem că există un polinom  $q_n \in \mathbb{P}_n$  astfel încât

$$\|f - q_n\|_\infty \leq \|f - p_n\|_\infty.$$

Atunci în punctele de alternanță Cebîșev avem

$$|f(x_i) - q_n(x_i)| < |f(x_i) - p_n(x_i)|,$$

ceea ce implică faptul că funcția  $g(x) \equiv (f(x) - p_n(x)) - (f(x) - q_n(x))$  are în punctele  $x_i$  același semn ca  $R(x) = f(x) - p_n(x)$ . Deoarece semnele lui  $R(x_i)$  alternează, există un zero în interiorul fiecărui subinterval  $[x_i, x_{i+1}]$ . Deci,  $g$  are  $n + 1$  zerori pe  $[a, b]$ . Aceasta nu se poate întâmpla dacă  $g$  nu este identic nul.  $\square$

**Exemplul 6.5.2.** Vom determina polinomul de cea mai bună aproximare uniformă de grad întâi pentru funcția  $f(x) = \sqrt{x}$  pe  $[a, b] \subset \mathbb{R}_+$ .

Polinomul căutat are forma  $P_1^* = c_0 + c_1 x$ . Eroarea de aproximare este  $e_1(x) = c_0 + c_1 x - \sqrt{x}$ . Derivata ei,  $e'_1(x) = c_1 - \frac{1}{2\sqrt{x}}$  se anulează în  $x_1 = \frac{1}{4c_1^2}$ . Conform teoremei lui Cebîșev abaterea maximă se realizează în trei puncte din  $[a, b]$  și obținem sistemul neliniar

$$\begin{cases} c_0 + c_1 a - \sqrt{a} = E_1 \\ c_0 + \frac{1}{4c_1} - \frac{2}{c_1} = -E_1 \\ c_0 + c_1 b - \sqrt{b} = E_1 \end{cases},$$

cu soluțiile

$$\begin{aligned} c_0 &= \frac{1}{2} \left( \sqrt{a} - \frac{a}{\sqrt{a} + \sqrt{b}} + \frac{\sqrt{a} + \sqrt{b}}{4} \right), \\ c_1 &= \frac{1}{\sqrt{a} + \sqrt{b}}, \\ E_1 &= c_0 + c_1 a - \sqrt{a}. \end{aligned}$$

$\diamond$

## Probleme

**Problema 6.1.** (a) Determinați matricea  $A = [a_{ij}]$  de dimensiune  $(m + 1) \times (m + 1)$ ,  $a_{ij} = (p_{m,i}, p_{m,j})$  a ecuațiilor normale relativ la baza Bernstein

$$p_{m,j}(t) = \binom{m}{j} t^j (1-t)^{m-j}, \quad j = \overline{0, m},$$

și funcția pondere  $w(t) \equiv 1$  pe  $[0, 1]$ .

(Indicație: utilizați funcția beta a lui Euler)

(b) Rezolvați sistemul de ecuații normale pentru  $m = 3(3)12$ , când funcția care urmează să fie aproximată este  $f(t) = 1$ . Care este soluția exactă? Afipați, pentru fiecare  $m$ , o estimare a numărului de condiționare, vectorul coeficienților și eroarea asociată (modulul diferenței dintre valoarea calculată și cea exactă). Comentați rezultatul.

**Problema 6.2.** Generați o literă  $\beta$  și o literă  $\omega$  folosind puncte de control adecvate și segmente de curbe B-spline de grad 2 și 3.

**Problema 6.3.** Determinați aproximanta B-spline continuă în sensul celor mai mici pătrate. Scrieți o rutină MATLAB care construiește și rezolvă sistemul de ecuații normale. Exploatați structura de bandă a matricei.

**Problema 6.4.** Considerăm curba Bézier cu punctele de control  $(1, 0), (-1, 1), (1, 1)$  și  $(-1, 0)$ . În  $t = 1/2$  această curbă are un punct cuspidal: prima sa derivată se anulează și are un colț. Verificați aceasta în MATLAB. Perturbați coordonata  $x$  a punctelor de control 2 și 3 cu cantități opuse, păstrând astfel simetria poligonului de control. Ce se întâmplă cu curba?

**Problema 6.5.** Investigați efectul alegerii diferitelor secvențe de noduri  $t$  asupra funcțiilor B-spline cubice (de exemplu, noduri uniforme, neuniforme, simple, multiple și.a.m.d.).

**Problema 6.6.** Alegeți  $n$  puncte de control pe curba

$$\begin{aligned} x(t) &= r \cos t, \\ y(t) &= r \sin t, \\ z(t) &= t, \end{aligned}$$

$t \in [0, 6\pi]$  și reprezentați curba B-spline cubică corespunzătoare.

**Problema 6.7 (Operatorul lui Baskakov).** Notăm cu  $C_N([0, \infty))$  spațiul de funcții:

$$C_N([0, \infty)) := \left\{ f \in C[0, \infty) : \exists \lim_{x \rightarrow \infty} \frac{f(x)}{1+x^N} \text{ finită} \right\},$$

unde  $N \in \mathbb{N}$ . Fie  $f : \mathbb{R} \rightarrow \mathbb{R}$  mărginită și operatorul

$$(L_m f)(x) = \sum_{k=0}^{\infty} \binom{m+k-1}{k} \frac{x^k}{(1+x)^{m+k}} f\left(\frac{k}{m}\right).$$

Să se arate că dacă  $f \in C_2([0, \infty))$ , avem  $\lim_{m \rightarrow \infty} L_m f = f$  uniform pe  $[0, a]$ ,  $0 < a < \infty$ . Implementați o funcție MATLAB care să aproximeze acest operator.

**Problema 6.8 (Operatorul Favard-Szász).** Fie  $f : [0, \infty) \rightarrow \mathbb{R}$  astfel încât  $\lim_{x \rightarrow \infty} f(x) = 0$  și  $a > 0$  fixat. Să se arate că dacă  $f \in C[0, a]$ , operatorul Favard-Szasz definit prin

$$(L_m f)(x) = \sum_{k=0}^{\infty} \frac{(mx)^k}{k!} e^{-mx} f\left(\frac{k}{m}\right)$$

are proprietatea

$$\lim_{m \rightarrow \infty} L_m f = f,$$

uniform pe  $[0, a]$ . Scrieți o funcție MATLAB care aproximează acest operator păstrând un număr finit de termeni din sumă.

**Problema 6.9.** Demonstrați că polinomul de cea mai bună aproximare de gradul al doilea pentru funcția  $\cosh t$  în  $[-1, 1]$  este  $a + bx^2$ , unde  $b = \cosh 1 - 1$ , iar  $a$  se obține rezolvând sistemul neliniar în  $a$  și  $t$ :

$$\begin{aligned} a &= 1 + \cosh t - t^2 b \\ \sinh t &= 2tb. \end{aligned}$$

# CAPITOLUL 7

## Aproximarea funcționalelor liniare

### 7.1. Introducere

Fie  $X$  un spațiu liniar,  $L_1, \dots, L_m$  funcționale liniare reale, liniar independente, definite pe  $X$  și  $L : X \rightarrow \mathbb{R}$  o funcțională liniară reală astfel încât  $L, L_1, \dots, L_m$  să fie liniar independente.

**Definiția 7.1.1.** O formulă de aproximare a funcționalei  $L$  în raport cu funcționalele  $L_1, \dots, L_m$  este o formulă de forma

$$L(f) = \sum_{i=1}^m A_i L_i(f) + R(f), \quad f \in X. \quad (7.1.1)$$

Parametrii reali  $A_i$  se numesc coeficienții formulei, iar  $R(f)$  termenul rest.

Pentru o formulă de aproximare de forma (7.1.1), dându-se funcționalele  $L_i$ , se pune problema determinării coeficienților  $A_i$  și a studiului termenului rest corespunzător valorilor obținute pentru coeficienți.

**Observația 7.1.2.** Forma funcționalelor  $L_i$  depinde de informațiile deținute asupra lui  $f$  (ele exprimând de fapt aceste informații), dar și de natura problemei de aproximare, adică de forma lui  $L$ . ◇

**Exemplul 7.1.3.** Dacă  $X = \{f \mid f : [a, b] \rightarrow \mathbb{R}\}$ ,  $L_i(f) = f(x_i)$ ,  $i = \overline{0, m}$ ,  $x_i \in [a, b]$  și  $L(f) = f(\alpha)$ ,  $\alpha \in [a, b]$ , formula de interpolare Lagrange

$$f(\alpha) = \sum_{i=0}^m \ell_i(\alpha) f(x_i) + (Rf)\alpha$$

este o formulă de tip (7.1.1), cu coeficienții  $A_i = \ell_i(\alpha)$ , iar una din reprezentările posibile pentru rest este

$$(Rf)(\alpha) = \frac{u(\alpha)}{(m+1)!} f^{(m+1)}(\xi), \quad \xi \in [a, b]$$

dacă există  $f^{(m+1)}$  pe  $[a, b]$ . ◇

**Exemplul 7.1.4.** Dacă  $X$  și  $L_i$  sunt ca în exemplul 7.1.3 și există  $f^{(k)}(\alpha)$ ,  $\alpha \in [a, b]$ ,  $k \in \mathbb{N}^*$ , iar  $L(f) = f^{(k)}(\alpha)$ , se obține o formulă de aproximare a valorii derivatei de ordinul  $k$  a lui  $f$  în punctul  $\alpha$

$$f^{(k)}(\alpha) = \sum_{i=0}^m A_i f(x_i) + R(f),$$

numită și *formulă de derivare numerică*. ◊

**Exemplul 7.1.5.** Dacă  $X$  este un spațiu de funcții definite pe  $[a, b]$ , integrabile pe  $[a, b]$  și pentru care există  $f^{(j)}(x_k)$ ,  $k = \overline{0, m}$ ,  $j \in I_k$ , cu  $x \in [a, b]$  și  $I_k$  mulțimi de indici date

$$L_{kj}(f) = f^{(j)}(x_k), \quad k = \overline{0, m}, \quad j \in I_k,$$

iar

$$L(f) = \int_a^b f(x) dx,$$

se obține formula

$$\int_a^b f(x) dx = \sum_{k=0}^m \sum_{j \in I_k} A_{kj} f^{(j)}(x_k) + R(f),$$

numită *formulă de integrare numerică*. ◊

**Definiția 7.1.6.** Dacă  $\mathbb{P}_r \subset X$ , numărul  $r \in \mathbb{N}$  cu proprietatea că  $\text{Ker}(R) = \mathbb{P}_r$  se numește grad de exactitate al formulei de aproximare (7.1.1).

**Observația 7.1.7.** Deoarece  $R$  este o funcțională liniară proprietatea  $\text{Ker}(R) = \mathbb{P}_r$  este echivalentă cu  $R(e_k) = 0$ ,  $k = \overline{0, r}$  și  $R(e_{r+1}) \neq 0$ , unde  $e_k(x) = x^k$ . ◊

Putem acum să formulăm *problema generală de aproximare*: dându-se o funcțională liniară  $L$  pe  $X$ ,  $m$  funcționale liniare  $L_1, L_2, \dots, L_m$  pe  $X$  și valorile lor („datele”)  $\ell_i = L_i f$ ,  $i = \overline{1, m}$  aplicate unei anumite funcții  $f$  și un subspațiu liniar  $\Phi \subset X$  cu  $\dim \Phi = m$ , dorim să găsim o formulă de aproximare de tipul

$$Lf \approx \sum_{i=1}^m a_i L_i f \tag{7.1.2}$$

care să fie exactă (adică să aibă loc egalitatea), pentru orice  $f \in \Phi$ .

Este natural (deoarece dorim să interpolăm) să facem următoarea

**Ipoteză:** „problema de interpolare“

Să se găsească  $\varphi \in \Phi$  astfel încât

$$L_i \varphi = s_i, \quad i = \overline{1, m} \tag{7.1.3}$$

are o soluție unică  $\varphi(\cdot) = \varphi(s, \cdot)$ , pentru  $s = [s_1, \dots, s_m]^T$ , arbitrar.

Putem exprima ipoteza noastră mai explicit în termenii unei baze date  $\varphi_1, \varphi_2, \dots, \varphi_m$  a lui  $\Phi$  și a

matricei Gram<sup>1</sup> asociate

$$G = [L_i \varphi_j] = \begin{vmatrix} L_1 \varphi_1 & L_1 \varphi_2 & \dots & L_1 \varphi_m \\ L_2 \varphi_1 & L_2 \varphi_2 & \dots & L_2 \varphi_m \\ \dots & \dots & \dots & \dots \\ L_m \varphi_1 & L_m \varphi_2 & \dots & L_m \varphi_m \end{vmatrix} \in \mathbb{R}^{m \times m}. \quad (7.1.4)$$

Cerem ca

$$\det G \neq 0. \quad (7.1.5)$$

Este ușor de văzut că această condiție este independentă de alegerea particulară a bazei. Pentru a arăta că solvabilitatea unică a lui (7.1.3) și condiția (7.1.5) sunt echivalente, exprimăm  $\varphi$  din (7.1.3) ca o combinație liniară a funcțiilor de bază

$$\varphi = \sum_{j=1}^{nm} c_j \varphi_j \quad (7.1.6)$$

și observăm că condițiile de interpolare

$$L_i \left( \sum_{j=1}^m c_j \varphi_j \right) = s_i, \quad i = \overline{1, m}$$

pot fi scrise (înănd cont de liniaritatea lui  $L_i$ ) sub forma

$$\sum_{j=1}^m c_j L_i \varphi_j = s_i, \quad i = \overline{1, m},$$

adică

$$Gc = s, \quad c = [c_1, c_2, \dots, c_m]^T, \quad s = [s_1, s_2, \dots, s_m]^T. \quad (7.1.7)$$

Aceasta are o soluție unică pentru  $s$  arbitrar dacă și numai dacă are loc (7.1.5).

Avem două abordări pentru rezolvarea acestei probleme.

**Metoda interpolării.** Rezolvăm problema generală de aproximare prin interpolare

$$Lf \approx L\varphi(\ell; \cdot), \quad \ell = [\ell_1, \ell_2, \dots, \ell_m]^T, \quad \ell_i = L_i f \quad (7.1.8)$$

Cu alte cuvinte aplicăm  $L$  nu lui  $f$ , ci soluției  $\varphi(\ell; \cdot)$  a problemei de aproximare (7.1.3) în care  $s = \ell$ . Ipoteza noastră ne garantează că  $\varphi(\ell; \cdot)$  este unic determinat. În particular, dacă  $f \in \Phi$ , atunci (7.1.8) are loc cu egalitate, deoarece  $\varphi(\ell; \cdot) = f(\cdot)$ , în mod trivial. Astfel, aproximanta noastră (7.1.8)



1

Jørgen Pedersen Gram (1850-1916), matematician danez, a studiat la Universitatea din Copenhaga. După absolvire a intrat la o companie de asigurări ca asistent calculator și apoi a promovat treptat până la ajuns director. A avut contribuții în domeniul dezvoltării în serie a funcțiilor și aproximării Cebîșev și în sensul celor mai mici pătrate. „Determinantul Gram” a fost introdus de el în legătură cu studiile sale asupra liniar independenței.

satisfac condițiile de exactitate cerute pentru (7.1.2). Rămâne doar să arătăm că (7.1.8) produce o aproximare de forma (7.1.2). Pentru aceasta să observăm că interpolantul în (7.1.8) este

$$\varphi(\ell; \cdot) = \sum_{j=1}^m c_j \varphi_j(\cdot)$$

unde vectorul  $c = [c_1, c_2, \dots, c_m]^T$  satisfac (7.1.7) cu  $s = \ell$

$$Gc = \ell, \quad \ell = [L_1 f, L_2 f, \dots, L_m f]^T.$$

Scriind

$$\lambda_j = L \varphi_j, \quad j = \overline{1, m}, \quad \lambda = [\lambda_1, \lambda_2, \dots, \lambda_m]^T, \quad (7.1.9)$$

avem din liniaritatea lui  $L$

$$L\varphi(\ell; \cdot) = \sum_{j=1}^m c_j L \varphi_j = \lambda^T c = \lambda^T G^{-1} \ell = [(G^T)^{-1} \lambda]^T \ell,$$

adică

$$L\varphi(\ell; \cdot) = \sum_{i=1}^m a_i L_i f, \quad a = [a_1, a_2, \dots, a_m]^T = (G^T)^{-1} \lambda. \quad (7.1.10)$$

**Metoda coeficienților nedeterminați.** Aici determinăm coeficienții din (7.1.3) astfel încât egalitatea să aibă loc  $\forall f \in \Phi$ , care, conform liniarității lui  $L$  și  $L_i$  este echivalentă cu egalitatea pentru  $f = \varphi_1, f = \varphi_2, \dots, f = \varphi_m$ , adică

$$\left( \sum_{j=1}^m a_j L_j \right) \varphi_i = L \varphi_i, \quad i = \overline{1, m},$$

sau conform (7.1.8)

$$\sum_{j=1}^m a_j L_j \varphi_i = \lambda_i, \quad i = \overline{1, m}.$$

Evident, matricea sistemului este  $G^T$ , deci

$$a = [a_1, a_2, \dots, a_m]^T = (G^T)^{-1} \lambda,$$

în concordanță cu (7.1.10). Astfel, metoda interpolării și cea a coeficienților nedeterminați sunt matematic echivalente – ele conduc la exact aceeași aproximare.

S-ar părea că, cel puțin în cazul polinoamelor (adică  $\Phi = \mathbb{P}_d$ ), prima metodă este mai puternică, deoarece poate conduce la o expresie a erorii de interpolare (aplicând funcționala formulei de interpolare  $f = a_n f + r_n f$ ). Dar și în cazul metodei coeficienților nedeterminați, din condiția de exactitate, se poate exprima restul cu ajutorul teoremei lui Peano (teorema 5.3.2).

## 7.2. Derivare numerică

Pentru simplitate vom considera doar derivata de ordinul I. Se pot aplica tehnici analoage și pentru alte derive. Vom rezolva problema prin interpolare: în loc să derivăm  $f \in C^{m+1}[a, b]$ , vom deriva polinomul său de interpolare:

$$f(x) = (L_m f)(x) + (R_m f)(x). \quad (7.2.1)$$

Scriem polinomul de interpolare în forma Newton

$$(L_m f)(x) = (N_m f)(x) = f_0 + (x - x_0)f[x_0, x_1] + \dots + (x - x_0) \dots (x - x_{m-1})f[x_0, x_1, \dots, x_m] \quad (7.2.2)$$

și restul sub forma

$$(R_m f)(x) = (x - x_0) \dots (x - x_m) \frac{f^{(m+1)}(\xi(x))}{(m+1)!}. \quad (7.2.3)$$

Derivând (7.2.2) în raport cu  $x$  și punând  $x = x_0$  obținem

$$(L_m f)(x_0) = f[x_0, x_1] + (x_0 - x_1)f[x_0, x_1, x_2] + \dots + (x_0 - x_1)(x_0 - x_2) \dots (x_0 - x_{m-1})f[x_0, x_1, \dots, x_m]. \quad (7.2.4)$$

Presupunând că  $f$  este continuu derivabilă pe un interval convenabil se obține pentru rest

$$(R_m f)'(x_0) = (x_0 - x_1) \dots (x_0 - x_m) \frac{f^{(m+1)}(\xi(x_0))}{(m+1)!}. \quad (7.2.5)$$

Deci, derivând (7.2.4) obținem

$$f'(x_0) = (L_m f)'(x_0) + \underbrace{(R_m f)'(x_0)}_{e_m}. \quad (7.2.6)$$

Dacă  $H = \max_i |x_0 - x_i|$ , eroarea are forma  $e_m = O(H^m)$ , când  $H \rightarrow 0$ .

Putem obține formule de aproximare de grad arbitrar, dar ele sunt de utilitate practică limitată.

**Observația 7.2.1.** Derivarea numerică este o operație critică și de aceea este bine să fie evitată pe cât posibil, deoarece chiar dacă aproximanta este bună, nu rezultă că derivata aproximantei este o aproximare bună a derivatei (vezi figura 7.1). Aceasta rezultă și din exemplul 7.2.2 ◇

**Exemplul 7.2.2.** Fie funcția

$$f(x) = g(x) + \frac{1}{n} \sin n^2(x-a), \quad g \in C^1[a, b].$$

Se constată că  $d(f, g) \rightarrow 0$  ( $n \rightarrow \infty$ ), dar  $d(f', g') = n \not\rightarrow 0$ . ◇

Formulele de derivare numerică sunt utile pentru deducerea unor metode numerice, în special pentru ecuații diferențiale ordinare și ecuații cu derive parțiale.

Se pot folosi și alte procedee de aproximare: Taylor, Hermite, spline, metoda celor mai mici pătrate.

## 7.3. Integrare numerică

Problema este de a calcula integrala definită a unei funcții date pe un interval mărginit  $[a, b]$ . Dacă  $f$  are o comportare bună, aceasta este o problemă de rutină, pentru care metodele cele mai simple de integrare cum ar fi regula trapezelor sau regula repetată a lui Simpson sunt satisfăcătoare, prima având anumite avantaje asupra celei de-a doua în cazul când  $f$  este periodică, cu perioada  $b-a$ . Complicațiile apar atunci când  $f$  are singularități (dar rămâne integrabilă), sau când intervalul de integrare este nemărginit (care este o altă manifestare a comportării singulare). Descompunând integrala pe subintervale, dacă

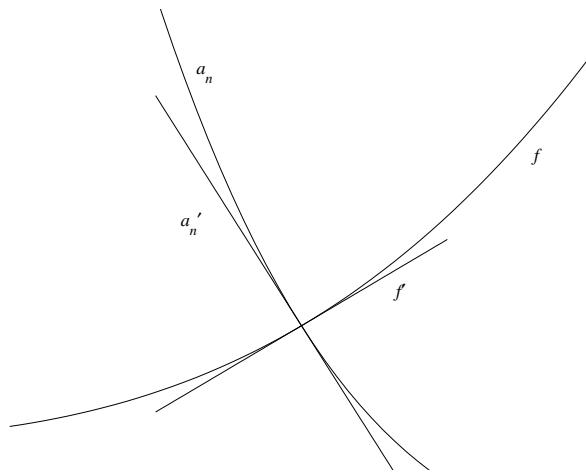


Figura 7.1: Neajunsurile derivării numerice

este necesar, în mai multe integrale, se poate presupune că singularitatea, dacă locul ei este cunoscut, este la unul sau la ambele capete ale intervalului  $[a, b]$ . Astfel de integrale improprii pot fi tratate ca și quadraturi cu ponderi, adică să încorporăm singularitatea într-o pondere, care devine un factor al integrandului, lăsând celălalt factor să aibă o comportare bună. Cel mai important exemplu este formula lui Gauss relativă la o astfel de pondere. În fine, este posibil să se accelereze convergența unei scheme de quadratură prin recombinări convenabile. Un astfel de exemplu este metoda lui Romberg.

Fie  $f : [a, b] \rightarrow \mathbb{R}$  integrabilă pe  $[a, b]$ ,  $F_k(f)$ ,  $k = \overline{0, m}$  informații despre  $f$  (de regulă funcționale liniare) și  $w : [a, b] \rightarrow \mathbb{R}_+$  o funcție pondere integrabilă pe  $[a, b]$ .

**Definiția 7.3.1.** O formulă de forma

$$\int_a^b w(x)f(x)dx = Q(f) + R(f), \quad (7.3.1)$$

unde

$$Q(f) = \sum_{j=0}^m A_j F_j(f),$$

se numește formulă de integrare numerică a funcției  $f$  sau formulă de quadratură. Parametrii  $A_j$ ,  $j = \overline{0, m}$  se numesc coeficienții formulei, iar  $R(f)$  termenul rest al ei.  $Q$  se numește funcțională de quadratură.

**Definiția 7.3.2.** Numărul natural  $d = d(Q)$  cu proprietatea că  $\forall f \in \mathbb{P}d$ ,  $R(f) = 0$  și  $\exists g \in \mathbb{P}d+1$  astfel încât  $R(g) \neq 0$  se numește grad de exactitate al formulei de quadratură.

Deoarece  $R$  este liniar, rezultă că o formulă de quadratură are gradul de exactitate  $d$  dacă și numai dacă  $R(e_j) = 0$ ,  $j = \overline{0, d}$  și  $R(e_{d+1}) \neq 0$ .

Dacă gradul de exactitate al unei formule de quadratură este cunoscut, restul se poate determina cu ajutorul teoremei lui Peano.

### 7.3.1. Formula trapezului și formula lui Simpson

Aceste formule au fost denumite de Gautschi în [24] „caii de bătaie” ai integrării numerice. Ele își fac bine munca când intervalul de integrare este mărginit și integrandul este neproblematic. Formula trapezelor este surprinzător de eficientă chiar și pentru intervale infinite.

Ambele reguli se obțin aplicând cele mai simple tipuri de interpolare subintervalelor diviziunii

$$a = x_0 < x_1 < x_2 < \dots < x_{n-1} < x_n = b, \quad x_k = a + kh, \quad h = \frac{b-a}{n}. \quad (7.3.2)$$

În cazul formulei trapezelor se interpolează liniar pe fiecare subinterval  $[x_k, x_{k+1}]$  și se obține

$$\int_{x_k}^{x_{k+1}} f(x)dx = \int_{x_k}^{x_{k+1}} (L_1 f)(x)dx + \int_{x_k}^{x_{k+1}} (R_1 f)(x)dx, \quad f \in C^2[a, b], \quad (7.3.3)$$

cu

$$(L_1 f)(x) = f_k + (x - x_k)f[x_k, x_{k+1}].$$

Integrând avem

$$\int_{x_k}^{x_{k+1}} f(x)dx = \frac{h}{2}(f_k + f_{k+1}) + R_1(f),$$

unde

$$R_1(f) = \int_{x_k}^{x_{k+1}} K_1(t)f''(t)dt,$$

iar

$$\begin{aligned} K_1(t) &= \frac{(x_{k+1} - t)^2}{2} - \frac{h}{2}[(x_k - t)_+ + (x_{k+1} - t)_+] = \\ &= \frac{(x_k - t)^2}{2} - \frac{h(x_{k+1} - t)}{2} = \\ &= \frac{1}{2}(x_{k+1} - t)(x_k - t) \leq 0. \end{aligned}$$

Deci

$$R_1(f) = -\frac{h^3}{12}f''(\xi_k), \quad \xi_k \in (x_k, x_{k+1})$$

și

$$\int_{x_k}^{x_{k+1}} f(x)dx = \frac{h}{2}(f_k + f_{k+1}) - \frac{1}{12}h^3f''(\xi_k). \quad (7.3.4)$$

Această formulă se numește *regula (elementară a) trapezului*.

Însuțînd pentru toate subintervalele se obține *regula trapezelor* sau *formula compusă a trapezului* sau *formula repetată a trapezului*.

$$\int_a^b f(x)dx = h \left( \frac{1}{2}f_0 + f_1 + \dots + f_{n-1} + \frac{1}{2}f_n \right) - \frac{1}{12}h^3 \sum_{k=0}^{n-1} f''(\xi_k).$$

Deoarece  $f''$  este continuă pe  $[a, b]$ , restul se poate scrie sub forma

$$R_{1,n}(f) = -\frac{(b-a)h^2}{12}f''(\xi) = -\frac{(b-a)^3}{12n^2}f''(\xi). \quad (7.3.5)$$

Cum  $f''$  este mărginită în modul pe  $[a, b]$  avem

$$R_{1,n}(f) = O(h^2),$$

**Sursa MATLAB 7.1 Aproximarea unei integrale prin formula repetată a trapezului**

```
function I=trapez(f,a,b,n);
%TRAPEZ formula trapezelor
%apel I=trapez(f,a,b,n);

h=(b-a)/n;
I=(f(a)+f(b)+2*sum(f([1:n-1]*h+a)))*h/2;
```

când  $h \rightarrow 0$  și deci regula trapezelor converge când  $h \rightarrow 0$  (sau echivalent,  $n \rightarrow \infty$ ), atunci când  $f \in C^2[a, b]$ .

În sursa MATLAB 7.1 se dă o implementare a formulei trapezelor.

Dacă în locul interpolării liniare se utilizează interpolarea pătratică pe două intervale consecutive se obține *regula lui Simpson repetată*. Varianta ei elementară, numită *regula lui Simpson*<sup>2</sup> sau *formula lui Simpson* este

$$\int_{x_k}^{x_{k+2}} f(x)dx = \frac{h}{3}(f_k + 4f_{k+1} + f_{k+2}) - \frac{1}{90}h^5 f^{(4)}(\xi_k), \quad x_k \leq \xi_k \leq x_{k+2}, \quad (7.3.6)$$

unde s-a presupus că  $f \in C^4[a, b]$ .

Să demonstrăm formula pentru restul formulei lui Simpson. Deoarece gradul de exactitate este 3, conform teoremei lui Peano avem

$$R_2(f) = \int_{x_k}^{x_{k+2}} K_2(t)f^{(4)}(t) dt.$$

unde

$$K_2(t) = \frac{1}{3!} \left\{ \frac{(x_{k+2}-t)^4}{4} - \frac{h}{3} [(x_k-t)_+^3 + 4(x_{k+1}-t)_+^3 + (x_{k+2}-t)_+^3] \right\},$$

adică

$$K_2(t) = \begin{cases} \frac{(x_{k+2}-t)^4}{4} - \frac{h}{3} [4(x_{k+1}-t)^3 + (x_{k+2}-t)^3], & t \in [x_k, x_{k+1}] \\ \frac{9(x_{k+2}-t)^4}{4} - \frac{h}{3}(x_{k+2}-t)^3, & t \in [x_{k+1}, x_{k+2}]. \end{cases}$$

Se verifică că pentru  $t \in [a, b]$ ,  $K_2(t) \leq 0$  și deci putem aplica corolarul la teorema lui Peano.

$$R_2(f) = \frac{1}{4!} f^{(4)}(\xi_k) R_2(e_4),$$

Thomas Simpson (1710-1761), matematician englez autodidact, autor al mai multor texte, populare la vremea respectivă. Simpson și-a publicat formula în 1743, dar ea a fost cunoscută deja, printre alții, de Cavalieri (1639), Gregory (1668) și Cotes (1722).



$$\begin{aligned}
R_2(e_4) &= \frac{x_{k+2}^5 - x_k^5}{5} - \frac{x_{k+2} - x_k}{6} \left[ x_k^4 + 4 \left( \frac{x_{k+2} + x_k}{2} \right)^4 + x_{k+2}^4 \right] \\
&= (x_{k+2} - x_k) \left[ \frac{x_{k+2}^4 + x_{k+2}^3 x_k + x_{k+2}^2 x_k^2 + x_{k+2} x_k^3 + x_k^4}{5} \right. \\
&\quad \left. - \frac{5x_k^4 + 4x_k^3 x_{k+2} + 6x_k^2 x_{k+2}^2 + 4x_k x_{k+2}^3 + 5x_{k+2}^4}{24} \right] \\
&= -\frac{x_{k+2} - x_k}{120} (-x_k^4 + 4x_k^3 x_{k+2} + 6x_k^2 x_{k+2}^2 + 4x_k x_{k+2}^3 - x_{k+2}^4) \\
&= -\frac{4}{15} h^5.
\end{aligned}$$

Deci,

$$R_2(f) = -\frac{h^5}{90} f^{(4)}(\xi_k).$$

Pentru regula repetată a lui Simpson obținem

$$\int_a^b f(x) dx = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + 4f_{n-1} + f_n) + R_{2,n}(f) \quad (7.3.7)$$

cu

$$R_{2,n}(f) = -\frac{1}{180}(b-a)h^4 f^{(4)}(\xi) = -\frac{(b-a)^5}{2880n^4} f^{(4)}(\xi), \quad \xi \in (a, b). \quad (7.3.8)$$

Se observă că  $R_{2,n}(f) = O(h^4)$ , de unde rezultă convergența când  $n \rightarrow \infty$ . Se observă și creșterea ordinului cu 1, ceea ce face ca regula repetată a lui Simpson să fie foarte populară și larg utilizată. Pentru o implementare a ei a se vedea sursa MATLAB 7.2.

---

### Sursa MATLAB 7.2 Aproximarea unei integrale prin formula repetată a lui Simpson

---

```

function I=Simpson(f,a,b,n);
%SIMPSON formula lui Simpson
%apel I=Simpson(f,a,b,n);

h=(b-a)/n;
x2=[1:n-1]*h+a;
x4=[0:n-1]*h+a+h/2;
I=h/6*(f(a)+f(b)+2*sum(f(x2))+4*sum(f(x4)));

```

---

Regula trapezelor lucrează foarte bine pentru polinoame trigonometrice. Presupunem fără a restrângă generalitatea că  $[a, b] = [0, 2\pi]$  și fie

$$\begin{aligned}
\mathbb{T}_m[0, 2\pi] &= \{t(x) : t(x) = a_0 + a_1 \cos x + a_2 \cos 2x + \cdots + a_m \cos mx + \\
&\quad + b_1 \sin x + b_2 \sin 2x + \cdots + b_m \sin mx\}
\end{aligned}$$

Atunci

$$R_{n,1}(f) = 0, \quad \forall f \in \mathbb{T}_{n-1}[0, 2\pi] \quad (7.3.9)$$

Aceasta se verifică luând

$$f(x) = e_\nu(x) := e^{i\nu x} = \cos \nu x + i \sin \nu x, \quad \nu = 0, 1, 2, \dots$$

$$\begin{aligned} R_{n,1}(e_\nu) &= \int_0^{2\pi} e_\nu(x) dx - \frac{2\pi}{n} \left[ \frac{1}{2} e_\nu(0) + \sum_{k=1}^{n-1} e_\nu\left(\frac{2k\pi}{n}\right) + \frac{1}{2} e_\nu(2\pi) \right] = \\ &= \int_0^{2\pi} e^{i\nu x} dx - \frac{2\pi}{n} \sum_{k=0}^{n-1} e^{\frac{2\pi i k \nu}{n}}. \end{aligned}$$

Când  $\nu = 0$ , aceasta este evident 0 și în caz contrar, deoarece

$$\begin{aligned} \int_0^{2\pi} e^{i\nu x} dx &= (i\nu)^{-1} e^{i\nu x} \Big|_0^{2\pi} = 0, \\ R_{n,1}(e_\nu) &= \begin{cases} -2\pi & \text{dacă } \nu = 0 \pmod{n}, \nu > 0 \\ -\frac{2\pi}{n} \frac{1 - e^{i\nu n \cdot 2\pi/n}}{1 - e^{i\nu \cdot 2\pi/n}} = 0 & \text{dacă } \nu \neq 0 \pmod{n} \end{cases} \end{aligned} \quad (7.3.10)$$

În particular,  $R_{n,1}(e_\nu) = 0$  pentru  $\nu = 0, 1, \dots, n-1$ , ceea ce demonstrează (7.3.9). Luând partea reală și imaginară în (7.3.10) se obține

$$R_{n,1}(\cos \nu \cdot) = \begin{cases} -2\pi, & \nu = 0 \pmod{n}, \nu \neq 0 \\ 0, & \text{în caz contrar} \end{cases}$$

$$R_{n,1}(\sin \nu \cdot) = 0.$$

De aceea, dacă  $f$  este  $2\pi$ -periodică și are o dezvoltare Fourier uniform convergentă

$$f(x) = \sum_{\nu=0}^{\infty} [a_\nu(f) \cos \nu x + b_\nu(f) \sin \nu x], \quad (7.3.11)$$

unde  $a_\nu(f), b_\nu(f)$  sunt coeficienții Fourier ai lui  $f$ , atunci

$$\begin{aligned} R_{n,1}(f) &= \sum_{\nu=0}^{\infty} [a_\nu(f) R_{n,1}(\cos \nu \cdot) + b_\nu(f) R_{n,1}(\sin \nu \cdot)] = \\ &= -2\pi \sum_{l=1}^{\infty} a_{ln}(f) \end{aligned} \quad (7.3.12)$$

Din teoria seriilor Fourier se știe că coeficienții Fourier ai lui  $f$  tind către zero cu atât mai repede cu cât  $f$  este mai netedă. Mai exact, dacă  $f \in C^r[\mathbb{R}]$ , atunci  $a_\nu(f) = O(\nu^{-r})$  când  $\nu \rightarrow \infty$  (și similar pentru  $b_\nu(f)$ ). Deoarece conform (7.3.12)

$$R_{n,1}(f) \simeq -2\pi a_n(f),$$

rezultă că

$$R_{n,1}(f) = O(n^{-r}) \text{ când } n \rightarrow \infty \quad (7.3.13)$$

( $f \in C^r[\mathbb{R}]$ ,  $2\pi$  periodică), care dacă  $r > 2$  este mai bună decât  $R_{n,1}(f) = O(n^{-2})$ , valabilă pentru funcții  $f$  neperiodice. În particular, dacă  $r = \infty$ , atunci regula trapezului converge mai repede decât orice putere a lui  $n^{-1}$ . Trebuie observat totuși că  $f$  trebuie să fie netedă pe întreg  $\mathbb{R}$ . Pornind de la o funcție  $f \in C^r[0, 2\pi]$  și prelungind-o pe întreaga axă reală prin periodicitate, în general nu se obține o funcție  $f \in C^r[\mathbb{R}]$ .

O altă împrejurare în care regula trapezelor excelează este pentru funcții definite pe  $\mathbb{R}$  și care au proprietatea următoare: pentru un anumit  $r \geq 1$

$$f \in C^{2r+1}[\mathbb{R}], \quad \int_{\mathbb{R}} |f^{(2r+1)}(x)| dx < \infty,$$

$$\lim_{x \rightarrow -\infty} f^{(2\rho-1)}(x) = \lim_{x \rightarrow +\infty} f^{(2\rho-1)}(x) = 0, \quad \rho = 1, 2, \dots, r. \quad (7.3.14)$$

În acest caz se poate arăta că

$$\int_{\mathbb{R}} f(x) dx = h \sum_{k=-\infty}^{\infty} f(kh) + R(f; h) \quad (7.3.15)$$

are o eroare  $R(f, h)$  ce satisfacă  $R(f, h) = O(h^{2r+1})$ ,  $h \rightarrow 0$ . Deci, dacă (7.3.14) are loc pentru orice  $r \in \mathbb{N}$ , atunci eroarea tinde la zero mai repede decât orice putere a lui  $h$ .

### 7.3.2. Formule Newton-Cotes cu ponderi și formule de tip Gauss

O formulă de cuadratură cu ponderi este o formulă de tipul

$$\int_a^b f(t) w(t) dt = \sum_{k=1}^n A_k f(t_k) + R_n(f) \quad (7.3.16)$$

unde  $w$  este nenegativă, integrabilă pe  $(a, b)$ .

Intervalul  $(a, b)$  poate fi mărginit sau nemărginit. Dacă este nemărginit trebuie să ne asigurăm că integrala din (7.3.16) este bine definită, cel puțin în cazul când  $f$  este polinom. Realizăm aceasta cerând ca toate momentele funcției pondere

$$\mu_s = \int_a^b t^s w(t) dt, \quad s = 0, 1, 2, \dots \quad (7.3.17)$$

să existe și să fie finite.

Spunem că (7.3.16) este de *tip interpolator*, dacă are gradul de exactitate  $d = n - 1$ . Formulele de tip interpolator sunt chiar formulele obținute prin interpolare, adică pentru care

$$\sum_{k=1}^n A_k f(t_k) = \int_a^b L_{n-1}(f; t_1, \dots, t_n, t) w(t) dt \quad (7.3.18)$$

sau echivalent

$$A_k = \int_a^b \ell_k(t) w(t) dt, \quad k = 1, 2, \dots, n, \quad (7.3.19)$$

unde

$$\ell_k(t) = \prod_{\substack{l=1 \\ l \neq k}}^n \frac{t - t_l}{t_k - t_l} \quad (7.3.20)$$

sunt polinoamele fundamentale Lagrange asociate nodurilor  $t_1, t_2, \dots, t_n$ . Faptul că (7.3.16) are gradul de exactitate  $d = n - 1$  este evident, deoarece pentru orice  $f \in \mathbb{P}_{n-1}$  avem  $L_{n-1}(f; \cdot) \equiv f(\cdot)$  în (7.3.18). Reciproc, dacă (7.3.16) are gradul de exactitate  $d = n - 1$ , atunci luând  $f(t) = \ell_r(t)$  în (7.3.17) obținem

$$\int_a^b \ell_r(t) w(t) dt = \sum_{k=1}^n A_k \ell_r(t_k) = A_r, \quad r = 1, 2, \dots, n,$$

adică (7.3.19).

Observăm că dacă se dau  $n$  noduri distincte  $t_1, \dots, t_n$  este posibil întotdeauna să construim o formulă de tip (7.3.16) care este exactă pentru orice polinom de grad  $\leq n - 1$ . În cazul  $w(t) \equiv 1$

pe  $[-1, 1]$  și  $t_k$  sunt echidistante pe  $[-1, 1]$  problema a fost intuită de Newton în 1687 și rezolvată în detaliu de Cotes<sup>3</sup> în jurul anului 1712. Prin extensie, vom numi formula (7.3.16) cu  $t_k$  prescrise și  $A_k$  date de (7.3.19) *formulă de tip Newton-Cotes*.

Chestiunea care se pune în mod natural este dacă nu am putea face aceasta mai bine, adică dacă nu am putea obține gradul de exactitate  $d > n - 1$  printr-o alegere judicioasă a nodurilor  $t_k$  (coeficienții  $A_k$  fiind dați în mod necesar de (7.3.19)). Răspunsul este surprinzător de simplu și de direct. Pentru a-l formula, considerăm polinomul nodurilor

$$u_n(t) = \prod_{k=1}^n (t - t_k). \quad (7.3.21)$$

**Teorema 7.3.3.** Dându-se un întreg  $k$ ,  $0 \leq k \leq n$ , formula de cuadratură (7.3.16) are gradul de exactitate  $d = n - 1 + k$  dacă și numai dacă sunt satisfăcute următoarele condiții:

- (a) formula (7.3.16) este de tip interpolator ;
- (b) polinomul nodurilor  $u_n$  din (7.3.21) satisface

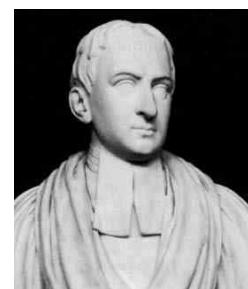
$$\int_a^b u_n(t)p(t)w(t) dt = 0, \forall p \in \mathbb{P}_{k-1}.$$

Condiția din (b) impune  $k$  restricții asupra nodurilor  $t_1, t_2, \dots, t_n$  din (7.3.16). (Dacă  $k = 0$ , nu avem nici o restricție, deoarece să cum știm putem atinge gradul de exactitate  $d = n - 1$ ). Într-adevăr,  $u_n$  trebuie să fie ortogonale pe  $\mathbb{P}_{k-1}$  relativ la funcția pondere  $w$ . Deoarece  $w(t) \geq 0$ , avem în mod necesar  $k \leq n$ . Altfel,  $u_n$  trebuie să fie ortogonal pe  $\mathbb{P}_n$ , în particular pe el însuși, ceea ce este imposibil. Astfel  $k = n$  este optimal, obținându-se o formulă de cuadratură cu gradul maxim de exactitate  $d_{max} = 2n - 1$ . Condiția (b) impune ortogonalitatea lui  $u_n$  pe toate polinoamele de grad mai mic, adică  $u_n(\cdot) = \pi_n(\cdot, w)$  este polinomul ortogonal în raport cu ponderea  $w$ . Această formulă optimală se numește *formulă de cuadratură de tip Gauss* asociată cu funcția pondere  $w$ . Deci nodurile ei sunt zerourile lui  $\pi_n(\cdot, w)$ , iar ponderile (coeficienții)  $A_k$  sunt dați de (7.3.19), adică

$$\begin{aligned} \pi_n(t_k; w) &= 0, \\ A_k &= \int_a^b \frac{\pi_n(t, w)}{(t - t_k)\pi'_n(t_k, w)} w(t) dt, \quad k = 1, 2, \dots, n \end{aligned} \quad (7.3.22)$$

Formula a fost dezvoltată de Gauss în 1814 în cazul special  $w(t) \equiv 1$  pe  $[-1, 1]$  și extinsă la funcții

Roger Cotes (1682-1716), fiul precoce al unui pastor de țară englez, a fost înșărcinat cu pregătirea celei de-a doua ediții a lucrării lui Newton *Principia*. El a dezvoltat ideea lui Newton referitoare la integrarea numerică și a publicat coeficienții pentru formulele de integrare numerică cu  $n$ -puncte, pentru  $n < 11$  (numere Cotes). La moartea sa prematură la vîrstă de 33 de ani, Newton a spus despre el: „Dacă ar fi trăit, am fi putut să iei ceva.”



pondere mai generale de către Christoffel <sup>4</sup> în 1877. De aceea se mai numește *formulă de cuadratură Gauss-Christoffel*.

*Demonstrația teoremei 7.3.3.* Vom demonstra întâi necesitatea lui (a) și (b). Deoarece gradul de exactitate este  $d = n - 1 + k \geq n - 1$ , condiția (a) este trivială. Condiția (b) rezultă de asemenea imediat, deoarece pentru orice  $p \in \mathbb{P}_{k-1}$ ,  $u_n p \in \mathbb{P}_{n-1+k}$ . Deci

$$\int_a^b u_n(t)p(t)w(t)dt = \sum_{k=1}^n A_k u_k(t_k)p(t_k),$$

care se anulează, căci  $u_n(t_k) = 0$  pentru  $k = 1, 2, \dots, n$ .

Pentru a demonstra suficiența lui (a) și (b) trebuie să arătăm că pentru orice  $p \in \mathbb{P}_{n-1+k}$  avem  $R_n(p) = 0$  în (7.3.16). Dându-se orice astfel de  $p$ , îl împărțim cu  $u_n$ , astfel încât

$$p = qu_n + r, \quad q \in \mathbb{P}_{k-1}, \quad r \in \mathbb{P}_{n-1}$$

unde  $q$  este câtul și  $r$  restul. Rezultă că

$$\int_a^b p(t)w(t)dt = \int_a^b q(t)u_n(t)w(t)dt + \int_a^b r(t)w(t)dt.$$

Prima integrală din dreapta este 0, conform lui (b), deoarece  $q \in \mathbb{P}_{k-1}$ , în timp ce a doua, conform lui (a), deoarece  $r \in \mathbb{P}_{n-1}$  este egală cu

$$\sum_{k=1}^n A_k r(t_k) = \sum_{k=1}^n A_k [p(t_k) - q(t_k)u_n(t_k)] = \sum_{k=1}^n A_k p(t_k),$$

ceea ce încheie demonstrația.  $\square$

Cazul  $k = n$  va fi discutat în secțiunea 7.3.3. Vom menționa două cazuri importante când  $k < n$ , care sunt de interes practic. Primul este formula de cuadratură Gauss-Radau în care o extremitate de interval, de exemplu  $a$ , este finită și servește ca nod, să zicem  $t_1 = a$ . Gradul maxim de exactitate care se poate obține este  $d = 2n - 2$  și corespunde lui  $k = n - 1$  în teorema (7.3.3). Partea (b) a teoremei ne spune că nodurile rămase  $t_2, \dots, t_n$  trebuie să fie rădăcinile polinomului  $\pi_{n-1}(\cdot; w_a)$ , unde  $w_a(t) = (t - a)w(t)$ . La fel, în formulele Gauss-Lobatto, ambele capete sunt finite și servesc ca noduri, să zicem  $t_1 = a$ ,  $t_n = b$ , iar nodurile rămase  $t_2, \dots, t_{n-1}$  sunt zerourile lui  $\pi_{n-2}(\cdot; w_{a,b})$ ,  $w_{a,b}(t) = (t - a)(b - t)w(t)$ , obținându-se astfel gradul de exactitate  $d = 2n - 3$ .



4

Elvin Bruno Christoffel (1829-1900), a activat pentru perioade scurte la Berlin și Zürich și cea mai mare parte a vieții la Strasbourg. Este cunoscut pentru lucrările sale de geometrie, în particular de analiza tensorilor, care a jucat un rol important în teoria relativității a lui Einstein.

### 7.3.3. Proprietăți ale cuadraturilor gaussiene

Regula de cuadratură a lui Gauss, dată de (7.3.16) și (7.3.22), pe lângă faptul că este optimală (adică are grad maxim de exactitate) are și unele proprietăți interesante.

(i) Toate nodurile sunt reale, distințe și situate în intervalul deschis  $(a, b)$ . Aceasta este o proprietate cunoscută satisfăcută de zerourile polinoamelor ortogonale.

(ii) Toți coeficienții (ponderile)  $A_k$  sunt pozitivi. Demonstrația se bazează pe o observație inginoasă a lui Stieltjes

$$0 < \int_a^b \ell_j^2(t) w(t) dt = \sum_{k=1}^n A_k \ell_j^2(t_k) = A_j, \quad j = 1, 2, \dots, n,$$

prima egalitate rezultând din faptul că gradul de exactitate este  $d = 2n - 1$ .

(iii) Dacă  $[a, b]$  este mărginit, atunci formula lui Gauss converge pentru orice funcție continuă. Adică  $R_n(f) \rightarrow 0$ , când  $n \rightarrow \infty$ , pentru orice  $f \in C[a, b]$ . Aceasta este o consecință a teoremei de aproximare a lui Weierstrass, din care rezultă că dacă  $\hat{p}_{2n-1}(f; \cdot)$  este polinomul de cea mai bună aproximare a lui  $f$  pe  $[a, b]$  în sensul normei uniforme atunci

$$\lim_{n \rightarrow \infty} \|f(\cdot) - \hat{p}_{2n-1}(f; \cdot)\|_\infty = 0.$$

Deoarece  $R_n(\hat{p}_{2n-1}) = 0$  (căci  $d = 2n - 1$ ), avem succesiv

$$\begin{aligned} |R_n(f)| &= |R_n(f - \hat{p}_{2n-1})| = \\ &= \left| \int_a^b [f(t) - \hat{p}_{2n-1}(f; t)] w(t) dt - \sum_{k=1}^n A_k [f(t_k) - \hat{p}_{2n-1}(f; t_k)] \right| \leq \\ &\leq \int_a^b |f(t) - \hat{p}_{2n-1}(f; t)| w(t) dt + \sum_{k=1}^n A_k |f(t_k) - \hat{p}_{2n-1}(f; t_k)| \leq \\ &\leq \|f(\cdot) - \hat{p}_{2n-1}(f; \cdot)\|_\infty \left[ \int_a^b w(t) dt + \sum_{k=1}^n A_k \right]. \end{aligned}$$

Aici pozitivitatea ponderilor  $A_k$  a intervenit crucial. Observând că

$$\sum_{k=1}^n A_k = \int_a^b w(t) dt = \mu_0,$$

concluzionăm că

$$|R_n(f)| \leq 2\mu_0 \|f - \hat{p}_{2n-1}\|_\infty \rightarrow 0, \text{ când } n \rightarrow \infty.$$

Proprietatea care urmează este baza unui algoritm eficient de obținere a unor formule de cuadratură gaussiană.

(iv) Fie  $\alpha_k = \alpha_k(w)$  și  $\beta_k = \beta_k(w)$  coeficienții din formula de recurență pentru polinoamele ortogonale

$$\begin{aligned} \pi_{k+1}(t) &= (t - \alpha_k) \pi_k(t) - \beta_k \pi_{k-1}(t), & k = 0, 1, 2, \dots \\ \pi_0(t) &= 1, \quad \pi_{-1}(t) = 0, \end{aligned} \tag{7.3.23}$$

unde

$$\begin{aligned} \alpha_k &= \frac{(t\pi_k, \pi_k)}{(\pi_k, \pi_k)} \\ \beta_k &= \frac{(\pi_k, \pi_k)}{(\pi_{k-1}, \pi_{k-1})}, \end{aligned} \tag{7.3.24}$$

cu  $\beta_0$  definit prin

$$\beta_0 = \int_a^b w(t)dt (= \mu_0).$$

*Matricea Jacobi* de ordinul  $n$  pentru funcția pondere  $w$  este o matrice simetrică tridiagonală definită prin

$$J_n(w) = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \sqrt{\beta_2} & \ddots & & \\ & & \ddots & \ddots & \sqrt{\beta_{n-1}} \\ 0 & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{bmatrix}. \quad (7.3.25)$$

**Teorema 7.3.4.** Nodurile  $t_k$  ale unei formule de tip Gauss sunt valorile proprii ale lui  $J_n$

$$J_n v_k = t_k v_k, \quad v_k^T v_k = 1, \quad k = 1, 2, \dots, n, \quad (7.3.26)$$

iar ponderile  $A_k$  sunt exprimabile cu ajutorul componentelor  $v_k$  ale vectorilor proprii normalizați corespunzători prin

$$A_k = \beta_0 v_{k,1}^2, \quad k = 1, 2, \dots, n. \quad (7.3.27)$$

Astfel, pentru a obține o formulă de quadratură Gaussiană trebuie rezolvată o problemă de vectori și valori proprii pentru o matrice tridiagonală simetrică. Pentru această problemă există metode foarte eficiente. Deci, abordarea bazată pe valori și vectori proprii este mai eficientă decât cea clasică. În plus, abordarea clasică se bazează pe două probleme prost condiționate: rezolvarea ecuațiilor polinomiale (coeficienții sunt obținuți prin aplicarea de relații de recurență, deci sunt deja perturbați) și rezolvarea unui sistem de ecuații având matricea Vandermonde.

*Demonstrația teoremei 7.3.4.* Fie  $\tilde{\pi}_k(\cdot) = \tilde{\pi}_k(\cdot, w)$  polinomul ortogonal normalizat, deci  $\pi_k = \sqrt{(\pi_k, \pi_k)_{d\lambda}} \tilde{\pi}_k$ . Inserând aceasta în (7.3.23), împărțind cu  $\sqrt{(\pi_k, \pi_k)_{d\lambda}}$  și utilizând (7.3.24), se obține

$$\tilde{\pi}_{k+1}(t) = (t - \alpha_k) \frac{\tilde{\pi}_k}{\sqrt{\beta_{k+1}}} - \beta_k \frac{\tilde{\pi}_{k-1}}{\sqrt{\beta_{k+1}\beta_k}},$$

sau înmulțind cu  $\sqrt{\beta_{k+1}}$  și reordonând,

$$t\tilde{\pi}_k(t) = \alpha_k \tilde{\pi}_k(t) + \sqrt{\beta_k} \tilde{\pi}_{k-1}(t) + \sqrt{\beta_{k+1}} \tilde{\pi}_{k+1}(t), \quad k = 0, 1, \dots, n-1. \quad (7.3.28)$$

Cu ajutorul matricei lui Jacobi,  $J_n$ , putem scrie această relație sub forma vectorială

$$t\tilde{\pi}(t) = J_n \tilde{\pi}(t) + \sqrt{\beta_n} \tilde{\pi}_n(t) e_n, \quad (7.3.29)$$

unde  $\tilde{\pi}(t) = [\tilde{\pi}_0(t), \tilde{\pi}_1(t), \dots, \tilde{\pi}_{n-1}(t)]^T$  și  $e_n(t) = [0, 0, \dots, 0, 1]^T$  sunt vectori din  $\mathbb{R}^n$ . Deoarece  $t_k$  sunt zerouri ale lui  $\tilde{\pi}_n$  rezultă din (7.3.29) că

$$t_k \tilde{\pi}(t_k) = J_n \tilde{\pi}(t_k), \quad k = 1, 2, \dots, n. \quad (7.3.30)$$

Aceasta demonstrează prima relație a teoremei 7.3.4, deoarece  $\tilde{\pi}$  este un vector nenul cu prima componentă

$$\tilde{\pi}_0 = \beta_0^{-1/2}. \quad (7.3.31)$$

Pentru a demonstra a doua relație, observăm din (7.3.30) că vectorul propriu normalizat  $v_k$  este

$$v_k = \frac{1}{[\tilde{\pi}(t_k)^T \tilde{\pi}(t_k)]} \tilde{\pi}(t_k) = \left( \sum_{\mu=1}^n \tilde{\pi}_{\mu-1}^2(t_k) \right)^{-1/2} \tilde{\pi}(t_k).$$

Comparând prima componentă din ambi membrii și ridicând la pătrat pe baza formulei (7.3.31) avem

$$\frac{1}{\sum_{\mu=1}^n \tilde{\pi}_{\mu-1}^2(t_k)} = \beta_0 v_{k,1}^2, \quad k = 1, 2, \dots, n. \quad (7.3.32)$$

Pe de altă parte, luând  $f(t) = \tilde{\pi}_{\mu-1}(t)$  în formula de cuadratură de tip Gauss (7.3.16), utilizând ortogonalitatea și din nou (7.3.31) se obține că

$$\beta_0^{1/2} \delta_{\mu-1,0} = \sum_{k=0}^n A_k \tilde{\pi}_{\mu-1}(t_k)$$

sau în formă matricială

$$Pa = \beta_0^{1/2} e_1, \quad (7.3.33)$$

unde  $\delta_{\mu-1,0}$  este simbolul lui Kronecker,  $P \in \mathbb{R}^{n \times n}$  este matricea vectorilor proprii,  $a \in \mathbb{R}^n$  este vectorul coeficienților gaussieni și  $e_1 = [1, 0, \dots, 0]^T \in \mathbb{R}^n$ . Deoarece coloanele lui  $P$  sunt ortogonale, avem

$$P^T P = D, \quad D = \text{diag}(d_1, d_2, \dots, d_n), \quad d_k = \sum_{\mu=1}^n \tilde{\pi}_{\mu-1}^2(t_k).$$

Înmulțim acum (7.3.33) la stânga cu  $P^T$  și obținem

$$Da = \beta_0^{1/2} P^T e_1 = \beta_0^{1/2} \cdot \beta_0^{-1/2} e = e, \quad e = [1, 1, \dots, 1]^T.$$

Deci,  $a = D^{-1}e$ , adică,

$$A_k = \frac{1}{\sum_{\mu=1}^n \tilde{\pi}_{\mu-1}^2(t_k)}, \quad k = 1, 2, \dots, n.$$

Comparând cu (7.3.32) se obține rezultatul dorit.  $\square$

Pentru detalii privind aspectele algoritmice legate de polinoame ortogonale și cuadraturi gaussiene a se vedea [25].

Dăm acum o tabelă cu funcțiile pondere clasice, polinoamele lor ortogonale corespunzătoare și coeficienții din formula de recurență  $\alpha_k, \beta_k$  (vezi tabela 7.1).

**Observația 7.3.5.** Pentru polinoamele Jacobi avem

$$\alpha_k = \frac{\beta^2 - \alpha^2}{(2k + \alpha + \beta)(2k + \alpha + \beta + 2)}$$

și

$$\begin{aligned} \beta_0 &= 2^{\alpha+\beta+1} B(\alpha+1, \beta+1), \\ \beta_k &= \frac{4k(k+\alpha)(k+\alpha+\beta)(k+\beta)}{(2k+\alpha+\beta-1)(2k+\alpha+\beta)^2(2k+\alpha+\beta+1)}, \quad k > 0. \end{aligned} \quad \diamond$$

Sursa MATLAB 7.3 calculează nodurile și coeficienții unei formule de cuadratură de tip Gauss cu ajutorul vectorilor și valorilor proprii ale matricei lui Jacobi. Parametrii de intrare sunt coeficienții  $\alpha$  și  $\beta$  ai relației de recurență. Se folosește funcția MATLAB `eig`. Deoarece în cazul unei matrice hermitiene, matricea vectorilor proprii este unitară în limita preciziei de lucru nu s-au mai normalizat vectorii proprii.

polinoamele	notația	ponderea	intervalul	$\alpha_k$	$\beta_k$
Legendre	$P_n(l_n)$	1	[-1,1]	0	$2 \ (k=0)$ $(4-k^{-2})^{-1} \ (k>0)$
Cebișev #1	$T_n$	$(1-t^2)^{-\frac{1}{2}}$	[-1,1]	0	$\pi \ (k=0)$ $\frac{1}{2}\pi \ (k=1)$ $\frac{1}{4} \ (k>0)$
Cebișev #2	$u_n(Q_n)$	$(1-t^2)^{\frac{1}{2}}$	[-1,1]	0	$\frac{1}{2}\pi \ (k=0)$ $\frac{1}{4} \ (k>0)$
Jacobi	$P_n^{(\alpha, \beta)}$	$(1-t)^\alpha (1-t)^\beta$ $\alpha > -1, \beta > -1$	[-1,1]	vezi observația 7.3.5	vezi observația 7.3.5
Laguerre	$L_n^{(\alpha)}$	$t^\alpha e^{-t}$ $\alpha > -1$	$[0, \infty)$	$2k+\alpha+1$	$\Gamma(1+\alpha) \ (k=0)$ $k(k+\alpha) \ (k>0)$
Hermite	$H_n$	$e^{-t^2}$	$\mathbb{R}$	0	$\sqrt{\pi} \ (k=0)$ $k/2 \ (k>0)$

Tabela 7.1: Polinoame ortogonale

**Sursa MATLAB 7.3** Calculul nodurilor și coeficienților unei formule de cuadratură gaussiene

```
function [g_nodes,g_coeff]=Gaussquad(alpha,beta)
%GAUSSQUAD - generare cuadratura Gauss
%calculeaza noduri si coeficienti pentru
%cuadraturi Gauss cu alpha si beta cunoscuti
%metoda - matrice Jacobi
n=length(alpha); rb=sqrt(beta(2:n));
J=diag(alpha)+diag(rb,-1)+diag(rb,1);
[v,d]=eig(J);
g_nodes=diag(d);
g_coeff=beta(1)*v(1,:).^2;
```

**Sursa MATLAB 7.4** Aproximarea unei integrale cu o formulă de tip Gauss

```
function I=vquad(g_nodes,g_coeff,f)
I=g_coeff*f(g_nodes);
```

Calculul valorii aproximative a unei integrale folosind o formulă de cuadratură cu coeficienții și nodurile calculate de Gaussquad se poate realiza într-o singură linie de cod – vezi sursa 7.4. Dăm acum funcțiile MATLAB care calculează nodurile și coeficienții pentru formule de tip Gauss de diverse tipuri, apelând funcția Gaussquad. Sursa 7.5 calculează nodurile și coeficienții Gauss-Legendre.

---

#### **Sursa MATLAB 7.5 Generare formulă Gauss-Legendre**

---

```
function [g_nodes,g_coeff]=Gauss_Legendre(n)
%GAUSS-LEGENDRE - noduri si coeficienti Gauss-Legendre

beta=[2, (4-([1:n-1]).^(-2)).^(-1)];
alpha=zeros(n,1);
[g_nodes,g_coeff]=Gaussquad(alpha,beta);
```

---

Deoarece la o formulă de tip Gauss-Cebîșev de speța I coeficienții sunt egali, iar nodurile sunt rădăcinile polinomului Cebîșev de speța I, în sursa 7.6 nu am mai folosit matricea Jacobi. Se mai dau funcțiile pentru formule Gauss-Cebîșev de speța a doua (sursa 7.7), Gauss-Hermite (sursa 7.8), Gauss-Laguerre (sursa 7.9) și Gauss-Jacobi (sursa 7.10).

---

#### **Sursa MATLAB 7.6 Generare formulă Gauss-Cebîșev de speța I**

---

```
function [g_nodes,g_coeff]=Gauss_Ceb1(n)
%GAUSS_CEB1 - noduri si coeficienti Gauss-Cebisev #1

g_coeff=pi/n*ones(1,n);
g_nodes=cos(pi*([1:n]'-0.5)/n);
```

---



---

#### **Sursa MATLAB 7.7 Generare formulă Gauss-Cebîșev de speța a II-a**

---

```
function [g_nodes,g_coeff]=Gauss_Ceb2(n)
%GAUSS_CEB1 - noduri si coeficienti Gauss-Cebisev #2

beta=[pi/2,1/4*ones(1,n-1)]; alpha=zeros(n,1);
[g_nodes,g_coeff]=Gaussquad(alpha,beta);
```

---



---

#### **Sursa MATLAB 7.8 Generare formulă Gauss-Hermite**

---

```
function [g_nodes,g_coeff]=Gauss_Hermite(n)
%GAUSS_HERMITE - noduri si coeficienti Gauss-Hermite

beta=[sqrt(pi),[1:n-1]/2]; alpha=zeros(n,1);
[g_nodes,g_coeff]=Gaussquad(alpha,beta);
```

---

**Sursa MATLAB 7.9** Generare formulă Gauss-Laguerre

---

```
function [g_nodes,g_coeff]=Gauss_Laguerre(n,a)
%GAUSS_HERMITE - noduri si coeficienti Gauss-Laguerre

k=1:n-1;
alpha=[a+1, 2*k+a+1];
beta=[gamma(1+a),k.*(k+a)];
[g_nodes,g_coeff]=Gaussquad(alpha,beta);
```

---

**Sursa MATLAB 7.10** Generare formulă Gauss-Jacobi

---

```
function [g_nodes,g_coeff]=Gauss_Jacobi(n,a,b)
%Gauss-Jacobi - noduri si coeficienti Gauss-Jacobi

k=0:n-1;
k2=2:n-1;
%rec. relation coeffs
bet1=4*(1+a)*(1+b)/((2+a+b)^2)/(3+a+b);
bet=[2^(a+b+1)*beta(a+1,b+1), bet1, 4*k2.* (k2+a+b).* (k2+a).*...
(k2+b)./(2*k2+a+b-1)./(2*k2+a+b).^2./ (2*k2+a+b+1)];
if a==b
    alpha=zeros(1,n);
else
    alpha=(b^2-a^2)./(2*k+a+b)./(2*k+a+b+2);
end
[g_nodes,g_coeff]=Gaussquad(alpha,bet);
```

---

(v) Markov<sup>5</sup> a observat că formula de quadratură a lui Gauss poate fi obținută cu ajutorul formulei de interolare a lui Hermite cu noduri duble:

$$f(x) = (H_{2n-1}f)(x) + u_n^2(x)f[x, x_1, x_1, \dots, x_n, x_n],$$



5

Andrei Andreevici Markov (1856-1922), matematician rus, activ în Sankt Petersburg. A avut contribuții importante în teoria probabilităților, teoria numerelor și teoria constructivă a aproximării.

$$\begin{aligned} \int_a^b w(x)f(x)dx &= \int_a^b w(x)(H_{2n-1}f)(x)dx + \\ &\quad + \int_a^b w(x)u_n^2(x)f[x, x_1, x_1, \dots, x_n, x_n]dx. \end{aligned}$$

Dar gradul de exactitate  $2n - 1$  implică

$$\begin{aligned} \int_a^b w(x)(H_{2n-1}f)(x)dx &= \sum_{i=1}^n A_i(H_{2n-1}f)(x_i) = \sum_{i=1}^n A_i f(x_i), \\ \int_a^b w(x)f(x)dx &= \sum_{i=1}^n A_i f(x_i) + \int_a^b w(x)u^2(x)f[x, x_1, x_1, \dots, x_n, x_n]dx, \end{aligned}$$

deci

$$R_n(f) = \int_a^b w(x)u_n^2(x)f[x, x_1, x_1, \dots, x_n, x_n]dx.$$

Cum  $w(x)u^2(x) \geq 0$ , aplicând teorema de medie pentru integrale și teorema de medie pentru diferențe divizate avem

$$\begin{aligned} R_n(f) &= f[\eta, x_1, x_1, \dots, x_n, x_n] \int_a^b w(x)u^2(x)dx = \\ &= \frac{f^{(2n)}(\xi)}{(2n)!} \int_a^b w(x)[\pi_n(x, w)]^2 dx, \quad \xi \in [a, b]. \end{aligned}$$

## 7.4. Cuadraturi adaptive

La metodele de integrare numerică erorile nu depind numai de dimensiunea intervalului utilizat, ci și de valoarea derivatelor de un anumit ordin ale funcției care urmează a fi integrată. Aceasta implică faptul că metodele nu vor lucra bine pentru funcții cu derivele mari pe unele subintervale sau pe tot intervalul. Este rezonabil să utilizăm subintervale mici acolo unde derivele sunt mari și subintervale mari acolo unde derivele sunt mici. O metodă care face aceasta într-o manieră sistematică se numește cuadratură adaptivă.

Abordarea generală într-o cuadratură adaptivă este de a utiliza două metode diferite pe fiecare subinterval, de a compara rezultatul și de a subdiviza intervalul dacă diferențele sunt mari. Există situația neferică în care se utilizează două metode proaste, rezultatele sunt proaste, dar diferența dintre ele este mică. Un mod de a evita o astfel de situație este de a ne asigura că o metodă supraestimează rezultatul, iar alta îl subestimează. Sursa MATLAB 7.11 dă un exemplu de structură generală de cuadratură adaptivă recursivă. Parametrul  $\text{g}$  este o funcție care implementează o formulă de cuadratură repetată, de exemplu formula trapezelor sau formula lui Simpson repetată. Structura algoritmului: divide et impera.

Spre deosebire de alte metode, la care se decide cât de mult se muncește pentru a asigura precizia dorită, la o cuadratură adaptivă se calculează doar atât cât este necesar. Aceasta înseamnă că eroarea absolută  $\text{eps}$  trebuie aleasă astfel încât să nu se intre într-un ciclu infinit pentru a atinge o precizie imposibil de atins. Numărul de pași depinde de natura funcției de integrat. Posibilități de îmbunătățire: precizia poate fi scalată cu raportul dintre dimensiunea intervalului curent și dimensiunea întregului interval.

**Exemplul 7.4.1.** Dorim să aproximăm lungimea unui arc de sinusoidă pe un interval egal cu o perioadă. Avem de aproximat

$$I = \int_0^{2\pi} \sqrt{1 + \cos^2(x)} dx.$$

**Sursa MATLAB 7.11 Cuadratură adaptivă**


---

```

function I=adaptquad(f,a,b,eps,g)
%ADAPTQUAD cuadratura adaptiva
%apel I=adaptquad(f,a,b,eps,g)
%f - functia
%a,b - limitele
%eps - eroarea
%g - cuadratura repetata utilizata
m=4;
I1=g(f,a,b,m);
I2=g(f,a,b,2*m);
if abs(I1-I2) < eps %succes
    I=I2;
    return
else %sudivizare recursiva
    I=adaptquad(f,a,(a+b)/2,eps,g)+adaptquad(f,(a+b)/2,b,eps,g);
end

```

---

Funcția de integrat arată în MATLAB astfel

```

function y=lssin(x)
y=sqrt(1+cos(x).^2);

```

Iată și două exemple de apel al lui adaptquad

```

>> format long
>> I=adaptquad(@lssin,0,2*pi,1e-8,@Simpson)
I =
    7.64039557805542
>> I=adaptquad(@lssin,0,2*pi,1e-8,@trapez)
I =
    7.64039557011458

```

Recomandăm cititorului să compare timpii de execuție.



Pentru detalii suplimentare asupra cuadraturilor adaptive recomandăm [23].

## 7.5. Cuadraturi iterate. Metoda lui Romberg

Un dezavantaj al cuadraturilor adaptive este acela că calculează repetat valorile funcției în noduri, iar atunci când este rulat un astfel de program apare un consum suplimentar de timp de calcul datorită recursivității sau gestiunii stivei într-o implementare iterativă. Cuadraturile iterative înălătură aceste inconveniente. Ele aplică la primul pas o cuadratură repetată și apoi subdivid intervalele în părți egale folosind la fiecare pas aproximantele calculate anterior. Vom exemplifica această tehnică printr-o metodă care pornește de la formula repetată a trapezului și îmbunătățește convergența utilizând extrapolarea Richardson.

Primul pas al procesului presupune aplicarea formulei repetate a trapezului cu  $n_1 = 1, n_2 = 2, \dots, n_p = 2^{p-1}$ , unde  $p \in \mathbb{N}^*$ . Valoarea pasului  $h_k$  corespunzătoare lui  $n_k$  va fi

$$h_k = \frac{b-a}{n_k} = \frac{b-a}{2^{k-1}}.$$

Cu aceste notații regula trapezului devine

$$\int_a^b f(x)dx = \frac{h_k}{2} \left[ f(a) + f(b) + 2 \sum_{i=1}^{2^{n-1}-1} f(a + ih_k) \right] - \frac{b-a}{12} h_k^2 f''(\mu_k) \quad (7.5.1)$$

$\mu_k \in (a, b)$ .

Notăm cu  $R_{k,1}$  rezultatul aproximării conform (7.5.1).

$$R_{1,1} = \frac{h_1}{2} [f(a) + f(b)] = \frac{b-a}{2} [f(a) + f(b)], \quad (7.5.2)$$

$$\begin{aligned} R_{2,1} &= \frac{h_2}{2} [f(a) + f(b) + 2f(a + h_2)] = \\ &= \frac{b-a}{4} \left[ f(a) + f(b) + 2f\left(a + \frac{b-a}{2}\right) \right] = \\ &= \frac{1}{2} \left[ R_{1,1} + h_1 f\left(a + \frac{1}{2}h_1\right) \right] \end{aligned}$$

și, în general,

$$R_{k,1} = \frac{1}{2} \left[ R_{k-1,1} + h_{k-1} \sum_{i=1}^{2^{k-2}} f\left(a + \left(i - \frac{1}{2}\right) h_{k-1}\right) \right], \quad k = \overline{2, n} \quad (7.5.3)$$

Urmează îmbunătățirea prin extrapolare Richardson <sup>6</sup> (deși a fost introdusă de Richardson și Gaunt

6



Lewis Fry Richardson (1881-1953), matematician englez. A avut contribuții la predicția numerică a vremii, propunând rezolvarea ecuațiilor hidro și termodinamice ale meteorologiei cu metode bazate pe diferențe finite. A realizat un studiu de referință asupra turbulenței atmosferice, introducând cantitățile numite astăzi „numerele Richardson”. La 50 de ani și-a luat licența în psihologie și a dezvoltat o teorie științifică a relațiilor internaționale. A fost ales membru al Royal Society în 1926.

se pare că este datorată lui Arhimede <sup>7)</sup>

$$I = \int_a^b f(x)dx = R_{k-1,1} - \frac{(b-a)}{12} h_k^2 f''(a) + O(h_k^4).$$

Vom elimina termenul în  $h_k^2$  combinând două ecuații

$$\begin{aligned} I &= R_{k-1,1} - \frac{(b-a)}{12} h_k^2 f''(a) + O(h_k^4), \\ I &= R_{k,1} - \frac{b-a}{48} h_k^2 f''(a) + O(h_k^4). \end{aligned}$$

Obținem

$$I = \frac{4R_{k,1} - R_{k-1,1}}{3} + O(h_k^4).$$

Definim

$$R_{k,2} = \frac{4R_{k,1} - R_{k-1,1}}{3}. \quad (7.5.4)$$

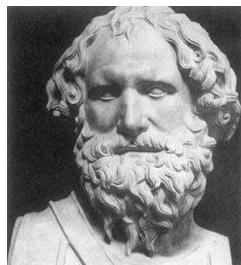
Se aplică extrapolarea Richardson și acestor valori. În general dacă  $f \in C^{2n+2}[a, b]$ , atunci pentru  $k = \overline{1, n}$  putem scrie

$$\begin{aligned} \int_a^b f(x)dx &= \frac{h_k}{2} \left[ f(a) + f(b) + 2 \sum_{i=1}^{2^{k-1}-1} f(a + ih_k) \right] + \\ &+ \sum_{i=1}^k K_i h_k^{2i} + O(h_k^{2k+2}), \end{aligned} \quad (7.5.5)$$

unde  $K_i$  nu depinde de  $h_k$ .

Formula (7.5.5) se justifică în modul următor. Fie  $a_0 = \int_a^b f(x)dx$  și

$$A(h) = \frac{h}{2} \left[ f(a) + 2 \sum_{k=1}^{n-1} f(a + kh) + f(b) \right], \quad h = \frac{b-a}{k}.$$



7

Arhimede (287 î.e.n. - 212 î.e.n.), matematician grec din Siracusa, unul dintre cei mai importanți ai întregii antichități. A pus la punct o metodă de integrare care i-a permis să determine arii, volume și suprafețe ale multor corpuși. A dat o aproximare precisă a lui  $\pi$ , metode de aproximare precisă a rădăcinii pătrate și un sistem de reprezentare a numerelor mari. În mecanică, Arhimede a descoperit teoremele fundamentale referitoare la centrul de greutate al figurilor plane și solidelor și principiul care îi poartă numele, referitor la un corp scufundat într-un lichid. Mașinile sale de război au contribuit la apărarea orașului său în timpul asediului romanilor, care a durat trei ani. A murit ucis de un soldat roman la sfârșitul asediului.

Dacă  $f \in C^{2k+1}[a, b]$ ,  $k \in \mathbb{N}^*$  are loc următorul rezultat datorat lui Euler<sup>8</sup> și MacLaurin<sup>9</sup>

$$A(h) = a_0 + a_1 h^2 + a_2 h^4 + \cdots + a_k h^{2k} + O(h^{2k+1}), \quad h \rightarrow 0 \quad (7.5.6)$$

unde

$$a_k = \frac{B_{2k}}{(2k)!} [f^{(2k-1)}(b) - f^{(2k-1)}(a)], \quad k = 1, 2, \dots, K.$$

Cantitățile  $B_k$  sunt numerele lui Bernoulli<sup>10</sup>, adică coeficienții dezvoltării

$$\frac{z}{e^z - 1} = \sum_{k=0}^{\infty} \frac{B_k}{k!} z^k, \quad |z| < 2\pi.$$

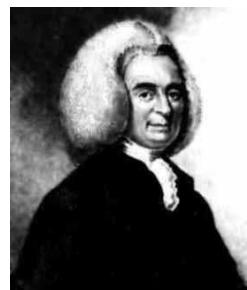
Formula (7.5.6) se numește *formula Euler-MacLaurin*.

Eliminând succesiv puterile lui  $h$  din (7.5.5) se obține

$$R_{k,j} = \frac{4^{j-1} R_{k,j-1} - R_{k-1,j-1}}{4^{j-1} - 1}, \quad k = \overline{2, n}, \quad j = \overline{2, i}.$$

Calculele se pot aranja tabelar, astfel:

Leonhard Euler (1707-1783), matematician elvețian, a urmat cursurile lui Jakob Bernoulli la Universitatea din Basel, luând și lecții particulare de la Johann Bernoulli. După ce la 20 de ani nu a reușit să obțină o catedră de fizică la Basel, a emigrat la Sankt Petersburg; mai târziu s-a mutat la Berlin și apoi din nou la Sankt Petersburg. Indiscutabil, Euler a fost cel mai prolific matematician al secolului al XVIII-lea,<sup>8</sup> lucrând în aproape toate ramurile calculului diferențial și integral și fiind unul dintre fondatorii calculului variațional. A elaborat lucrări de pionierat în științele aplicate: hidrodinamică, mecanica materialelor deformabile și solidului rigid, optică, astronomie. Nici chiar orbirea sa la vîrstă de 59 de ani nu i-a afectat productivitatea fenomenală. Se pare că opera sa nu a fost încă editată în întregime, apărând până în 1998 71 de volume.



Colin Maclaurin (1698-1768), matematician scoțian. A aplicat calculul infinitesimal la probleme de geometrie. Este cunoscut pentru dezvoltarea în serie în jurul originii, dar a avut și contribuții la teoria ecuațiilor.



Jacob Bernoulli (1654-1705), fratele mai mare al lui Johann Bernoulli, activ în Basel. A fost unul dintre primii matematicieni care a apreciat importanța introducerii de către Leibniz și Newton a calculului diferențial și integral, pe care l-a îmbogățit cu contribuții originale, în competiție (nu întotdeauna amicală) cu fratele său mai mic. Este unul dintre fondatorii Teoriei probabilităților, prin lucrarea sa *Ars conjectandi* și prin „legea numerelor mari”.

<sup>10</sup>

$$\begin{matrix}
 & R_{1,1} \\
 R_{2,1} & R_{2,2} \\
 R_{3,1} & R_{3,2} & R_{3,3} \\
 \vdots & \vdots & \vdots & \ddots \\
 R_{n,1} & R_{n,2} & R_{n,3} & \dots & R_{n,n}
 \end{matrix}$$

Deoarece  $(R_{n,1})$  este convergent și  $(R_{n,n})$  converge, mai rapid decât  $(R_{n,1})$ . Drept criteriu de oprire se poate folosi  $|R_{n-1,n-1} - R_{n,n}| \leq \varepsilon$ .

Dăm o implementare în MATLAB a metodei lui Romberg (sursa MATLAB 7.12, fișierul Romberg.m).

**Exemplul 7.5.1.** Problema din exemplul 7.4.1 se poate rezolva și în modul următor:

```
>> I=Romberg(@lsin,0,2*pi,1e-8)
```

```
I =
7.64039557805609
```

## 7.6. Cuadraturi adaptive II

Coloana a doua din metoda lui Romberg corespunde aproximării prin metoda lui Simpson. Notăm

$$S_{k,1} = R_{k,2}.$$

Coloana a treia este deci o combinație a două aproximante de tip Simpson:

$$S_{k,2} = S_{k,1} + \frac{S_{k,1} - S_{k-1,1}}{15} = R_{k,2} + \frac{R_{k,2} - R_{k-1,2}}{15}.$$

Relația

$$S_{k,2} = S_{k,1} + \frac{S_{k,1} - S_{k-1,1}}{15}, \quad (7.6.1)$$

va fi folosită la elaborarea unui algoritm de quadratură adaptivă. Fie  $c = (a+b)/2$ . Formula elementară a lui Simpson este

$$S = \frac{h}{6} (f(a) + 4f(c) + f(b)).$$

Pentru două subintervale se obține

$$S_2 = \frac{h}{12} (f(a) + 4f(d) + 2f(c) + 4f(e) + f(b)),$$

unde  $d = (a+c)/2$  și  $e = (c+b)/2$ . Cantitatea  $Q$  se obține aplicând (7.6.1) celor două aproximante:

$$Q = S_2 + (S_2 - S)/15.$$

Putem să dam acum un algoritm recursiv pentru aproximarea integralei. Funcția adquad evaluatează integrandul aplicând regula lui Simpson. Ea apelează recursiv quadstep și aplică extrapolarea. Implementarea se dă în sursa MATLAB 7.13.

**Exemplul 7.6.1.** Problema din exemplul 7.4.1 se poate rezolva și cu ajutorul funcției adquad:

**Sursa MATLAB 7.12 Metoda lui Romberg**

---

```
function I=romberg(f,a,b,epsi,nmax)
%ROMBERG - calculul aproximativ al unei integrale
%prin metoda lui Romberg
%apel I=romberg(f,a,b,epsi,nmax)
%f -functia
%a,b - limitele de integrare
%epsi - eroarea
%nmax - numar maxim de iteratii
if nargin < 5
    nmax=10;
end
if nargin < 4
    epsi=1e-3;
end
R=zeros(nmax,nmax);
h=b-a;
% prima iteratie
R(1,1)=h/2*(sum(f([a,b])));
for k=2:nmax
    %formula trapezelor;
    x=a+([1:2^(k-2)]-0.5)*h;
    R(k,1)=0.5*(R(k-1,1)+h*sum(f(x)));
    %extrapolare
    plj=4;
    for j=2:k
        R(k,j)=(plj*R(k,j-1)-R(k-1,j-1))/(plj-1);
        plj=plj*4;
    end
    if (abs(R(k,k)-R(k-1,k-1))<epsi) & (k>3)
        I=R(k,k);
        return
    end
    %dublare noduri
    h=h/2;
end
error('prea multe iteratii')
```

---

---

**Sursa MATLAB 7.13 Cuadratura adaptivă, varianta**

---

```

function [Q,fcount] = adquad(F,a,b,tol,varargin)
%ADQUAD Cuadratura adaptiva
%apel [Q,fcount] = adquad(F,a,b,tol,varargin)
% F - functia
% a,b - intervalul
% tol precizia, implicit 1.e-6.
% restul argumentelor se transmit integrandului,F(x,p1,p2,...)

% Face F apelabila prin feval.
if ischar(F) & exist(F) ~=2
    F = inline(F);
elseif isa(F,'sym')
    F = inline(char(F));
end

if nargin < 4 | isempty(tol), tol = 1.e-6; end

% Initializare
c = (a + b)/2;
fa = F(a,varargin{:}); fc = F(c,varargin{:});
fb = F(b,varargin{:});

% Apel recursiv
[Q,k] = quadstep(F, a, b, tol, fa, fc, fb, varargin{:});
fcount = k + 3;

% -----
function [Q,fcount] = quadstep(F,a,b,tol,fa,fc,fb,varargin)
% Subfunctie recursiva utilizata de adquad.

h = b - a;
c = (a + b)/2;
fd = F((a+c)/2,varargin{:});
fe = F((c+b)/2,varargin{:});
Q1 = h/6 * (fa + 4*fc + fb);
Q2 = h/12 * (fa + 4*fd + 2*fc + 4*fe + fb);
if abs(Q2 - Q1) <= tol
    Q = Q2 + (Q2 - Q1)/15;
    fcount = 2;
else
    [Qa,ka] = quadstep(F, a, c, tol, fa, fd, fc, varargin{:});
    [Qb,kb] = quadstep(F, c, b, tol, fc, fe, fb, varargin{:});
    Q = Qa + Qb;
    fcount = ka + kb + 2;
end

```

---

```
>> I=adquad(@lsin,0,2*pi,1e-8)
```

```
I =
7.64039557801944
```

## 7.7. Integrare numerică în MATLAB

MATLAB are două funcții de bază pentru integrare numerică, quad și quadl. Ambele necesită ca intervalul de integrare  $[a, b]$  să fie finit și integrandul să nu aibă nici o singularitate pe acest interval. Pentru tratarea limitelor infinite sau singularităților se pot încerca diverse trucuri cunoscute în analiza numerică cum ar fi schimbarea de variabilă, integrare prin părți, cuadraturi gaussiene, s.a.m.d. (vezi [66, 24, 74, 14]).

Cea mai frecventă formă de apel este  $q = \text{quad}(\text{fun}, a, b, \text{tol})$  (și similar pentru quadl), unde fun este funcția de integrat. Ea poate fi dată sub formă de sir de caractere, obiect inline sau function handle. Important este să accepte la intrare vectori și să returneze vectori. Argumentul tol este eroarea absolută (implicit  $10^{-6}$ ).

Forma  $q = \text{quad}(\text{fun}, a, b, \text{tol}, \text{trace})$  cu trace nenul trasează (urmărește) valorile [fcount a b-a Q] calculate în timpul aplicării recursive.

Forma  $q = \text{quad}(\text{fun}, a, b, \text{tol}, \text{trace}, p1, p2, \dots)$  transmite argumentele suplimentare p1, p2, ..., direct lui fun, fun(x, p1, p2, ...). În acest caz, pentru a utiliza valori implicate ale lui tol sau trace în locul lor pe lista de parametri se vor trece matrice vide.

Forma  $[q, fcount] = \text{quad}(\dots)$  returnează numărul de evaluări de funcții.

Să presupunem că dorim să approximăm  $\int_0^\pi x \sin x dx$ . Integrandul îl putem păstra în fișierul xsin.m:

```
function y=xsin(x)
y=x.*sin(x);
```

Aproximanta se obține astfel:

```
>> quad(@xsin,0,pi)
ans =
3.1416
```

Rutina quad este o implementare a unei cuadraturi adaptive de tip Simpson, aşa cum se descrie în secțiunea 7.6 sau în [46]. quadl este mai precisă și se bazează pe o cuadratură de tip Gauss-Lobatto cu 4 puncte (și grad de exactitate 5) și o extensie a ei de tip Kronrod cu 7 puncte (și grad de exactitate 9), ambele descrise în [23]. Cuadratura este adaptivă. Ambele funcții dau mesaje de avertismen dacă subintervalele devin prea mici sau dacă s-au făcut excesiv de multe evaluări. Astfel de mesaje indică posibile singularități.

Pentru a ilustra modul de lucru al lui quad și quadl vom approxima integrala

$$\int_0^1 \left( \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.09)^2 + 0.04} - 6 \right) dx.$$

Integrandul este funcția MATLAB humps, folosită la testarea rutinelor de integrare numerică sau de demo-urile din MATLAB. Vom aplica quad acestei funcții cu tol=1e-4. Figura 7.2 reprezintă integrandul și marchează punctele de pe axa x în care se evaluatează integrandul; cercurile corespund valorilor integrandului. Figura arată că subintervalele sunt mai mici acolo unde integrandul variază mai

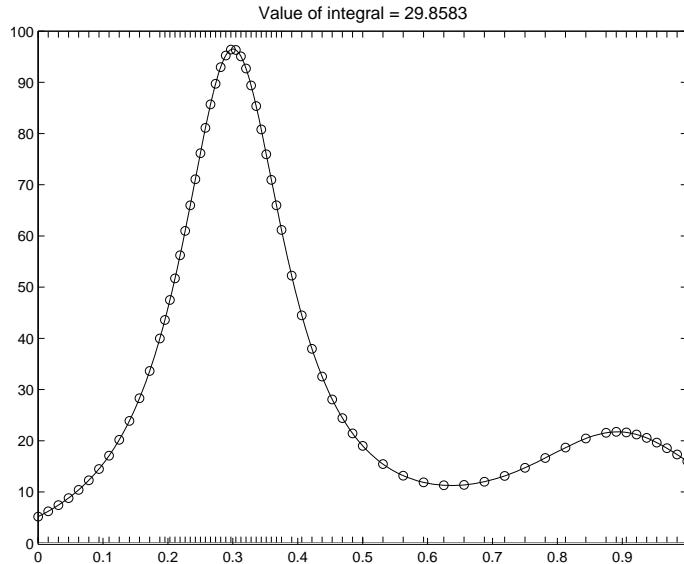


Figura 7.2: Integrarea numerică a lui humps prin quad

rapid. Ea a fost obținută modificând funcția quad din MATLAB. Următorul exemplu aproximează integralele lui Fresnel

$$x(t) = \int_0^t \cos(u^2) du, \quad y(t) = \int_0^t \sin(u^2) du.$$

Acestea sunt ecuațiile parametrice ale unei curbe, numită spirala lui Fresnel. Ea a fost reprezentată în figura 7.3, considerând 1000 de puncte echidistante din intervalul  $[-4\pi, 4\pi]$ . Din motive de eficiență, vom exploata simetria și vom evita integrarea repetată pe  $[0, t]$ , integrând pe fiecare subinterval și evaluând integralele cu cumsum:

```
n = 1000; x = zeros(1,n); y = x;
i1 = inline('cos(x.^2)');
i2 = inline('sin(x.^2)');
t=linspace(0,4*pi,n);
for i=1:n-1
    x(i) = quadl(i1,t(i),t(i+1),1e-3);
    y(i) = quadl(i2,t(i),t(i+1),1e-3);
end
x = cumsum(x); y = cumsum(y);
plot([-x(end:-1:1),0,x], [-y(end:-1:1),0,y])
axis equal
```

Pentru a integra funcții date prin valori, nu prin expresia lor analitică, se folosește funcția trapz. Ea implementează regula trapezelor (nodurile nu trebuie să fie echidistante). Așa cum am văzut în secțiunea 7.3.1, aceasta dă rezultate bune la integrarea funcțiilor periodice pe intervale a căror lungime este un multiplu întreg al perioadei. Exemplu:

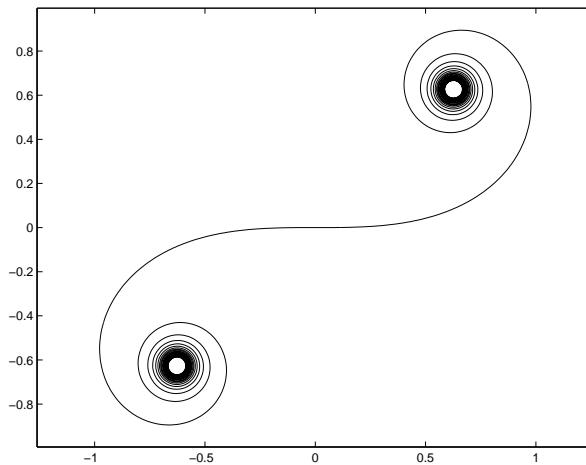


Figura 7.3: Spirala lui Fresnel

```

>> x=linspace(0,2*pi,10);
>> y=1./(2+sin(x));
>> trapz(x,y)
ans =
    3.62759872810065
>> 2*pi*sqrt(3)/3-ans
ans =
    3.677835813675756e-010

```

Valoarea exactă a integralei fiind  $\frac{2}{3}\sqrt{3}\pi$ , eroarea este mai mică decât  $10^{-9}$ .

## Probleme

**Problema 7.1.** Să se aproximeze

$$\int_0^1 \frac{\sin x}{x} dx,$$

folosind o cuadratură adaptivă și metoda lui Romberg. Ce probleme pot să apară? Să se compare rezultatul cu cel furnizat de quad sau quad1.

**Problema 7.2.** Pornind de la o integrală convenabilă, să se aproximeze  $\pi$  cu 8 zecimale exacte, folosind metoda lui Romberg și o cuadratură adaptivă.

**Problema 7.3.** Aproximați

$$\int_{-1}^1 \frac{2}{1+x^2} dx$$

folosind formula trapezelor și formula repetată a lui Simpson, pentru diverse valori ale lui  $n$ . Cum variază precizia odată cu  $n$ ? Reprezentați grafic.

**Problema 7.4.** Funcția eroare,  $\text{erf}$ , se definește prin

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

Tabelați valorile acestei funcții pentru  $x = 0.1, 0.2, \dots, 1$ , utilizând funcția `adquad`. Să se compare rezultatele cu cele furnizate de funcțiile MATLAB `quad` și `erf`.

**Problema 7.5.** (a) Utilizați `adquad` și funcția `quad` din MATLAB pentru a aproxima

$$\int_{-1}^2 \frac{1}{\sin \sqrt{|t|}} dt.$$

(b) De ce nu apar probleme de tip împărțire la zero în  $t = 0$ ?

**Problema 7.6.** Pentru un număr  $p \in \mathbb{N}$  considerăm integrala

$$I_p = \int_0^1 (1-t)^p f(t) dt.$$

Să se compare formula trapezelor pentru  $n$  subintervale cu formula Gauss-Jacobi cu  $n$  noduri și parametrii  $\alpha = p$  și  $\beta = 0$ . Luăți, de exemplu,  $f(t) = \text{tgt}$ ,  $p = 5(5)20$  și  $n = 10(10)50$  în cazul formulei trapezelor și  $n = 1(1)5$  pentru formula Gauss-Jacobi.

**Problema 7.7.** Fie

$$f(x) = \ln(1+x) \ln(1-x).$$

- (a) Utilizați `ezplot` pentru a reprezenta grafic  $f(x)$  pentru  $x \in [-1, 1]$ .  
 (b) Utilizați Maple sau toolbox-ul Symbolic pentru a obține valoarea exactă a integralei

$$\int_{-1}^1 f(x) dx.$$

- (c) Găsiți valoarea numerică a expresiei de la (b).  
 (d) Ce se întâmplă dacă încercăm să utilizăm

$$\text{adquad}(' \log(1+x) * \log(1-x)', -1, 1).$$

la aproximarea valorii integralei?

- (e) Cum evitați dificultatea? Justificați soluția.  
 (f) Utilizați `adquad` cu diverse precizii (toleranțe). Reprezentați grafic eroarea și numărul de evaluări în funcție de toleranță.



# CAPITOLUL 8

## Rezolvarea numerică a ecuațiilor neliniare

### 8.1. Ecuații neliniare

Problema discutată în acest capitol se poate scrie generic sub forma

$$f(x) = 0, \quad (8.1.1)$$

dar admite diverse interpretări, depinzând de semnificația lui  $x$  și  $f$ . Cel mai simplu caz este cel al unei singure ecuații cu o singură necunoscută, caz în care  $f$  este o funcție dată de o variabilă reală sau complexă și încercăm să găsim valorile acestei variabile pentru care  $f$  se anulează. Astfel de valori se numesc rădăcini ale ecuației (8.1.1) sau zerouri ale funcției  $f$ .

Dacă  $x$  din (8.1.1) este un vector, să zicem  $x = [x_1, x_2, \dots, x_d]^T \in \mathbb{R}^d$  și  $f$  este de asemenea un vector ale cărui componente sunt funcții de cele  $d$  variabile  $x_1, x_2, \dots, x_d$ , atunci (8.1.1) reprezintă un sistem de ecuații.

Se spune că sistemul este nelinier dacă cel puțin una dintre componente ale lui  $f$  depinde nelinier de cel puțin una din variabilele  $x_1, x_2, \dots, x_d$ . Dacă toate componente ale lui  $f$  sunt funcții liniare de  $x_1, \dots, x_d$  avem de-a face cu un sistem de ecuații algebrice liniare. Mai general (8.1.1) ar putea reprezenta o ecuație funcțională, dacă  $x$  este un element al unui spațiu de funcții și  $f$  este un operator (liniar sau nelinier) ce acționează pe acest spațiu. În fiecare din aceste situații zeroul din dreapta lui (8.1.1) poate avea diverse interpretări: numărul zero în primul caz, vectorul nul în al doilea și funcția identic nulă în cel de-al treilea.

Mare parte din acest capitol este consacrată unei ecuații neliniare scalare. Astfel de ecuații apar frecvent în analiza sistemelor în vibrație, unde rădăcinile corespund frecvențelor critice (rezonanță). Cazul special al ecuațiilor algebrice, unde  $f$  din (8.1.1) este un polinom, este de importanță considerabilă și merită un tratament special.

## 8.2. Iterații, convergență și eficiență

Nici chiar cele mai simple ecuații - de exemplu cele algebrice - nu admit soluții care să fie exprimabile prin expresii raționale sau radicali. Din acest motiv, este imposibil, în general, să calculăm rădăcinile ecuațiilor neliniare printr-un număr finit de operații aritmetice. Este nevoie de o metodă iterativă, adică de o procedură care generează o secvență infinită de aproximării  $\{x_n\}_{n \in \mathbb{N}}$  astfel încât

$$\lim_{n \rightarrow \infty} x_n = \alpha, \quad (8.2.1)$$

unde  $\alpha$  este o rădăcină a ecuației. În cazul unui sistem,  $x_k$  și  $\alpha$  sunt vectori de dimensiune adecvată, iar convergența trebuie înțeleasă în sensul convergenței pe componente.

Deși convergența unui proces iterativ este de dorit, pentru a putea fi practică, este necesar ceva mai mult decât convergența. Ceea ce se dorește este o convergență rapidă. Conceptul de bază pentru măsurarea vitezei de convergență este ordinul de convergență.

**Definiția 8.2.1.** Spunem că  $x_n$  converge către  $\alpha$  (cel puțin) liniar dacă

$$|x_n - \alpha| \leq e_n, \quad (8.2.2)$$

unde  $\{e_n\}$  este un sir pozitiv ce satisface

$$\lim_{n \rightarrow \infty} \frac{e_{n+1}}{e_n} = c, \quad 0 < c < 1. \quad (8.2.3)$$

Dacă (8.2.2) și (8.2.3) au loc cu egalitate în (8.2.2), atunci c se numește eroare asimptotică.

În această definiție, expresia „cel puțin“ se leagă de faptul că avem doar inegalitate în (8.2.2), ceea ce dorim în practică. De fapt, strict vorbind, marginea  $e_n$  converge liniar, însemnând că în cele din urmă (pentru  $n$  suficient de mare) fiecare din aceste margini ale erorii este aproximativ o fracție constantă din precedenta.

**Definiția 8.2.2.** Se spune că  $x_n$  converge către  $\alpha$  (cel puțin) cu ordinul  $p \geq 1$  dacă (8.2.2) are loc cu

$$\lim_{n \rightarrow \infty} \frac{e_{n+1}}{e_n^p} = c, \quad c > 0. \quad (8.2.4)$$

Astfel convergența de ordinul 1 coincide cu convergența liniară, în timp ce convergența de ordinul  $p > 1$  este mai rapidă. De notat că în acest ultim caz nu se pune nici o restricție asupra constantei  $c$ : odată ce  $e_n$  este suficient de mic, exponentul  $p$  va avea grija de convergență. Și în acest caz, dacă avem egalitate în (8.2.2),  $c$  se numește eroare asimptotică.

Aceleași definiții se aplică și sirurilor vectoriale, cu modulul înlocuit cu orice normă vectorială.

Clasificarea convergenței în raport cu ordinul este destul de rudimentară, deoarece sunt tipuri de convergență la care definițiile (8.2.1) și (8.2.2) nu se aplică. Astfel, un sir  $\{e_n\}$  poate converge către zero mai întâi decât liniar, de exemplu dacă  $c = 1$  în (8.2.3). Acest tip de convergență se numește subliniară. La fel,  $c = 0$  în (8.2.3) conduce la convergență superliniară, dacă (8.2.4) nu are loc pentru nici un  $p > 1$ .

Este instructiv să examinăm comportarea lui  $e_n$ , dacă în loc de relația la limită avem egalitate pentru un anumit  $n$ , să zicem

$$\frac{e_{n+1}}{e_n^p} = c, \quad n = n_0, n_0 + 1, n_0 + 2, \dots \quad (8.2.5)$$

Pentru  $n_0$  suficient de mare, relația este aproape adevărată. Prinț-o simplă inducție se obține că

$$e_{n_0+k} = c^{\frac{p^k-1}{p-1}} e_{n_0}^{p^k}, \quad k = 0, 1, 2, \dots, \quad (8.2.6)$$

care desigur are loc pentru  $p > 1$ , dar și pentru  $p = 1$  când  $p \downarrow 1$ :

$$e_{n_0+k} = c^k e_{n_0}, \quad k = 0, 1, 2, \dots, \quad (p = 1). \quad (8.2.7)$$

Presupunând că  $e_{n_0}$  este suficient de mare astfel încât aproximarea  $x_{n_0}$  are un număr de zecimale corecte, scriem  $e_{n_0+k} = 10^{-\delta_k} e_{n_0}$ . Atunci  $\delta_k$ , în conformitate cu (8.2.2), reprezintă numărul suplimentar de cifre zecimale corecte din aproximația  $x_{n_0+k}$  (în contrast cu  $x_{n_0}$ ). Logaritmând (8.2.6) și (8.2.7) obținem

$$\delta_k = \begin{cases} k \log \frac{1}{c}, & \text{dacă } p = 1 \\ p^k \left[ \frac{1-p^{-k}}{p-1} \log \frac{1}{c} + (1-p^{-k}) \log \frac{1}{e_{n_0}} \right], & \text{dacă } p > 1 \end{cases}$$

deci când  $k \rightarrow \infty$

$$\delta_k \sim c_1 k \quad (p = 1), \quad \delta_k \sim c_p p^k \quad (p > 1), \quad (8.2.8)$$

unde  $c_1 = \log \frac{1}{c} > 0$ , dacă  $p = 1$  și

$$c_p = \frac{1}{p-1} \log \frac{1}{c} + \log \frac{1}{e_{n_0}}$$

(presupunem că  $n_0$  este suficient de mare și deci  $e_{n_0}$  suficient de mic, pentru a avea  $c_p > 0$ ). Aceasta ne arată că numărul de cifre zecimale corecte crește liniar odată cu  $k$  când  $p = 1$  și exponential când  $p > 1$ . În ultimul caz  $\delta_{k+1}/\delta_k \sim p$ , însemnând că (pentru  $k$  mare) numărul de cifre zecimale corecte crește, pe iterație, cu un factor  $p$ .

Dacă fiecare iterație necesită  $m$  unități de lucru (o „unitate de lucru” este efortul necesar pentru a calcula o valoare a funcției sau a unei anumite derivate a sa), atunci *indicele de eficiență* al iterării poate fi definit prin

$$\lim_{k \rightarrow \infty} [\delta_{k+1}/\delta_k]^{1/m} = p^{1/m}.$$

Aceasta ne dă o bază comună de comparare între diversele metode iterative. Metodele liniare au indicele de eficiență 1.

Calculele practice necesită o regulă de oprire care să termine iterăția atunci când s-a obținut (sau se crede că s-a obținut) precizia dorită. Ideal, ne oprim atunci când  $\|x_n - \alpha\| < tol$ ,  $tol$  dat. Deoarece  $\alpha$  nu este cunoscut se obișnuiește să se înlătăruască  $x_n - \alpha$  cu  $x_n - x_{n-1}$  și se impune cerința ca

$$\|x_n - x_{n-1}\| \leq tol, \quad (8.2.9)$$

unde

$$tol = \|x_n\| \varepsilon_r + \varepsilon_a \quad (8.2.10)$$

cu  $\varepsilon_r, \varepsilon_a$  valori date ale erorii. Ca o măsură de siguranță, am putea cere ca (8.2.9) să aibă loc pentru mai multe valori consecutive ale lui  $n$ , nu doar pentru una singură. Alegând  $\varepsilon_r = 0$  sau  $\varepsilon_a = 0$  se obține un test de eroare absolută sau relativă. Este totuși prudent să utilizăm un test mixt, cum ar fi, să zicem  $\varepsilon_e = \varepsilon_a = \varepsilon$ . Atunci, dacă  $\|x_n\|$  este mic sau moderat de mare, se controlează efectiv eroarea absolută, în timp ce pentru  $\|x_n\|$  foarte mare se controlează eroarea relativă. Testele de mai sus se pot combina cu un test de tipul  $\|f(x)\| \leq \varepsilon$ .

## 8.3. Metoda şirurilor Sturm

Există situații în care este de dorit să selectăm o rădăcină particulară dintre mai multe și să avem scheme iterative care converg către ea. Aceasta este cazul, de exemplu, pentru polinoame ortogonale, ale căror rădăcini sunt reale și distințe. De asemenea, am putea dori să facem o selecție a unei anumite

rădăcini: cea mai mare, a doua ca mărime, a treia și.m.d. și să o calculăm fără a mai calcula și altele. Aceasta este posibil dacă combinăm înjumătățirea cu teorema lui Sturm<sup>1</sup>.

Să considerăm ecuația

$$f(x) := \pi_d(x) = 0, \quad (8.3.1)$$

unde  $\pi_d$  este un polinom de grad  $d$ , ortonormal în raport cu o anumită măsură. Știm că  $\pi_d$  este polinomul caracteristic al unei matrice simetrice triadiagonale și poate fi calculat recursiv printr-o relație de recurență de forma

$$\begin{aligned} \pi_0(x) &= 1, \quad \pi_1(x) = x - \alpha_0 \\ \pi_{k+1}(x) &= (x - \alpha_k)\pi_k(x) - \beta_k\pi_{k-1}(x), \quad k = 1, 2, \dots, d-1, \end{aligned} \quad (8.3.2)$$

cu  $\beta_k$  pozitiv. Recurența (8.3.2) este utilă nu numai pentru calculul lui  $\pi_d(x)$ , dar și pentru că are următoarea proprietate utilă, datorată lui Sturm.

**Teorema 8.3.1 (Sturm).** *Fie  $\sigma(x)$  numărul de schimbări de semn (zerourile nu contează) în secvența de numere*

$$\pi_d(x), \pi_{d-1}(x), \dots, \pi_1(x), \pi_0(x). \quad (8.3.3)$$

*Atunci, pentru orice două numere  $a, b$  cu  $a < b$ , numărul de zerouri ale lui  $\pi_d$  pe intervalul  $a < x \leq b$  este egal cu  $\sigma(a) - \sigma(b)$ .*

Deoarece  $\pi_k(x) = x^k + \dots$ , este evident că  $\sigma(-\infty) = d$ ,  $\sigma(+\infty) = 0$ , astfel încât numărul de rădăcini reale ale lui  $\pi_d$  este  $\sigma(-\infty) - \sigma(\infty) = d$ . Mai mult dacă  $\xi_1 > \xi_2 > \dots > \xi_d$  desemnează zerourile lui  $\pi_d$  în ordine descrescătoare, avem comportarea lui  $\sigma$  ca în figura 8.1.

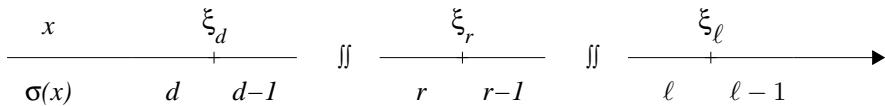


Figura 8.1: Ilustrarea metodei lui Sturm

Este ușor de văzut că:

$$\sigma(x) \leq r-1 \iff x \geq \xi_r. \quad (8.3.4)$$

Într-adevăr, presupunem că  $x \geq \xi_r$ . Atunci  $\#\text{zerouri} \leq x \geq d+1-r$ . Deci conform teoremei lui Sturm,  $\sigma(-\infty) - \sigma(x) = d - \sigma(x) = \#\text{zerouri} \leq x \leq d_1 - r$ , adică  $\sigma(x) \leq r-1$ . Reciproc, dacă  $\sigma(x) \leq r-1$ , atunci, din teorema lui Sturm,  $\#\text{zerouri} \leq x \geq d+1-r$ , ceea ce implică  $x \geq \xi_r$  (vezi figura 8.1).

<sup>1</sup>Jaques Charles François Sturm (1803-1855), matematician și fizician elvețian, cunoscut pentru teorema sa asupra șirurilor Sturm, descoperită în 1829 și pentru teoria sa asupra ecuației diferențiale Sturm-Liouville. A avut contribuții semnificative și în domeniul geometriei proiective și diferențiale.



Ideea de bază este de a controla procesul de înjumătățire nu ca mai sus, ci mai degrabă verificând inegalitatea (8.3.4) pentru a vedea dacă suntem în stânga sau în dreapta lui  $\xi_r$ . Pentru a inițializa procedura, avem nevoie de valorile  $a_1 = a$ ,  $b_1 = b$  astfel încât  $a < \xi_d$  și  $b > \xi_1$ . Acestea se obțin trivial ca și capete ale intervalului de ortogonalitate al lui  $\pi_d$ , dacă acesta este finit. Mai general putem aplica teorema lui Gershgorin matricei lui Jacobi  $J_d$  (7.3.25) asociate polinomului (8.3.2),

$$J_n = \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & 0 \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & \\ & \sqrt{\beta_2} & \alpha_2 & \ddots \\ & & \ddots & \ddots & \sqrt{\beta_{n-1}} \\ 0 & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{bmatrix},$$

înănd cont că zerourile lui  $\pi_d$  sunt valori proprii ale lui  $J_d$ .

Teorema lui Gershgorin afirmă că valorile proprii ale matricei  $A = [a_{ij}]$  de ordin  $d$  sunt localizate în reuniunea discurilor

$$\left\{ z \in \mathbb{C} : |z - a_{ii}| \leq r_i, r_i = \sum_{j \neq i} |a_{ij}| \right\}, \quad i = \overline{1, d}.$$

În acest mod,  $a$  poate fi ales ca fiind cel mai mic și  $b$  cel mai mare dintre cele  $d$  numere  $\alpha_0 + \sqrt{\beta_1}$ ,  $\alpha_1 + \sqrt{\beta_1} + \sqrt{\beta_2}, \dots, \alpha_{d-2} + \sqrt{\beta_{d-2}} + \sqrt{\beta_{d-1}}, \alpha_{d-1} + \sqrt{\beta_{d-1}}$ . Metoda șirurilor lui Sturm continuă după cum urmează, pentru orice  $r$  cu  $1 \leq r \leq d$ :

```

for  $n := 1, 2, 3, \dots$  do
   $x_n := \frac{1}{2}(a_n + b_n);$ 
  if  $\sigma(x_n) > r - 1$  then
     $a_{n+1} := x_n; b_{n+1} := b_n;$ 
  else
     $a_{n+1} := a_n; b_{n+1} := x_n;$ 
  end if
end for
```

Deoarece inițial  $\sigma(a) = d > r - 1$ ,  $\sigma(b) = 0 \leq r - 1$ , rezultă din construcție că

$$\sigma(a_n) > r - 1, \quad \sigma(b_n) \leq r - 1, \quad n = 1, 2, 3, \dots$$

însemnând că  $\xi_r \in [a_n, b_n]$ , pentru orice  $n = 1, 2, 3, \dots$ . Mai mult, deoarece în metoda înjumătățirii,  $b_n - a_n = \frac{b - a}{2^{n-1}}$ , metoda converge cel puțin liniar către rădăcina  $\xi_r$ .

## 8.4. Metoda falsei poziții

Ca în metoda înjumătățirii, presupunem că avem două numere  $a < b$  astfel încât

$$f \in C[a, b], \quad f(a)f(b) < 0 \tag{8.4.1}$$

și generăm un șir descendente de intervale  $[a_n, b_n]$ ,  $n = 1, 2, 3, \dots$ , cu  $a_1 = a$ ,  $b_1 = b$  astfel încât  $f(a_n)f(b_n) < 0$ . Spre deosebire de metoda înjumătățirii, pentru a determina următorul interval nu luăm mijlocul lui  $[a_n, b_n]$ , ci soluția  $x = x_n$  a ecuației liniare

$$(L_1 f)(x; a_n, b_n) = 0.$$

Aceasta pare să fie o alegere mai flexibilă decât în metoda înjumătățirii deoarece  $x_n$  va fi mai apropiat de capătul de care  $|f|$  este mai mic.

Procedura decurge după cum urmează:

```

for  $n := 1, 2, \dots$  do
     $x_n := a_n - \frac{a_n - b_n}{f(a_n) - f(b_n)} f(a_n);$ 
    if  $f(a_n)f(x_n) > 0$  then
         $a_{n+1} := x_n; b_{n+1} := b_n;$ 
    else
         $a_{n+1} := a_n; b_{n+1} := x_n;$ 
    end if
end for

```

Iterația se poate termina când  $\min(x_n - a_n, b_n - x_n) \leq tol$ , unde  $tol$  este o valoare dată.

Convergența se analizează mai ușor dacă presupunem că  $f$  este convexă sau concavă pe  $[a, b]$ . Presupunem că avem

$$f''(x) > 0, \quad x \in [a, b] \text{ (} f \text{ convexă}), \quad f(a) < 0, \quad f(b) > 0. \quad (8.4.2)$$

Șirul

$$x_{n+1} = x_n - \frac{x_n - b}{f(x_n) - f(b)} f(x_n), \quad n \in \mathbb{N}^*, \quad x_1 = a \quad (8.4.3)$$

este monoton crescător și mărginit superior de  $\alpha$ , deci convergent către o limită  $x$ , iar  $f(x) = 0$  (vezi figura 8.2).

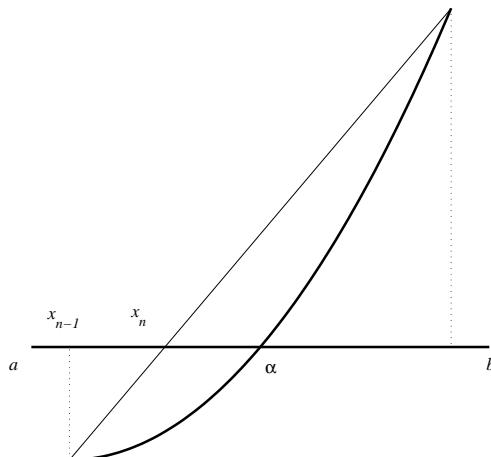


Figura 8.2: Metoda falsei poziții

Viteza de convergență se determină scăzând  $\alpha$  din ambii membri ai lui (8.4.3) și utilizând faptul că  $f(\alpha) = 0$ :

$$x_{n+1} - \alpha = x_n - \alpha - \frac{x_n - b}{f(x_n) - f(b)} [f(x_n) - f(\alpha)].$$

Împărțind cu  $x_n - \alpha$  avem

$$\frac{x_{n+1} - \alpha}{x_n - \alpha} = 1 - \frac{x_n - b}{f(x_n) - f(b)} \frac{f(x_n) - f(\alpha)}{x_n - \alpha}.$$

Făcând  $n \rightarrow \infty$  și utilizând faptul că  $x_n \rightarrow \alpha$ , obținem

$$\lim_{n \rightarrow \infty} \frac{x_{n+1} - \alpha}{x_n - \alpha} = 1 - (b - \alpha) \frac{f'(\alpha)}{f(b)}. \quad (8.4.4)$$

Deci metoda converge liniar, cu eroarea asimptotică

$$c = 1 - (b - a) \frac{f'(\alpha)}{f(b)}.$$

Datorită ipotezei convexității avem  $c \in (0, 1)$ . Analog se face demonstrația în cazul când  $f$  este concavă. Dacă  $f$  nu este nici convexă nici concavă pe  $[a, b]$ , ci  $f \in C^2[a, b]$  și  $f''(\alpha) \neq 0$ ,  $f''$  are semn constant pe o vecinătate a lui  $\alpha$  și pentru un  $n$  suficient de mare  $x_n$  ajunge în acea vecinătate și se poate proceda ca mai sus.

Dezavantaje: (i) convergența lentă; (ii) faptul că unul din capete poate rămâne fix. Dacă  $f$  este turtită în vecinătatea rădăcinii și  $a$  este apropiat de  $\alpha$  și  $b$  depărtat convergența poate fi foarte lentă.

## 8.5. Metoda secantei

Este o variantă a metodei falsei poziții, în care nu se mai cere ca  $f$  să aibă valori de semne contrare, nici măcar la capetele intervalului inițial.

Se aleg două valori arbitrarе de pornire  $x_0, x_1$  și se continuă cu

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n), \quad n \in \mathbb{N}^*. \quad (8.5.1)$$

Aceasta preîntâmpină apariția unei false poziții și sugerează o convergență mai rapidă. Din păcate, nu mai are loc convergența „globală“ pe  $[a, b]$  ci doar convergența „locală“, adică numai dacă  $x_0$  și  $x_1$  sunt suficient de apropiate de rădăcină.

Vom avea în continuare nevoie de o relație între trei erori consecutive

$$\begin{aligned} x_{n+1} - \alpha &= x_n - \alpha - \frac{f(x_n)}{f[x_{n-1}, x_n]} = (x_n - \alpha) \left( 1 - \frac{f(x_n) - f(\alpha)}{(x_n - \alpha)f[x_{n-1}, x_n]} \right) \\ &= (x_n - \alpha) \left( 1 - \frac{f[x_n, \alpha]}{f[x_{n-1}, x_n]} \right) = (x_n - \alpha) \frac{f[x_{n-1}, x_n] - f[x_n, \alpha]}{f[x_{n-1}, x_n]} \\ &= (x_n - \alpha)(x_{n-1} - \alpha) \frac{f[x_n, x_{n-1}, \alpha]}{f[x_{n-1}, x_n]}. \end{aligned}$$

Deci

$$(x_{n+1} - \alpha) = (x_n - \alpha)(x_{n-1} - \alpha) \frac{f[x_n, x_{n-1}, \alpha]}{f[x_{n-1}, x_n]}, \quad n \in \mathbb{N}^*. \quad (8.5.2)$$

Din (8.5.2) rezultă imediat că dacă  $\alpha$  este o rădăcină simplă ( $f(\alpha) = 0, f'(\alpha) \neq 0$ ) și  $x_n \rightarrow \alpha$  și dacă  $f \in C^2$  pe o vecinătate a lui  $\alpha$ , convergența este superliniară. Cât este ordinul de convergență?

Înlocuim raportul diferențelor divizate din (8.5.2) cu o constantă, ceea ce este aproape adevărat când  $n$  este mare. Punând  $c_k = |x_k - \alpha|$ , avem

$$e_{n+1} = e_n e_{n-1} C, \quad C > 0$$

Înmulțind ambii membri cu  $C$  și punând  $E_n = Ce_n$  obținem

$$E_{n+1} = E_n E_{n-1}, \quad E_n \rightarrow 0.$$

Logaritând și punând  $y_n = \frac{1}{E_n}$ , se obține

$$y_{n+1} = y_n + y_{n-1}, \quad (8.5.3)$$

care este recurența pentru sirul lui Fibonacci. Soluția este

$$y_n = c_1 t_1^n + c_2 t_2^n,$$

unde  $c_1, c_2$  sunt constante și

$$t_1 = \frac{1}{2}(1 + \sqrt{5}), \quad t_2 = \frac{1}{2}(1 - \sqrt{5}).$$

Deoarece  $y_n \rightarrow \infty$ , avem  $c_1 \neq 0$  și  $y_n \sim c_1 t_1^n$ , căci  $|t_2| < 1$ . Revenind la substituție  $\frac{1}{E_n} \sim e^{c_1 t_1^n}$ ,  $\frac{1}{e_n} \sim C e^{c_1 t_1^n}$  și deci

$$\frac{e_{n+1}}{e_n^{t_1}} \sim \frac{C^{t_1} e^{c_1 t_1^n t_1}}{C e^{c_1 t_1^{n+1}}} = C^{t_1 - 1}, \quad n \rightarrow \infty.$$

Deci ordinul de convergență este  $t_1 = \frac{1 + \sqrt{5}}{2} \approx 1.61803 \dots$  (secțiunea de aur).

**Teorema 8.5.1.** Fie  $\alpha$  un zero simplu al lui  $f$  și fie  $I_\varepsilon = \{x \in \mathbb{R} : |x - \alpha| < \varepsilon\}$  și presupunem că  $f \in C^2[I_\varepsilon]$ . Definim pentru  $\varepsilon$  suficient de mic

$$M(\varepsilon) = \max_{\substack{s \in I_\varepsilon \\ t \in I_\varepsilon}} \left| \frac{f''(s)}{2f'(t)} \right|. \quad (8.5.4)$$

Presupunem că

$$\varepsilon M(\varepsilon) < 1 \quad (8.5.5)$$

Atunci metoda secantei converge către rădăcina unică  $\alpha \in I_\varepsilon$  pentru orice valori de pornire  $x_0 \neq x_1$  cu  $x_0 \in I_\varepsilon$ ,  $x_1 \in I_\varepsilon$ .

**Observația 8.5.2.** Se observă că  $\lim_{\varepsilon \rightarrow 0} M(\varepsilon) = \left| \frac{f''(\alpha)}{2f'(\alpha)} \right| < \infty$ , deci (8.5.5) poate fi satisfăcută pentru  $\varepsilon$  suficient de mic. Natura locală a convergenței este cuantificată prin cerința ca  $x_0, x_1 \in I_\varepsilon$ . ◇

*Demonstrație.* Se observă că  $\alpha$  este singurul zero al lui  $f$  în  $I_\varepsilon$ . Aceasta rezultă din formula lui Taylor pentru  $x = \alpha$ :

$$f(x) = f(\alpha) + (x - \alpha)f'(\alpha) + \frac{(x - \alpha)^2}{2}f''(\xi),$$

unde  $f(\alpha) = 0$  și  $\xi \in (x, \alpha)$  (sau  $(\alpha, x)$ ). Astfel dacă  $x \in I_\varepsilon$ , atunci și  $\xi \in I_\varepsilon$  și avem

$$f(x) = (x - \alpha)f'(\alpha) \left[ 1 + \frac{x - \alpha}{2} \frac{f''(\xi)}{f'(\alpha)} \right].$$

Aici, dacă  $x \neq \alpha$ , toți trei factorii sunt diferenți de 0, căci

$$\left| \frac{x - \alpha}{2} \frac{f''(\xi)}{f'(\alpha)} \right| \leq \varepsilon M(\varepsilon) < 1.$$

Deci  $f$  se poate anula pe  $I_\varepsilon$  numai în  $x = \alpha$ . Să arătăm că  $x_n \in I_\varepsilon$  pentru orice  $n$ , în afară de cazul când  $f(x_n) = 0$ , în care  $x_n = \alpha$  și metoda converge într-un număr finit de pași. Vom demonstra aceasta prin inducție: presupunem că  $x_{n-1}, x_n \in I_\varepsilon$  și  $x_n \neq x_{n-1}$ . Acest lucru este adevărat pentru  $n = 1$  din ipoteză. Deoarece  $f \in C^2[I_\varepsilon]$

$$f[x_{n-1}, x_n] = f'(\xi_1), \quad f[x_{n-1}, x_n, \alpha] = \frac{1}{2}f''(\xi_2), \quad \xi_i \in I_\varepsilon, \quad i = 1, 2,$$

din (8.5.2) rezultă

$$|x_{n+1} - \alpha| \leq \varepsilon^2 \left| \frac{f''(\xi_2)}{2f'(\xi_1)} \right| \leq \varepsilon \varepsilon M(\varepsilon) < \varepsilon,$$

adică  $x_{n+1} \in I_\varepsilon$ . Mai mult, din relația între trei erori consecutive, (8.5.2), rezultă  $x_{n+1} \neq x_n$  în afară de cazul când  $f(x_n) = 0$  (și atunci  $x_n = \alpha$ ). Utilizând (8.5.2) avem

$$|x_{n+1} - \alpha| \leq |x_n - \alpha| \varepsilon M(\varepsilon)$$

care aplicată repetat ne dă

$$|x_{n+1} - \alpha| \leq |x_n - \alpha| \varepsilon M(\varepsilon) \leq \dots \leq [\varepsilon M(\varepsilon)]^{n-1} |x_1 - \alpha|.$$

Cum  $\varepsilon M(\varepsilon) < 1$ , rezultă că metoda este convergentă și  $x_n \rightarrow \alpha$  când  $n \rightarrow \infty$ .  $\square$

Deoarece este nevoie de o singură evaluare a lui  $f$  pe pas, indicele de eficiență este  $p = \frac{1+\sqrt{5}}{2} \approx 1.61803 \dots$  O implementare a metodei este dată în funcția MATLAB 8.1.

## 8.6. Metoda lui Newton

Poate fi privită ca un caz la limită al metodei secantei, când  $x_{n-1} \rightarrow x_n$ . Obținem iterația

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \quad (8.6.1)$$

O altă interpretare mult mai fructuoasă este liniarizarea ecuației  $f(x) = 0$  în  $x = x_n$ :

$$f(x) \approx f(x_n) + (x - x_n)f'(x_n) = 0.$$

Privită în acest mod metoda lui Newton se poate generaliza la ecuații neliniare de toate tipurile (sisteme neliniare, ecuații funcționale, caz în care  $f'$  trebuie înțeleasă ca derivată Fréchet), iar iterația este

$$x_{n+1} = x_n - [f'(x_n)]^{-1} f(x_n). \quad (8.6.2)$$

Studiul erorii în metoda lui Newton se face la fel ca la metoda secantei.

$$\begin{aligned} x_{n+1} - \alpha &= x_n - \alpha - \frac{f(x_n)}{f'(x_n)} \\ &= (x_n - \alpha) \left[ 1 - \frac{f(x_n) - f(\alpha)}{(x_n - \alpha)f'(x_n)} \right] \\ &= (x_n - \alpha) \left( 1 - \frac{f[x_n, \alpha]}{f[x_n, x_n]} \right) = (x_n - \alpha)^2 \frac{f[x_n, x_n, \alpha]}{f[x_n, x_n]}. \end{aligned} \quad (8.6.3)$$

De aceea, dacă  $x_n \rightarrow \alpha$ , atunci

$$\lim_{n \rightarrow \infty} \frac{x_{n+1} - \alpha}{(x_n - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)}$$

și ordinul de convergență al metodei lui Newton este 2, dacă  $f''(\alpha) \neq 0$ . Referitor la convergența locală a metodei lui Newton avem:

**Sursa MATLAB 8.1 Secant method for nonlinear equations in  $\mathbb{R}$** 

```

function [z,ni]=secant(f,x0,x1,ea,er,Nmax)
%SECANT - secant method in R
%input
%f - function
%x0,x1 - starting values
%ea,er - absolute and relative error, respectively
%Nmax - maximim number of iterations
%output
%z - approximate root
%ni - actual no. of iterations

if nargin<6, Nmax=50; end
if nargin<5, er=0; end
if nargin<4, ea=1e-3; end
xv=x0; fv=f(xv); xc=x1; fc=f(xc);
for k=1:Nmax
    xn=xc-fc*(xc-xv)/(fc-fv);
    if abs(xn-xc)<ea+er*xn %success
        z=xn;
        ni=k;
        return
    end
    %prepare next iteration
    xv=xc; fv=fc; xc=xn; fc=feval(f,xn);
end
%failure
error('maximum iteration number exceeded')

```

**Teorema 8.6.1.** Fie  $\alpha$  o rădăcină simplă a ecuației  $f(x) = 0$  și  $I_\varepsilon = \{x \in \mathbb{R} : |x - \alpha| \leq \varepsilon\}$ . Presupunem că  $f \in C^2[I_\varepsilon]$ . Definim

$$M(\varepsilon) = \max_{\substack{s \in I_\varepsilon \\ t \in I_\varepsilon}} \left| \frac{f''(s)}{2f'(t)} \right|. \quad (8.6.4)$$

Dacă  $\varepsilon$  este suficient de mic astfel încât

$$2\varepsilon M(\varepsilon) < 1, \quad (8.6.5)$$

atunci pentru orice  $x_0 \in I_\varepsilon$ , metoda lui Newton este bine definită și converge pătratic către singura rădăcină  $\alpha \in I_\varepsilon$ .

Criteriul de oprire pentru metoda lui Newton

$$|x_n - x_{n-1}| < \varepsilon$$

se bazează pe următoarea propoziție:

**Propoziția 8.6.2.** Fie  $(x_n)$  sirul de aproximante generat prin metoda lui Newton. Dacă  $\alpha$  este o rădăcină simplă din  $[a, b]$ ,  $f \in C^2[a, b]$  și metoda este convergentă, atunci există un  $n_0 \in \mathbb{N}$  astfel încât

$$|x_n - \alpha| \leq |x_n - x_{n-1}|, \quad n > n_0.$$

*Demonstrație.* Vom arăta întâi că

$$|x_n - \alpha| \leq \frac{1}{m_1} |f(x_n)|, \text{ unde } m_1 := \inf_{x \in [a, b]} |f'(x)|. \quad (8.6.6)$$

Utilizând teorema lui Lagrange,  $f(\alpha) - f(x_n) = f'(\xi)(\alpha - x_n)$ , cu  $\xi \in (\alpha, x_n)$  (sau  $(x_n, \alpha)$ ). Din relațiile  $f(\alpha) = 0$  și  $|f'(x)| \geq m_1$  pentru  $x \in (a, b)$  rezultă că  $|f(x_n)| \geq m_1 |\alpha - x_n|$ , adică chiar (8.6.6).

Pe baza formulei lui Taylor avem

$$f(x_n) = f(x_{n-1}) + (x_n - x_{n-1})f'(x_{n-1}) + \frac{1}{2}(x_n - x_{n-1})^2 f''(\mu), \quad (8.6.7)$$

cu  $\mu \in (x_{n-1}, x_n)$  sau  $\mu \in (x_n, x_{n-1})$ . Înținând cont de modul de obținere a unei aproximății în metoda lui Newton, avem  $f(x_{n-1}) + (x_n - x_{n-1})f'(x_{n-1}) = 0$  și din (8.6.7) se obține

$$|f(x_n)| = \frac{1}{2}(x_n - x_{n-1})^2 |f''(\mu)| \leq \frac{1}{2}(x_n - x_{n-1})^2 \|f''\|_\infty,$$

iar pe baza formulei (8.6.6) rezultă că

$$|\alpha - x_n| \leq \frac{\|f''\|_\infty}{2m_1} (x_n - x_{n-1})^2.$$

Cum am presupus că metoda este convergentă, există un  $n_0$  natural cu proprietatea că

$$\frac{\|f''\|_\infty}{2m_1} (x_n - x_{n-1}) < 1, \quad n > n_0$$

și deci

$$|x_n - \alpha| \leq |x_n - x_{n-1}|, \quad n > n_0.$$

□

Interpretarea geometrică a metodei lui Newton apare în figura 8.3, iar o implementare în sursa MATLAB 8.2.

Alegerea valorii de pornire este, în general, o problemă dificilă. În practică, se alege o valoare, iar dacă după un număr maxim fixat de iterații nu s-a obținut precizia dorită, testată prin unul din criteriile uzuale, se încearcă cu altă valoare de pornire. De exemplu, dacă rădăcina este izolată într-un interval  $[a, b]$  și  $f''(x) \neq 0$ ,  $x \in (a, b)$ , un criteriu de alegere este  $f(x_0)f''(x_0) > 0$ .

## 8.7. Metoda aproximățiilor succesive

Adesea, în aplicații, ecuațiile neliniare apar sub forma unei *probleme de punct fix*: să se determine  $x$  astfel încât

$$x = \varphi(x). \quad (8.7.1)$$

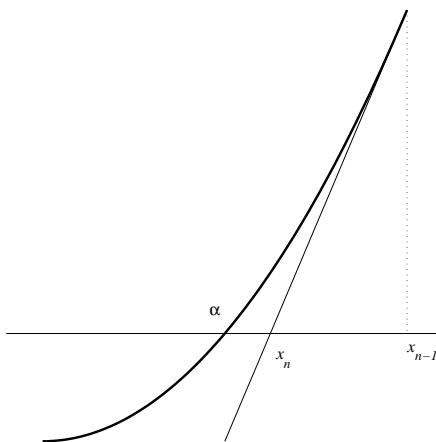


Figura 8.3: Metoda lui Newton

**Sursa MATLAB 8.2 Metoda lui Newton pentru ecuații neliniare în  $\mathbb{R}$** 

```

function [z,ni]=Newtons(f,fd,x0,ea,er,Nmax)
%NEWTONS - metoda lui Newton pentru ecuatii in R
%Intrare
%f - functia
%fd - derivata
%x0 - valoarea de pornire
%ea,er - eroarea absoluta, respectiv relativa
%Nmax - numar maxim de iteratii
%Iesire
%z - aproximatia solutiei
%ni - numar de iteratii

if nargin<6, Nmax=50; end
if nargin<5, er=0; end
if nargin<4, ea=1e-3; end
xv=x0;
for k=1:Nmax
    xc=xv-f(xv)/fd(xv);
    if abs(xc-xv)<ea+er*xc %succes
        z=xc; ni=k;
        return
    end
    xv=xc; %pregatesc iteratia urmatoare
end
%esec
error('s-a depasit numarul maxim de iteratii')

```

Un număr  $\alpha$  ce satisface această ecuație se numește punct fix al lui  $\varphi$ . Orice ecuație  $f(x) = 0$  se poate scrie (în multe moduri diferite) în forma echivalentă (8.7.1). De exemplu, dacă  $f'(x) \neq 0$  în intervalul de interes, putem lua

$$\varphi(x) = x - \frac{f(x)}{f'(x)}. \quad (8.7.2)$$

Dacă  $x_0$  este o aproximare inițială a unui punct fix  $\alpha$  a lui (8.7.1), atunci metoda aproximăriilor succesive generează un sir de aproximări

$$x_{n+1} = \varphi(x_n). \quad (8.7.3)$$

Dacă acest sir converge și  $\varphi$  este continuă, atunci sirul converge către un punct fix a lui  $\varphi$ . De notat că (8.7.3) este chiar metoda lui Newton dacă  $\varphi$  este dată de (8.7.2). Astfel, metoda lui Newton poate fi privită ca o iterare de tip punct fix, dar nu și metoda secantei. Pentru o iterare de forma (8.7.3), presupunând că  $x_n \rightarrow \alpha$  când  $n \rightarrow \infty$ , ordinul de convergență este ușor de determinat. Să presupunem că în punctul fix  $\alpha$  avem

$$\varphi'(\alpha) = \varphi''(\alpha) = \dots = \varphi^{(p-1)}(\alpha) = 0, \quad \varphi^p(\alpha) \neq 0. \quad (8.7.4)$$

Presupunem că  $\varphi \in C^p$  pe o vecinătate a lui  $\alpha$ . Avem atunci, conform teoremei lui Taylor

$$\begin{aligned} \varphi(x_n) &= \varphi(\alpha) + (x_n - \alpha)\varphi'(\alpha) + \dots + \frac{(x_n - \alpha)^{p-1}}{(p-1)!}\varphi^{(p-1)}(\alpha) \\ &\quad + \frac{(x_n - \alpha)^p}{p!}\varphi^{(p)}(\xi_n) = \varphi(\alpha) + \frac{(x_n - \alpha)^p}{p!}\varphi^{(p)}(\xi_n), \end{aligned}$$

unde  $\xi_n \in (\alpha, x_n)$  (sau  $(x_n, \alpha)$ ). Deoarece  $\varphi(x_n) = x_{n+1}$  și  $\varphi(\alpha) = \alpha$  obținem

$$\frac{x_{n+1} - \alpha}{(x_n - \alpha)^p} = \frac{1}{p!}\varphi^{(p)}(\xi_n).$$

Când  $x_n \rightarrow \alpha$ , deoarece  $\xi_n$  este între  $x_n$  și  $\alpha$ , deducem pe baza continuității că

$$\lim_{n \rightarrow \infty} \frac{x_{n+1} - \alpha}{(x_n - \alpha)^p} = \frac{1}{p!}\varphi^{(p)}(\alpha) \neq 0. \quad (8.7.5)$$

Aceasta ne arată că ordinul de convergență este exact  $p$  și eroarea asimptotică este

$$c = \frac{1}{p!}\varphi^{(p)}(\alpha). \quad (8.7.6)$$

Combinând aceasta cu condiția uzuală de convergență locală se obține:

**Teorema 8.7.1.** *Fie  $\alpha$  un punct fix al lui  $\varphi$  și  $I_\varepsilon = \{x \in \mathbb{R} : |x - \alpha| \leq \varepsilon\}$ . Presupunem că  $\varphi \in C^p[I_\varepsilon]$  și satisface (8.7.4). Dacă*

$$M(\varepsilon) := \max_{t \in I_\varepsilon} |\varphi'(t)| < 1 \quad (8.7.7)$$

atunci iterarea (8.7.3) converge către  $\alpha$ ,  $\forall x_0 \in I_\varepsilon$ . Ordinul de convergență este  $p$ , iar eroarea asimptotică este dată de (8.7.6).

## 8.8. Metoda lui Newton pentru rădăcini multiple

Dacă  $\alpha$  este o rădăcină multiplă de ordinul  $m$ , atunci ordinul de convergență a metodei lui Newton este doar 1. Într-adevăr, fie

$$\varphi(x) = x - \frac{f(x)}{f'(x)}.$$

Deoarece

$$\varphi'(x) = \frac{f(x)f''(x)}{[f'(x)]^2},$$

procesul va fi convergent dacă  $\varphi'(\alpha) = 1 - 1/m < 1$ .

O modalitate de a evita aceasta este să rezolvăm ecuația

$$u(x) := \frac{f(x)}{f'(x)} = 0$$

care are aceleași rădăcini ca și  $f$ , dar simple. Metoda lui Newton pentru problema modificată are forma

$$x_{k+1} = x_k - \frac{u(x_k)}{u'(x_k)} = \frac{f(x_k)f'(x_k)}{[f'(x_k)]^2 - f(x_k)f''(x_k)}. \quad (8.8.1)$$

Deoarece  $\alpha$  este o rădăcină simplă a lui  $u$ , convergența lui (8.8.1) este pătratică. Singurul dezavantaj teoretic al lui (8.8.1) este derivata a două necesară suplimentar și complexitatea mai mare a calculului lui  $x_{k+1}$  din  $x_k$ . În practică aceasta este o slăbiciune, deoarece numitorul lui (8.8.1) poate lua valori foarte mici în vecinătatea lui  $\alpha$  când  $x_k \rightarrow \alpha$ .

Convergența pătratică a metodei lui Newton se poate realiza nu numai prin modificarea problemei, ci și prin modificarea metodei. În vecinătatea unei soluții multiple de ordinul  $m$ ,  $\alpha$ , avem

$$f(x) = (x - \alpha)^m \varphi(x) \approx (x - \alpha)^m \cdot c, \quad (8.8.2)$$

de unde rezultă

$$\frac{f(x)}{f'(x)} \approx \frac{x - \alpha}{m} \Rightarrow \alpha \approx x - m \frac{f(x)}{f'(x)}.$$

Metoda modificată corespunzătoare

$$x_{k+1} := x_k - m \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \dots \quad (8.8.3)$$

converge pătratic către rădăcina multiplă de ordinul  $m$  când se întrebunează o valoare corectă a lui  $m$  în (8.8.3). Eficiența variantei (8.8.3) a metodei lui Newton depinde de utilizarea unei valori de aproximare bune pentru  $m$ , dacă această valoare nu este cunoscută din alte surse.

În ipoteza

$$|x_k - \alpha| < |x_{k-1} - \alpha| \wedge |x_k - \alpha| < |x_{k-2} - \alpha|$$

putem înlocui în (8.8.2)  $\alpha$  prin  $x_k$

$$\begin{aligned} f(x_{k-1}) &\approx (x_{k-1} - x_k)^m \cdot c, \\ f(x_{k-2}) &\approx (x_{k-2} - x_k)^m \cdot c. \end{aligned}$$

În continuare se obține  $m$ :

$$m \approx \frac{\log [f(x_{k-1})/f(x_{k-2})]}{\log [(x_{k-1} - x_k)/(x_{k-2} - x_k)]}.$$

Această valoare poate fi utilizată în (8.8.3).

## 8.9. Ecuații algebrice

Există multe metode special concepute pentru a rezolva ecuații algebrice. Aici vom descrie numai metoda lui Newton aplicată în acest context, concentrându-ne asupra unui mod eficient de a evalua simultan valoarea polinomului și a primei derivate.

**Metoda lui Newton aplicată ecuațiilor algebrice.** Considerăm o ecuație algebraică de grad  $d$

$$f(x) = 0, \quad f(x) = x^d + a_{d-1}x^{d-1} + \cdots + a_0, \quad (8.9.1)$$

în care coeficientul dominant se presupune (fără a restrânge generalitatea) să fie egal cu 1 și unde putem presupune, fără a restrânge generalitatea că  $a_0 \neq 0$ . Pentru simplitate, vom presupune că toți coeficienții sunt reali. Pentru a aplica metoda lui Newton ecuației (8.9.1) este nevoie de a metodă bună de evaluare a polinomului și derivatei.

Schema lui Horner este bună pentru așa ceva:

```

bd := 1; cd := 1;
for k = d - 1 downto 1 do
    bk := tbk+1 + ak;
    ck := tcck+1 + bk;
end for
b0 := tb1 + a0;
```

Atunci  $f(t) = b_0$ ,  $f'(t) = c_1$ .

Deci procedăm astfel:

Se aplică metoda lui Newton, calculând simultan  $f(x_n)$  și  $f'(x_n)$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Se aplică apoi metoda lui Newton polinomului  $\frac{f(x)}{x - \alpha}$ . Pentru rădăcini complexe se începe cu  $x_0$  complex și toate calculele se fac în aritmetică complexă. Este posibil să se împartă cu factori pătratici și să se folosească aritmetică reală – se ajunge astfel la metoda lui Bairstow. Folosind metoda aceasta de scădere a gradului erorile pot fi mari. O modalitate de îmbunătățire este de a utiliza rădăcinile astfel calculate ca aproximări initiale și a aplica metoda lui Newton polinomului original.

## 8.10. Metoda lui Newton în $\mathbb{R}^n$

Metoda lui Newton este ușor de generalizat la sisteme neliniare

$$F(x) = 0, \quad (8.10.1)$$

unde  $F : \Omega \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$ , iar  $x, F(x) \in \mathbb{R}^n$ . Sistemul (8.10.1) se scrie pe componente

$$\begin{cases} F_1(x_1, \dots, x_n) = 0 \\ \vdots \\ F_n(x_1, \dots, x_n) = 0 \end{cases}$$

Fie  $F'(x^{(k)})$  jacobianul lui  $F$  în  $x^{(k)}$ :

$$J := F'(x^{(k)}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}(x^{(k)}) & \cdots & \frac{\partial F_1}{\partial x_n}(x^{(k)}) \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1}(x^{(k)}) & \cdots & \frac{\partial F_n}{\partial x_n}(x^{(k)}) \end{bmatrix}. \quad (8.10.2)$$

Cantitatea  $1/f'(x)$  se înlocuiește în acest caz cu inversa jacobianului în  $x^{(k)}$ :

$$x^{(k+1)} = x^{(k)} - [F'(x^{(k)})]^{-1} F(x^{(k)}). \quad (8.10.3)$$

Scriem iterată sub forma

$$x^{(k+1)} = x^{(k)} + w^{(k)}. \quad (8.10.4)$$

Se observă că  $w_k$  este soluția sistemului de  $n$  ecuații liniare cu  $n$  necunoscute

$$F'(x^{(k)})w^{(k)} = -F(x^{(k)}). \quad (8.10.5)$$

Este mai eficient și mai convenabil ca, în loc să inversăm jacobianul la fiecare pas, să rezolvăm sistemul (8.10.5) și să folosim iterată în forma (8.10.4).

**Teorema 8.10.1.** *Fie  $\alpha$  o soluție a ecuației  $F(x) = 0$  și presupunem că în bila închisă  $B(\delta) \equiv \{x : \|x - \alpha\| \leq \delta\}$  matricea Jacobi a lui  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  există, este nesingulară și satisfac condiția Lipschitz*

$$\|F'(x) - F'(y)\|_{\infty} \leq c\|x - y\|_{\infty}, \quad \forall x, y \in B(\delta), \quad c > 0.$$

Punem  $\gamma = c \max \{\|F'(x)\|_{\infty} : \|\alpha - x\|_{\infty} \leq \delta\}$  și  $0 < \varepsilon < \min\{\delta, \gamma^{-1}\}$ . Atunci pentru orice aproximare inițială  $x^{(0)} \in B(\varepsilon) := \{x : \|x - \alpha\|_{\infty} \leq \varepsilon\}$  metoda lui Newton este convergentă, iar vectorii  $e^{(k)} := \alpha - x^{(k)}$  satisfac următoarele inegalități:

$$(a) \|e^{(k+1)}\|_{\infty} \leq \gamma\|e^{(k)}\|_{\infty}^2$$

$$(b) \|e^{(k)}\|_{\infty} \leq \gamma^{-1}(\gamma\|e^{(0)}\|_{\infty})^{2^k}.$$

*Demonstrație.* Dacă  $F'$  este continuă pe segmentul ce unește punctele  $x, y \in \mathbb{R}^n$ , conform teoremei lui Lagrange

$$F(x) - F(y) = J_k(x - y),$$

unde

$$J_k = \begin{bmatrix} \frac{\partial F_1}{\partial x_1}(\xi_1) & \dots & \frac{\partial F_1}{\partial x_n}(\xi_1) \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1}(\xi_n) & \dots & \frac{\partial F_n}{\partial x_n}(\xi_n) \end{bmatrix} \Rightarrow$$

$$\begin{aligned} e^{(k+1)} &= e^{(k)} - [F'(x^{(k)})]^{-1}(F(\alpha) - F(x^{(k)})) = e^{(k)} - [F'(x^{(k)})]^{-1} J_k e^{(k)} \\ &= [F'(x^{(k)})]^{-1}(F'(x^{(k)}) - J_k) e^{(k)} \end{aligned}$$

și de aici rezultă imediat (a). Din condiția Lipschitz

$$\|F'(x^{(k)}) - J_k\|_{\infty} \leq c \max_{j=1,n} \|x^{(k)} - \xi^{(j)}\| \leq c\|x^{(k)} - \alpha\|.$$

Deci, dacă  $\|\alpha - x^{(k)}\|_{\infty} \leq \varepsilon$ , atunci  $\|\alpha - x^{(k+1)}\|_{\infty} \leq (\gamma\varepsilon)\varepsilon \leq \varepsilon$ . Deoarece (a) este adevărată pentru orice  $k$ , se obține (b) imediat.  $\square$

Funcția MATLAB 8.3 dă o implementare a metodei lui Newton care funcționează și pentru ecuații scalare și pentru sisteme.

**Sursa MATLAB 8.3 Metoda lui Newton în  $\mathbb{R}$  și  $\mathbb{R}^n$** 

```

function [z,ni]=Newton(f,fd,x0,ea,er,nmax)
%NEWTON - metoda lui Newton pentru ecuatii neliniare
% in R si R^n
%apel [z,ni]=Newton(f,fd,x0,ea,er,nmax)
%Intrare
%f - functia
%fd - derivata
%x0 - aproximatia initiala
%ea - eroarea absoluta (implicit 1e-3)
%er - eroarea relativa (implicit 0)
%nmax - numarul maxim de iteratii (implicit 50)
%Iesire
%z - aproximatia radacinii
%ni - numarul de iteratii

if nargin < 6, nmax=50; end
if nargin < 5, er=0; end
if nargin < 4, ea=1e-3; end
xp=x0(:); %x precedent
for k=1:nmax
    xc=xp-feval(fd,xp)\ feval(f,xp);
    if norm(xc-xp,inf)<ea+er*norm(xc,inf)
        z=xc; %succes
        ni=k;
        return
    end
    xp=xc;
end
error('S-a depasit numarul maxim de iteratii');

```

**Exemplul 8.10.2.** Să considerăm sistemul neliniar:

$$\begin{aligned} 3x_1 - \cos(x_1 x_2) - \frac{1}{2} &= 0, \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 &= 0, \\ e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} &= 0 \end{aligned}$$

Funcția și jacobianul în MATLAB se dau în continuare :

```

function y=fs3(x)
y=[3*x(1)-cos(x(2)*x(3))-1/2;...
    x(1)^2-81*(x(2)+0.1)^2+sin(x(3))+1.06;...
    exp(-x(1)*x(2))+20*x(3)+(10*pi-3)/3];

function y=fs3d(x)
y=[3,x(3)*sin(x(2)*x(3)), x(2)*sin(x(2)*x(3));...

```

```

2*x(1), -162*(x(2)+0.1), cos(x(3));...
-x(2)*exp(-x(1)*x(2)), -x(1)*exp(-x(1)*x(2)), 20];

```

Iată și un exemplu de apel:

```
>> [z, ni] = Newton(@fs3, @fs3d, x0, 1e-9);
```

Aplicând metoda lui Newton acestui sistem cu valoarea de pornire  $x^{(0)} = [0.1, 0.1, -0.1]^T$  se obțin rezultatele din tabela 8.1. Precizia  $10^{-9}$  se atinge după 5 iterări. ◇

$k$	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\ x^{(k)} - x^{(k-1)}\ _\infty$
0	0.1000000000	0.1000000000	-0.1000000000	—
1	0.4998696729	0.0194668485	-0.5215204719	0.42152
2	0.5000142402	0.0015885914	-0.5235569643	0.0178783
3	0.5000001135	0.0000124448	-0.5235984501	0.00157615
4	0.5000000000	0.0000000008	-0.5235987756	1.2444e-005
5	0.5000000000	0.0000000000	-0.5235987756	7.75786e-010
6	0.5000000000	-0.0000000000	-0.5235987756	1.11022e-016

Tabela 8.1: Rezultatele din exemplul 8.10.2

## 8.11. Metode quasi-Newton

O slabiciune semnificativă a metodei lui Newton pentru rezolvarea sistemelor de ecuații neliniare este necesitatea ca la fiecare pas să calculăm matricea jacobiană și să rezolvăm un sistem  $n \times n$  cu această matrice. Pentru a ilustra dimensiunile unei astfel de slabiciuni, să evaluăm volumul de calcule asociat cu o iterare a metodei lui Newton. Matricea jacobiană asociată unui sistem de  $n$  ecuații neliniare scris în forma  $F(x) = 0$  necesită evaluarea celor  $n^2$  derivate parțiale ale celor  $n$  funcții componente ale lui  $F$ . În cele mai multe situații, evaluarea exactă a derivatelor parțiale este neconvenabilă și de multe ori imposibilă. Efortul total pentru o iterare a metodei lui Newton va fi de cel puțin  $n^2 + n$  evaluări de funcții scalare ( $n^2$  pentru evaluarea jacobianului și  $n$  pentru evaluarea lui  $F$ ) și  $O(n^3)$  operații aritmetice pentru a rezolva sistemul liniar. Acest volum de calcule este prohibitiv, exceptând valori mici ale lui  $n$  și funcții scalare ușor de evaluat. Este firesc ca atenția să fie îndreptată spre reducerea numărului de evaluări și evitarea rezolvării unui sistem liniar la fiecare pas.

La metoda secantei aproximarea  $x^{(k+1)}$  se obține ca soluție a ecuației liniare

$$\bar{l}_k = f(x^{(k)}) + (x - x^{(k)}) \frac{f(x_{(k)} + h_k) - f(x_{(k)})}{h_k} = 0.$$

Aici funcția  $\bar{l}_k$  poate fi interpretată în două moduri:

1. ca aproximare a ecuației tangentei

$$l_k(x) = f(x^{(k)}) + (x - x^{(k)})f'(x^{(k)});$$

2. ca interpolare liniară între punctele  $x^{(k)}$  și  $x^{(k+1)}$ .

Se pot obține diverse generalizări ale metodei secantei la sisteme de ecuații neliniare în funcție de modul în care se interpretează  $\bar{l}_k$ . Prima interpretare conduce la metode de tip Newton discretizate, iar a doua la metode bazate pe interpolare.

Metodele de tip Newton discretizate se obțin dacă în metoda lui Newton (8.10.3)  $F'(x)$  se înlocuiește cu o aproximare discretă  $A(x, h)$ . Derivatele parțiale din matricea jacobiană (8.10.2) se vor înlocui prin diferențele divizate

$$A(x, h)e_i := [F(x + h_i e_i) - F(x)]/h_i, \quad i = \overline{1, n}, \quad (8.11.1)$$

unde  $e_i \in \mathbb{R}^n$  este al  $i$ -lea vector al bazei canonice și  $h_i = h_i(x)$  este mărimea pasului de discretizare. O alegere posibilă a pasului este de exemplu

$$h_i := \begin{cases} \varepsilon|x_i|, & \text{dacă } x_i \neq 0 \\ \varepsilon, & \text{altfel,} \end{cases}$$

cu  $\varepsilon := \sqrt{\text{eps}}$ , unde eps este epsilon-ul mașinii.

### 8.11.1. Interpolare liniară

La interpolare fiecare dintre planele tangente se înlocuiește cu un (hiper)plan care interpolează funcțiile componente  $F_i$  ale lui  $F$  în  $n+1$  puncte date  $x^{k,j}$ ,  $j = \overline{0, n}$ , într-o vecinătate a lui  $x^{(k)}$ , adică se determină vectorii  $a^{(i)}$  și scalarii  $\alpha_i$ , astfel încât pentru

$$L_i(x) = \alpha_i + a^{(i)T}x, \quad i = \overline{1, n} \quad (8.11.2)$$

are loc

$$L_i(x^{k,j}) = F_i(x^{k,j}), \quad i = \overline{1, n}, \quad j = \overline{0, n}.$$

Următoarea aproximare  $x^{(k+1)}$  se obține ca punct de intersecție între cele  $n$  hiperplane (8.11.2) din  $\mathbb{R}^{n+1}$  cu hiperplanul  $y = 0$ . Aproximanta  $x^{(k+1)}$  rezultă ca soluție a sistemului de ecuații liniare

$$L_i(x) = 0, \quad i = \overline{1, n}. \quad (8.11.3)$$

În funcție de alegerea punctelor de interpolare se obțin diferite metode, dintre care cele mai cunoscute sunt metoda lui Brown și metoda lui Brent. Metoda lui Brown combină aproximarea lui  $F'$  și rezolvarea sistemului prin eliminare gaussiană. În metoda lui Brent se întrebunează la rezolvarea sistemului metoda QR. Ambele metode aparțin unei clase de metode, care, la fel ca metoda lui Newton, converg pătratic, dar au nevoie doar de  $(n^2 + 3n)/2$  evaluări de funcții pe iterație.

Într-un studiu comparativ, Moré și Cosnard [47] au ajuns la concluzia că metoda lui Brent este adeseori de preferat metodei lui Brown și că pentru sisteme de ecuații neliniare, la care evaluarea lui  $F$  necesită un efort mai mic, metoda lui Newton discretizată este cea mai eficientă metodă de rezolvare.

### 8.11.2. Metode de modificare

Din punct de vedere al efortului de calcul, sunt deosebit de convenabile metodele în care la fiecare pas se întrebunează o aproximare  $A_k$  a lui  $F'(x^{(k)})$ , care se obține din  $A_{k-1}$  printr-o modificare de rang 1, adică prin adăugarea unei matrice de rang 1:

$$A_{k+1} := A_k + u^{(k)} \left[ v^{(k)} \right]^T, \quad u^{(k)}, v^{(k)} \in \mathbb{R}^n, \quad k = 0, 1, 2, \dots$$

Pe baza formulei Sherman-Morrison (vezi [16])

$$\left( A + uv^T \right)^{-1} = A^{-1} - \frac{1}{1 + v^T A^{-1} u} A^{-1} u v^T A^{-1},$$

pentru  $B_{k+1} := A_{k+1}^{-1}$  are loc relația de recurență

$$B_{k+1} = B_k - \frac{B_k u^{(k)} \begin{bmatrix} v^{(k)} \end{bmatrix}^T B_k}{1 + [v^{(k)}]^T B_k u^{(k)}}, \quad k = 0, 1, 2, \dots,$$

atât timp cât  $1 + [v^{(k)}]^T B_k u^{(k)} \neq 0$ . Astfel, nu mai este necesară rezolvarea unui sistem liniar la fiecare pas; ea se înlocuiește cu înmulțiri matrice-vector, ceea ce corespunde unei reduceri a efortului de calcul de la  $O(n^3)$  la  $O(n^2)$ . Acest avantaj va fi plătit prin aceea că nu vom mai avea o convergență pătratică ca la metoda lui Newton, ci doar una superliniară:

$$\lim_{k \rightarrow \infty} \frac{\|x^{(k+1)} - \alpha\|}{\|x^{(k)} - \alpha\|} = 0. \quad (8.11.4)$$

În metoda lui Broyden alegerea vectorilor  $u^{(k)}$  și  $v^{(k)}$  are loc după principiul aproximăției secantei. În cazul scalar aproxiarea  $a_k \approx f'(x^{(k)})$  se face unic prin

$$a_{k+1}(x^{(k+1)} - x^{(k)}) = f(x^{(k+1)}) - f(x^{(k)}).$$

Pentru  $n > 1$ , din contră, aproximarea

$$A_{k+1}(x^{(k+1)} - x^{(k)}) = F(x^{(k+1)}) - F(x^{(k)}) \quad (8.11.5)$$

(aşa numita ecuație quasi-Newton) nu mai este unic determinată; orice altă matrice de forma

$$\bar{A}_{k+1} := A_{k+1} + pq^T$$

cu  $p, q \in \mathbb{R}^n$  și  $q^T(x^{(k+1)} - x^{(k)}) = 0$  verifică de asemenea ecuația (8.11.5). Pe de altă parte,

$$y_k := F(x^{(k)}) - F(x^{(k-1)}) \text{ și } s_k := x^{(k)} - x^{(k-1)}$$

conțin numai informații despre derivata parțială a lui  $F$  în direcția  $s_k$ , dar nici o informație în direcții ortogonale pe  $s_k$ . Pe această direcție trebuie ca efectul lui  $A_{k+1}$  și  $A_k$  să coincidă

$$A_{k+1}q = A_kq, \quad \forall q \in \{v : v \neq 0, v^T s_k = 0\}. \quad (8.11.6)$$

Pornind de la prima aproximare  $A_0 \approx F'(x^{(0)})$ , se generează sirul  $A_1, A_2, \dots$  utilizând formulele (8.11.5) și (8.11.6) (Broyden [9], Dennis și Moré [16]).

Pentru sirul  $B_0 = A_0^{-1} \approx [F(x^{(0)})]^{-1}, B_1, B_2, \dots$  cu ajutorul formulei Sherman-Morisson se obține relația de recurență

$$B_{k+1} := B_k + \frac{(s_{k+1} - B_k y_{k+1}) s_{k+1}^T B_k}{s_{k+1}^T B_k y_{k+1}}, \quad k = 0, 1, 2, \dots$$

care necesită doar înmulțiri matrice vector și a cărei complexitate este doar  $O(n^2)$ . Cu ajutorul matricelor  $B_k$  se poate defini metoda lui Broyden prin

$$x^{(k+1)} := x^{(k)} - B_k F(x^{(k)}), \quad k = 0, 1, 2, \dots$$

Această metodă converge superliniar în sensul lui (8.11.4), dacă pașii  $s_k$  se apropiie asymptotic (când  $k \rightarrow \infty$ ) de vectorii de actualizare (corecție) ai metodei lui Newton. Se poate recunoaște în aceasta semnificația centrală a principiului linearizării locale la rezolvarea ecuațiilor neliniare.

Funcția MATLAB 8.4 dă o implementare a metodei lui Broyden.

**Sursa MATLAB 8.4 Metoda lui Broyden pentru sisteme neliniare**

```
function [z,ni]=Broyden1(f,fd,x0,ea,er,nmax)
%BROYDEN1 - metoda lui Broyden pentru sisteme neliniare
%apel [z,ni]=Broyden1(f,x0,ea,er,nmax)
%Intrare
%f - functia
%fd - derivata
%x0 - aproximatia initiala
%ea - eroarea absoluta
%er - eroarea relativa
%nmax - numarul maxim de iteratii
%Iesire
%z - aproximatia radacinii
%ni - numarul de iteratii

if nargin < 6, nmax=50; end
if nargin < 5, er=0; end
if nargin < 4, ea=1e-3; end
x=zeros(length(x0),nmax+1);
F=x;
x(:,1)=x0(:);
F(:,1)=f(x(:,1));
B=inv(fd(x));
x(:,2)=x(:,1)+B*F(:,1);
for k=2:nmax
    F(:,k)=f(x(:,k));
    y=F(:,k)-F(:,k-1); s=x(:,k)-x(:,k-1);
    B=B+((s-B*y)*s'*B)/(s'*B*y);
    x(:,k+1)=x(:,k)-B*F(:,k);
    if norm(x(:,k+1)-x(:,k),inf)<ea+er*norm(x(:,k+1),inf)
        z=x(:,k+1); %succes
        ni=k;
        return
    end
end
error('S-a depasit numarul maxim de iteratii');
```

---

**Exemplul 8.11.1.** Sistemul neliniar:

$$\begin{aligned} 3x_1 - \cos(x_1 x_2) - \frac{1}{2} &= 0, \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 &= 0, \\ e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} &= 0 \end{aligned}$$

a fost rezolvat în exemplul 8.10.2 prin metoda lui Newton. Aplicând acestui sistem metoda lui Broyden cu aceeași funcție și același jacobian ca în exemplul 8.10.2 și cu valoarea de pornire  $x^{(0)} = [0.1, 0.1, -0.1]^T$  se obțin rezultatele din tabela 8.2. Iată și un exemplu de apel:

```
>> [z, ni] = Broyden1(@fs3, @fs3d, x0, 1e-9);
Precizia  $10^{-9}$  se atinge după 8 iterații. ◇
```

$k$	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$\ x^{(k)} - x^{(k-1)}\ _\infty$
0	0.1000000000	0.1000000000	-0.1000000000	—
1	-0.2998696729	0.1805331515	0.3215204719	0.42152
2	0.5005221618	0.0308676159	-0.5197695490	0.84129
3	0.4999648231	0.0130913099	-0.5229213211	0.0177763
4	0.5000140840	0.0018065260	-0.5235454326	0.0112848
5	0.5000009766	0.0001164840	-0.5235956998	0.00169004
6	0.5000000094	0.0000011019	-0.5235987460	0.000115382
7	0.5000000000	0.0000000006	-0.5235987756	1.10133e-006
8	0.5000000000	0.0000000000	-0.5235987756	5.69232e-010
9	0.5000000000	-0.0000000000	-0.5235987756	4.29834e-013

Tabela 8.2: Rezultatele din exemplul 8.11.1

## 8.12. Ecuații neliniare în MATLAB

MATLAB are puține rutine pentru determinarea rădăcinilor unei funcții de o variabilă. Dacă funcția este polinomială, am văzut că `roots(p)` returnează rădăcinile lui  $p$ , unde  $p$  este vectorul coeficienților ordonați descrescător după puterile variabilei.

Funcția `fzero` determină o rădăcină a unei funcții de o variabilă. Algoritmul folosit de `fzero` este o combinație de metode: înjumătățire, secantă și interpolare inversă pătratică [46]. Cea mai simplă variantă de apel a ei este  $x = fzero(f, x0)$ , cu  $x0$  scalar, care încearcă să găsească un zero al lui  $f$  în vecinătatea lui  $x0$ . De exemplu,

```
>> fzero('cos(x)-x', 0)
ans =
    0.7391
```

Precizia și modul de afișare a rezultatelor sunt controlate de un al treilea argument de intrare, structura `options`, care se poate seta cu ajutorul funcției `optimset`. Funcția `fzero` utilizează două câmpuri ale lui `options`: `TolX` care dă precizia (toleranța) și `Display` care specifică nivelul de raportare, cu valorile `off` pentru nici o ieșire, `iter` pentru ieșire la fiecare iterație, `final` pentru a afișa doar rezultatul final și `notify` pentru a afișa rezultatul numai dacă funcția nu converge (implicit). Pentru exemplul precedent, utilizând `Display` cu `final` obținem:

```
>> fzero('cos(x)-x', 0, optimset('Display','final'))
Zero found in the interval [-0.905097, 0.905097]
ans =
    0.7391
```

Argumentul de intrare  $x_0$  poate fi și un interval la ale cărui capete funcția să aibă valori de semne contrare. Un astfel de argument este util dacă funcția are singularități. Vom seta în continuare `Display` pe `final` cu comanda

```
os=optimset('Display','final');
```

Considerăm exemplul (mesajul nu apare):

```
>> [x,fval]=fzero('tan(x)-x',1,os)
...
x =
    1.5708
fval =
 -1.2093e+015
```

Cel de-al doilea argument de ieșire este valoarea funcției în zeroul calculat. Deoarece funcția  $f(x) = \tan x - x$  are o singularitate în  $\pi/2$  (vezi figura 8.4) vom da ca argument de pornire un interval ce conține un zero, dar fără singularități:

```
>> [x,fval]=fzero('tan(x)-x',[-1,1],os)
Zero found in the interval: [-1, 1].
x =
    0
fval =
    0
```

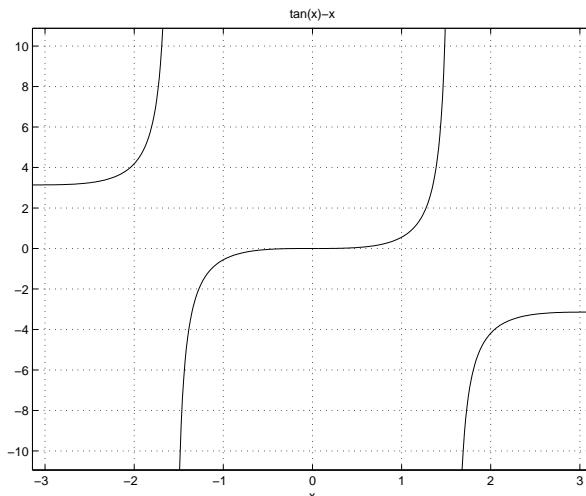


Figura 8.4: Singularitate a funcției  $f(x) = \tan x - x$ , evidențiată cu `ezplot('tan(x)-x', [-pi,pi]), grid`

Se pot transmite parametrii suplimentari  $p_1, p_2, \dots$  funcției  $f$  cu apeluri de forma

$x = fzero(f, x_0, options, p_1, p_2, \dots)$

Dacă se dorește ca  $options$  să aibă valori implicate se poate folosi pe poziția respectivă matricea vidă  $[]$ .

MATLAB nu are nici o funcție pentru rezolvarea sistemelor de ecuații neliniare. Totuși, se poate încerca rezolvarea unor astfel de sisteme prin minimizarea sumei pătratelor reziduurilor. Toolbox-ul Optimization conține un rezolvator de ecuații neliniare.

Funcția `fminsearch` căută un minim local al unei funcții reale de  $n$  variabile reale. O formă posibilă de apel este  $x = fminsearch(f, x_0, options)$ . Structura  $options$  este organizată la fel ca în cazul lui `fzero`, dar sunt folosite mai multe câmpuri. Amintim `MaxFunEvals` (numărul maxim de evaluări de funcții permise), `MaxIter` (numărul maxim de iterații permise), `TolFun` (precizia de terminare pentru valoarea funcției). Valoarea implicită pentru `TolX` și `TolFun` este  $1e-4$ .

**Exemplul 8.12.1.** Sistemul neliniar din exemplele 8.10.2 și 8.11.1, adică

$$\begin{aligned}f_1(x_1, x_2, x_3) &:= 3x_1 - \cos(x_1 x_2) - \frac{1}{2} = 0, \\f_2(x_1, x_2, x_3) &:= x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 = 0, \\f_3(x_1, x_2, x_3) &:= e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} = 0\end{aligned}$$

ar putea fi rezolvat încercând minimizarea sumei pătratelor membrilor stângi:

$$F(x_1, x_2, x_3) = [f_1(x_1, x_2, x_3)]^2 + [f_2(x_1, x_2, x_3)]^2 + [f_3(x_1, x_2, x_3)]^2.$$

Funcția de minimizat este dată în fișierul `fminob.m`:

```
function y = fminob(x)
y=(3*x(1)-cos(x(2)*x(3))-1/2)^2+(x(1)^2-81*(x(2)+0.1)^2+...
sin(x(3))+1.06)^2+(exp(-x(1)*x(2))+20*x(3)+(10*pi-3)/3)^2;
```

Vom alege vectorul de pornire  $x^{(0)} = [0.5, 0.5, 0.5]^T$  și precizia  $10^{-9}$  și pentru  $x$ . Comenzile MATLAB și rezultatul lor se dau mai jos

```
>> os=optimset('Display','final','TolX',1e-9,'TolFun',1e-9);
>> [xm,fval]=fminsearch(@fminob,[0.5,0.5,0.5]',os)
Optimization terminated:
the current x satisfies the termination criteria using
OPTIONS.TolX of 1.000000e-009 and F(X) satisfies the
convergence criteria using OPTIONS.TolFun of 1.000000e-009
xm =
    0.4999999959246
    0.00000000001815
   -0.52359877559440
fval =
    1.987081116616629e-018
```

Comparativ cu rezolvările bazate pe metoda lui Newton sau metoda lui Broyden, se constată că timpul de execuție este mai lung și precizia nu este la fel de bună (și datorită aspectului mai complicat al funcției obiectiv). Aproximanta obținută astfel poate fi folosită ca vector de pornire pentru metoda lui Newton. ◇

Funcția `fminsearch` se bazează pe varianta Nelder-Mead a algoritmului simplex. Algoritmul este lent, dar are avantajul că nu utilizează derive și este insensibil la discontinuități ale funcției. Toolbox-ul Optimization conține metode mai sofisticate de minimizare.

## Probleme

**Problema 8.1.** Găsiți primele 10 valori pozitive pentru care  $x = \operatorname{tg} x$ .

**Problema 8.2.** Investigați comportarea metodei lui Newton și a secantei pentru funcția

$$f(x) = \operatorname{sign}(x - a)\sqrt{|x - a|}.$$

**Problema 8.3 (Adaptată după [46]).** Considerăm polinomul

$$x^3 - 2x - 5.$$

Wallis a folosit acest exemplu pentru a prezenta metoda lui Newton în fața academiei franceze. El are o rădăcină reală în intervalul  $(2, 3)$  și o pereche de rădăcini complexe conjugate.

- (a) Utilizați Maple sau toolbox-ul Symbolic pentru a calcula rădăcinile. Rezultatele sunt urâte. Convertiți-le în valori numerice.
- (b) Determinați toate rădăcinile cu funcția `roots`.
- (c) Determinați rădăcina reală cu `fzero`.
- (d) Determinați toate rădăcinile cu metoda lui Newton (pentru cele complexe folosiți valori de pornire complexe).
- (e) Se poate utiliza metoda înjumătățirii sau a falsei poziții la determinarea unei rădăcini complexe? De ce sau de ce nu?

**Problema 8.4.** Să se rezolve numeric sistemele

$$\begin{aligned} f_1(x, y) &= 1 - 4x + 2x^2 - 2y^3 = 0 \\ f_2(x, y) &= -4 + x^4 + 4y + 4y^4 = 0, \end{aligned}$$

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 - x_2 + 0.25 = 0 \\ f_2(x_1, x_2) &= -x_1 + x_2^2 + 0.25 = 0, \end{aligned}$$

$$\begin{aligned} f_1(x_1, x_2) &= 2x_1 + x_2 - x_1 x_2 / 2 - 2 = 0 \\ f_2(x_1, x_2) &= x_1 + 2x_2^2 - \cos(x_2) / 2 - \frac{3}{2} = 0. \end{aligned}$$

**Problema 8.5.** Să se rezolve numeric sistemul

$$\begin{aligned} 9x^2 + 36y^2 + 4z^2 - 36 &= 0, \\ x^2 - 2y^2 - 20z &= 0, \\ x^2 - y^2 + z^2 &= 0 \end{aligned}$$

*Indicație.* Sunt patru soluții. Valori bune de pornire  $[\pm 1, \pm 1, 0]^T$ .

**Problema 8.6.** Să considerăm sistemul, inspirat dintr-un exemplu din industria chimică

$$f_i := \beta a_i^2 + a_i - a_{i-1} = 0.$$

Sistemul are  $n$  ecuații și  $n+2$  necunoscute. Vom lua  $a_0 = 5$ ,  $a_n = 0.5$  mol/litru. Să se rezolve sistemul pentru  $n = 10$  și valoarea de pornire  $\mathbf{x} = [1 : -0.1 : 0.1]'$ .



# CAPITOLUL 9

## Valori și vectori proprii

În acest capitol ne ocupăm de determinarea valorilor proprii ale unei matrice pătratice  $A \in \mathbb{R}^{n \times n}$  și vectorilor proprii corespunzători, adică a valorilor  $\lambda \in \mathbb{C}$  și vectorilor  $x \in \mathbb{C}^n \setminus \{0\}$  pentru care

$$Ax = \lambda x. \quad (9.0.1)$$

**Definiția 9.0.1.** Numărul  $\lambda \in \mathbb{C}$  se numește valoare proprie a matricei  $A \in \mathbb{R}^{n \times n}$ , dacă există un vector  $x \in \mathbb{C}^n \setminus \{0\}$  numit vector propriu astfel încât  $Ax = \lambda x$ .

**Observația 9.0.2.**

1. Cerința  $x \neq 0$  este importantă, căci vectorul nul este un vector propriu corespunzător oricărei valori proprii.
2. Chiar dacă  $A$  este reală, ea poate avea valori proprii complexe. În acest caz ele apar întotdeauna în perechi conjugate.
3. Problemele de vectori și valori proprii se numesc uneori prescurtat *probleme proprii*. ◊

### 9.1. Valori proprii și rădăcini ale polinoamelor

Orice problemă de calcul al valorilor proprii se poate reduce la calculul zerourilor unui polinom: valorile proprii ale unei matrice  $A \in \mathbb{R}^{n \times n}$  sunt rădăcinile *polinomului caracteristic*

$$p_A \lambda = \det(A - \lambda I), \quad \lambda \in \mathbb{C},$$

căci determinatul este nul exact atunci când sistemul  $(A - \lambda I)x = 0$  are o soluție nebanală, adică atunci când  $\lambda$  este o valoare proprie.

O metodă de rezolvare a problemelor proprii ar putea fi calculul polinomului caracteristic și apoi determinarea rădăcinilor. Natural, calculul unui determinant *în general* fiind o problemă complexă și

instabilă, transformarea matricei ar fi mai potrivită. Reciproc, problema găsirii rădăcinilor unui polinom poate fi formulată ca o problemă de determinare a valorilor proprii. Fie  $p \in \mathbb{P}_n$  un polinom cu coeficienți reali, pe care îl putem scrie (cu ajutorul rădăcinilor sale,  $z_1, \dots, z_n$ , eventual complexe) sub forma

$$p(x) = a_n x^n + \dots + a_0 = a_n(x - z_1) \dots (x - z_n), \quad a_n \in \mathbb{R}, \quad a_n \neq 0.$$

Pe spațiul vectorial  $\mathbb{P}_{n-1}$ , „înmulțirea modulo  $p$ ”

$$\mathbb{P}_{n-1} \ni q \rightarrow r \quad xq(x) = \alpha p(x) + r(x), \quad r \in \mathbb{P}_n \quad (9.1.1)$$

este o *transformare liniară* și deoarece

$$x^n = \frac{1}{a_n} p(x) - \sum_{j=0}^{n-1} \frac{a_j}{a_n} x^j, \quad x \in \mathbb{R},$$

vom reprezenta  $p$  relativ la baza  $1, x, \dots, x^{n-1}$  prin aşa-numita matrice *companion* a lui Frobenius (de dimensiune  $n \times n$ )

$$M = \begin{bmatrix} 0 & & & -\frac{a_0}{a_n} \\ 1 & 0 & & -\frac{a_1}{a_n} \\ \ddots & \ddots & \ddots & \vdots \\ & 1 & 0 & -\frac{a_{n-2}}{a_n} \\ & & 1 & -\frac{a_{n-1}}{a_n} \end{bmatrix}. \quad (9.1.2)$$

Fie  $v_j = (v_{jk} : k = \overline{1, n}) \in \mathbb{C}^n$ ,  $j = \overline{1, n}$  aleși astfel ca

$$\ell_j(x) = \frac{p(x)}{x - z_j} = a_n \prod_{k \neq j} (x - z_k) = \sum_{k=1}^n v_{jk} x^{k-1}, \quad j = \overline{1, n};$$

atunci

$$\sum_{k=1}^n (Mv_j - z_j v_j)_k x^{k-1} = x\ell_j(x) - z_j \ell_j(x) = (x - z_j)\ell_j(x) = p(x) \approx 0,$$

și cu aceasta  $Mv_j = z_j v_j$ ,  $j = \overline{1, n}$ .

*Valorile proprii ale lui  $M$  sunt deci rădăcini ale lui  $p$ .*

Matricea Frobenius dată de (9.1.2) este doar o modalitate dintre multe altele prin care se poate reprezenta „înmulțirea” din (9.1.1); orice altă bază a lui  $\mathbb{P}_{n-1}$  furnizează o matrice  $M$  ale cărei valori proprii să fie rădăcini ale lui  $p$ . Singurul mijloc auxiliar de manipulare a polinoamelor de care avem nevoie este realizarea unei „împărțiri cu rest”.

## 9.2. Terminologie și descompunere Schur

Așa cum ne arată exemplul

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad p_A(\lambda) = \lambda^2 + 1 = (\lambda + i)(\lambda - i),$$

o matrice reală poate avea valori proprii complexe. De aceea (cel puțin în teorie) este avantajos să ne ocupăm de matrice complexe  $A \in \mathbb{C}^{n \times n}$ .

**Definiția 9.2.1.** Două matrice  $A, B \in \mathbb{C}^{n \times n}$  se numesc similare dacă există o matrice  $T \in \mathbb{C}^{n \times n}$ , astfel încât

$$A = TBT^{-1}.$$

**Lema 9.2.2.** Dacă  $A, B \in \mathbb{C}^{n \times n}$  sunt similare, ele au aceleași valori proprii.

*Demonstrație.* Fie  $\lambda \in \mathbb{C}$  o valoare proprie a lui  $A = TBT^{-1}$  și  $x \in \mathbb{C}^n$  vectorul propriu corespunzător. Atunci avem

$$B(T^{-1}x) = T^{-1}ATT^{-1}x = T^{-1}Ax = \lambda T^{-1}x$$

și deci,  $\lambda$  este, de asemenea, valoare proprie a lui  $B$ .  $\square$

Din algebra liniară se cunoaște următorul rezultat important.

**Teorema 9.2.3 (Forma normală Jordan).** Orice matrice  $A \in \mathbb{C}^{n \times n}$  este similară cu o matrice

$$J = \begin{bmatrix} J_1 & & & \\ & \ddots & & \\ & & J_k & \end{bmatrix}, \quad J_\ell = \begin{bmatrix} \lambda_\ell & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & \\ & & & 1 \\ & & & \lambda_\ell \end{bmatrix} \in \mathbb{C}^{n_\ell \times n_\ell}, \quad \sum_{\ell=1}^k n_\ell = n,$$

numită forma normală Jordan a lui  $A$ .

Pentru demonstrație vezi [31].

**Definiția 9.2.4.** O matrice se numește diagonalizabilă, dacă toate blocurile sale Jordan  $J_\ell$  au dimensiunea 1, adică  $n_\ell = 1$ ,  $\ell = \overline{1, n}$ . O matrice se numește nederogatorie dacă pentru fiecare valoare proprie  $\lambda_\ell$  există exact un bloc Jordan în care ea apare pe diagonală.

**Observația 9.2.5.** Dacă o matrice  $A \in \mathbb{R}^{n \times n}$  are  $n$  valori proprii simple, atunci ea este diagonalizabilă și de asemenea nederogatorie și potrivită pentru a fi tratată numeric.  $\diamond$

**Teorema 9.2.6 (Descompunere Schur).** Pentru orice matrice  $A \in \mathbb{C}^{n \times n}$  există o matrice unitară  $U \in \mathbb{C}^{n \times n}$  și o matrice triunghiulară superior

$$R = \begin{bmatrix} \lambda_1 & * & \dots & * \\ & \ddots & \ddots & \vdots \\ & & \ddots & * \\ & & & \lambda_n \end{bmatrix} \in \mathbb{C}^{n \times n},$$

astfel încât  $A = URU^*$ .

**Observația 9.2.7.** 1. Elementele diagonale ale lui  $R$  sunt, natural, valorile proprii ale lui  $A$ . Deoarece  $A$  și  $R$  sunt similare, ele au aceleași valori proprii.

2. Între  $A$  și  $R$  are loc o formă mai puternică de similaritate: ele sunt *unitar-similare*.  $\diamond$

*Demonstrația teoremei 9.2.6.* Demonstrația se face prin inducție. Cazul  $n = 1$  este trivial. Presupunem teorema adevărată pentru  $n \in \mathbb{N}$  și fie  $A \in \mathbb{C}^{(n+1) \times (n+1)}$ . Fie  $\lambda \in \mathbb{C}$  o valoare proprie a lui  $A$  și  $x \in \mathbb{C}^{n+1}$ ,  $\|x\|_2 = 1$ , vectorul propriu corespunzător. Luăm vectorul  $u_1 = x$  și alegem  $u_2, \dots, u_{n+1}$  astfel încât  $u_1, \dots, u_{n+1}$  să formeze o bază ortonormală a lui  $\mathbb{C}^{n+1}$ , sau echivalent, matricea  $U = [u_1, \dots, u_{n+1}]$  să fie unitară. Așadar,

$$U^* A U e_1 = U^* A u_1 = U^* A x = \lambda U^* x = \lambda e_1,$$

adică

$$U^* A U = \begin{bmatrix} \lambda & * \\ 0 & B \end{bmatrix}, \quad B \in \mathbb{C}^{n \times n}.$$

Conform ipotezei inducției există o matrice unitară  $V \in \mathbb{C}^{n \times n}$ , astfel încât  $B = V S V^*$ , unde  $S \in \mathbb{C}^{n \times n}$  este o matrice triunghiulară superior. De aceea

$$A = U \begin{bmatrix} \lambda_1 & * \\ 0 & V S V^* \end{bmatrix} U^* = \underbrace{U \begin{bmatrix} 1 & 0 \\ 0 & V \end{bmatrix}}_{=:U} \underbrace{\begin{bmatrix} \lambda_1 & * \\ 0 & S \end{bmatrix}}_{=:R} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & V^* \end{bmatrix}}_{=:U^*} U^*$$

și demonstrația este completă.  $\square$

Să dăm acum două consecințe nemijlocite ale descompunerii Schur.

**Corolarul 9.2.8.** Oricărei matrice hermitiene  $A \in \mathbb{C}^{n \times n}$  îi corespunde o matrice ortogonală  $U \in \mathbb{C}^{n \times n}$  astfel încât

$$A = U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} U^*, \quad \lambda_j \in \mathbb{R}, \quad j = \overline{1, n}.$$

*Demonstrație.* Matricea  $R$  din teorema 9.2.6 verifică  $R = U^* A U$ . Deoarece

$$R^* = (U^* A U) = U^* A^* U = U^* A U = R,$$

$R$  trebuie să fie diagonală, cu elemente reale pe diagonală (fiind hermitiană).  $\square$

Altfel spus, corolarul 9.2.8 ne asigură că orice matrice hermitiană este unitar similară cu o matrice diagonală și posedă o bază formată din vectori proprii ortonormali. În plus, toate valorile proprii ale unei matrice hermitiene sunt reale. Este interesant, nu numai din punct de vedere teoretic, ce matrice sunt unitar diagonalizabile.

**Teorema 9.2.9.** O matrice  $A \in \mathbb{C}^{n \times n}$  este unitar diagonalizabilă, adică există o matrice unitară  $U \in \mathbb{C}^{n \times n}$  astfel încât

$$A = U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} U^*, \quad \lambda_j \in \mathbb{R}, \quad j = 1, \dots, n, \tag{9.2.1}$$

dacă și numai dacă  $A$  este normală, adică

$$AA^* = A^* A. \tag{9.2.2}$$

*Demonstrație.* Punem  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ . Conform lui (9.2.1),  $A$  are forma  $A = U\Lambda U^*$ , deci

$$AA^* = U\Lambda U^* U\Lambda^* U^* = U|\Lambda|^2 U^* \text{ și } A^* A = U^* \Lambda^* U^* U\Lambda U^* = U|\Lambda|^2 U^*,$$

adică (9.2.2). Pentru reciprocă, folosim descompunerea Schur a lui  $A$  sub forma  $R = U^* AU$ . Atunci

$$|\lambda_1|^2 = (R^* R)_{11} = (RR^*)_{11} = |\lambda_1|^2 + \sum_{k=2}^n |r_{1k}|^2,$$

de unde rezultă  $r_{12} = \dots = r_{1n} = 0$ . Prin inducție, se vede că pentru  $j = \overline{2, n}$

$$(R^* R)_{jj} = |\lambda_j|^2 + \sum_{k=1}^{j-1} |r_{kj}|^2 = (RR^*)_{jj} = |\lambda_j|^2 + \sum_{k=j+1}^n |r_{jk}|^2,$$

din care cauză  $R$  trebuie să fie diagonală.  $\square$

Pentru matrice reale *descompunerea Schur reală* este un pic mai complicată.

**Teorema 9.2.10.** *Pentru orice matrice  $A \in \mathbb{R}^{n \times n}$  există o matrice ortogonală  $U \in \mathbb{R}^{n \times n}$  astfel încât*

$$A = U \begin{bmatrix} R_1 & * & \dots & * \\ * & \ddots & \ddots & \vdots \\ & \ddots & \ddots & * \\ & & & R_k \end{bmatrix} U^*, \quad (9.2.3)$$

în care fie  $R_j \in \mathbb{R}^{1 \times 1}$ , fie  $R_j \in \mathbb{R}^{2 \times 2}$ , cu două valori proprii complexe conjugate,  $j = \overline{1, k}$ .

Descompunerea Schur reală transformă  $A$  într-o matrice Hessenberg superioară

$$U^T A U = \begin{bmatrix} * & \dots & \dots & * \\ * & \ddots & & \vdots \\ & \ddots & \ddots & \vdots \\ & & * & * \end{bmatrix}.$$

*Demonstrație.* Dacă  $A$  are doar valori proprii reale, atunci se procedează ca la descompunerea Schur complexă. Altfel, fie  $\lambda = \alpha + i\beta$ ,  $\beta \neq 0$ , o valoare proprie complexă a lui  $A$  și  $x + iy$  vectorul propriu corespunzător. Atunci

$$A(x + iy) = \lambda(x + iy) = (\alpha + i\beta)(x + iy) = (\alpha x - \beta y) + i(\beta x + \alpha y)$$

sau matricial

$$A \underbrace{\begin{bmatrix} x & y \end{bmatrix}}_{\in \mathbb{R}^{n \times 2}} = \begin{bmatrix} x & y \end{bmatrix} \underbrace{\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}}_{:= R}.$$

Deoarece  $R = \alpha^2 + \beta^2 > 0$ , căci  $\beta \neq 0$ ,  $\text{span}\{x, y\}$  este un subspațiu bidimensional  $A$ -invariant al lui  $\mathbb{R}^n$ . Atunci alegem  $u_1, u_2$  astfel încât să formeze o bază a acestui spațiu, completată cu  $u_3, \dots, u_n$  la o bază ortonormală a lui  $\mathbb{R}^n$  și raționând analog cu cazul complex obținem

$$U^T A U = \begin{bmatrix} R & * \\ 0 & B \end{bmatrix}$$

și inducția se derulează ca la descompunerea Schur complexă.  $\square$

### 9.3. Iterația vectorială

Iterația vectorială (numită și metoda puterii) este cea mai simplă metodă atunci când dorim o valoare proprie și vectorul propriu corespunzător.

Pornind de la un vector  $y^{(0)} \in \mathbb{C}^n$  se construiește sirul  $y^{(k)}$ ,  $k \in \mathbb{N}$  prin intermediul iterării

$$\begin{aligned} z^{(k)} &= Ay^{(k-1)}, \\ y^{(k)} &= \frac{z^{(k)}}{\|z^{(k)}\|_\infty}, \quad j_* = \min \left\{ 1 \leq j \leq n : |z_j^{(k)}| \geq \left(1 - \frac{1}{k}\right) \|z^{(k)}\|_\infty \right\} \end{aligned} \quad (9.3.1)$$

și se afirmă că, în condiții determinate, acest sir converge către vectorul propriu dominant.

**Propoziția 9.3.1.** Fie  $A \in \mathbb{C}^{n \times n}$  o matrice diagonalizabilă ale cărei valori proprii  $\lambda_1, \dots, \lambda_n$  verifică condiția

$$|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_n|.$$

Atunci sirul  $y^{(k)}$ ,  $k \in \mathbb{N}$ , converge către un multiplu al vectorului normat  $x_1$  corespunzător valorii proprii  $\lambda_1$ , pentru aproape orice  $y^{(0)}$ .

*Demonstrație.* Fie  $x_1, \dots, x_n$  vectorii proprii ortonormali ai lui  $A$  corespunzător valorilor proprii  $\lambda_1, \dots, \lambda_n$  – existența lor rezultă din diagonalizabilitatea lui  $A$ . Scriem

$$y^{(0)} = \sum_{j=1}^n \alpha_j x_j, \quad \alpha_j \in \mathbb{C}, \quad j = \overline{1, n},$$

și afirmăm că

$$A^k y^{(0)} = \sum_{j=1}^n \alpha_j A^k x_j = \sum_{j=1}^n \alpha_j \lambda^k x_j = \lambda_1^k \sum_{j=1}^n \alpha_j \left(\frac{\lambda_j}{\lambda_1}\right)^k x_j.$$

De aici rezultă, deoarece  $|\lambda_1| > |\lambda_j|$ ,  $j = \overline{2, n}$  că

$$\lim_{k \rightarrow \infty} \lambda_1^{-k} A^k y^{(0)} = \alpha_1 x_1 + \lim_{k \rightarrow \infty} \sum_{j=2}^n \alpha_j \left(\frac{\lambda_j}{\lambda_1}\right)^k x_j = \alpha_1 x_1,$$

precum și

$$\lim_{k \rightarrow \infty} |\lambda_1|^{-k} \|A^k y^{(0)}\|_\infty = \left\| \sum_{j=1}^n \alpha_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \alpha_j x_j \right\| = |\alpha_1| \|x_1\|_\infty.$$

Dacă  $\alpha_1 = 0$  și deci  $y^{(0)}$  aparține hiperplanului

$$x_1^+ = \{x \in \mathbb{C}^n, x^* x_1 = 0\},$$

atunci ambele limite sunt nule și nu se poate spune nimic despre convergența sirului  $y^k$ ,  $k \in \mathbb{N}$ ; acest hiperplan este o mulțime de măsură nulă, așa că în continuare vom presupune că  $\alpha_1 \neq 0$ .

Datorită lui (9.3.1),  $y^k = \gamma_k A^k y^{(0)}$ ,  $\gamma_k \in \mathbb{C}$  și pe lângă aceasta  $\|y^k\|_\infty = 1$ , aşadar

$$\lim_{k \rightarrow \infty} |\lambda_1|^k |\gamma_k| = \lim_{k \rightarrow \infty} \frac{1}{|\lambda_1|^{-1} \|A^{(k)} y^{(0)}\|} = \frac{1}{|\alpha_1| \|x_1\|_\infty}.$$

Astfel

$$y^{(k)} = \gamma_k A^k y^{(0)} = \underbrace{\frac{\gamma_k \lambda_1^k}{|\gamma_k \lambda_1^k|}}_{=: e^{-2\pi i \theta_k}} \underbrace{\frac{\alpha_1 x_1}{|\alpha_1| \|x_1\|_\infty}}_{=: \alpha x_1} + O\left(\frac{|\lambda_2|^k}{|\lambda_1|^k}\right), \quad k \in \mathbb{N}, \quad (9.3.2)$$

unde  $\theta_k \in [0, 1]$ . Aici intervine normarea mai ciudată din (9.3.1): fie  $j$  indicele minim pentru care  $|(\alpha x_1)_j| = \|\alpha x_1\|_\infty$ ; atunci conform lui (9.3.2) pentru un  $k$  suficient de mare de asemenea în (9.3.1)  $j^* = j$ . Deci are loc

$$\lim_{k \rightarrow \infty} y_j^{(k)} = 1 \Rightarrow \lim_{k \rightarrow \infty} e^{2\pi i \theta_k} = \lim_{k \rightarrow \infty} \frac{y_j^{(k)}}{(\alpha x_1)_j} = \frac{1}{(\alpha x_1)_j}.$$

Înlocuind aceasta în (9.3.2) obținem convergența sirului  $y^{(k)}$ ,  $k \in \mathbb{N}$ .  $\square$

Se poate aplica de asemenea iterația vectorială pentru a găsi toate valorile proprii și toți vectorii proprii, în măsura în care valorile proprii ale lui  $A$  sunt diferite în modul. Pentru aceasta se determină cea mai mare în modul valoare proprie  $\lambda_1$  a lui  $A$  și vectorul propriu corespunzător  $x_1$  și se continuă cu

$$A^{(1)} = A - \lambda_1 x_1 x_1^T.$$

Matricea diagonalizabilă  $A^{(1)}$  are aceeași vectori proprii ortonormali ca și  $A$ , doar că  $x_1$  este vectorul propriu corespunzător valorii proprii 0 și nu mai joacă nici un rol în iterare, atât timp cât nu se pornește chiar cu un multiplu al lui  $x_1$ . Aplicând încă o dată iterăția vectorială lui  $A^{(1)}$  se obține a doua valoare proprie ca mărime în modul  $\lambda_2$  și vectorul propriu corespunzător; iterăția

$$A^{(j)} = A^{(j-1)} - \lambda_j x_j x_j^T, \quad j = \overline{1, n}, \quad A^{(0)} = A$$

calculează succesiv toate valorile proprii și toți vectorii proprii ai lui  $A$ , presupunând că valorile proprii sunt diferite în modul.

### **Observația 9.3.2 (Dezavantajele iterăției vectoriale).**

1. Metoda funcționează în general doar când există un vector propriu dominant, adică când există pentru o valoare proprie dominantă exact un vector propriu. Dacă se consideră, de exemplu, matricea

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix},$$

ea transformă vectorul  $[x_1 \ x_2]^T$  în vectorul  $[x_2 \ x_1]^T$  și convergența are loc exact atunci când iterăția pornește cu unul din vectorii proprii.

2. Metoda dă rezultate numai pentru vectori de pornire „potrivită”. Sună nemaipomenit că toți vectorii de pornire cu excepția celor dintr-un hiperplan sunt potriviti, dar nu este chiar aşa de simplu. Dacă valoarea proprie cea mai mare în modul a unei matrice reale este complexă, atunci se poate itera la nesfârșit cu valori de pornire reale, căci oricum nu se va găsi vectorul propriu.
3. Ar trebui făcute tot timpul calcule în complex, ceea ce ar ridica serios complexitatea (la adunare un număr dublu de calcule, la înmulțire de sase ori mai multe).
4. Viteza de convergență depinde de raportul

$$\frac{|\lambda_2|}{|\lambda_1|} < 1,$$

care poate fi oricât de aproape de 1. Dacă dominanța vectorului propriu dominant nu este bine reliefată, atunci convergența poate fi extrem de lentă.  $\diamond$

Tinând cont de toate acestea, conchidem că iterăția vectorială nu este o metodă prea bună pentru problemele de valori proprii.

## 9.4. Metoda QR – teoria

Metoda practică actuală pentru tratarea problemelor de valori proprii este metoda QR, descoperită de Francis [22] și Kublanovskaya [39], o extensie unitară a metodei LR a lui Rutishauser [54]. Vom începe cu cazul complex.

Metoda este una iterativă extrem de simplă: se pornește cu  $A^{(0)} = A$  și se calculează iterativ cu ajutorul descompunerii QR,

$$A^{(k)} = Q_k R_k, \quad A^{(k+1)} = R_k Q_k, \quad k \in \mathbb{N}_0. \quad (9.4.1)$$

În anumite ipoteze, acest sir va converge către o matrice ale cărei elemente diagonale sunt valorile proprii ale lui  $A$ .

**Lema 9.4.1.** *Matricele  $A^{(k)}$  construite prin (9.4.1),  $k \in \mathbb{N}$ , sunt unitar similară cu  $A$  și au aceleași valori proprii ca  $A$ .*

*Demonstrație.* Are loc

$$A^{(k+1)} = Q_k^* Q_k R_k Q_k = Q_k^* A^{(k)} Q_k = \dots = \underbrace{Q_k^* \dots Q_0^*}_{=: U_k^*} A \underbrace{Q_0 \dots Q_k}_{=: U_k}.$$

□

Pentru a arăta convergența, vom interpreta iterația QR ca o generalizare a iterației vectoriale (9.3.1) (fără normarea ciudată) la spații vectoriale. Pentru aceasta vom scrie baza ortonormală  $u_1, \dots, u_m \in \mathbb{C}^n$  a unui subspațiu  $m$ -dimensional  $U \subset \mathbb{C}^n$ ,  $m \leq n$ , ca vectori coloane ai unei matrice unitare  $U \in \mathbb{R}^{n \times m}$  și iterăm subspațiul vectorial (respectiv matricele) peste descompunerea QR

$$U_{k+1} R_k = A U_k, \quad k \in \mathbb{N}_0, \quad U_0 \in \mathbb{C}^n. \quad (9.4.2)$$

De aici rezultă imediat că

$$U_{k+1}(R_k \dots R_0) = A U_k(R_{k-1} \dots R_0) = A^2 U_{k-1}(R_{k-2} \dots R_0) = \dots = A^{k+1} U_0. \quad (9.4.3)$$

Dacă definim acum, pentru  $m = n$ ,  $A^{(k)} = U_k^* A U_k$ , atunci conform lui (9.4.2) au loc relațiile

$$\begin{aligned} A^{(k)} &= U_k^* A U_k = U_k^* U_{k+1} R_k \\ A^{(k+1)} &= U_{k+1}^* A U_{k+1} = U_{k+1}^* A U_k U_k^* U_{k+1} \end{aligned}$$

și punând  $Q_k := U_k^* U_{k+1}$ , obținem regula de iterare (9.4.1). Ca matrice de pornire putem alege  $U_0 = I$ .

Pentru a obține rezultate despre metoda QR mai avem nevoie de un tip de matrice.

**Definiția 9.4.2.** O matrice de fază  $\Theta \in \mathbb{C}^{n \times n}$  este o matrice diagonală de forma

$$\Theta = \begin{bmatrix} e^{-i\theta_1} & & & \\ & \ddots & & \\ & & e^{-i\theta_n} & \end{bmatrix}, \quad \theta_j \in [0, 2\pi), \quad j = \overline{1, n}.$$

**Propoziția 9.4.3.** Presupunem că matricea  $A \in \mathbb{C}^{n \times n}$  are valori proprii distincte în modul,  $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| > 0$ . Dacă matricea  $X^{-1}$  din forma normală Jordan  $A = X\Lambda X^{-1}$  a lui  $A$  are o descompunere  $LU$

$$X^{-1} = ST, \quad S = \begin{bmatrix} 1 & & & \\ * & 1 & & \\ \vdots & \ddots & \ddots & \\ * & \dots & * & 1 \end{bmatrix}, \quad T = \begin{bmatrix} * & \dots & * \\ & \ddots & \vdots \\ & & * \end{bmatrix},$$

atunci există matricele de fază  $\Theta_k$ ,  $k \in \mathbb{N}_0$ , astfel încât sirul de matrice  $(\Theta_k U_k)$ ,  $k \in \mathbb{N}$  să conveargă.

**Observația 9.4.4 (La propoziția 9.4.3).** 1. Convergența sirului  $(\Theta_k U_k)$ , înseamnă, mai ales, că dacă bazele ortonormale aparținătoare converg către o bază ortonormală a lui  $\mathbb{C}^n$ , avem de asemenea convergența spațiilor vectoriale.

2. Existența descompunerii LU a lui  $X^{-1}$  este fără nici o limitare: deoarece  $X^{-1}$  este inversabilă în mod trivial, există întotdeauna o permutare  $P$ , astfel încât  $X^{-1}P^T = (PX)^{-1} = LU$  și  $PX$  este într-adevăr o matrice inversabilă. Aceasta înseamnă că matricea  $\hat{A} = P^TAP$ , în care  $A$  se transformă prin permutarea liniilor și coloanelor și care are aceleași valori proprii ca și  $A$ , verifică ipoteza propoziției 9.4.3.
3. Demonstrația propoziției 9.4.3 este o modificare a demonstrației din [68, pag. 54–56] pentru convergența metodei LR, care își are originea în lucrarea lui Wilkinson<sup>1</sup> [75]. Ce este de fapt metoda LR? Se procedează ca la metoda QR, dar lui  $A$  i se aplică eliminarea gausiană,  $A^{(k)} = L_k R_k$  și apoi se construiește  $A^{(k+1)} = R_k L_k$ . De asemenea, această metodă converge în anumite condiții către o matrice triunghiulară superior. ◇

Înainte de a demonstra propoziția 9.4.3 să vedem întâi de ce convergența sirului  $(U_k)$  atrage convergența metodei QR. Își anume, dacă avem  $\|U_{k+1} - U_k\| \leq \varepsilon$  sau echivalent

$$U_{k+1} = U_k + E, \quad \|E\|_2 \leq \varepsilon,$$

atunci

$$Q_k = U_{k+1}^* U_k = (U_k + E)^* U_k = I + E^* U_k = I + F, \quad \|F\|_2 \leq \|E\|_2 \underbrace{\|U_k\|_2}_{=1} \leq \varepsilon,$$

și cu aceasta

$$A^{(k+1)} = R_k Q_k = R_k (I + F) = R_k + G, \quad \|G\|_2 \leq \varepsilon \|R_k\|_2,$$

James Hardy Wilkinson (1919-1986), matematician englez. Contribuții importante în domeniul Analizei numerice, Algebrei liniare numerice și Informaticii. Membru al Royal Society, laureat al premiului Turing al ACM. Pe lângă numeroasele sale lucrări în domeniul Analizei numerice, a lucrat și la dezvoltarea de biblioteci de rutine numerice. Grupul NAG (Numerical Algorithms Group) și-a început activitatea în 1970 și multe dintre rutinele de algebră liniară numerică s-au datorat lui Wilkinson.



deci șirul  $A^{(k)}$ ,  $k \in \mathbb{N}$ , converge, de asemenea, către o matrice triunghiulară superior, numai dacă normele lui  $R_k$ ,  $k \in \mathbb{N}$  sunt uniform mărginite. Chiar aşa se întâmplă, căci

$$\|R_k\|_2 = \|Q_k^* A^{(k)}\|_2 = \|A^{(k)}\|_2 = \|Q_{k-1}^* \dots Q_0^* A Q_0 \dots Q_{k-1}\| = \|A\|_2.$$

Mai avem nevoie de un rezultat ajutător despre „unicitatea” descompunerii QR.

**Lema 9.4.5.** Fie  $U, V \in \mathbb{C}^{n \times n}$  matrice unitare și fie  $R, S \in \mathbb{C}^{n \times n}$  matrice triunghiulare superior inversabile. Atunci are loc  $UR = VS$  dacă și numai dacă există o matrice de fază

$$\Theta = \begin{bmatrix} e^{-i\theta_1} & & & \\ & \ddots & & \\ & & e^{-i\theta_n} & \end{bmatrix}, \quad \theta_j \in [0, 2\pi), j = \overline{1, n},$$

astfel încât  $U = V\Theta^*$ ,  $R = \Theta S$ .

*Demonstrație.* Deoarece  $UR = V\Theta^*\Theta S = VS$ , ” $\Leftarrow$ ” este trivială. Pentru necesitate, din  $UR = VS$  rezultă că  $V^*U = SR^{-1}$  trebuie să fie o matrice triunghiulară superior astfel încât  $(V^*U)^* = U^*V = RS^{-1}$ . Așadar  $\Theta = V^*U$  este o matrice diagonală unitară și are loc  $U = VV^*U = V\Theta$ .  $\square$

*Demonstrația propoziției 9.4.3.* Fie  $A = X\Lambda X^{-1}$  forma normală Jordan a lui  $A$ , unde  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ . Pentru  $U_0 = I$  și  $k \in \mathbb{N}_0$

$$U_k \left( \prod_{j=k-1}^0 R_j \right) = (X^{-1}\Lambda X)^k = X\Lambda^k X^{-1} = X\Lambda^k ST = X \underbrace{(\Lambda^k S \Lambda^{-k})}_{=: L_k} \Lambda^k T,$$

în care  $L_k$  este o matrice triunghiulară inferior cu elementele

$$(L_k)_{jm} = \left( \frac{\lambda_j}{\lambda_m} \right)^k, \quad 1 \leq m \leq j \leq n, \quad (9.4.4)$$

astfel încât pentru  $k \in \mathbb{N}$

$$|L_k - I| \leq \left( \max_{1 \leq m < j \leq n} |s_{jm}| \right) \left( \max_{1 \leq m < j \leq n} \left| \frac{\lambda_j}{\lambda_m} \right| \right)^k \begin{bmatrix} 0 & & & \\ 1 & \ddots & & \\ \vdots & \ddots & \ddots & \\ 1 & \dots & 1 & 0 \end{bmatrix}, \quad (k \in \mathbb{N}). \quad (9.4.5)$$

Fie  $\widehat{U}_k \widehat{R}_k = X L_k$  descompunerea QR a lui  $X L_k$ , care datorită lui (9.4.5) și lemei 9.4.5 converge până la o matrice de fază către descompunerea QR  $X = UR$  a lui  $X$ . Dacă aplicăm acum lema 9.4.5 identității

$$U_k \left( \prod_{j=k-1}^0 R_j \right) = \widehat{Q}_k \widehat{R}_k \Lambda^k T,$$

atunci există matricele de fază  $\Theta_k$ , astfel încât

$$U_k = \widehat{Q}_k \Theta_k^* \text{ și } \left( \prod_{j=k-1}^0 R_j \right) = \Theta_k \widehat{R}_k \Lambda^k T,$$

deci există matricele de fază  $\widehat{\Theta}_k$ , astfel încât  $U_k \widehat{\Theta}_k \rightarrow U$ , când  $k \rightarrow \infty$ .  $\square$

Merită să aruncăm o scurtă privire asupra „termenului de eroare” din (9.4.4), ale căruia elemente subdiagonale verifică relația

$$|L_k|_{jm} \leq \left( \frac{|\lambda_j|}{|\lambda_m|} \right) |s_{jm}|, \quad 1 \leq m < j \leq n.$$

Așadar are loc

*Convergența unui element subdiagonal către 0 este cu atât mai rapidă cu cât elementul este mai îndepărtat de diagonală.*

## 9.5. Metoda QR – practica

### 9.5.1. Metoda QR clasică

Am văzut că metoda QR generează un sir de matrice  $A^{(k)}$  care în condiții determinante trebuie să conveargă către o matrice triunghiulară superior care are pe diagonală valorile proprii. Putem întrebuița această metodă pentru matrice reale.

**Exemplul 9.5.1.** Fie

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 2 & 1 \end{bmatrix}.$$

Această matrice are valorile proprii

$$\lambda_1 \approx 4.56155, \quad \lambda_2 = -1, \quad \lambda_3 \approx 0.43845.$$

Iterând prin metoda QR se obțin pentru elementele subdiagonale rezultatele din tabela 9.1 (o implementare MATLAB brută). Se poate vedea că, după  $k$  iterații, elementele  $a_{m\ell}^{(k)}$ ,  $\ell < m$ , se apropie de

#iterații	$a_{21}$	$a_{31}$	$a_{32}$
10	6.64251e-007	-2.26011e-009	0.00339953
20	1.70342e-013	-1.52207e-019	8.9354e-007
30	4.36711e-020	-1.02443e-029	2.34578e-010
40	1.11961e-026	-6.89489e-040	6.15829e-014

Tabela 9.1: Rezultate pentru exemplul 9.5.1

0 la fel ca  $|\lambda_\ell / \lambda_k|$ . ◊

**Exemplul 9.5.2.** Matricea

$$\begin{bmatrix} 1 & 5 & 7 \\ 3 & 0 & 6 \\ 4 & 3 & 1 \end{bmatrix}$$

are valorile proprii

$$\lambda_1 \approx 9.7407, \quad \lambda_2 \approx -3.8703 + 0.6480i, \quad \lambda_3 \approx -3.8703 - 0.6480i.$$

În acest caz nu ne putem aștepta ca metoda QR să conveargă către o matrice triunghiulară superioară, căci atunci toate valorile proprii ale lui  $A$  ar fi reale. De fapt, după 100 de iterații se obține matricea

$$A^{(100)} \approx \begin{bmatrix} 9.7407 & -4.3355 & 0.94726 \\ 8.552e-039 & -4.2645 & 0.7236 \\ 3.3746e-039 & -0.79491 & -3.4762 \end{bmatrix},$$

care ne furnizează corect valoarea proprie reală. Pe lângă aceasta, matricea  $2 \times 2$  din colțul din dreapta jos ne furnizează valorile proprii complexe  $-3.8703 \pm 0.6480i$ .  $\diamond$

Al doilea exemplu recomandă următoarea strategie: dacă elementele situate sub diagonală nu vor să dispară, ar fi indicat să privim mai atent matricea  $2 \times 2$  corespunzătoare.

**Definiția 9.5.3.** Dacă  $A \in \mathbb{R}^{n \times n}$  are descompunerea  $QR$ , atunci transformarea  $RQ$  a lui  $A$  se definește prin  $A_* = RQ$ .

Ce probleme apar la realizarea practică a metodei QR? Deoarece complexitatea descompunerii QR este  $O(n^3)$ , nu este prea înțelept să utilizăm o metodă care se bazează pe un astfel de pas iterativ. Pentru a evita problema, vom converti matricele inițiale în matrice similare a căror descompunere QR poate fi calculată mai repede. Astfel de matrice sunt matricele Hessenberg superioare a căror descompunere QR se poate calcula cu  $n$  iterații Givens, deci un total de  $O(n^2)$  operații: de vreme ce numai elementele  $h_{j-1,j}$ ,  $j = \overline{2, n}$  trebuie eliminate, vom determina unghurile  $\phi_2, \dots, \phi_n$ , astfel ca

$$G(n-1, n; \phi_n) \dots G(1, 2; \phi_2)H = R$$

și are loc

$$H_* = RG^T(1, 2; \phi_2) \dots G^T(n-1, n; \phi_n). \quad (9.5.1)$$

Această idee este implementată în fișierul `HessenRQ.m` (vezi funcția MATLAB 9.1).

**Lema 9.5.4.** Dacă  $H \in \mathbb{R}^{n \times n}$  este o matrice Hessenberg superioară, atunci  $H_*$  este de asemenea matrice Hessenberg superioară.

*Demonstrație.* Rezultă direct din reprezentarea (9.5.1). Înmulțirea la dreapta cu o matrice Givens  $G^T(j, j+1, \phi_{j+1})$ ,  $j = \overline{1, n-1}$  înseamnă o combinație a coloanelor  $j$  și  $j+1$  și crează valori diferite de zero doar în prima subdiagonală –  $R$  este triunghiulară superior.  $\square$

Să vedem cum aducem matricea inițială la forma Hessenberg. În acest scop vom utiliza pentru variație transformările Householder. Să presupunem că am găsit deja o matrice  $Q_k$ , astfel încât primele  $k$  coloane ale matricei transformate să aibă deja forma Hessenberg, adică

$$Q_k A Q_k^T = \left[ \begin{array}{ccc|cc|ccc} * & \dots & * & * & & * & \dots & * \\ * & \dots & * & * & & * & \dots & * \\ \ddots & \vdots & \vdots & \vdots & & \vdots & \ddots & \vdots \\ & * & * & * & \dots & * & \dots & * \\ \hline & & & a_1^{(k)} & & * & \dots & * \\ & & & \vdots & & \vdots & \ddots & \vdots \\ & & & a_{n-k-1}^{(k)} & & * & \dots & * \end{array} \right].$$

**Sursa MATLAB 9.1 Tranformarea RQ a unei matrice Hessenberg**

```

function HH=HessenRQ(H)
%HESSENRO - transformarea RQ a unei matrice Hessenberg
%utilizand rotatii Givens
%intrare H -matrice Hesenberg
%iesire HH - transformarea RQ a lui H

[m,n]=size(H);
Q=eye(m,n);

for k=2:n
    a=H(k-1:k,k-1);
    an=sqrt(a'*a); %norma euclidiana
    c=sign(a(2))*abs(a(1))/an; %sinus
    s=sign(a(1))*abs(a(2))/an; %cosinus
    Jm=eye(n);
    Jm(k-1,k-1)=c; Jm(k,k)=c;
    Jm(k-1,k)=s; Jm(k,k-1)=-s;
    H=Jm*H;
    Q=Q*Jm';
end
HH=H*Q;

```

Apoi determinăm  $\hat{y} \in \mathbb{R}^{n-k-1}$  și  $\alpha \in \mathbb{R}$  (care rezultă automat), astfel ca

$$H(\hat{y}) \begin{bmatrix} a_1^{(k)} \\ \vdots \\ \vdots \\ a_{n-k-1}^{(k)} \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} \Rightarrow U_{k+1} := \begin{bmatrix} I_{k+1} & H(\hat{y}) \end{bmatrix}$$

și obținem că

$$\underbrace{U_{k+1}Q_k}_{{=:Q_{k+1}}} \underbrace{A Q_k U_{k+1}}_{{=:Q_{k+1}^T}} = \left[ \begin{array}{ccc|c|ccc} * & \dots & * & * & * & \dots & * \\ * & \dots & * & * & * & \dots & * \\ \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \alpha & * & \dots & * \\ 0 & * & * & 0 & * & \dots & * \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & 0 & * & \dots & * \end{array} \right] U_{k+1};$$

matricea unitate  $I_{k+1}$  din stânga sus în matricea  $U_{k+1}$  are grija să avem în primele  $k + 1$  coloane o structură Hessenberg. Versiunea MATLAB a acestei metode este implementată în fișierul hessen.m (sursa MATLAB 9.2). În concluzie, metoda noastră QR va fi o metodă în două faze:

1. Transformă pe  $A$  în forma Hessenberg utilizând o transformare ortogonală:

$$H^{(0)} = Q A Q^T, \quad Q^T Q = Q Q^T = I.$$

---

**Sursa MATLAB 9.2 Trecerea la forma Hessenberg**

---

```

function [A,Q]=hessen_h(A)
%HESSEN_H - reducere Householder la forma Hessenberg
%intrare
%A - matrice
%rezultate
%A - forma Hessenberg (se rescrie peste A)
%Q - matricea de transformare (optional)

[m,n]=size(A);
%obtine transformare
v=zeros(m,m);
Q=eye(m,m);
for k=1:m-2
    x=A(k+1:m,k);
    vk=mysign(x(1))*norm(x,2)*[1;zeros(length(x)-1,1)]+x;
    vk=vk/norm(vk);
    A(k+1:m,k:m)=A(k+1:m,k:m)-2*vk*(vk'*A(k+1:m,k:m));
    A(1:m,k+1:m)=A(1:m,k+1:m)-2*(A(1:m,k+1:m)*vk)*vk';
    v(k+1:m,k)=vk;
end
%obtine matricea Q cu care se transforma
if nargout==2
    Q=eye(m,m);
    for j=1:m
        for k=m:-1:1
            Q(k:m,j)=Q(k:m,j)-2*v(k:m,k)*(v(k:m,k)'*Q(k:m,j));
        end
    end
end

```

---

## 2. Execuță iterațiile QR

$$H^{(k+1)} = H_*^{(k)}, \quad k \in \mathbb{N}_0,$$

în speranță că elementele de pe prima subdiagonală converg toate către zero.

Deoarece elementele subdiagonale converg cel mai întâi, putem folosi drept criteriu de oprire maximul modulului. Aceasta ne conduce la *metoda QR simplă*, vezi QRMethod1.m, sursa MATLAB 9.3. Desigur, la apariția unor valori proprii complexe această metodă iterează la nesfârșit.

**Exemplul 9.5.5.** Aplicăm noua metodă matricei din exemplul 9.5.1. Pentru diverse toleranțe  $\varepsilon$  date obținem rezultatele din tabela 9.2. Se observă că la fiecare trei iterații se câștigă o zecimală la elementele vecine cu diagonala. ◇

*Putem încerca să accelerăm metoda descompunând problema în subprobleme.* Dacă avem o matrice

**Sursa MATLAB 9.3 Metoda QR simplă**

```

function [lambda,it]=QRMethod1(A,t)
%QRMETHOD1 - Calcululeaza valorile proprii ale unei matrice
%reale, implementarea naiva
%intrare
%    A - matricea
%    t - toleranta
%iesire
%    lambda - valorile proprii - diagonalala lui R
%    it - numar de iteratii

H=hessen_h(A); %forma Hessenberg
it=0;
while norm(diag(H,-1),inf) > t %crit. de oprire
    H=HessenRQ(H);
    it=it+1;
end
lambda=diag(H);

```

$\varepsilon$	#iterații	$\lambda_1$	$\lambda_2$	$\lambda_3$
$10^{-3}$	11	4.56155	-0.999834	0.438281
$10^{-4}$	14	4.56155	-1.000001	0.438461
$10^{-5}$	17	4.56155	-0.999999	0.438446
$10^{-10}$	31	4.56155	-1	0.438447

Tabela 9.2: Rezultatele pentru exemplul 9.5.5

Hessenberg de forma

$$H = \left[ \begin{array}{cccc|cc} * & \dots & \dots & * & & & \\ * & \ddots & & \vdots & & & * \\ & \ddots & \ddots & \vdots & & & \\ & & * & * & & & \\ \hline & & & * & \dots & \dots & * \\ & & & * & \ddots & & \vdots \\ & & & & \ddots & \ddots & \vdots \\ & & & & & * & * \end{array} \right] = \begin{bmatrix} H_1 & * \\ & H_2 \end{bmatrix},$$

atunci problema de valori proprii referitoare la  $H$  se poate descompune într-o problemă de valori proprii referitoare la  $H_1$  și una referitoare la  $H_2$ .

Conform lui [29], un element subdiagonal  $h_{j+1,j}$  este considerat „suficient de mic” dacă

$$|h_{j+1,j}| \leq \text{eps} (|h_{jj}| + |h_{j+1,j+1}|). \quad (9.5.2)$$

Aici vom proceda mai simplu și vom descompune o matrice dacă cel mai mic element în modul situat pe prima subdiagonală devine mai mic decât o toleranță dată. Procedeul este după cum urmează:

$\varepsilon$	#iterații	$\lambda_1$	$\lambda_2$	$\lambda_3$
$10^{-3}$	12	9.7406	$-3.8703 + 0.6479i$	$-3.8703 - 0.6479i$
$10^{-4}$	14	9.7407	$-3.8703 + 0.6479i$	$-3.8703 - 0.6479i$
$10^{-5}$	17	9.7407	$-3.8703 + 0.6480i$	$-3.8703 - 0.6480i$
$10^{-5}$	19	9.7407	$-3.8703 + 0.6480i$	$-3.8703 - 0.6480i$
$10^{-5}$	22	9.7407	$-3.8703 + 0.6480i$	$-3.8703 - 0.6480i$

Tabela 9.3: Rezultate pentru exemplul 9.5.6

funcția de determinare a valorilor proprii determină cu ajutorul unei iterații QR o descompunere în matricele  $H_1$  și  $H_2$  și se apelează recursiv pe sine pentru fiecare din aceste matrice.

Dacă una dintre matrice este  $1 \times 1$ , valoarea proprie este automat determinată, iar dacă este  $2 \times 2$ , atunci polinomul său caracteristic este

$$\begin{aligned} p_A(x) &= \det(A - xI) = x^2 - \text{tr}(A)x + \det(A) \\ &= x^2 \underbrace{(a_{11} + a_{22})}_{=:b} x + \underbrace{(a_{11}a_{22} - a_{12}a_{21})}_{=:c}. \end{aligned}$$

Dacă discriminantul  $b^2 - 4c$  este pozitiv,  $A$  are două valori proprii reale și distințe

$$x_1 = \frac{1}{2} \left( -b - \text{sgn}(b)\sqrt{b^2 - 4c} \right) \text{ și } x_2 = \frac{c}{x_1},$$

altfel valorile proprii sunt complexe și anume

$$\frac{1}{2} \left( -b \pm i\sqrt{4c - b^2} \right).$$

Astfel, avem sub control și cazul valorilor proprii complexe. Ideea este implementată în fișierul `QRSplit1a.m`, sursa MATLAB 9.4. Iterațiile QR se execută efectiv în fișierul `QRIter.m`, iar cazul complex în `Eigen2x2.m`.

**Exemplul 9.5.6.** Să considerăm din nou matricea din exemplul 9.5.2 căreia îi aplicăm rutina `QRSplit1a`. Rezultatele apar în tabela 9.3.  $\diamond$

## 9.5.2. Deplasare spectrală

Utilizarea matricelor Hessenberg ne permite, într-adevăr, să executăm fiecare pas de iterație într-un timp mai scurt. Vom încerca acum să micșorăm numărul de iterații, aşadar să creștem viteza de convergență deoarece

Rata de convergență a elementelor subdiagonale  $h_{j+1,j}$  are ordinul de mărime

$$\left( \frac{\lambda_{j+1}}{\lambda_j} \right)^k, \quad j = \overline{1, n-1}.$$

Cuvântul de ordine este aici *deplasare (translație) spectrală*. Se observă că, pentru  $\mu \in \mathbb{R}$ , matricea  $A - \mu I$  are valorile proprii  $\lambda_1 - \mu, \dots, \lambda_n - \mu$ . Pentru o matrice oarecare inversabilă,  $B$ , matricea  $B(A - \mu I)B^{-1} + \mu I$  are valorile proprii  $\lambda_1, \dots, \lambda_n$  – se poate aşadar deplasa spectrul matricei înainte și înapoi printr-o transformare de similaritate. Se ordonează valorile proprii  $\mu_1, \dots, \mu_n$  astfel ca

$$|\mu_1 - \mu| > |\mu_2 - \mu| > \dots > |\mu_n - \mu|, \quad \{\mu_1, \dots, \mu_n\} = \{\lambda_1, \dots, \lambda_n\}$$

**Sursa MATLAB 9.4** Metoda QR cu partitionare și tratarea cazurilor  $2 \times 2$ 

```

function [lambda, It]=QRSPsplit1a(A,t)
%QRSPPLIT1 calculul valorilor proprii: metoda QR, partitionare
%si tratare speciala a matricelor 2x2
%Intrare
%A - matricea
%t - toleranta
%Rezultate
%lambda - valorile proprii
%It - numar de iteratii

[m,n]=size(A);
if n==1
    It=0;
    lambda=A;
    return
elseif n==2
    It=0;
    lambda=Eigen2x2(A);
    return
else
    H=hessen_h(A); %aducere la forma Hessenberg
    [H1,H2,It]=QRIter(H,t); %descompunere H->H1,H2
    %apel recursiv
    [l1,It1]=QRSPsplit1a(H1,t);
    [l2,It2]=QRSPsplit1a(H2,t);
    It=It+It1+It2;
    lambda=[l1;l2];
end

```

și dacă  $\mu$  este apropiat de  $\mu_n$ , atunci dacă metoda QR începe cu  $H^0 = A - \mu I$ , elementul subdiagonal  $h_{n-1,n}^{(n)}$  converge foarte repede către zero. Mai bine este dacă deplasarea spectrală se realizează la fiecare pas individual. Pe lângă aceasta, putem lua ca aproximatie pentru  $\mu$  (în mod euristic) valoarea  $h_{nn}^{(k)}$ . Se obține astfel următoarea schemă iterativă

$$H^{(k+1)} = \left( H^{(k)} - \mu_k I \right)_* + \mu_k I, \quad \mu_k := h_{nn}^{(k)}, \quad k \in \mathbb{N}_0,$$

cu matricea de pornire  $H^0 = QAQ^T$ . Această metodă este implementată în QRMethod2.m. În fișierul QRSPsplit2.m se dă o implementare care tratează și valorile proprii complexe. Această funcție apelează QRIter2.

**Observația 9.5.7.** Dacă valoarea de deplasare  $\mu$  este suficient de apropiată de o valoare proprie  $\lambda$ , atunci matricea se descompune într-un singur pas iterativ. ◇

**Exemplul 9.5.8.** Să considerăm din nou matricea din exemplul 9.5.1. Dăm numai toleranțele și comparăm numărul de iterații necesare pentru atingerea lor (tabela 9.4). Deplasarea spectrală accelerează metoda, în special în legătură cu descompunerea matricei. ◇

**Sursa MATLAB 9.5 Eigenvalues of a  $2 \times 2$  matrix**

---

```
function lambda=Eigen2x2 (A)
%EIGEN2X2 - Calculeaza valorile proprii ale unei matrice 2x2
%A - matricea 2x2
%lambda - valorile proprii

b=trace (A); c=det (A);
d=b^2/4-c;
if d > 0
    if b == 0
        lambda = [sqrt(c); -sqrt(c)];
    else
        x = (b/2+sign(b)*sqrt(d));
        lambda=[x; c/x];
    end
else
    lambda=[b/2+i*sqrt(-d); b/2-i*sqrt(-d)];
end
```

---

**Sursa MATLAB 9.6 Iterația QR**

---

```
function [H1,H2,It]=QRIter (H,t)
%QRITER - executa iteratia QR pe matricea HouseHolder
%pana cand cel mai mic element subdiagonal este < t
%Intrare
% H - matricea Hessenberg
% t - toleranta
%Rezultate
%H1, H2 - descompunerea dupa elementul minim
%It - numar de iteratii

It=0; [m,n]=size(H);
[m, j]=min(abs(diag(H,-1)));
while m>t
    It=It+1;
    H=HessenRQ(H);
    [m, j]=min(abs(diag(H,-1)));
end
H1=H(1:j,1:j);
H2=H(j+1:n, j+1:n);
```

---

**Sursa MATLAB 9.7 Metoda QR cu deplasare spectrală și partitioare**


---

```

function [lambda,it]=QRMethod2(A,t)
%QRMETHOD2 - Calcululeaza valorile proprii ale unei matrice
%reale cu metoda QR si deplasare
%intrare
%    A - matricea
%    t - toleranta
%iesire
%    lambda - valorile proprii - diagonalala lui R
%    it - numar de iteratii

[m,n]=size(A); II=eye(n); H=hessen_h(A); it=0;

while norm(diag(H,-1),inf) > t
    m = H(n,n);
    H = HessenRQ(H - m * II) + m*II;
    it=it+1;
end
lambda=diag(H);

```

---

$\varepsilon$	#iterații			
	QRMethod1	QrMethod2	QRSplit1	QRSplit2
$10^{-5}$	17	11	17	1
$10^{-6}$	19	12	19	1
$10^{-7}$	22	14	22	1
$10^{-8}$	25	16	25	1
$10^{-9}$	28	18	28	1
$10^{-10}$	31	19	31	1
$10^{-20}$	58	36	58	9

Tabela 9.4: Comparații pentru exemplul 9.5.8

---

**Sursa MATLAB 9.8** Metoda QR cu deplasare spectrală, partitōnare și tratarea valorilor proprii complexe
 

---

```

function [lambda,It]=QRSPsplit2(A,t)
%QRSPLIT2 - calculul valorilor proprii metoda QR,
%partitionare, deplasare si tratare
%speciala a matricelor 2x2
%Intrare
%A - matricea
%t - toleranta
%Rezultate
%lambda - valorile proprii
%It - numar de iteratii

[m,n]=size(A);
if n==1
    It=0;
    lambda=A;
    return
elseif n==2
    It=0;
    lambda=Eigen2x2(A);
    return
else
    H=hessen_h(A); %aducere la forma Hessenberg
    [H1,H2,It]=QRITER2(H,t); %descompunere H->H1,H2
    %apel recursiv
    [l1,It1]=QRSPsplit2(H1,t);
    [l2,It2]=QRSPsplit2(H2,t);
    It=It+It1+It2;
    lambda=[l1;l2];
end

```

---

### 9.5.3. Metoda QR cu pas dublu

Se poate arăta că metoda QR cu deplasare spectrală converge *pătratic*, eroarea fiind pentru un  $\rho < 1$  doar

$$O(\rho^{2k}) \text{ în loc de } O(\rho^k).$$

Oricât de frumoasă ar fi această idee, ea funcționează bine doar pentru valori proprii reale, în cazul valorilor proprii complexe fiind problematică. Cu toate acestea, putem exploata faptul că valorile proprii apar în perechi. Aceasta ne conduce la ideea de „metode cu pas dublu”:

*în loc să deplasăm spectrul cu o valoare proprie, aproximată în mod euristic cu  $h_{n,n}^{(k)}$ , se execută două deplasări într-un pas, și anume cu valorile proprii ale lui*

$$B = \begin{bmatrix} h_{n-1,n-1}^{(k)} & h_{n-1,n}^{(k)} \\ h_{n-1,n}^{(k)} & h_{n,n}^{(k)} \end{bmatrix}.$$

**Sursa MATLAB 9.9 Iterație QR și partionare**

```

function [H1,H2,It]=QRIter2(H,t)
%QRITER - executa iteratia QR pe matricea HouseHolder
%pana cand cel mai mic element subdiagonal este < t
%Parametrii - ca la QRIter

It=0; [m,n]=size(H);
II=eye(n);
[m, j]=min(abs(diag(H,-1)));
while m>t
    It=It+1;
    H=HessenRQ(H-H(n,n)*II)+H(n,n)*II;
    [m, j]=min(abs(diag(H,-1)));
end
H1=H(1:j,1:j);
H2=H(j+1:n, j+1:n);

```

Există două posibilități: fie ambele valori  $\mu$  și  $\mu'$  ale lui  $B$  sunt reale și atunci se procedează ca mai sus, fie sunt complexe conjugate și avem valorile proprii  $\mu$  și  $\bar{\mu}$ . Așa cum vom vedea, al doilea caz se poate de asemenea trata în aritmetică reală. Fie  $Q_k, Q'_k \in \mathbb{C}^{n \times n}$  și  $R_k, R'_k \in \mathbb{C}^{n \times n}$  matricele descompunerii QR complexe

$$\begin{aligned} Q_k R_k &= H^{(k)} - \mu I, \\ Q'_k R'_k &= R_k Q_k + (\mu - \bar{\mu}) I. \end{aligned}$$

Atunci are loc

$$\begin{aligned} H^{(k+1)} &:= R'_k Q'_k + \mu I = (Q'_k)^* (R_k Q_k + (\mu - \bar{\mu}) I) Q'_k + \bar{\mu} I \\ &= Q_k^* R_k Q_k Q'_k + \mu I = (Q'_k)^* Q_k^* (H^{(k)} - \mu I) Q_k Q'_k + \mu I \\ &= \underbrace{(Q_k Q'_k)^*}_{=U^*} \underbrace{H^{(k)}}_{=U} \underbrace{Q_k Q'_k}_{=U}. \end{aligned}$$

Utilizând matricea  $S = R'_k R_k$  avem

$$\begin{aligned} US &= Q_k Q'_k R'_k R_k = Q_k (R_k Q_k + (\mu - \bar{\mu}) I) R_k \\ &= Q_k R_k Q_k R_k + (\mu - \bar{\mu}) Q_k R_k = (H^{(k)} - \mu I)^2 + (\mu - \bar{\mu})(H^{(k)} - \mu I) \\ &= (H^{(k)})^2 - 2\mu H^{(k)} + \mu^2 I + (\mu - \bar{\mu})H^{(k)} - (\mu^2 - \mu\bar{\mu})I \\ &= (H^{(k)})^2 - (\mu + \bar{\mu})H^{(k)} + \mu\bar{\mu}I =: X \end{aligned} \tag{9.5.3}$$

Dacă  $\mu = \alpha + i\beta$ , atunci  $\mu + \bar{\mu} = 2\alpha$  și  $\mu\bar{\mu} = |\mu|^2 = \alpha^2 + \beta^2$ , aşadar matricea  $X$  din membrul drept al lui (9.5.3), deci are o descompunere QR  $X = QR$  reală și conform lemei 9.4.5 există o matrice de fază  $\Theta \in \mathbb{C}^{n \times n}$  astfel încât  $U = \Theta Q$ . Dacă vom itera în real mai departe, vom obține *metoda QR cu pas dublu*

$$\begin{aligned} Q_k R_k &= (H^{(k)})^2 - (h_{n-1,n-1}^{(k)} + h_{n,n}^{(k)})H^{(k)} \\ &\quad + \left( (h_{n-1,n-1}^{(k)} h_{n,n}^{(k)} - h_{n-1,n}^{(k)} H_{n,n-1}^{(k)}) I \right), \end{aligned} \tag{9.5.4}$$

$$H^{(k+1)} = Q_k^T H^{(k)} Q_k.$$

**Observația 9.5.9 (Metoda QR cu pas dublu).**

1. Matricea  $X$  din (9.5.3) nu mai este o matrice Hessenberg, ea având o diagonală suplimentară. Cu toate acestea se poate calcula descompunerea QR a lui  $X$  destul de simplu, cu doar  $2n - 3$  rotații Jacobi, în loc de  $n - 1$ , cât este necesar pentru o matrice Hessenberg.

2. Datorită complexității sale ridicate, înmulțirea  $Q_k^T H^{(k)} Q_k$  nu mai este o metodă efectivă de iteratie; acest lucru se poate remedia, a se vedea de exemplu [22] sau [60, pag. 272–278].
3. Natural,  $H^{(k+1)}$  poate fi adusă la forma Hessenberg.
4. Metoda cu dublu pas se aplică numai atunci când matricea  $A$  are valori proprii complexe; din contră, la matrice simetrice nu este avantajoasă.  $\diamond$

Metoda QR cu pas dublu, partitioanare și tratarea matricelor  $2 \times 2$  este dată în sursa MATLAB 9.10, fișierul `QRSplit3`. Ea apelează `QRDouble`.

**Sursa MATLAB 9.10 Metoda QR cu dublu pas, partitioanare și tratarea matricelor  $2 \times 2$** 

```
function [lambda,It]=QRSplit3(A,t)
%QRSPLIT3 - calculul val. proprii cu metoda QR partitionare,
%deplasare si tratare speciala a matricelor 2x2
%Intrare
%A - matricea
%t - toleranta
%Rezultate
%lambda - valorile proprii
%It - numar de iteratii

[m,n]=size(A);
if n==1
    It=0;
    lambda=A;
    return
elseif n==2
    It=0;
    lambda=Eigen2x2(A);
    return
else
    H=hessen_h(A); %aducere la forma Hessenberg
    [H1,H2,It]=QRDouble(H,t); %descompunere H->H1,H2
    %apel recursiv
    [l1,It1]=QRSplit3(H1,t);
    [l2,It2]=QRSplit3(H2,t);
    It=It+It1+It2;
    lambda=[l1;l2];
end
```

**Exemplul 9.5.10.** Aplicăm metodele `QRSplit2` și `QRSplit3` matricelor din exemplele 9.5.1 și 9.5.2. Se obțin rezultatele din tabela 9.5. Buna comportare a metodei pasului dublu se justifică prin aceea că obține două valori proprii dintr-o dată.  $\diamond$

$\varepsilon$	#iterații în real		#iterații în complex	
	QRSSplit2	QRSSplit3	QRSSplit2	QRSSplit3
1e-010	1	1	9	4
1e-020	9	2	17	5
1e-030	26	3	45	5

Tabela 9.5: Comparațiile din exemplul 9.5.10

**Sursa MATLAB 9.11 Iterație QR cu dublu pas și transformare Hessenberg**

```
function [H1,H2,It]=QRDouble(H,t)
%QRDOUBLE - executa iteratia QR cu dublu pas si transformare
%inversa pe matricea HouseHolder pana cand cel mai
%mic element subdiagonal este < t

It=0; [m,n]=size(H);
II=eye(n);
[m, j]=min(abs(diag(H,-1)));
while m>t
    It=It+1;
    X = H*H ... %Matricea X
    - (H(n-1,n-1) + H(n,n)) * H ...
    + (H(n-1,n-1)*H(n,n) - H(n,n-1)*H(n-1,n))*II;
    [Q,R]=qr(X);
    H=hessen_h(Q'*H*Q);
    [m, j]=min(abs(diag(H,-1)));
end
H1=H(1:j,1:j);
H2=H(j+1:n,j+1:n);
```

**9.6. Valori și vectori proprii în MATLAB**

MATLAB utilizează rutine LAPACK pentru a calcula valori și vectori proprii. Valorile proprii ale unei matrice se calculează cu funcția `eig`: `e=eig(A)` pune valorile proprii ale lui `A` în vectorul `e`. Forma `[V,D]=eig(A)`, unde `A` este matrice pătratică de ordinul  $n$ , returnează în coloanele lui `V`  $n$  vectori proprii ai lui `A` și în matricea diagonală `D` valorile proprii ale lui `A`. Are loc relația  $A \cdot V = V \cdot D$ . Nu orice matrice are  $n$  vectori proprii liniari independenți, deci matricea `V` returnată de `eig` poate fi singulară (sau datorită erorilor de rotunjire nesingulară, dar foarte prost condiționată). Matricea din exemplul următor are o valoare proprie dublă 1, dar numai un vector propriu liniar independent:

```
>> [V,D]=eig([2, -1; 1, 0])
V =
    0.7071    0.7071
    0.7071    0.7071
D =
    1         0
```

0        1

Vectorii proprii sunt scalări astfel ca norma lor euclidiană să fie egală cu unu (lucru posibil, căci dacă  $x$  este un vector propriu, atunci orice multiplu al său este de asemenea vector propriu).

Dacă  $A$  este hermitiană, atunci MATLAB returnează valorile proprii sortate crescător și matricea vectorilor proprii unitară (în limita preciziei de lucru):

```
>> [V, D]=eig([2,-1;-1,1])
V =
-0.5257 -0.8507
-0.8507 0.5257
D =
0.3820 0
0 2.6180
>> norm(V'*V-eye(2))
ans =
2.2204e-016
```

În exemplul următor vom calcula valorile proprii ale matricei lui Frank (nehermitiană):

```
>> F = gallery('frank', 5)
F =
5 4 3 2 1
4 4 3 2 1
0 3 3 2 1
0 0 2 2 1
0 0 0 1 1
>> e = eig(F)'
e =
10.0629 3.5566 1.0000 0.0994 0.2812
```

Dacă  $\lambda$  este valoare proprie a matricei  $F$ , atunci  $1/\lambda$  este de asemenea valoare proprie:

```
>> 1./e
ans =
0.0994 0.2812 1.0000 10.0629 3.5566
```

Motivul este acela că polinomul caracteristic este anti-palindromic, adică termenii egali depărtați de extrem sunt numere opuse:

```
>> poly(F)
ans =
1.0000 -15.0000 55.0000 -55.0000 15.0000 -1.0000
```

Astfel,  $\det(F - \lambda I) = -\lambda^5 \det(F - \lambda^{-1} I)$ .

Funcția `condeig` calculează numărul de condiționare pentru valori proprii. Aceasta se definește prin

$$\Gamma(A) = \inf \{ \text{cond}(X) : X^{-1}AX = \text{diag}(\lambda_i) \}.$$

Forma `c=condeig(A)` returnează un vector al numerelor de condiționare ale valorilor proprii ale lui  $A$ . Forma `[V, D, s] = condeig(A)` este echivalentă cu: `[V, D] = eig(A)`, `s = condeig(A)`. Un număr de condiționare mare indică o valoare proprie sensibilă la perturbații ale matricei. Exemplul următor afișează în prima linie valorile proprii ale matricei lui Frank de ordinul 6 și în a doua linie numerele lor de condiționare:

```
>> A = gallery('frank', 6);
>> [V,D,s] = condeig(A);
>> [diag(D)'; s']
ans =
12.9736    5.3832    1.8355    0.5448    0.0771    0.1858
1.3059    1.3561    2.0412   15.3255   43.5212   56.6954
```

Dăm în continuare câteva informații despre modul de lucru al funcției `eig`. Ea lucrează în mai multe stadii. Întâi, dacă  $A$  este nesimetrică, ea echilibrează matricea, adică, realizează transformări de similaritate  $A \leftarrow Y^{-1}AY$ , unde  $Y$  este o permutare a unei matrice diagonale aleasă astfel încât să facă liniile și coloanele lui  $A$  de norme aproximativ egale. Motivarea echilibrării este aceea că poate conduce la un calcul mai precis al vectorilor și valorilor proprii. Totuși, uneori echilibrarea nu este necesară și poate fi inhibată cu `eig(A, 'nobalance')` (a se vedea `doc eig` pentru un exemplu).

După echilibrare, `eig` reduce  $A$  la forma Hessenberg superioară (sau triagonală dacă  $A$  este hermitiană). Forma Hessenberg se poate calcula cu  $H = \text{hess}(A)$  sau  $[Q, H] = \text{hess}(A)$ . Ultima formă dă și matricea unitară prin care se face transformarea. Comanda  $T = \text{schur}(A)$  sau  $[Q, T] = \text{schur}(A)$  produce descompunerea Schur a lui  $A$  reală sau complexă, după cum  $A$  este o matrice reală sau complexă. Descompunerea Schur complexă a unei matrice reale se poate obține cu `schur(A, 'complex')`.

MATLAB poate rezolva și probleme de valori proprii generalizate, adică probleme de forma: fiind date două matrice pătratice de ordinul  $n$ ,  $A$  și  $B$ , să se găsească scalarii  $\lambda$  și vectorii  $x \neq 0$  astfel încât  $Ax = \lambda Bx$ . Valorile proprii generalizate se pot calcula cu  $e = \text{eig}(A, B)$ , iar  $[V, D] = \text{eig}(A, B)$  returnează o matrice diagonală  $D$  a valorilor proprii și o matrice pătratică de ordinul  $n$  a vectorilor proprii  $V$  astfel încât  $A \cdot V = B \cdot V \cdot D$ . Teoria corespunzătoare este mai complicată decât cea a valorilor proprii standard: putem să nu avem nici o valoare proprie, putem avea un număr finit de valori proprii sau o infinitate, sau valori proprii infinit de mari. Dacă  $B$  este singulară, se pot obține valori proprii NaN. Dăm un exemplu de rezolvare a unei probleme proprii generalizate:

```
>> A = gallery('triw', 3), B = magic(3)
A =
1     -1     -1
0      1     -1
0      0      1
B =
8      1      6
3      5      7
4      9      2
>> [V,D]=eig(A,B); V, eigvals = diag(D)'
V =
-1.0000    -1.0000    0.3526
 0.4844    -0.4574    0.3867
 0.2199    -0.2516   -1.0000
eigvals =
 0.2751    0.0292   -0.3459
```

Se numește *descompunere cu valori singulare* (singular value decomposition – SVD) descompunerea

$$A = U\Sigma V^*, \quad (9.6.1)$$

unde  $\Sigma$  este o matrice diagonală reală, iar  $U$  și  $V$  sunt matrice unitare (ortogonale în cazul real).

Există două variante de SVD: una completă, care se aplică unei matrice dreptunghiulare  $m \times n$  și care returnează matricele  $U$  de dimensiune  $m \times m$ ,  $\Sigma$  de dimensiune  $m \times n$  și  $V$  de dimensiune  $n \times n$  și una economică sau redusă în care  $U$  are dimensiunea  $m \times n$ ,  $\Sigma$  are dimensiunea  $n \times n$  și  $V$  are dimensiunea  $n \times n$ .

SVD este un instrument util de analiză a aplicațiilor dintr-un spațiu vectorial cu valori în alt spațiu, posibil de dimensiune diferită. Dacă  $A$  este pătratică, simetrică și pozitiv definită SVD (9.6.1) și descompunerea cu valori proprii coincid. Spre deosebire de descompunerea cu valori proprii, SVD există întotdeauna.

Fie matricea

$$A = \begin{matrix} 9 & 4 \\ 6 & 8 \\ 2 & 7 \end{matrix}$$

Descompunerea sa SVD completă este

```
>> [U, S, V]=svd(A)
U =
-0.6105    0.7174    0.3355
-0.6646   -0.2336   -0.7098
-0.4308   -0.6563    0.6194

S =
14.9359      0
      0    5.1883
      0      0

V =
-0.6925    0.7214
-0.7214   -0.6925
```

iar cea redusă

```
>> [U, S, V]=svd(A, 0)
U =
-0.6105    0.7174
-0.6646   -0.2336
-0.4308   -0.6563

S =
14.9359      0
      0    5.1883

V =
-0.6925    0.7214
-0.7214   -0.6925
```

În ambele cazuri se poate verifica că  $U \cdot S \cdot V'$  este egală cu  $A$ , în limita erorilor de rotunjire.

## Probleme

**Problema 9.1.** Calculați valorile proprii ale matricei Hilbert pentru  $n = 10, 11, \dots, 20$  și numerele de condiționare corespunzătoare.

**Problema 9.2.** Matricele

```
P=gallery('pascal',12)
F=gallery('frank',12)
```

au proprietatea că dacă  $\lambda$  este valoare proprie, atunci și  $1/\lambda$  este de asemenea valoare proprie. Cât de bine conservă valorile proprii această proprietate? Utilizați `condeig` pentru a explica comportarea diferită a celor două matrice.

**Problema 9.3.** Care este cea mai mare valoare proprie a lui `magic(n)`? De ce?

**Problema 9.4.** Încercați secvența de comenzi:

```
n=100;
d=ones(n,1);
A=diag(d,1)+diag(d,-1);
e=eig(A);
plot(-n/2:n/2,e,'.')
```

Recunoașteți curba rezultată? Ați putea găsi o formulă pentru valorile proprii ale acestei matrice?

**Problema 9.5.** Studiați valorile și vectorii proprii corespunzători ai matricei la care se ajunge la problema 4.7. Reprezentați grafic valorile și vectorii proprii pentru un  $N$  dat.

**Problema 9.6.** Fie  $T_N$  matricea la care se ajunge la discretizarea cu diferențe a ecuației lui Poisson (problema 4.8). Valorile ei proprii sunt

$$\lambda_j = 2 \left( 1 - \cos \frac{\pi j}{N+1} \right),$$

iar vectorii proprii  $z_j$  au componente

$$z_j(k) = \sqrt{\frac{2}{N+1}} \sin \frac{jk\pi}{N+1}.$$

Să se reprezinte grafic valorile și vectorii proprii ai lui  $T_{21}$ .

**Problema 9.7.** (a) Implementați metoda puterii (iterația vectorială).

(b) Testați funcția de la punctul precedent pentru matricea și vectorul inițial

$$A = \begin{pmatrix} 6 & 5 & -5 \\ 2 & 6 & -2 \\ 2 & 5 & -1 \end{pmatrix}, \quad x = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.$$

(r) Aproximați raza spectrală  $\rho(A)$  a matricei

$$A = \begin{pmatrix} 2 & 0 & -1 \\ -2 & -10 & 0 \\ -1 & -1 & 4 \end{pmatrix},$$

utilizând metoda puterii și vectorul inițial  $(1, 1, 1)^T$ .

**Problema 9.8.** Determinați valorile proprii ale matricei

$$\begin{pmatrix} 190 & 66 & -84 & 30 \\ 66 & 303 & 42 & -36 \\ 336 & -168 & 147 & -112 \\ 30 & -36 & 28 & 291 \end{pmatrix}$$

folosind metoda QR cu dublu pas. Să se compare rezultatul cu cel furnizat de e.i.g.

**Problema 9.9.** Determinați descompunerile cu valori singulare ale următoarelor matrice:

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 7 \\ 0 & 0 & 0 \end{bmatrix} \quad [2 \quad 1] \quad \begin{bmatrix} 5 \\ 4 \end{bmatrix}.$$

# CAPITOLUL 10

## Rezolvarea numerică a ecuațiilor diferențiale ordinare

### 10.1. Ecuații diferențiale

Considerăm problema cu valori inițiale (problema Cauchy<sup>1)</sup>: să se determine o funcție cu valori vectoriale  $y \in C^1[a, b]$ ,  $y : [a, b] \rightarrow \mathbb{R}^d$ , astfel încât

$$\begin{cases} \frac{dy}{dx} = f(x, y), & x \in [a, b] \\ y(a) = y_0 \end{cases} \quad (10.1.1)$$

Vom evidenția două clase importante de astfel de probleme:

- (i) pentru  $d = 1$  avem o singură ecuație diferențială scalară de ordinul I

$$\begin{cases} y' = f(x, y) \\ y(a) = y_0 \end{cases}$$

- (ii) pentru  $d > 1$  avem un sistem de ecuații diferențiale ordinare de ordinul I

$$\begin{cases} \frac{dy^i}{dx} = f^i(x, y^1, y^2, \dots, y^d), & i = \overline{1, d} \\ y^i(a) = y_0^i, & i = \overline{1, d} \end{cases}$$

Augustin Louis Cauchy (1789-1857), matematician francez, considerat părintele analizei moderne. A fundamentat solid analiza pe baza conceptului riguros de limită. Este de asemenea creatorul analizei complexe, în care „formula lui Cauchy” ocupă un loc central. Numele său este legat și de contribuții de pionierat în domeniul ecuațiilor diferențiale și cu derivate partiale, în particular legate de problema existenței și unicității. La fel ca în cazul multor mari matematicieni din secolele al optșprezecelea și al nouăsprezecelea, lucrările sale au tratat probleme din geometrie, algebră, teoria numerelor, mecanică, dar și fizică teoretică.



Reamintim următoarea teoremă clasică referitoare la existență și unicitate.

**Teorema 10.1.1.** *Presupunem că  $f(x, y)$  este continuă în prima variabilă pentru  $x \in [a, b]$  și în raport cu cea de-a doua variabilă satisfacă o condiție Lipschitz uniformă*

$$\|f(x, y) - f(x, y^*)\| \leq L\|y - y^*\|, \quad y, y^* \in \mathbb{R}^d, \quad (10.1.2)$$

unde  $\|\cdot\|$  este o anumită normă vectorială. Atunci problema Cauchy (10.1.1) are o soluție unică  $y(x)$ ,  $a \leq x \leq b$ ,  $\forall y_0 \in \mathbb{R}^d$ . Mai mult,  $y(x)$  depinde continuu de  $a$  și  $y_0$ .

Condiția Lipschitz (10.1.2) are singur loc dacă toate funcțiile  $\frac{\partial f^i}{\partial y^j}(x, y)$ ,  $i, j = \overline{1, d}$  sunt continue în raport cu variabilele  $y$  și sunt mărginite pe  $[a, b] \times \mathbb{R}^d$ . Aceasta este situația în cazul sistemelor de ecuații diferențiale ordinare liniare, unde

$$f^i(x, y) = \sum_{j=1}^d a_{ij}(x)y^j + b_i(x), \quad i = \overline{1, d}$$

și  $a_{ij}(x), b_i(x)$  sunt funcții continue pe  $[a, b]$ .

De multe ori condiția Lipschitz (10.1.2) are loc într-o vecinătate a lui  $x_0$  astfel încât  $y(x)$  să rămână într-un compact  $D$ .

## 10.2. Metode numerice

Se face distincție între *metode de aproximare analitice* și *metode discrete*. În cadrul primei categorii se încearcă să se găsească aproximări  $y_a(x) \approx y(x)$  ale soluției exacte, valabile pentru orice  $x \in [a, b]$ . Acestea de obicei au forma unei dezvoltări într-o serie trunchiată, fie după puterile lui  $x$ , fie în polinoame Cebâșev, fie într-un alt sistem de funcții de bază. În cazul metodelor discrete, se încearcă să se găsească aproximării  $u_n \in \mathbb{R}^d$  ale lui  $y(x_n)$  pe o grilă de puncte  $x_n \in [a, b]$ . Abscisele  $x_n$  pot fi predeterminate (de exemplu puncte echidistante pe  $[a, b]$ ), sau mai convenabil sunt generate dinamic ca parte a procesului de integrare.

Dacă se dorește, se pot obține din aceste aproximante discrete  $\{u_n\}$  aproximante  $y_n(x)$  definite pe întreg intervalul  $[a, b]$ , fie prin interpolare, sau mai natural, printr-un mecanism de continuare conținut în metoda de aproximare însăși. Ne vom ocupa numai de metode discrete cu un pas, adică metode în care  $u_{n+1}$  este determinat cunoscând numai  $x_n, u_n$  și pasul  $h$  pentru a trece de la  $x_n$  la  $x_{n+1} = x_n + h$ . Într-o metodă cu  $k$  pași ( $k > 1$ ) este necesară cunoașterea a  $k - 1$  puncte adiționale  $(x_{n-j}, u_{n-j})$ ,  $j = 1, 2, \dots, k - 1$  pentru a obține o nouă componentă a soluției.

Când se descrie o metodă cu un pas este suficient să arătăm cum se trece de la un punct generic  $(x, y)$ ,  $x \in [a, b]$ ,  $y \in \mathbb{R}^d$  la punctul următor  $(x+h, y_{next})$ . Ne vom referi la aceasta ca fiind *descrierea locală* a unei metode cu un pas. Aceasta include de asemenea o discuție a preciziei locale, adică cât de apropiat este  $y_{next}$  de soluție în  $x+h$ . O metodă cu un pas pentru rezolvarea problemei Cauchy (10.1.1) generează efectiv o funcție grilă  $\{u_n\}$ ,  $u_n \in \mathbb{R}^d$ , pe o grilă  $a = x_0 < x_1 < x_2 < \dots < x_{N-1} < x_N = b$  ce acoperă intervalul  $[a, b]$ , prin care se intenționează ca  $u_n$  să aproximeze soluția exactă  $y(x)$  în  $x = x_n$ . Punctul  $(x_{n+1}, u_{n+1})$  se obține din punctul  $(x_n, u_n)$  aplicând o metodă cu un pas, având un pas  $h_n = x_{n+1} - x_n$  adevărat ales. Ne vom referi la aceasta ca *descrierea globală* a unei metode cu un pas. Chestiunile de interes aici sunt comportarea erorii globale  $u_n - y(x_n)$ , în particular stabilitatea, convergența și alegerea lui  $h_n$  pentru a trece de la un punct  $x_n$  al grilei la următorul,  $x_{n+1} = x_n + h_n$ .

### 10.3. Descrierea locală a metodelor cu un pas

Dându-se un punct generic  $x \in [a, b]$ ,  $y \in \mathbb{R}^d$ , definim un pas al metodei cu un pas prin

$$y_{next} = y + h\Phi(x, y; h), \quad h > 0. \quad (10.3.1)$$

Funcția  $\Phi : [a, b] \times \mathbb{R}^d \times \mathbb{R}_+ \rightarrow \mathbb{R}^d$  poate fi gândită ca un increment aproximativ pe unitatea de pas sau ca o aproximare a diferenței divizate și ea definește metoda. Împreună cu (10.3.1) considerăm soluția  $u(t)$  a ecuației diferențiale (10.1.1) ce trece prin punctul  $(x, y)$ , adică problema locală cu valoarea inițială

$$\begin{cases} \frac{du}{dt} = f(t, u) \\ u(x) = y \end{cases} \quad t \in [x, x + h] \quad (10.3.2)$$

Vom numi  $u(t)$  soluție de referință. Se intenționează ca vectorul  $y_{next}$  din (10.3.1) să aproximeze  $u(x + h)$ . Cu cât succes se realizează aceasta se măsoară prin eroarea locală de trunchiere, definită după cum urmează.

**Definiția 10.3.1.** Eroarea de trunchiere a metodei  $\Phi$  în punctul  $(x, y)$  este definită prin

$$T(x, y; h) = \frac{1}{h}[y_{next} - u(x + h)]. \quad (10.3.3)$$

Eroarea de trunchiere este o funcție cu valori vectoriale de  $d + 2$  variabile. Utilizând (10.3.1) și (10.3.2) o putem scrie sub formă

$$T(x, y; h) = \Phi(x, y; h) - \frac{1}{h}[u(x + h) - u(x)], \quad (10.3.4)$$

ceea ce arată că  $T$  este diferența între incrementul aproximativ și cel exact pe unitatea de pas.

**Definiția 10.3.2.** Metoda  $\Phi$  se numește consistentă dacă

$$T(x, y; h) \rightarrow 0 \text{ când } h \rightarrow 0, \quad (10.3.5)$$

uniform pentru  $(x, y) \in [a, b] \times \mathbb{R}^d$ .

Conform lui (10.3.4) și (10.3.3) avem consistență dacă și numai dacă

$$\Phi(x, y; 0) = f(x, y), \quad x \in [a, b], \quad y \in \mathbb{R}^d. \quad (10.3.6)$$

O descriere mai fină a preciziei locale este furnizată de definiția următoare, bazată pe conceptul de eroare de trunchiere.

**Definiția 10.3.3.** Spunem că metoda  $\Phi$  are ordinul  $p$  dacă pentru o anumită normă vectorială  $\|\cdot\|$

$$\|T(x, y; h)\| \leq Ch^p, \quad (10.3.7)$$

uniform pe  $[a, b] \times \mathbb{R}^d$ , cu constanta  $C$  independentă de  $x, y$  și  $h$ .

Această proprietate se mai poate exprima sub forma

$$T(x, y; h) = O(h^p), \quad h \rightarrow 0. \quad (10.3.8)$$

De notat că  $p > 0$  implică consistență. De obicei  $p \in \mathbb{N}^*$ . El se numește *ordin exact*, dacă (10.3.7) nu are loc pentru nici un  $p$  mai mare.

**Definiția 10.3.4.** O funcție  $\tau : [a, b] \times \mathbb{R}^d \rightarrow \mathbb{R}^d$  care satisface  $\tau(x, y) \not\equiv 0$  și

$$T(x, y; h) = \tau(x, y)h^p + O(h^{p+1}), \quad h \rightarrow 0 \quad (10.3.9)$$

se numește funcție de eroare principală.

Funcția de eroare principală determină termenul principal (dominant) al erorii de trunchiere. Numărul  $p$  din (10.3.9) este ordinul exact al metodei deoarece  $\tau \not\equiv 0$ .

Toate definițiile precedente sunt formulate în ideea că  $h > 0$  este un număr mic. Cu cât  $p$  este mai mare, cu atât metoda este mai precisă.

## 10.4. Exemple de metode cu un pas

Unele dintre metodele cele mai vechi sunt motivate prin considerații geometrice simple asupra pantei definite de membrul drept al ecuației diferențiale. În această categorie intră metoda lui Euler și metoda lui Euler modificată. Alte metode mai precise și mai sofisticate se bazează pe dezvoltarea Taylor.

### 10.4.1. Metoda lui Euler

Euler a propus metoda sa în 1768, la începutul istoriei calculului diferențial și integral. Ea constă pur și simplu în a urma panta în punctul generic  $(x, y)$  pe un interval de lungime  $h$

$$y_{next} = y + hf(x, y). \quad (10.4.1)$$

(vezi figura 10.1).

Astfel  $\Phi(x, y; h) = f(x, y)$  nu depinde de  $h$  și conform lui (10.3.6) metoda este consistentă. Pentru eroarea de trunchiere avem conform lui (10.3.3)

$$T(x, y; h) = f(x, y) - \frac{1}{h}[u(x+h) - u(x)], \quad (10.4.2)$$

unde  $u(t)$  este soluția de referință definită de (10.3.2). Deoarece  $u' = f(x, u(x)) = f(x, y)$ , putem scrie, utilizând formula lui Taylor

$$\begin{aligned} T(x, y; h) &= u'(x) - \frac{1}{h}[u(x+h) - u(x)] = \\ &= u'(x) - \frac{1}{h}[u(x) + hu'(x) + \frac{1}{2}h^2u''(\xi) - u(x)] = \\ &= -\frac{1}{2}hu''(\xi), \quad \xi \in (x, x+h), \end{aligned} \quad (10.4.3)$$

presupunând că  $u \in C^2[x, x+h]$ . Aceasta este adevărată dacă  $f \in C^1([a, b] \times \mathbb{R}^d)$ . Diferențind acum total (10.3.2) în raport cu  $t$  și făcând  $t = \xi$ , suntem conduși la

$$T(x, y; h) = -\frac{1}{2}h[f_x + f_y f](\xi, u(\xi)), \quad (10.4.4)$$

unde  $f_x$  este derivata parțială a lui  $f$  în raport cu  $x$  și  $f_y$  este jacobianul lui  $f$  în raport cu variabila  $y$ . Dacă în spiritul teoremei 10.1.1, presupunem că  $f$  și toate derivatele sale parțiale de ordinul I sunt uniform mărginite în  $[a, b] \times \mathbb{R}^d$ , există o constantă  $C$ , independentă de  $x, y$  și  $h$  astfel încât

$$\|T(x, y; h)\| \leq Ch. \quad (10.4.5)$$

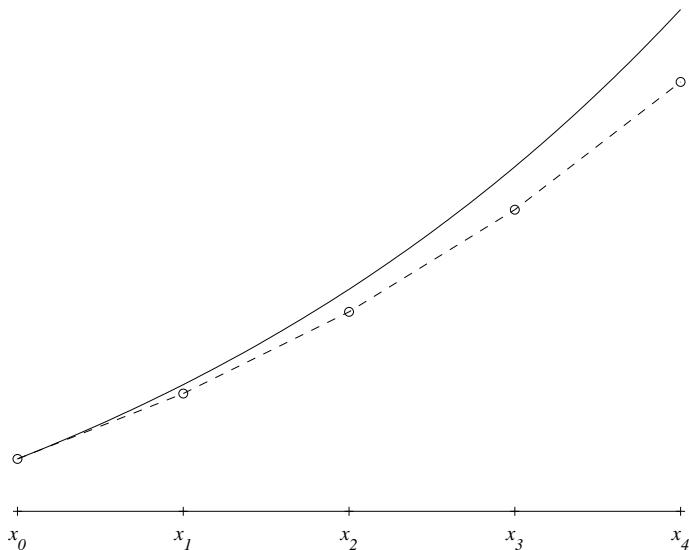


Figura 10.1: Metoda lui Euler – soluția exactă (linie continuă) și soluția aproximativă (linie punctată)

Astfel, metoda lui Euler are ordinul  $p = 1$ . Dacă facem aceeași presupunere și despre derivatele parțiale de ordinul doi ale lui  $f$  avem

$$u''(\xi) = u''(x) + O(h)$$

și de aceea din (10.4.3) rezultă

$$T(x, y; h) = -\frac{1}{2}h[f_x + f_y f](x, y) + O(h^2), \quad h \rightarrow 0, \quad (10.4.6)$$

arătând că funcția eroare principală este dată de

$$\tau(x, y) = -\frac{1}{2}[f_x + f_y f](x, y). \quad (10.4.7)$$

Exceptând situația când  $f_x + f_y f \equiv 0$ , ordinul exact al metodei lui Euler este  $p = 1$ .

### 10.4.2. Metoda dezvoltării Taylor

Am văzut că metoda lui Euler se bazează pe trunchierea dezvoltării Taylor a soluției de referință după cel de-al doilea termen. Este o idee naturală, propusă încă de Euler, de a utiliza mai mulți termeni din dezvoltarea Taylor. Aceasta necesită calculul succesiv al „derivatelor totale“ ale lui  $f$ ,

$$\begin{aligned} f^{[0]}(x, y) &= f(x, y) \\ f^{[k+1]}(x, y) &= f_x^{[k]}(x, y) + f_y^{[k]}(x, y)f(x, y), \quad k = 0, 1, 2, \dots \end{aligned} \quad (10.4.8)$$

care determină derivatele succesive ale soluției de referință  $u(t)$  a lui (10.3.2) în virtutea relației

$$u^{(k+1)}(t) = f^{[k]}(t, u(t)), \quad k = 0, 1, 2, \dots \quad (10.4.9)$$

Acestea, pentru  $t = x$  devin

$$u^{(k+1)}(x) = f^{[k]}(x, y), \quad k = 0, 1, 2, \dots \quad (10.4.10)$$

și sunt utilizate pentru a scrie dezvoltarea Taylor conform cu

$$y_{next} = y + h \left[ f^{[0]}(x, y) + \frac{1}{2}hf^{[1]}(x, y) + \dots + \frac{1}{p!}h^{p-1}f^{[p-1]}(x, y) \right], \quad (10.4.11)$$

adică

$$\Phi(x, y; h) = f^{[0]}(x, y) + \frac{1}{2}hf^{[1]}(x, y) + \dots + \frac{1}{p!}h^{p-1}f^{[p-1]}(x, y). \quad (10.4.12)$$

Pentru eroarea de trunchiere, utilizând (10.4.10) și (10.4.12) și presupunând că  $f \in C^p([a, b] \times \mathbb{R}^d)$  se obține din teorema lui Taylor

$$\begin{aligned} T(x, y; h) &= \Phi(x, y; h) - \frac{1}{h}[u(x+h) - u(x)] = \\ &= \Phi(x, y; h) - \sum_{k=0}^{p-1} u^{(k+1)}(x) \frac{h^k}{(k+1)!} - u^{(p+1)}(\xi) \frac{h^p}{(p+1)!} = \\ &= -u^{(p+1)}(\xi) \frac{h^p}{(p+1)!}, \quad \xi \in (x, x+h), \end{aligned}$$

așa că

$$\|T(x, y; h)\| \leq \frac{C_p}{(p+1)!} h^p,$$

unde  $C_p$  este o margine a derivatei totale de ordin  $p$  a lui  $f$ . Astfel metoda are ordinul exact  $p$  (în afară de cazul când  $f^{[p]}(x, y) \equiv 0$ ) și funcția eroare principală este

$$\tau(x, y) = -\frac{1}{(p+1)!} f^{[p]}(x, y). \quad (10.4.13)$$

Necesitatea calculului multor derivate parțiale în (10.4.8) a fost un factor descurajant în trecut, când se făcea cu mâna. Dar în zilele noastre această sarcină poate cădea în seama calculatorului, astfel încât metoda a devenit din nou o opțiune viabilă.

### 10.4.3. Metode de tip Euler îmbunătățite

Există prea multă inerție în metoda lui Euler: nu ar trebui urmată aceeași pantă pe întreg intervalul de lungime  $h$ , deoarece de-a lungul acestui segment de dreapta panta definită de ecuația diferențială se schimbă. Aceasta sugerează mai multe alternative. De exemplu am putea să reevaluăm panta la mijlocul segmentului – să luăm pulsul ecuației diferențiale – și apoi să urmăram panta actualizată pe întreg intervalul (vezi figura 10.2). Formula este

$$y_{next} = y + hf \left( x + \frac{1}{2}h, y + \frac{1}{2}hf(x, y) \right) \quad (10.4.14)$$

sau

$$\Phi(x, y; h) = f \left( x + \frac{1}{2}h, y + \frac{1}{2}hf(x, y) \right). \quad (10.4.15)$$

Observați „imbricarea” necesară aici. Pentru programarea acestei metode este indicat să se scrie

$$\begin{aligned} K_1(x, y) &= f(x, y) \\ K_2(x, y; h) &= f \left( x + \frac{1}{2}h, u + \frac{1}{2}hK_1 \right) \\ y_{next} &= y + hK_2 \end{aligned} \quad (10.4.16)$$

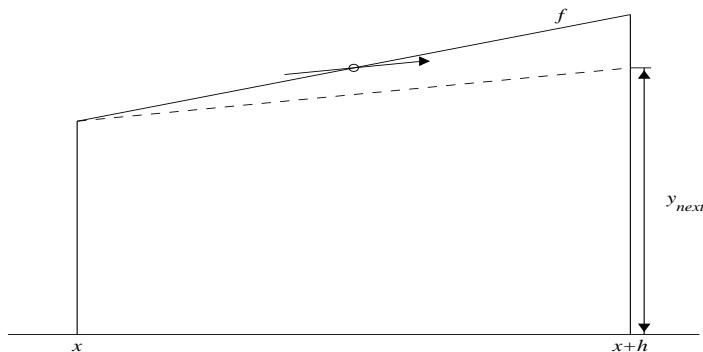


Figura 10.2: Metoda lui Euler modificată

Cu alte cuvinte, încercăm să luăm două pante de test  $K_1$  și  $K_2$ , una în punctul inițial și a doua în apropiere și apoi să-o alegem pe ultima ca pantă finală. Metoda se numește *metoda lui Euler modificată*.

Putem la fel de bine să luăm o și două pantă de încercare  $(x+h, y+hf(x, y))$ , dar atunci, deoarece trebuie să aşteptăm prea mult înainte de a reevalua panta, luăm ca pantă finală media două pante

$$\begin{aligned} K_1(x, y) &= f(x, y) \\ K_2(x, y; h) &= f(x + h, y + hK_1) \\ y_{\text{next}} &= y + \frac{1}{2}h(K_1 + K_2). \end{aligned} \tag{10.4.17}$$

Această metodă se numește *metoda lui Heun*. Efectul ambelor modificări este creșterea ordinului cu 1, așa cum se va vedea în continuare.

## 10.5. Metode Runge-Kutta

Se caută  $\Phi$  de forma:

$$\begin{aligned} \Phi(x, y; h) &= \sum_{s=1}^r \alpha_s K_s \\ K_1(x, y) &= f(x, y) \\ K_s(x, y) &= f \left( x + \mu_s h, y + h \sum_{j=1}^{s-1} \lambda_{sj} K_j \right), \quad s = 2, 3, \dots, r \end{aligned} \tag{10.5.1}$$

Este natural să impunem în (10.5.1) condițiile

$$\mu_s = \sum_{j=1}^{s-1} \lambda_{sj}, \quad s = 2, 3, \dots, r, \quad \sum_{s=1}^r \alpha_s = 1, \tag{10.5.2}$$

unde primul set de condiții este echivalent cu

$$K_s(x, y; h) = u'(x + \mu_s h) + O(h^2), \quad s \geq 2,$$

iar a doua este condiția de consistență (10.3.6) (adică  $\Phi(x, y; h) = f(x, y)$ ).

Vom numi metoda (10.5.1) *metodă Runge-Kutta explicită în r stadii* deoarece necesită r evaluări ale funcției  $f$  din membrul drept al ecuației diferențiale. Condițiile (10.5.2) conduc la un sistem neliniar. Fie  $p^*(r)$  ordinul maxim pentru o metodă Runge-Kutta explicită în r stadii. Kutta<sup>2</sup> a arătat în 1901 că

$$p^*(r) = r, \quad r = \overline{1, 4}.$$

Se pot considera *metode Runge-Kutta implice* cu r stadii

$$\begin{aligned} \Phi(x, y; h) &= \sum_{s=1}^r \alpha_s K_s(x, y; h), \\ K_s &= f \left( x + \mu_s h, y + \sum_{j=1}^r \lambda_{sj} K_j \right), \quad s = \overline{1, r}, \end{aligned} \tag{10.5.3}$$

în care ultimele  $r$  ecuații formează un sistem de ecuații (în general neliniar) cu necunoscutele  $K_1, K_2, \dots, K_r$ . Deoarece fiecare dintre necunoscute este un vector din  $\mathbb{R}^d$ , înainte de construirea incrementului aproximativ  $\Phi$  trebuie să rezolvăm un sistem de  $rd$  ecuații cu  $rd$  necunoscute. *Metodele Runge-Kutta semiimplicite*, la care limitele de sumare merg de la  $j = 1$  la  $j = s$  necesită un efort mai mic. Se ajunge la un sistem de  $r$  ecuații, fiecare având  $d$  necunoscute, componentele lui  $K_s$ . Volumul considerabil de calcule necesar în metodele implice și semiimplicite se justifică numai în împrejurări speciale, de exemplu la rezolvarea problemelor stiff. Motivul este acela că metodele implice pot avea ordin mai mare și proprietăți de stabilitate mai bune.

Parametrii se aleg astfel ca ordinul să fie cât mai mare posibil.

**Exemplul 10.5.1.** Fie

$$\Phi(x, y; h) = \alpha_1 K_1 + \alpha_2 K_2,$$

unde

$$\begin{aligned} K_1(x, y) &= f(x, y), \\ K_2(x, y; h) &= f(x + \mu_2 h, y + \lambda_{21} h K_1), \\ \lambda_{21} &= \mu_2. \end{aligned}$$

Avem deci 3 parametri,  $\alpha_1, \alpha_2, \mu$ . Un mod sistematic de a determina ordinul maxim  $p$  este de a dezvolta atât  $\Phi(x, y; h)$  cât și  $h^{-1}[u(x + h) - u(x)]$  după puterile lui  $h$  și să impunem coincidența a cât mai multor termeni posibili, fără a impune restricții asupra lui  $f$ . Pentru a dezvolta  $\Phi$  avem nevoie de dezvoltarea Taylor a unei funcții vectoriale de mai multe variabile

$$\begin{aligned} f(x + \Delta x, y + \Delta y) &= f + f_x \Delta x + f_y \Delta y + \\ &+ \frac{1}{2} [f_{xx}(\Delta x)^2 + 2f_{xy}\Delta x \Delta y + (\Delta y)^T f_{yy}(\Delta y)] + \dots, \end{aligned} \tag{10.5.4}$$

Wilhelm Martin Kutta (1867-1944), matematician german, cu pre-ocupări în domeniul matematicilor aplicate. Cunoscut pentru lucrările sale în domeniul rezolvării numerice a ecuațiilor diferențiale ordinare, a avut și contribuții al aplicarea transformărilor conforme la probleme de hidro și aerodinamică (formula Kutta-Jukovski).



unde  $f_y$  este jacobianul lui  $f$ , iar  $f_{yy} = [f_{yy}^i]$  este vectorul matricelor hessiene ale lui  $f$ . În formula de mai sus toate funcțiile și derivatele parțiale se evaluatează în  $(x, y)$ . Punând  $\Delta x = \mu h$ ,  $\Delta y = \mu h f$  obținem

$$\begin{aligned} K_2(x, y; h) &= f + \mu h(f_x + f_y f) \\ &\quad + \frac{1}{2}\mu^2 h^2(f_{xx} + 2f_{xy}f + f^T f_{yy}f) + O(h^3) \end{aligned} \quad (10.5.5)$$

$$\frac{1}{h}[u(x+h) - u(x)] = u'(x) + \frac{1}{2}hu''(x) + \frac{1}{6}u'''(x) + O(h^3), \quad (10.5.6)$$

unde

$$\begin{aligned} u'(x) &= f \\ u''(x) &= f^{[1]} = f_x + f_y f \\ u'''(x) &= f^{[2]} = f_x^{[1]} + f_y^{[1]} f = f_{xx} + f_x f_y f + f_y f_x + (f_{xy} + (f_y f)_y) f = \\ &= f_{xx} + 2f_{xy}f + f^T f_{yy}f + f_y(f_x + f_y)f \end{aligned}$$

și unde în ultima ecuație s-a utilizat

$$(f_y f)_y f = f^T f_{yy}f + f_y^2 f.$$

Avem

$$T(x, y; h) = \alpha_1 K_1 + \alpha_2 K_2 - \frac{1}{h}[u(x+h) - u(x)]$$

în care înlocuim (10.5.5) și (10.5.6). Găsim

$$\begin{aligned} T(x, y; h) &= (\alpha_1 + \alpha_2 - 1)f + \left(\alpha_2\mu - \frac{1}{2}\right)h(f_x + f_y f) + \\ &\quad + \frac{1}{2}h^2 \left[\left(\alpha_2\mu^2 - \frac{1}{3}\right)(f_{xx} + 2f_{xy}f + f^T f_{yy}f) - \frac{1}{3}f_y(f_x + f_y f)\right] + O(h^3) \end{aligned} \quad (10.5.7)$$

Nu putem impune asupra coeficientului  $h^2$  condiția ca el să fie zero decât dacă impunem restricții severe asupra lui  $f$ . Astfel ordinul maxim este 2 și el se obține pentru

$$\begin{cases} \alpha_1 + \alpha_2 = 1 \\ \alpha_2\mu = \frac{1}{2} \end{cases}$$

Soluția

$$\alpha_1 = 1 - \alpha_2$$

$$\mu = \frac{1}{2\alpha_2}$$

deinde de un parametru,  $\alpha_2 \neq 0$ , arbitrar. Pentru  $\alpha_2 = 1$  avem metoda lui Euler modificată, iar pentru  $\alpha_2 = \frac{1}{2}$  metoda lui Heun.  $\diamond$

Vom menționa formula Runge-Kutta clasică de ordin  $p = 4$ .

$$\begin{aligned} \Phi(x, y; h) &= \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ K_1(x, y; h) &= f(x, y) \\ K_2(x, y; h) &= f\left(x + \frac{1}{2}h, y + \frac{1}{2}hK_1\right) \\ K_3(x, y; h) &= f\left(x + \frac{1}{2}h, y + \frac{1}{2}hK_2\right) \\ K_4(x, y; h) &= f(x + h, y + hK_3) \end{aligned} \quad (10.5.8)$$

Dacă  $f$  nu depinde de  $y$ , atunci metoda (10.5.8) coincide cu formula lui Simpson. Runge<sup>3</sup> a avut ideea de a generaliza formula lui Simpson la ecuații diferențiale ordinare. El a reușit doar parțial, formula sa având  $r = 4$  și  $p = 3$ . Metoda (10.5.8) a fost descoperită de Kutta în 1901 printr-o căutare sistematică.

În cazul când  $f$  nu depinde de  $y$ , atunci (10.5.8) se reduce la formula lui Simpson. Metoda Runge-Kutta clasică de ordinul 4 pentru o grilă de  $N + 1$  puncte echidistante este dată de funcția MATLAB 10.1.

---

### Sursa MATLAB 10.1 Metoda Runge-Kutta de ordinul 4

---

```
function [t,w]=RK4(f,tspan,alfa,N)
%RK4 metoda Runge-Kutta clasica cu noduri echidistante
%apel [t,w]=RK4(f,tspan,alfa,N)
%f - functia din membrul drept
%tspan - intervalul
%alfa - valoarea de pornire
%N - numarul de subintervale
%t - abscisele solutiei
%w - ordonatele

tc=tspan(1); wc=alfa(:);
h=(tspan(end)-tspan(1))/N;
t=tc; w=wc';
for k=1:N
    K1=f(tc,wc);
    K2=f(tc+1/2*h,wc+1/2*h*K1);
    K3=f(tc+1/2*h,wc+1/2*h*K2);
    K4=f(tc+h, wc+h*K3);
    wc=wc+h/6*(K1+2*K2+2*K3+K4);
    tc=tc+h;
    t=[t;tc]; w=[w;wc'];
end
```

---

**Exemplul 10.5.2.** Utilizând metoda Runge-Kutta de ordinul 4 pentru a aproxima soluția problemei Cauchy

$$\begin{aligned} y' &= -y + t + 1, \quad t \in [0, 1] \\ y(0) &= 1, \end{aligned}$$


---

Carle David Tolm  Runge (1856-1927), matematician german, membru al școlii matematice de la G tingen și unul dintre pionierii matematicii numerice. Este cunoscut pentru metodele Runge-Kutta din domeniul rezolv rii numerice a ecua ilor diferen iale ordinare, ale c rori idei de baz  i se datorează. A avut contribu ii notabile și  n domeniul aproxim rilor  n planul complex.



$t_i$	Aproximante	Valori exacte	Eroarea
0.0	1	1	0
0.1	1.00483750000	1.00483741804	8.19640e-008
0.2	1.01873090141	1.01873075308	1.48328e-007
0.3	1.04081842200	1.04081822068	2.01319e-007
0.4	1.07032028892	1.07032004604	2.42882e-007
0.5	1.10653093442	1.10653065971	2.74711e-007
0.6	1.14881193438	1.14881163609	2.98282e-007
0.7	1.19658561867	1.19658530379	3.14880e-007
0.8	1.24932928973	1.24932896412	3.25617e-007
0.9	1.30656999120	1.30656965974	3.31459e-007
1.0	1.36787977441	1.36787944117	3.33241e-007

Tabela 10.1: Rezultate numerice pentru exemplul 10.5.2

cu  $h = 0.1$ ,  $N = 10$  și  $t_i = 0.1i$  se obțin rezultatele din tabelul 10.1. Soluția exactă este  $y(t) = e^{-t} + t$ , iar sevența de apel

`[t, w]=rk4(@edex1, [0, 1], 1, 10);`

Funcția MATLAB

```
function df=edex1(t,y)
df=-y+t+1;
```

definește membrul drept.

◇

Se obișnuiește să se asocieze unei metode Runge-Kutta cu  $r$  stadii (10.5.3) tabloul

$$\begin{array}{c|cccc} \mu_1 & \lambda_{11} & \lambda_{12} & \dots & \lambda_{1r} \\ \mu_2 & \lambda_{21} & \lambda_{22} & \dots & \lambda_{2r} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \mu_r & \lambda_{r1} & \lambda_{r2} & \dots & \lambda_{rr} \\ \hline & \alpha_1 & \alpha_2 & \dots & \alpha_r \end{array} \quad \left( \text{în formă matricială } \begin{array}{c|c} \mu & \Lambda \\ \hline & \alpha^T \end{array} \right)$$

numit *tabelă Butcher*. Pentru o metodă explicită  $\mu_1 = 0$  și  $\Lambda$  este triunghiulară inferior, cu zerouri pe diagonală. Putem asocia primelor  $r$  linii ale tabelei Butcher o formulă de quadratură  $\int_0^{\mu_s} u(t) dt \approx \sum_{j=1}^r \lambda_{sj} u(\mu_j)$ ,  $s = \overline{1, r}$  și ultimei linii formula de quadratură  $\int_0^1 u(t) dt \approx \sum_{s=1}^r \alpha_s u(\mu_s)$ . Dacă gradele de exactitate respective sunt  $d_s = q_s - 1$ ,  $1 \leq s \leq r + 1$  ( $d_s = \infty$  dacă  $\mu_s = 0$  și toti  $\lambda_{sj} = 0$ ) atunci din teorema lui Peano rezultă că în reprezentarea restului apar derivatele de ordinul  $q_s$  ale lui  $u$  și deci punând  $u(t) = y'(x + th)$  se obține

$$\frac{y(x + \mu_s h) - y(x)}{h} - \sum_{j=1}^r \lambda_{sj} y'(x + \mu_j h) = O(h^{q_s}), \quad s = \overline{1, r}$$

și

$$\frac{y(x + h) - y(x)}{h} - \sum_{s=1}^r \alpha_s y'(x + \mu_s h) = O(h^{q_{r+1}}).$$

Pentru metoda Runge-Kutta clasică de ordinul patru (10.5.8) tabela Butcher este:

0	0			
$\frac{1}{2}$	$\frac{1}{2}$	0		
$\frac{1}{2}$	0	$\frac{1}{2}$	0	
1	0	0	1	0
	$\frac{1}{6}$	$\frac{2}{6}$	$\frac{2}{6}$	$\frac{1}{6}$

Sursa MATLAB 10.2 este un exemplu de implementare a unei metode Runge-Kutta cu pas constant când se cunoaște tabela Butcher. Ultimul parametru al funcției Runge\_Kutta este o procedură de

---

### Sursa MATLAB 10.2 Implementarea unei metode Runge-Kutta cu pas constant cu tabelă Butcher

---

```

function [x,y,nfev]=Runge_Kutta(f,tspan,y0,N,BT)
%RUNGE-KUTTA metoda Runge-Kutta cu pas constant
%apel [t,y,nfev]=Runge_Kutta(f,tspan,y0,N,BT)
%f -functia din membrul drept
%tspan - intervalul [a,b]
%y0 - valoarea de pornire
%N - numarul de pasi
%BT - procedura care furnizeaza tabela Butcher sub forma
%      [lambda,alfa,mu,s] - s numarul de stadii
%t -abscisele
%y ordonatele componentelor
%nfev - numarul de evaluari de functii

[lambda,alfa,mu,r]=BT(); %initializare tabela Butcher
h=(tspan(end)-tspan(1))/N; %lungime pas
xc=tspan(1); yc=y0(:); x=xc; y=yc'; K=zeros(length(y0),r);
for k=1:N %iteratia RK
    K(:,1)=f(xc,yc);
    for i=2:r
        K(:,i)=f(xc+mu(i)*h,yc+h*(K(:,1:i-1)*lambda(i,1:i-1)'));
    end
    yc=yc+h*(K*alfa);
    xc=xc+h; %pregatesc iteratia urmatoare
    x=[x;xc]; y=[y;yc'];
end if nargout==3
nfev=r*N;
end

```

---

initializare a tabelei Butcher care returnează elementele  $\mu$ ,  $\lambda$  și  $\alpha$  ale tabelei și numărul de stadii  $r$  al metodei. Pentru metoda Runge-Kutta clasică de ordinul 4 procedura de inițializare a tabelei Butcher este dată în sursa 10.3.

Problema din exemplul 10.5.2 se poate rezolva cu secvența de apel

```
>> [t2,w2]=Runge_Kutta(@edex1,[0,1],1,10,@RK4tab);
```

De remarcat că funcțiile RK4 și Runge\_Kutta funcționează atât pentru ecuații diferențiale scalare cât și pentru sisteme.

**Sursa MATLAB 10.3 Inițializare tabelă Butcher RK4**

```
function [a,b,c,s]=RK4tab
%RK4 - tabela Butcher pentru metoda RK4 clasica
s=4;
a=zeros(s,s-1);
a(2:s,1:s-1)=[1/2,0,0; 0, 1/2,0;0,0,1];
b=[1,2,2,1]'/6;
c=sum(a');
```

## 10.6. Descrierea globală a metodelor cu un pas

Descrierea metodelor se face cel mai bine în termeni de grile și funcții grilă.

O grilă pe intervalul  $[a, b]$  este o mulțime de puncte  $\{x_n\}_{n=0}^N$  astfel încât

$$a = x_0 < x_1 < x_2 < \dots < x_{N-1} < x_N = b, \quad (10.6.1)$$

cu lungimile grilei definite prin

$$h_n = x_{n+1} - x_n, \quad n = 0, 1, \dots, N-1. \quad (10.6.2)$$

Finețea grilei este măsurată prin

$$|h| = \max_{0 \leq n \leq N-1} h_n. \quad (10.6.3)$$

Vom utiliza litera  $h$  pentru a desemna colecția de lungimi  $h = \{h_n\}$ . Dacă  $h_1 = h_2 = \dots = h_N = (b-a)/N$ , grila se numește *uniformă*, iar în caz contrar *neuniformă*. O funcție cu valori vectoriale  $v = \{v_n\}$ ,  $v_n \in \mathbb{R}^d$ , definită pe grila (10.6.1) se numește *funcție grilă*. Astfel,  $v_n$  este valoarea funcției  $v$  în punctul  $x_n$  al grilei. Orice funcție  $v(x)$  definită pe  $[a, b]$  induce o funcție grilă prin restricție. Vom desemna multimea funcțiilor grilă pe  $[a, b]$  prin  $\Gamma_h[a, b]$  și pentru fiecare funcție grilă  $v = \{v_n\}$  definim norma sa prin

$$\|v\|_\infty = \max_{0 \leq n \leq N} \|v_n\|, \quad v \in \Gamma_h[a, b]. \quad (10.6.4)$$

O metodă cu un pas – de fapt orice metodă discretă – este o metodă care produce o funcție grilă  $u = \{u_n\}$  astfel încât  $u \approx y$ , unde  $y = \{y_n\}$  este funcția grilă indușă de soluția exactă a problemei Cauchy. Fie metoda

$$\begin{aligned} x_{n+1} &= x_n + h_n \\ u_{n+1} &= u_n + h_n \Phi(x_n, u_n; h_n) \end{aligned} \quad (10.6.5)$$

unde  $x_0 = a$ ,  $u_0 = y_0$ .

Pentru a clarifica analogia dintre (10.1.1) și (10.6.5) vom introduce operatorii  $R$  și  $R_h$  care acționează pe  $C^1[a, b]$  și respectiv pe  $\Gamma_h[a, b]$ . Aceștia sunt operatorii reziduali

$$(Rv)(x) := v'(x) - f(x, v(x)), \quad v \in C^1[a, b] \quad (10.6.6)$$

$$(R_h v)_n := \frac{1}{h_n} (v_{n+1} - v_n) - \Phi(x_n, v_n; h_n), \quad n = 0, 1, \dots, N-1, \quad (10.6.7)$$

unde  $v = \{v_n\} \in \Gamma_h[a, b]$ . (Funcția grilă  $\{(R_h v)_n\}$  nu este definită pentru  $n = N$ , dar putem lua arbitrar  $(R_h v)_N = (R_h v)_{N-1}$ ). Atunci problema Cauchy și analogul său discret (10.6.5) se pot scrie transparent ca

$$Ry = 0 \text{ pe } [a, b], \quad y(a) = y_0, \quad (10.6.8)$$

$$R_h u = 0 \text{ pe } [a, b], \quad u_0 = y_0. \quad (10.6.9)$$

De notat că operatorul rezidual (10.6.7) este strâns înrudit cu eroarea locală de trunchiere (10.3.3) când aplicăm operatorul într-un punct  $(x_n, y(x_n))$  pe traectoria soluției exacte. Atunci, într-adevăr, soluția de referință  $u(t)$  coincide cu soluția  $y(t)$  și

$$\begin{aligned} (R_h y)_n &= \frac{1}{h_n} [y(x_{n+1}) - y(x_n)] - \Phi(x_n, y(x_n); h_n) = \\ &= -T(x_n, y(x_n); h_n). \end{aligned} \quad (10.6.10)$$

### 10.6.1. Stabilitatea

Stabilitatea este o proprietate numai a schemei numerice (10.6.5) și nu are nimic de-a face apriori cu puterea de aproximare. Ea caracterizează robustețea schemei în raport cu perturbații mici. Totuși stabilitatea combinată cu consistența conduce la convergența soluției numerice către soluția adevărată. Definim stabilitatea în termeni de operatori reziduali discreți  $R_h$  din (10.6.7). Ca de obicei, presupunem că  $\Phi(x, y; h)$  este definită pe  $[a, b] \times \mathbb{R}^d \times [0, h_0]$ , unde  $h_0 > 0$  este un număr pozitiv adecvat.

**Definiția 10.6.1.** Metoda (10.6.5) se numește stabilă pe  $[a, b]$  dacă există o constantă  $K > 0$  care nu depinde de  $h$  astfel încât pentru o grilă arbitrară  $h$  pe  $[a, b]$  și pentru două funcții grilă arbitrară  $v, w \in \Gamma_h[a, b]$  are loc

$$\|v - w\|_\infty \leq K (\|v_0 - w_0\|_\infty + \|R_h v - R_h w\|_\infty), \quad v, w \in \Gamma_h[a, b] \quad (10.6.11)$$

pentru orice  $h$  cu  $|h|$  suficient de mică. În (10.6.11) norma este definită prin (10.6.4).

Vom numi în continuare (10.6.11) *inegalitatea de stabilitate*. Motivația ei este următoarea. Să presupunem că avem două funcții grilă  $u, w$  ce satisfac

$$R_h u = 0, \quad u_0 = y_0 \quad (10.6.12)$$

$$R_h w = \varepsilon, \quad w_0 = y_0 + \eta_0, \quad (10.6.13)$$

unde  $\varepsilon = \{\varepsilon_n\} \in \Gamma_h[a, b]$  este o funcție grilă cu  $\|\varepsilon_n\|$  mic și  $\|\eta_0\|$  este de asemenea mic. Putem interpreta  $u \in \Gamma_h[a, b]$  ca fiind rezultatul aplicării schemei numerice (10.6.5) cu precizie infinită, în timp ce  $w \in \Gamma_h[a, b]$  ar putea fi soluția lui (10.6.5) în aritmetică în virgulă flotantă. Atunci, dacă stabilitatea are loc, avem

$$\|u - w\|_\infty \leq K(\|\eta_0\|_\infty + \|\varepsilon\|_\infty), \quad (10.6.14)$$

adică, schimbarea locală în  $u$  este de același ordin de mărime ca și eroarea reziduală locală  $\{\varepsilon_n\}$  și eroarea inițială  $\eta_0$ . Trebuie apreciat, totuși, că prima ecuație în (10.6.13) spune

$$w_{n+1} - w_n - h_n \Phi(x_n, w_n, h_n) = h_n \varepsilon_n,$$

însemnând că erorile de rotunjire trebuie să tindă către zero atunci când  $|h| \rightarrow \infty$ .

Interesant, pentru stabilitate este necesară doar o condiție Lipschitz asupra lui  $\Phi$ .

**Teorema 10.6.2.** Dacă  $\Phi(x, y; h)$  satisface o condiție Lipschitz în raport cu variabila  $y$ ,

$$\|\Phi(x, y; h) - \Phi(x, y^*; h)\| \leq M \|y - y^*\| \text{ pe } [a, b] \times \mathbb{R}^d \times [0, h_0], \quad (10.6.15)$$

atunci metoda (10.6.5) este stabilă.

Vom pregăti demonstrația prin următoarea lemă utilă.

**Lema 10.6.3.** Fie  $\{e_n\}$  o secvență de numere,  $e_n \in \mathbb{R}$ , ce satisface

$$e_{n+1} \leq a_n e_n + b_n, \quad n = 0, 1, \dots, N-1 \quad (10.6.16)$$

unde  $a_n > 0$  și  $b_n \in \mathbb{R}$ . Atunci

$$e_n \leq E_n, \quad E_n = \left( \prod_{k=0}^{n-1} a_k \right) e_0 + \sum_{k=0}^{n-1} \left( \prod_{\ell=k+1}^{n-1} a_\ell \right) b_k, \quad n = 0, 1, \dots, N \quad (10.6.17)$$

Adoptăm aici convenția uzuală că un produs vid are valoarea 1 și o sumă vidă are valoarea 0.

*Demonstrația lemei 10.6.3.* Se verifică că

$$E_{n+1} = a_n E_n + b_n, \quad n = 0, 1, \dots, N-1, \quad E_0 = e_0.$$

Scăzând aceasta din (10.6.16) se obține

$$e_{n+1} - E_{n+1} \leq a_n(e_n - E_n), \quad n = 0, 1, \dots, N-1.$$

Acum,  $e_0 - E_0 = 0$ , aşa că  $e_1 - E_1 \leq 0$ , căci  $a_0 > 0$ . Prin inducție, mai general,  $e_n - E_n \leq 0$ , deoarece  $a_{n-1} > 0$ .  $\square$

*Demonstrația teoremei 10.6.2.* Fie  $h = \{h_n\}$  o grilă arbitrară pe  $[a, b]$  și  $v, w \in \Gamma_h[a, b]$  două funcții grilă cu valori vectoriale arbitrare. Din definiția lui  $R_h$  putem scrie

$$v_{n+1} = v_n + h_n \Phi(x_n, v_n; h_n) + h_n(R_h v)_n, \quad n = 0, 1, \dots, N-1$$

și similar pentru  $w_{n+1}$ . Scăzându-le obținem

$$\begin{aligned} v_{n+1} - w_{n+1} &= v_n - w_n + h_n[\Phi(x_n, v_n; h_n) - \Phi(x_n, w_n; h_n)] + \\ &\quad + h_n[(R_h v)_n - (R_h w)_n], \quad n = 0, 1, \dots, N-1. \end{aligned} \quad (10.6.18)$$

Definim acum

$$e_n = \|v_n - w_n\|, \quad d_n = \|(R_h v)_n - (R_h w)_n\|, \quad \delta = \|d_n\|_\infty. \quad (10.6.19)$$

Utilizând inegalitatea triunghiului în (10.6.18) și condiția Lipschitz (10.6.19) pentru  $\Phi$  obținem

$$e_{n+1} \leq (1 + h_n M) e_n + h_n \delta, \quad n = 0, 1, \dots, N-1 \quad (10.6.20)$$

Aceasta este inegalitatea (10.6.16) cu  $a_n = 1 + h_n M$ ,  $b_n = h_n \delta$ . Deoarece  $k = 0, 1, \dots, n-1$ ,  $n \leq N$  avem

$$\begin{aligned} \prod_{\ell=k+1}^{n-1} a_\ell &\leq \prod_{\ell=0}^{n-1} a_\ell = \prod_{\ell=0}^{N-1} (1 + h_\ell M) \leq \prod_{\ell=0}^{N-1} e^{h_\ell M} \\ &= e^{(h_0 + h_1 + \dots + h_{N-1})M} = e^{(b-a)M}, \end{aligned}$$

unde în a doua inegalitate a fost utilizată inegalitatea clasică  $1 + x \leq e^x$ , din lema 10.6.3 se obține că

$$\begin{aligned} e_n &\leq e^{(b-a)M} e_0 + e^{(b-a)M} \sum_{k=0}^{n-1} h_k \delta \leq \\ &\leq e^{(b-a)M} (e_0 + (b-a)\delta), \quad n = 0, 1, \dots, N-1. \end{aligned}$$

De aceea

$$\|e\|_\infty = \|v - w\|_\infty \leq e^{(b-a)M} (\|v_0 - w_0\| + (b-a)\|R_h v - R_h w\|_\infty),$$

căre este (10.6.11) cu  $K = e^{(b-a)M} \max\{1, b-a\}$ .  $\square$

Am demonstrat de fapt stabilitatea pentru orice  $|h| \leq h_0$ , nu numai pentru  $h$  suficient de mic.

Toate metodele cu un pas utilizate în practică satisfac o condiție Lipschitz dacă  $f$  satisfac o astfel de condiție și constanta  $M$  pentru  $\Phi$  poate fi aproimată cu ajutorul constantei  $L$  pentru  $f$ . Este evident pentru metoda lui Euler și nu este dificil de demonstrat pentru celelalte. Este util de observat că  $\Phi$  nu este nevoie să fie continuă în  $x$ ; continuitatea pe porțiuni fiind suficientă atât timp cât (10.6.15) are loc pentru orice  $x \in [a, b]$ , luând limitele laterale în punctele de discontinuitate.

Vom folosi următoarea aplicație a lemei 10.6.3, relativă la o funcție grilă  $v \in \Gamma_h[a, b]$  ce satisfac

$$v_{n+1} = v_n + h_n(A_n v_n + b_n), \quad n = 0, 1, \dots, N-1, \quad (10.6.21)$$

unde  $A_n \in \mathbb{R}^{d \times d}$ ,  $b_n \in \mathbb{R}^d$ , și  $h_n$  este o grila arbitrară pe  $[a, b]$ .

**Lema 10.6.4.** Presupunem că în (10.6.21)

$$\|A_n\| \leq M, \quad \|b_n\| \leq \delta, \quad n = 0, 1, \dots, N-1, \quad (10.6.22)$$

unde constantele  $M$  și  $\delta$  nu depind de  $h$ . Atunci, există o constantă  $K > 0$ , independentă de  $h$ , dar depinzând de  $\|v_0\|$ , astfel încât

$$\|v\|_\infty \leq K. \quad (10.6.23)$$

*Demonstrație.* Lemma rezultă observând că

$$\|v_{n+1}\| \leq (1 + h_n M) \|v_n\| + h_n \delta, \quad n = 0, 1, \dots, N-1,$$

care este chiar inegalitatea (10.6.19) din demonstrația teoremei 10.6.2, deci

$$\|v_n\| \leq e^{(b-a)M} [\|v_0\| + (b-a)\delta]. \quad (10.6.24)$$

□

## 10.6.2. Convergență

Stabilitatea este un concept puternic. Ea implică aproape imediat convergență și este un instrument de deducere a estimării erorii globale. Vom începe prin a defini precis ce înseamnă convergență.

**Definiția 10.6.5.** Fie  $a = x_0 < x_1 < x_2 < \dots < x_N = b$  o grilă pe  $[a, b]$  cu lungimea grilei  $|h| = \max_{1 \leq n \leq N} (x_n - x_{n-1})$ . Fie  $u = \{u_n\}$  o funcție grilă definită aplicând metoda (10.6.5) pe  $[a, b]$  și  $y = \{y_n\}$  funcția grilă indușă de soluția exactă a problemei Cauchy. Metoda (10.6.5) se numește convergentă pe  $[a, b]$  dacă are loc

$$\|u - y\|_\infty \rightarrow 0, \text{ când } |h| \rightarrow 0 \quad (10.6.25)$$

**Teorema 10.6.6.** Dacă metoda (10.6.5) este consistentă și stabilă pe  $[a, b]$ , atunci ea converge. Mai mult, dacă  $\Phi$  are ordinul  $p$ , atunci

$$\|u - y\|_\infty = O(|h|^p) \text{ când } |h| \rightarrow 0. \quad (10.6.26)$$

*Demonstrație.* Din inegalitatea de stabilitate (10.6.11) aplicată funcțiilor grilă  $v = u$  și  $w = y$  din definiția 10.6.5 avem pentru  $|h|$  suficient de mic

$$\|u - y\|_\infty \leq K(\|u_0 - y(x_0)\| + \|R_h u - R_h y\|_\infty) = K\|R_h y\| \quad (10.6.27)$$

deoarece  $u_0 = y(x_0)$  și  $R_h u = 0$  conform (10.6.5). Dar, conform lui (10.6.10)

$$\|R_h y\|_\infty = \|T(\cdot, y; h)\|_\infty, \quad (10.6.28)$$

unde  $T$  este eroare de trunchiere a metodei  $\Phi$ . Din definiția consistenței

$$\|T(\cdot, y; h)\|_\infty \rightarrow 0, \text{ când } |h| \rightarrow 0,$$

cea ce demonstrează prima parte a teoremei. Partea a doua rezultă imediat din (10.6.27) și (10.6.28), deoarece ordinul  $p$ , prin definiție, înseamnă că

$$\|T(\cdot, y; h)\|_\infty = O(|h|^p), \text{ când } |h| \rightarrow 0. \quad (10.6.29)$$

□

### 10.6.3. Asimptotica erorii globale

Deoarece funcția eroare principală descrie contribuția termenului principal al erorii locale de trunchiere este de interes să identificăm termenul dominant în eroarea globală  $u_n - y(x_n)$ . Pentru a simplifica lucrările, vom presupune că avem o grilă de lungime constantă  $h$ , deși nu este dificil să lucrăm cu o grilă de lungime variabilă de forma  $h_n = \vartheta(x_n)h$ , unde  $\vartheta(x)$  este o funcție continuă pe porțiuni și  $0 < \vartheta(x) < \theta$  pentru  $a \leq x \leq b$ . Astfel, considerăm că metoda cu un pas având forma

$$\begin{aligned} x_{n+1} &= x_n + h \\ u_{n+1} &= u_n + h\Phi(x_n, u_n; h); \quad n = 0, 1, \dots, N-1 \\ x_0 &= a, \quad u_0 = y_0, \end{aligned} \quad (10.6.30)$$

definește o funcție grilă  $u = \{u_n\}$  pe o grilă uniformă pe  $[a, b]$ . Suntem interesați în comportarea asimptotică a lui  $u_n - y(x_n)$  când  $h \rightarrow 0$ , unde  $y(x)$  este soluția exactă a problemei Cauchy

$$\begin{cases} \frac{dy}{dx} = f(x, y) & x \in [a, b] \\ y(a) = y_0 \end{cases} \quad (10.6.31)$$

**Teorema 10.6.7.** Presupunem că

- (1)  $\Phi(x, y, h) \in C^2([a, b] \times \mathbb{R}^d \times [0, h_0])$ ;
- (2)  $\Phi$  este o metodă de ordin  $p \geq 1$  ce admite o funcție de eroare principală  $\tau(x, y) \in C([a, b] \times \mathbb{R}^d)$ ;
- (3)  $e(x)$  este soluția problemei Cauchy liniare

$$\begin{cases} \frac{de}{dx} = f_y(x, y(x))e + \tau(x, y(x)), & a \leq x \leq b \\ e(a) = 0 \end{cases} \quad (10.6.32)$$

Atunci, pentru  $n = \overline{0, N}$ ,

$$u_n - y(x_n) = e(x_n)h^p + O(h^{p+1}), \quad \text{când } h \rightarrow 0. \quad (10.6.33)$$

Înainte de a demonstra teorema, facem următoarele observații:

1. Semnificația precisă a lui (10.6.33) este

$$\|u - y - h^p e\|_\infty = O(h^{p+1}), \text{ când } h \rightarrow 0,$$

unde  $u, y, e$  sunt funcțiile grilă  $u = \{u_n\}$ ,  $y = \{y(x_n)\}$  și  $e = \{e(x_n)\}$ .

2. Datorită consistenței  $\Phi(x, y; 0) = f(x, y)$ , condiția (1) din enunț implică  $f \in C^2([a, b] \times \mathbb{R}^d)$ , ceea ce este mai mult decât suficient pentru a garanta existența și unicitatea soluției  $e(x)$  a lui (10.6.32) pe întreg intervalul  $[a, b]$ .
3. Faptul că anumite componente, dar nu toate ale lui  $\tau(x, y)$  ar putea să fie identice nule nu implică faptul că  $e(x)$  se anulează de asemenea, deoarece (10.6.32) este un sistem cuplat de ecuații diferențiale.

*Demonstrația teoremei 10.6.7.* Vom începe cu un calcul ajutător, estimarea lui

$$\Phi(x_n, u_n; h) - \Phi(x_n, y(x_n); h). \quad (10.6.34)$$

Conform teoremei lui Taylor (pentru funcții de mai multe variabile), aplicată celei de-a  $i$ -a componente a lui (10.6.34), avem

$$\begin{aligned} \Phi^i(x_n, u_n; h) - \Phi^i(x_n, y(x_n); h) &= \sum_{j=1}^d \Phi_{y^j}^i(x_n, y(x_n); h) [u_n^j - y^j(x_n)] \\ &\quad + \frac{1}{2} \sum_{j,k=1}^d \Phi_{y^j y^k}^i(x_n, \bar{u}_n; h) [u_n^j - y^j(x_n)] [u_n^k - y^k(x_n)], \end{aligned} \quad (10.6.35)$$

unde  $\bar{u}_n$  este pe segmentul ce unește  $u_n$  și  $y(x_n)$ . Utilizând teorema lui Taylor încă o dată, în variabila  $h$ , putem scrie

$$\Phi_{y^j}^i(x_n, y(x_n); h) = \Phi_{y^j}^i(x_n, y(x_n); 0) + h \Phi_{y^j h}^i(x_n, y(x_n); \bar{h}),$$

unde  $0 < \bar{h} < h$ . Deoarece, conform consistenței,  $\Phi(x, y; 0) \equiv f(x, y)$  pe  $[a, b] \times \mathbb{R}^d$ , avem

$$\Phi_{y^j}^i(x, y; 0) = f_{y^j}^i(x, y), \quad x \in [a, b], \quad y \in \mathbb{R}^d,$$

și condiția (1) din ipoteză ne permite să scriem

$$\Phi_{y^j}^i(x_n, y(x_n); h) = f_{y^j}^i(x_n, y(x_n)) + O(h), \quad h \rightarrow 0. \quad (10.6.36)$$

Observând acum că  $u_n - y(x_n) = O(h^p)$ , în virtutea teoremei 10.6.6 și utilizând (10.6.36) în (10.6.35), obținem conform ipotezei (1),

$$\begin{aligned} \Phi^i(x_n, u_n; h) - \Phi^i(x_n, y(x_n); h) &= \sum_{j=1}^d f_{y^j}^i(x_n, y(x_n)) [u_n^j - y^j(x_n)] + \\ &\quad O(h^{p+1}) + O(h^{2p}). \end{aligned}$$

Dar  $O(h^{2p})$  este de ordinul  $O(h^{p+1})$ , căci  $p \geq 1$ . Astfel, în notație vectorială,

$$\Phi(x_n, u_n; h) - \Phi(x_n, y(x_n); h) = f_y(x_n, y(x_n)) [u_n - y(x_n)] + O(h^{p+1}). \quad (10.6.37)$$

Acum, pentru a evidenția termenul dominant în eroarea globală, definim funcția grilă  $r = \{r_n\}$  prin

$$r = h^{-p}(u - y). \quad (10.6.38)$$

Atunci

$$\begin{aligned} \frac{1}{h}(r_{n+1} - r_n) &= \frac{1}{h} [h^{-p}(u_{n+1} - y(x_{n+1})) - h^{-p}(u_n - y(x_n))] = \\ &= h^p \left[ \frac{1}{h}(u_{n+1} - u_n) - \frac{1}{h}(y(x_{n+1}) - y(x_n)) \right] = \\ &= h^{-p} \{ \Phi(x_n, u_n; h) - [\Phi(x_n, y(x_n); h) - T(x_n, y(x_n); h)] \}, \end{aligned}$$

unde am utilizat (10.6.30) și relația (10.6.10) pentru eroarea de trunchiere  $T$ . Deci, exprimând  $T$  cu ajutorul funcției eroare principală  $\tau$ , obținem

$$\begin{aligned} \frac{1}{h}(r_{n+1} - r_n) &= h^{-p} [\Phi(x_n, u_n; h) - \Phi(x_n, y(x_n); h) + \tau(x_n, y(x_n))h^p \\ &\quad + O(h^{p+1})] \end{aligned}$$

Pentru primii doi termeni din paranteză utilizăm (10.6.37) și definiția lui  $r$  (10.6.38) pentru a obține

$$\begin{aligned} \frac{1}{h}(r_{n+1} - r_n) &= f_y(x_n, y(x_n))r_n + \tau(x_n, y(x_n)) + O(h), \quad n = \overline{0, N-1} \\ r_0 &= 0. \end{aligned} \quad (10.6.39)$$

Acum punând

$$g(x, y) := f_y(x, y(x))y + \tau(x, y(x)) \quad (10.6.40)$$

putem interpreta (10.6.39) scriind

$$\left( R_h^{Euler, g} r \right)_n = \varepsilon_n, \quad n = \overline{0, N-1}, \quad \varepsilon_n = O(h),$$

unde  $R_h^{Euler, g}$  este operatorul rezidual discret (10.6.7) corespunzător metodei lui Euler aplicată lui  $e' = g(x, e)$ ,  $e(a) = 0$ . Deoarece metoda lui Euler este stabilă pe  $[a, b]$  și  $g$  fiind liniară în  $y$  satisfacă o condiție Lipschitz uniformă, avem conform inegalității de stabilitate (10.6.11)

$$\|r - e\|_\infty = O(h),$$

și conform lui (10.6.38)

$$\|u - y - h^p e\|_\infty = O(h^{p+1}),$$

înălțându-se astfel că cum trebuia arătat.  $\square$

## 10.7. Monitorizarea erorilor și controlul pasului

Vom încerca să realizăm monitorizarea erorilor globale, cel puțin asymptotic, implementând rezultatul din teorema 10.6.7. Aceasta necesită evaluarea matricei jacobiene  $f_y(x, y)$  de-a lungul sau în apropierea traiectoriei soluției; acest lucru este natural, deoarece  $f_y(x, y)$  guvernează, într-o primă aproximare, efectul perturbațiilor prin ecuația diferențială variațională (10.6.32). În această ecuație tonul este dat de funcția de eroare principală, evaluată de-a lungul traiectoriei, așa că estimarea erorii locale de trunchiere (mai exact a funcției de eroare principală) este de asemenea necesară în această abordare. Pentru simplitate vom presupune că grila are o lungime constantă.

### 10.7.1. Estimarea erorii globale

Ideeia estimării este de a integra „ecuația variațională“ (10.6.32) împreună cu ecuația principală (10.6.31). Deoarece avem nevoie de  $e(x)$  în (10.6.31) numai în limita unei erori de  $O(h)$  (orice termen de eroare  $O(h)$  din  $e(x_n)$ , înmulțit cu  $h^p$ , fiind absorbit de termenul  $O(h^{p-1})$ ), putem utiliza în acest scop metoda lui Euler, care va furniza aproximarea dorită  $v_n \approx e(x_n)$ .

**Teorema 10.7.1.** Presupunem că

- (1)  $\Phi(x, y; h) \in C^2([a, b] \times \mathbb{R}^d \times [0, h_0])$ ;
- (2)  $\Phi$  este o metodă de ordin  $p \geq 1$  ce admite o funcție de eroare principală  $\tau(x, y) \in C^1([a, b] \times \mathbb{R}^d)$ ;

(3) este disponibilă o estimare  $r(x, y; h)$  pentru funcția de eroare principală ce satisface

$$r(x, y; h) = \tau(x, y) + O(h), \quad h \rightarrow 0, \quad (10.7.1)$$

uniform pe  $[a, b] \times \mathbb{R}^d$ ;

(4) odată cu funcția grilă  $u = \{u_n\}$  generăm funcția grilă  $v = \{v_n\}$  în modul următor

$$\begin{aligned} x_{n+1} &= x_n + h; \\ u_{n+1} &= u_n + h\Phi(x_n, u_n; h) \\ v_{n+1} &= v_n + h [f_y(x_n, v_n)v_n + r(x_n, u_n; h)] \\ x_0 &= a, \quad u_0 = y_0, \quad v_0 = 0. \end{aligned} \quad (10.7.2)$$

Atunci pentru orice  $n = \overline{0, N-1}$ ,

$$u_n - y(x_n) = v_n h^p + O(h^{p+1}), \quad \text{când } h \rightarrow 0. \quad (10.7.3)$$

*Demonstrație.* Demonstrația începe cu stabilirea următoarelor estimări

$$f_y(x_n, u_n) = f_y(x_n, y(x_n)) + O(h), \quad (10.7.4)$$

$$r(x_n, u_n; h) = \tau(x_n, y(x_n)) + O(h). \quad (10.7.5)$$

Din ipoteza (1) observăm pe baza consistenței  $f(x, y) = \Phi(x, y; 0)$  că  $f(x, y) \in C^2([a, b] \times \mathbb{R}^d)$ . Înținând cont de teorema 10.6.6, avem  $u_n = y(x_n) + O(h^p)$ , și deci

$$f_y(x_n, u_n) = f_y(x_n, y_n) + O(h^p),$$

relație ce implică (10.7.4), deoarece  $p \geq 1$ . În continuare, deoarece  $\tau(x, y) \in C^1([a, b] \times \mathbb{R}^d)$ , conform ipotezei (2) avem

$$\begin{aligned} \tau(x_n, u_n) &= \tau(x_n, y(x_n)) + \tau_y(x_n, \bar{u}_n)(u_n - y(x_n)) \\ &= \tau(x_n, y(x_n)) + O(h^p) \end{aligned}$$

și aplicând apoi ipoteza (3) obținem

$$r(x_n, u_n; h) = \tau(x_n, u_n) + O(h) = \tau(x_n, y(x_n)) + O(h^p) + O(h),$$

din care rezultă imediat (10.7.5).

Fie (a se compara cu (10.6.40))

$$g(x, y) = f_y(x, y(x))y + \tau(x, y(x)). \quad (10.7.6)$$

Ecuația pentru  $v_{n+1}$  în (10.7.2) are forma

$$v_{n+1} = v_n + h(A_n v_n + b_n),$$

unde  $A_n$  sunt matrice mărginite și  $b_n$  sunt vectori mărginiți. Conform lemei 10.6.6, avem mărginirea lui  $v_n$ ,

$$v_n = O(1), \quad h \rightarrow 0. \quad (10.7.7)$$

Înlocuind (10.7.4) și (10.7.5) în ecuația lui  $v_{n+1}$  și înținând cont de (10.7.7) obținem

$$\begin{aligned} v_{n+1} &= v_n + h [f_y(x_n, y(x_n))v_n + \tau(x_n, y(x_n)) + O(h)] \\ &= v_n + hg(x_n, v_n) + O(h^2). \end{aligned}$$

Astfel, cu notația utilizată în demonstrația teoremei 10.6.7

$$\left( R_h^{Euler,g} v \right)_n = O(h), \quad v_0 = 0.$$

Deoarece metoda lui Euler este stabilă,

$$v_n - e(x_n) = O(h),$$

unde  $e(x)$  este, ca mai înainte, soluția ecuației

$$\begin{aligned} e' &= g(x, e) \\ e(a) &= 0 \end{aligned}$$

Deci, conform lui (10.6.33)

$$u_n - y(x_n) = e(x_n)h^p + O(h^{p+1}).$$

□

## 10.7.2. Estimarea erorii de trunchiere

Pentru a aplica teorema 10.7.1 avem nevoie de estimări  $r(x, y; h)$  ale funcției de eroare principală  $\tau(x, y)$  care să aibă precizia  $O(h)$ . Vom descrie două dintre ele, în ordinea crescătoare a eficienței.

### Extrapolare Richardson la zero

Aceasta funcționează pentru orice metodă cu un pas  $\Phi$ , dar de obicei este considerată prea costisitoare. Dacă  $\Phi$  are ordinul  $p$ , procedura este următoarea

$$\begin{aligned} y_h &= y + h\Phi(x, y; h), \\ y_{h/2} &= y + \frac{1}{2}h\Phi\left(x, y; \frac{1}{2}h\right), \\ y_h^* &= y_{h/2} + \frac{1}{2}h\Phi\left(x + \frac{1}{2}h, y_{h/2}; \frac{1}{2}h\right), \\ r(x, y; h) &= \frac{1}{1 - 2^{-p}} \frac{1}{h^{p+1}} (y_h - y_h^*). \end{aligned} \tag{10.7.8}$$

De notat că  $y_h^*$  este rezultatul aplicării lui  $\Phi$  pentru doi pași consecutivi, fiecare de lungime  $h/2$ , pe cătă vreme  $y_h$  este rezultatul aplicării lui  $\Phi$  pentru un pas de lungime  $h$ .

Să verificăm acum că  $r(x, y; h)$  datează de grupul de formule (10.7.8) este o estimare acceptabilă. Pentru aceasta trebuie să presupunem că  $\tau(x, y) \in C^1([a, b] \times \mathbb{R}^d)$ . Conform lui (10.3.4) și (10.3.8), utilizând soluția de referință  $u(t)$  ce trece prin  $(x, y)$  avem

$$\Phi(x, y; h) = \frac{1}{h}[u(x+h) - u(x)] + \tau(x, y)h^p + O(h^{p+1}). \tag{10.7.9}$$

Mai mult,

$$\begin{aligned} \frac{1}{h}(y_h - y_h^*) &= \frac{1}{h}(y_h - y_{h/2}) + \Phi(x, y; h) - \frac{1}{2}h\Phi\left(x + \frac{1}{2}h, y_{h/2}; \frac{1}{2}h\right) \\ &= \Phi(x, y; h) - \frac{1}{2}\Phi\left(x, y; \frac{1}{2}h\right) - \frac{1}{2}h\Phi\left(x + \frac{1}{2}h, y_{h/2}; \frac{1}{2}h\right). \end{aligned}$$

Aplicând (10.7.9) fiecărui termen din membrul drept găsim

$$\begin{aligned} \frac{1}{h}(y_h - y_h^*) &= \frac{1}{h}[u(x+h) - u(x)] + \tau(x, y)h^p + O(h^{p+1}) \\ &\quad - \frac{1}{2} \frac{1}{h/2} \left[ u\left(x + \frac{1}{2}h\right) - u(x) \right] - \frac{1}{2}\tau(x, y)\left(\frac{1}{2}h^p\right) + O(h^{p+1}) \\ &\quad - \frac{1}{2} \frac{1}{h/2} \left[ u(x+h) - u\left(x + \frac{1}{2}h\right) \right] - \frac{1}{2}\tau\left(x + \frac{1}{2}h, y + O(h)\right)\left(\frac{1}{2}h^p\right) \\ &\quad + O(h^{p+1}) = \tau(x, y)(1 - 2^{-p})h^p + O(h^{p+1}). \end{aligned}$$

În consecință

$$\frac{1}{1 - 2^{-p}} \frac{1}{h}(y_h - y_h^*) = \tau(x, y)h^p + O(h^{p+1}), \quad (10.7.10)$$

așa cum s-a dorit.

Scăzând (10.7.10) din (10.7.9), rezultă incidental că

$$\Phi^*(x, y; h) := \Phi(x, y; h) - \frac{1}{1 - 2^{-p}} \frac{1}{h}(y_h - y_h^*) \quad (10.7.11)$$

definește o metodă cu un pas de ordin  $p + 1$ .

Procedura (10.7.8) este costisitoare. Pentru un proces Runge-Kutta de ordinul 4 sunt necesare în total 11 evaluări ale lui  $f$  pe pas, aproape de trei ori mai mult decât pentru un pas Runge-Kutta. De aceea, extrapolarea Richardson este utilizată numai după fiecare doi pași ai lui  $\Phi$ , adică se continuă în conformitate cu formulele

$$\begin{aligned} y_h &= y + h\Phi(x, y; h), \\ y_{2h}^* &= y_h + h\Phi(x + h, y_h; h) \\ y_{2h} &= y + 2h\Phi(x, y; 2h). \end{aligned} \quad (10.7.12)$$

Atunci (10.7.10) ne dă

$$\frac{1}{2(2^p - 1)} \frac{1}{h^{p+1}} (y_{2h} - y_{2h}^*) = \tau(x, y) + O(h), \quad (10.7.13)$$

așa că expresia din membrul drept este un estimator acceptabil al lui  $r(x, y; h)$ . Dacă cei doi pași din (10.7.12) conduc la o precizie acceptabilă (a se vedea subsecțiunea 10.7.3), atunci pentru un proces Runge-Kutta de ordinul 4 procedura necesită numai trei evaluări adiționale ale lui  $f$ , deoarece  $y_h$  și  $y_{2h}^*$  trebuie calculat oricum. Vom vedea că există scheme mai eficiente.

### Metode scufundate (imbricate)

Ideea de bază a acestei abordări este următoarea: se consideră o metodă  $\Phi$  de ordinul  $p$  și o metodă  $\Phi^*$  de ordinul  $p^* = p + 1$  și se definește

$$r(x, y; h) = \frac{1}{h^p} [\Phi(x, y; h) - \Phi^*(x, y; h)]. \quad (10.7.14)$$

Acesta este un estimator acceptabil, așa cum rezultă scăzând relațiile

$$\begin{aligned} \Phi(x, y; h) - \frac{1}{h} [u(x+h) - u(x)] &= \tau(x, y)h^p + O(h^{p+1}) \\ \Phi^*(x, y; h) - \frac{1}{h} [u(x+h) - u(x)] &= O(h^{p+1}) \end{aligned}$$

și împărțind rezultatul cu  $h^p$ .

Cheia problemei este de a face această procedură eficientă. Urmând o idee a lui Fehlberg, putem încerca să facem aceasta inclusiv un proces Runge-Kutta de ordinul  $p$  în altul de ordin  $p+1$ . Mai concret, fie  $\Phi$  o metodă Runge-Kutta explicită în  $r$  stadii

$$\begin{aligned} K_1(x, y) &= f(x, y) \\ K_s(x, y; h) &= f\left(x + \mu_s h; y + h \sum_{j=1}^{s-1} \lambda_{sj} K_j\right), \quad s = 2, 3, \dots, r \\ \Phi(x, y; h) &= \sum_{s=1}^r \alpha_s K_s \end{aligned}$$

Atunci pentru  $\Phi^*$  alegem un proces similar în  $r^*$ -stadii, cu  $r^* > r$ , astfel încât

$$\mu_s^* = \mu_s, \quad \lambda_{sj}^* = \lambda_{sj}, \text{ pentru } s = 2, 3, \dots, r.$$

Estimarea (10.7.14) costă atunci din  $r^* - r$  evaluări suplimentare ale lui  $f$ . Dacă  $r^* = r + 1$  putem încă să mai facem economii de evaluări suplimentare, selectând (dacă este posibil)

$$\mu_r^* = 1, \quad \lambda_{r^*j} = \alpha_j \text{ pentru } j = \overline{1, r^* - 1} \quad (r^* = r + 1) \quad (10.7.15)$$

Atunci, într-adevăr,  $K_r^*$  va fi identic cu  $K_1$  pentru pasul următor.

Perechi de astfel de formule Runge-Kutta imbricate  $(p, p+1)$  au fost dezvoltate la sfârșitul anilor '60 de E. Fehlberg[20, 21]. Este un grad considerabil de libertate în alegerea acestor parametri. Alegerile lui Fehlberg au fost ghidate de încercarea de a reduce mărimea coeficienților tututor derivatelor parțiale care intervin în funcția de eroare principală  $\tau(x, y)$  a lui  $\Phi$ . El a reușit să obțină pentru parametrii  $p, r, r^*$  valorile date în tabela 10.2

$p$	3	4	5	6	7	8
$r$	4	5	6	8	11	15
$r^*$	5	6	8	10	13	17

Tabela 10.2: Formule Runge-Kutta imbricate

Pentru procesul de ordinul 3 (și numai pentru acesta) se pot alege parametrii astfel ca să aibă loc și (10.7.15).

### 10.7.3. Controlul pasului

Orice estimare  $r(x, y; h)$  a funcției de eroare principală  $\tau(x, y)$  implică o estimare

$$h^p r(x, y; h) = T(x, y; h) + O(h^{p+1}) \quad (10.7.16)$$

a erorii de trunchiere, care poate fi utilizată pentru a monitoriza eroarea locală de trunchiere în timpul procesului de integrare. Totuși, trebuie avut în vedere faptul că eroarea locală de trunchiere este chiar diferită de eroarea globală, eroare pe care vrem de fapt să o controlăm. Pentru a obține o mai bună cunoaștere a relației dintre aceste două erori reamintim teorema următoare, care cuantifică continuitatea soluției problemei Cauchy în raport cu valorile inițiale.

**Teorema 10.7.2.** Fie  $f(x, y)$  continuă în  $x \in [a, b]$  și care satisface o condiție Lipschitz cu constanta  $L$ , uniform pe  $[a, b] \times \mathbb{R}$ , adică

$$\|f(x, y) - f(x, y^*)\| \leq L \|y - y^*\|.$$

Atunci problema Cauchy

$$\begin{aligned} \frac{dy}{dx} &= f(x, y), \quad x \in [a, b], \\ y(c) &= y_c \end{aligned} \tag{10.7.17}$$

are o soluție unică pentru orice  $c \in [a, b]$  și orice  $y_c \in \mathbb{R}^d$ . Fie  $y(x, s)$  și  $y(x; s^*)$  soluțiile lui (10.7.17) ce corespund lui  $y_c = s$  și respectiv  $y_c = s^*$ . Atunci, pentru orice normă vectorială  $\|\cdot\|$ ,

$$\|y(x; s) - y(x; s^*)\| \leq e^{L|x-c|} \|s - s^*\|. \tag{10.7.18}$$

Rezolvarea numerică a problemei (10.6.31) printr-o metodă cu un pas (nu neapărat constant) înseamnă în realitate că se urmărește o secvență de „piste ale soluției“ (expresia este din [24]) prin care în fiecare punct al grilei se sare de la o pistă la următoarea cu o cantitate egală cu eroarea de trunchiere în  $x_n$  (vezi figura 10.3). Aceasta rezultă din definiția erorii de trunchiere, soluția de referință fiind una din pistele soluției. Mai concret, a  $n$ -a pistă,  $n = \overline{0, N}$ , este dată de soluția problemei Cauchy

$$\begin{aligned} \frac{dv_n}{dx} &= f(x, v_n), \quad x \in [x_n, b], \\ v_n(x_n) &= u_n, \end{aligned} \tag{10.7.19}$$

și

$$u_{n+1} = v(x_{n+1}) + h_n T(x_n, u_n; h_n), \quad n = \overline{0, N-1}. \tag{10.7.20}$$

Deoarece conform lui (10.7.19) avem  $u_{n+1} = v_{n+1}(x_{n+1})$ , putem aplica teorema 10.7.2 soluțiilor  $v_{n+1}$  și  $v_n$ , luând  $c = x_{n+1}$ ,  $s = u_{n+1}$ ,  $s^* = u_{n+1} - h_n T(x_n, u_n; h_n)$  (conform lui (10.7.20)) și astfel obținem

$$\|v_{n+1}(x) - v_n(x)\| \leq h_n e^{L|x-x_n|} \|T(x_n, u_n; h_n)\|, \quad n = \overline{0, N-1}. \tag{10.7.21}$$

Acum

$$\sum_{n=0}^{N-1} [v_{n+1}(x) - v_n(x)] = v_N(x) - v_0(x) = v_N(x) - y(x), \tag{10.7.22}$$

și deoarece  $v_N(x_N) = u_N$ , luând  $x = x_N$ , obținem din (10.7.21) și (10.7.22) că

$$\begin{aligned} \|u_N - y(x_N)\| &\leq \sum_{n=0}^{N-1} \|v_{n+1}(x_N) - v_n(x_N)\| \\ &\leq \sum_{n=0}^{N-1} h_n e^{L|x_N-x_{n+1}|} \|T(x_n, u_n; h_n)\|. \end{aligned}$$

De aceea, dacă ne asigurăm că

$$\|T(x_n, u_n; h_n)\| \leq \varepsilon_T, \quad n = \overline{0, N-1}, \tag{10.7.23}$$

atunci

$$\|u_N - y(x_N)\| \leq \varepsilon_T \sum_{n=0}^{N-1} (x_{n+1} - x_n) e^{L|x_N-x_{n+1}|}.$$

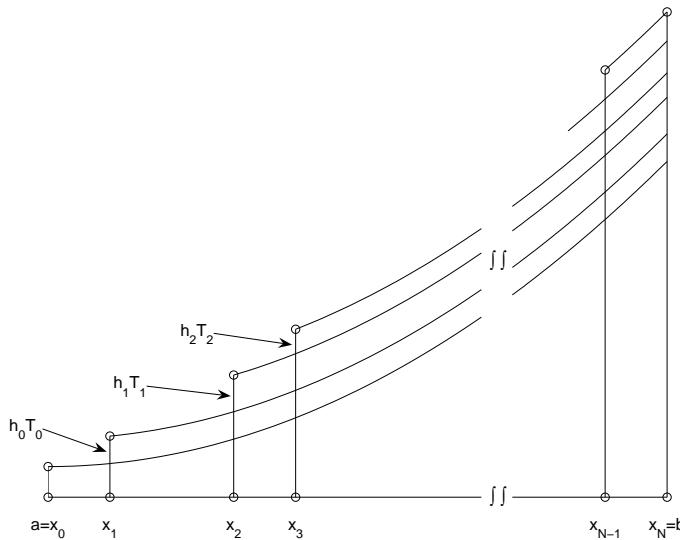


Figura 10.3: Acumularea erorilor într-o metodă cu un pas

Interpretând suma din dreapta ca o sumă Riemann pentru o integrală definită, obținem în final aproximarea

$$\|u_N - y(x_N)\| \leq \varepsilon_T \int_a^b e^{L(b-x)} dx = \frac{\varepsilon_T}{L} (e^{L(b-a)} - 1).$$

Astfel, cunoașterea unei estimări pentru  $L$  ne va permite să găsim un  $\varepsilon_T$

$$\varepsilon_T = \frac{L}{e^{L(b-a)} - 1} \varepsilon, \quad (10.7.24)$$

care să ne garanteze o eroare  $\|u_N - y(x_N)\| \leq \varepsilon$ . Ceea ce are loc pentru întreaga grilă pe  $[a, b]$  are loc, desigur, pentru orice grilă pe subintervalul  $[a, x]$ ,  $a \leq x \leq b$ . Astfel, în principiu, dându-se precizia dorită  $\varepsilon$  pentru soluția  $y(x)$ , putem determina un „nivel de toleranță”  $\varepsilon_T$  (din (10.7.24)) și putem asigura precizia dorită păstrând eroarea locală de trunchiere sub limita  $\varepsilon_T$  (a se compara cu (10.7.23)). De notat că dacă  $L \rightarrow 0$ , avem  $\varepsilon_T \rightarrow \varepsilon/(b-a)$ . Această valoare limită pentru  $\varepsilon_T$  ar fi adecvată pentru o problemă de cuadratură, dar nu pentru o ecuație diferențială veritabilă, unde  $\varepsilon_T$ , în general, trebuie ales mult mai mic decât eroarea finală  $\varepsilon$ .

Considerații ca acestea motivează următorul mecanism de control al pasului: fiecare pas de integrare (de la  $x_n$  la  $x_{n+1} = x_n + h_n$ ) constă din următoarele părți:

1. Estimăm  $h_n$ .
2. Se calculează  $u_{n+1} = u_n + h_n \Phi(x_n, u_n; h_n)$  și  $r(x_n, u_n; h_n)$ .
3. Se testează dacă  $h_n^p \|r(x_n, u_n; h_n)\| \leq \varepsilon_T$ . Dacă testul este satisfăcut se trece la pasul următor. Dacă nu, repetăm pasul cu un  $h_n$  mai mic, până când testul este satisfăcut.

Pentru a estima  $h_n$ , presupunem întâi că  $n \geq 1$ , astfel ca estimatorul din pasul precedent,  $r(x_{n-1}, u_{n-1}; h_{n-1})$  (sau cel puțin norma sa) să fie disponibil. Atunci, neglijând termenul  $O(h)$ ,

$$\|\tau(x_{n-1}, u_{n-1})\| \approx \|r(x_{n-1}, u_{n-1}; h_{n-1})\|,$$

și deoarece  $\tau(x_n, u_n) \approx \tau(x_{n-1}, u_{n-1})$ , în plus.

$$\|\tau(x_n, u_n)\| \approx \|r(x_{n-1}, u_{n-1}; h_{n-1})\|.$$

Ceea ce dorim este

$$\|\tau(x_n, u_n)\| h_n^p \approx \theta \varepsilon_T,$$

unde  $\theta$  este un factor de siguranță (să zicem  $\theta = 0.8$ ). Eliminând  $\tau(x_n, u_n)$  găsim

$$h_n \approx \left\{ \frac{\theta \varepsilon_T}{\|r(x_{n-1}, u_{n-1}; h_{n-1})\|} \right\}^{1/p}.$$

De notat că din pasul precedent avem

$$h_{n-1}^p \|r(x_{n-1}, u_{n-1}; h_{n-1})\| \leq \varepsilon_T,$$

așa că

$$h_n \geq \theta^{1/p} h_{n-1}$$

și tendința este de creștere a pasului.

Dacă  $n = 0$ , procedăm la fel, alegând o valoare inițială  $h_0^{(0)}$  a lui  $h_0$  și calculăm  $r(x_0, y_0; h_0^{(0)})$  pentru a obține

$$h_0^{(1)} = \left\{ \frac{\theta \varepsilon_T}{r(x_0, y_0; h_0^{(0)})} \right\}^{1/p}.$$

Procesul se poate repeta odată sau de două ori pentru a obține estimarea finală a lui  $h_0$  și  $r(x_0, y_0; h_0^{(0)})$ .

Pentru o descriere sintetică a metodelor Runge-Kutta cu pas variabil tabela Butcher se completează cu o linie suplimentară care servește la calculul lui  $\Phi^*$  (și deci a lui  $r(x, y; h)$ ):

$\mu_1$	$\lambda_{11}$	$\lambda_{12}$	$\dots$	$\lambda_{1r}$	
$\mu_2$	$\lambda_{21}$	$\lambda_{22}$	$\dots$	$\lambda_{2r}$	
$\vdots$	$\vdots$	$\vdots$	$\dots$	$\vdots$	
$\mu_r$	$\lambda_{r1}$	$\lambda_{r2}$	$\dots$	$\lambda_{rr}$	
	$\alpha_1$	$\alpha_2$	$\dots$	$\alpha_r$	
	$\alpha_1^*$	$\alpha_2^*$		$\alpha_r^*$	$\alpha_{r+1}^*$

Ca exemplu, în tabela 10.3 dăm tabela Butcher pentru o metodă de ordinul 2-3. Pentru deducerea elementelor talelei a se consulta [69, paginile 451–452].

Tabela 10.4 este tabela Butcher pentru metoda Bogacki-Shampine [8]. Ea stă la baza rezolvitorului `ode23` din MATLAB.

Un alt exemplu important este DOPRI5 sau RK5(4)7FM, o pereche cu ordinele 4-5 și cu 7 stadii (tabela 10.5). Aceasta este o pereche foarte eficientă, ea stând la baza rezolvitorului `ode45` din MATLAB, dar și a altor rezolvitori importanți.

Vom da în continuare un exemplu de implementare pentru o metodă Runge-Kutta cu pas variabil. În spiritul lucrării [17] am implementat o funcție MATLAB mai generală, `oderk`, care utilizează tabela Butcher. Modul de tratare al erorilor și organizarea generală se inspiră din `ode23` și `ode45` din MATLAB, dar și din rutina `ode23tx` din [46].

$\mu_j$	$\lambda_{ij}$			
0	0			
$\frac{1}{4}$	$\frac{1}{4}$	0		
$\frac{27}{40}$	$-\frac{189}{800}$	$\frac{729}{800}$	0	
1	$\frac{214}{891}$	$\frac{1}{33}$	$\frac{650}{891}$	0
$\alpha_i$	$\frac{214}{891}$	$\frac{1}{33}$	$\frac{650}{891}$	0
$\alpha_i^*$	$\frac{533}{2106}$	0	$\frac{800}{1053}$	$-\frac{1}{78}$

Tabela 10.3: O pereche 2-3

$\mu_j$	$\lambda_{ij}$			
0	0			
$\frac{1}{2}$	$\frac{1}{2}$	0		
$\frac{3}{4}$	0	$\frac{3}{4}$	0	
1	$\frac{2}{9}$	$\frac{3}{9}$	$\frac{4}{9}$	0
$\alpha_i$	$\frac{2}{9}$	$\frac{3}{9}$	$\frac{4}{9}$	0
$\alpha_i^*$	$\frac{7}{24}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{8}$

Tabela 10.4: Tabela Butcher pentru metoda Bogacki-Shampine

$\mu_j$	$\lambda_{ij}$						
0	0						
$\frac{1}{5}$	$\frac{1}{5}$	0					
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$	0				
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$	0			
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$	0		
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$	0	
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$\alpha_i$	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$\alpha_i^*$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

Tabela 10.5: Perechea inclusă RK5(4)7FM (DOPRI5)

Primul argument al funcției `oderk` specifică funcția din membrul drept  $f(t, y)$ . El poate fi un handler de funcție, un sir de caractere sau o funcție inline. Al doilea argument este un vector `tspan`, cu două componente `t0` și `tfinal`. Ele dau capetele intervalului pe care se face rezolvarea (integrarea). Al treilea argument `y0` furnizează valorile de pornire  $y_0 = y(t_0)$ . Lungimea lui `y0` spune rutinei numărul de ecuații diferențiale din sistem. Al patrulea argument este un handler de funcție care inițializează tabela Butcher. Dacă lipsește, metoda implicită folosită este metoda Bogacki-Shampine (tabela 10.4, funcția `BS23`). Al cincilea argument `opts` conține opțiunile rezolvatorului. Ele se pot inițializa cu funcția folosită de MATLAB, `odeset`. În `oderk` se iau în considerare numai opțiunile `RelTol` (eroarea relativă, implicit `1e-3`), `AbsTol` (eroarea absolută, implicit `1e-6`), `OutputFcn` (funcția de ieșire, implicit `odeplot`) și `Stats` (specifică dacă se doresc statistici și are valoarea `on` sau `off`). Instrucțiunea

```
opts=odeset('RelTol', 1e-5, 'AbsTol', 1e-8, 'OutputFcn',...
    myodeplot)
```

inițializează eroarea relativă pe  $10^{-5}$ , cea absolută pe  $10^{-8}$  și declară `myodeplot` ca funcție de ieșire.

Ieșirea lui `oderk` poate fi numerică sau grafică. Fără nici un argument de ieșire se generează un grafic al tuturor componentelor soluției. Cu două argumente de ieșire, instrucțiunea

```
[tout,yout] = oderk(F,tspan,y0)
generează o tabelă a absciselor și ordonatelor soluției.
```

Să examinăm codul acestei funcții.

```
function [tout,yout] = oderk(F,tspan,y0,BT,opts,varargin)
%ODERK rezolva ecuatii diferențiale nonstiff
% ODERK utilizeaza doua metode continue date
% prin tabela Butcher
%
% ODERK(F,TSPAN,Y0) cu TSPAN = [T0 TFINAL] integreaza
% sistemul de ecuatii diferențiale y' = f(t,y) de la
% t = T0 la t = TFINAL. Conditia initiala este y(T0) = Y0.
% F este numele unui fisier M, sau o functie
% inline sau un sir de caractere ce defineste f(t,y).
% Aceasta functie trebuie sa aiba doua argumente, t si y
% si trebuie sa returneze un vector coloana al
% derivatelor, y'.
%
% Cu doua argumente de iesire, [T,Y] = ODERK(...)
% returneaza un vector coloana T si un tablou Y unde
% Y(:,k) este solutia in T(k).
%
% Fara argumente de iesire ODERK reprezinta grafic solutia.
%
% ODERK(F,TSPAN,Y0,RTOL) utilizeaza eroarea relativă RTOL,
% in locul celei implicite 1.e-3.
%
% ODERK(F,TSPAN,Y0,BT) utilizeaza tabela Butcher. Daca BT
% este vid, se utilizeaza BS23 (Bogacki-Shampine)
%
% ODERK(F,TSPAN,Y0,BT,OPTS) unde OPTS = ODESET('reltol',...
% RTOL,'abstol',ATOL,'outputfcn',@PLOTFUN) utilizeaza
```

```
% eroarea relativa RTOL in locul valorii implicate 1.e-3,
% eroarea absoluta ATOL in loc de 1.e-6 si apeleaza PLOTFUN
% in loc de ODEPLOT dupa fiecare pas terminat cu succes.
%
% Daca se apeleaza cu mai mult de 5 argumente de intrare,
% ODERK(F,TSPAN,Y0,BT,RTOL,P1,P2,...), argumentele
% suplimentare se transmit lui F, F(T,Y,P1,P2,...).
%
% Optiunea Stats pe on furnizeaza statistici
%
% Exemplu
%     tspan = [0 2*pi];
%     y0 = [1 0]';
%     F = '[0 1; -1 0]*y';
%     oderk(F,tspan,y0);
```

Urmează acum initializarea variabilelor și prelucrarea opțiunilor.

```
% Initializare variabile

rtol = 1.e-3;
atol = 1.e-6;
plotfun = @odeplot;
statflag = 0;
statflag=strncmp(optsStats,'on');
if (nargin >= 4) & ~isempty(BT) %tabela Butcher
    [lambda,alfa,alfas,mu,s,oop,fsal]=BT();
else
    [lambda,alfa,alfas,mu,s,oop,fsal]=BS23();
end
if statflag %statistici
    stat=struct('ns',0,'nrej',0,'nfunc',0);
end

if nargin >= 5 & isnumeric(opts)
    rtol = opts;
elseif nargin >= 5 & isstruct(opts)
    statflag=strncmp(optsStats,'on');
    if ~isempty(opts.RelTol), rtol = opts.RelTol; end
    if ~isempty(opts.AbsTol), atol = opts.AbsTol; end
    if ~isempty(opts.OutputFcn),
        plotfun = opts.OutputFcn;
    end
end
if statflag %statistici
    stat=struct('ns',0,'nrej',0,'nfunc',0);
end

t0 = tspan(1);
tfinal = tspan(2);
```

```

tdir = sign(tfinal - t0);
plotit = (nargout == 0);
threshold = atol / rtol;
hmax = abs(0.1*(tfinal-t0));
t = t0;
y = y0(:);

% Face F apelabila prin feval

if ischar(F) & exist(F) ~=2
    F = inline(F,'t','y');
elseif isa(F,'sym')
    F = inline(char(F),'t','y');
end

% Initializeaza iesirile

if plotit
    plotfun(tspan,y,'init');
else
    tout = t;
    yout = y.';
end

```

Calculul lungimii pasului inițial este o chestiune delicată deoarece necesită unele cunoștințe despre scara globală a problemei.

```

% Calculeaza pasul initial

K=zeros(length(y0),s);
K(:,1)=F(t,y,varargin:); %prima evaluare
if statflag, stat.nfunc=stat.nfunc+1; end
r = norm(K(:,1))./max(abs(y),threshold),inf) + realmin;
h = tdir*0.8*rtol^(oop)/r;

```

Urmează acum ciclul principal. Procesul de integrare începe din  $t = t_0$  și  $t$  se incrementează până când se atinge  $t_{final}$ . Este posibil să se meargă și în sens invers dacă  $t_{final} < t_0$ .

```

% ciclul principal

while t ~= tfinal

    hmin = 16*eps*abs(t);
    if abs(h) > hmax, h = tdir*hmax; end
    if abs(h) < hmin, h = tdir*hmin; end

    % corectare pas final

```

```

if 1.1*abs(h) >= abs(tfinal - t)
    h = tfinal - t;
end

```

Iată calculul efectiv. Prima pantă  $K(:, 1)$  a fost deja calculată. Urmează acum  $s-1$  evaluări ale pantei, unde  $s$  este de numărul de stadii.

```

% tentativa calcul pas

for i=2:s
    K(:,i)=F(t+mu(i)*h,y+h*K(:,1:i-1)*...
        (lambda(i,1:i-1)'));
end
if statflag, stat.nfunc=stat.nfunc+s-1; end
tnew=t+h;
ynew=y+h*K*alfas;

```

Urmează estimarea erorii. Norma vectorului de eroare este scalată cu raportul dintre eroarea absolută și cea relativă. Utilizarea celui mai mic număr flotant, `realmin`, evită situația când `err` este zero.

```

% Estimeaza eroarea

e = h*K*(alfa-alfas);
err = norm(e./max(max(abs(y),abs(ynew)),threshold),...
    inf) + realmin;

```

Acum se testează dacă pasul s-a terminat cu succes. Dacă da, rezultatul este afișat sau adăugat la vectorul de ieșire. Dacă nu și se cer statistică, pasul nereușit se contorizează. Dacă metoda este de tip FSAL (First Same As Last), adică ultimul stadiu al pasului precedent este la fel cu primul stadiu al pasului următor, atunci ultima valoare de funcție calculată se reutilizează.

```

% Accepta solutia daca eroarea estimata < toleranta

if err <= rtol %pas acceptat
    t = tnew;
    y = ynew;
    if plotit
        if plotfun(t,y,'');
            break
        end
    else
        tout(end+1,1) = t;
        yout(end+1,:) = y.';
        if statflag
            stat.ns=stat.ns+1;
        end
    end
end

```

```

        end
    end
    if fsal % Reutilizare valoare finală dacă e cazul
        K(:,1)=K(:,s);
    else
        K(:,1)=F(t,y);
        if statflag, stat.nfunc=stat.nfunc+1; end
    end
else %pas respins
    if statflag, stat.nrej=stat.nrej+1; end
end

```

Estimarea erorii se utilizează pentru a calcula o nouă lungime de pas. Raportul `rtol/err` este supravîntar dacă pasul curent este terminat cu succes și subunitar dacă pasul curent a eşuat. Factorii 0.8 și 5 evită schimbările excesive ale lungimii pasului.

```

% Calcul pas nou
h = h*min(5,0.8*(rtol/err)^(oop));

```

Aici este singurul loc unde s-ar putea detecta o eventuală singularitate.

```

% Iesire dacă pasul este prea mic

if abs(h) <= hmin
    warning(sprintf('...
        dimensiune pas %e prea mica în t = %e.\n',h,t));
    t = tfinal;
end
end

```

Aici se încheie ciclul principal. Funcția de afișare trebuie să-și încheie munca.

```

if plotit
    plotfun([],[],'done');
end
if statflag %statistici
    fprintf('%d pasi cu succes\n',stat.ns)
    fprintf('%d pasi esuati\n', stat.nrej)
    fprintf('%d evaluari de functii\n', stat.nfunc)
end

```

Rezolvitor	tip problemă	Tip algoritm
ode45	Nonstiff	Pereche Runge-Kutta explicită, cu ordinele 4 și 5
ode23	Nonstiff	Pereche Runge-Kutta explicită, cu ordinele 2 și 3
ode113	Nonstiff	Metodă cu mai mulți pași explicită, cu ordin variabil, ordinele de la 1 la 13
ode15s	Stiff	Metodă cu mai mulți pași implicită, cu ordin variabil, ordinele de la 1 la 15
ode23s	Stiff	Pereche Rosenbrock modificată (cu un pas), cu ordinele 2 și 3
ode23t	Stiff	Regula implicită a trapezului, cu ordinele 2 și 3
ode23tb	Stiff	Algoritm Runge-Kutta implicit, ordinele 2 și 3

Tabela 10.6: Rezolvitori MATLAB pentru ecuații diferențiale ordinare

## 10.8. Ecuații diferențiale ordinare în MATLAB

### 10.8.1. Rezolvitori

MATLAB are facilități foarte puternice de rezolvare a problemelor cu valori inițiale pentru ecuații diferențiale ordinare:

$$\frac{dy}{dt} = f(t, y(t)), \quad y(t_0) = y_0.$$

Cel mai simplu mod de a rezolva o astfel de problemă este de a scrie o funcție care evaluează  $f$  și de a apela unul dintre rezolvitorii MATLAB. Informația minimă pe care un rezolvitor trebuie să o primească este numele funcției, multimea valorilor lui  $t$  pe care se cere soluția și valoarea inițială  $y_0$ . Rezolvitorii MATLAB acceptă argumente de intrare și ieșire opționale care permit să se specifice mai mult despre problema matematică și modul de rezolvare a ei. Fiecare rezolvitor MATLAB este conceput să fie eficient în anumite situații, dar toți sunt în esență interschimbabili. Toți rezolvitorii au aceeași sintaxă, ceea ce ne permite să încercăm diferite metode numerice atunci când nu știm care ar fi cea mai potrivită. Sintaxa este

`[t, y]=rezolvitor (@fun, tspan, y0, optiuni, p1, p2, ...)`  
unde `rezolvitor` este unul din rezolvitorii dați în tabela 10.6.

Argumentele de intrare sunt

- `fun` – specifică funcția din membrul drept. În versiunile 6.x este un handler de funcție, iar în versiunile 5.x este un nume de funcție (în acest caz se scrie '`fun`' nu `@fun`);
- `tspan` – vector ce specifică intervalul de integrare. Dacă este un vector cu două elemente `tspan=[t0 tfinal]`, rezolvitorul integrează de la `t0` la `tfinal`. Dacă `tspan` are mai mult de două elemente rezolvitorul returnează soluțiile în acele puncte. Abscisele trebuie ordonate crescător sau descrescător. Rezolvitorul nu își alege pașii după valorile din `tspan`, ci obține valorile în aceste puncte prin prelungiri continue ale formulelor de bază care au același ordin de precizie ca și soluțiile calculate în puncte.
- `optiuni` – opțiunile permit setarea unor parametrii ai rezolvitorului și se crează cu `odeset`.

Parametrii de ieșire sunt:

- `t` – vectorul coloană al absciselor;
- `y` – tabloul soluțiilor: o linie corespunde unei abscise, iar o coloană unei componente a soluției.

După opțiuni pot să apară parametrii variabili,  $p_1, p_2, \dots$  care sunt transmiși funcției  $\text{fun}$  la fiecare apel. De exemplu

```
[t,y]=rezolvitor(@fun,tspan,y0,optiuni,p1,p2,...)
apelează
    fun(T,Y,flag,p1,p2,...).
```

### 10.8.2. Exemple non-stiff

Să considerăm ecuația scalară

$$y'(t) = -y(t) + 5e^{-t} \cos 5t, \quad y(0) = 0,$$

pentru  $t \in [0, 3]$ . Membrul drept este conținut în fișierul `f1scal.m`:

```
function yder=f1scal(t,y)
%F1SCAL Exemplu de EDO scalara
yder = -y+5*exp(-t).*cos(5*t);
```

Vom folosi rezolvitorul `ode45`. Secvența de comenzi MATLAB

```
>> tspan = [0,3]; yzero=0;
>> [t,y]=ode45(@f1scal,tspan,yzero);
>> plot(t,y,'k--*')
>> xlabel('t'), ylabel('y(t)')
```

produce graficul din figura 10.4. Soluția exactă este  $y(t) = e^{-t} \sin 5t$ . Verificăm aceasta calculând

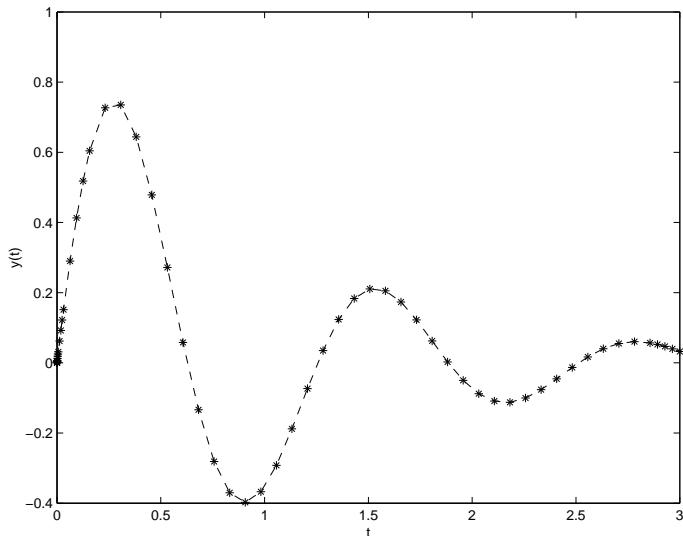


Figura 10.4: Un exemplu de ecuație diferențială ordinată scalară

maximul modulului diferențelor dintre valorile furnizate de `ode45` și valorile soluției exacte calculate în abscisele furnizate de `ode45`:

```
>> norm(y-exp(-t).*sin(5*t),inf)
ans =
3.8416e-004
```

Să considerăm acum ecuația pendulului simplu [17, secțiunea 1.4]:

$$\frac{d^2}{dt^2}\theta(t) = -\frac{g}{L} \sin \theta(t),$$

unde  $g$  este acceleratia gravitațională și  $L$  este lungimea pendulului. Această ecuație de ordinul al doilea se poate transforma într-un sistem de ecuații de ordinul I, introducând necunoscutele  $y_1(t) = \theta(t)$  și  $y_2(t) = d\theta(t)/dt$ :

$$\begin{aligned}\frac{d}{dt}y_1(t) &= y_2(t), \\ \frac{d}{dt}y_2(t) &= -\frac{g}{L} \sin y_1(t).\end{aligned}$$

Aceste ecuații sunt codificate în fișierul pend.m, dat în continuare:

```
function yp=pend(t,y,g,L)
%PEND - pendul simplu
%g - acceleratia gravitaționala, L - lungimea
yp=[y(2); -g/L*sin(y(1))];
```

Aici,  $g$  și  $L$  sunt parametrii suplimentari care vor fi furnizați lui pend de către rezolvator. Vom calcula soluția pentru  $t \in [0, 10]$  și trei condiții inițiale diferite.

```
g=10; L=10;
tspan = [0,10];
yazero = [1; 1]; ybzero = [-5; 2];
yczero = [5; -2];
[ta,ya] = ode45(@pend,tspan,yazero,[],g,L);
[tb,yb] = ode45(@pend,tspan,ybzero,[],g,L);
[tc,yc] = ode45(@pend,tspan,yczero,[],g,L);
```

În apelurile de forma

```
[ta,ya] = ode45(@pend,tspan,yazero,[],g,L);
```

[] reprezintă opțiunile, iar următorii sunt parametrii suplimentari care vor fi transmiși membrului drept. Pentru a obține grafice de fază vom reprezenta pe  $y_2(t)$  în funcție de  $y_1(t)$ . Este sugestiv să reprezentăm pe același grafic și câmpul de vectori cu quiver. Săgețile generate de quiver au direcția gradientului  $[y_2, -\sin y_1]$  și lungimea proporțională norma euclidiană a acestui vector. Imaginea obținută apare în figura 10.5.

```
[y1,y2] = meshgrid(-5:0.5:5,-3:0.5:3);
Dy1Dt = y2; Dy2Dt = -sin(y1);
quiver(y1,y2,Dy1Dt,Dy2Dt)
hold on
plot(ya(:,1),ya(:,2),yb(:,1),yb(:,2),yc(:,1),yc(:,2))
axis equal, axis([-5,5,-3,3])
xlabel y_1(t), ylabel y_2(t), hold off
```

Orice soluție a ecuației pendulului conservă energia: cantitatea  $y_2(t)^2 - \cos y_1(t)$  este constantă. Vom verifica aceasta prin

```
>> Ec = 0.5*yc(:,2).^2-cos(yc(:,1));
>> max(abs(Ec(1)-Ec))
ans =
0.0263
```

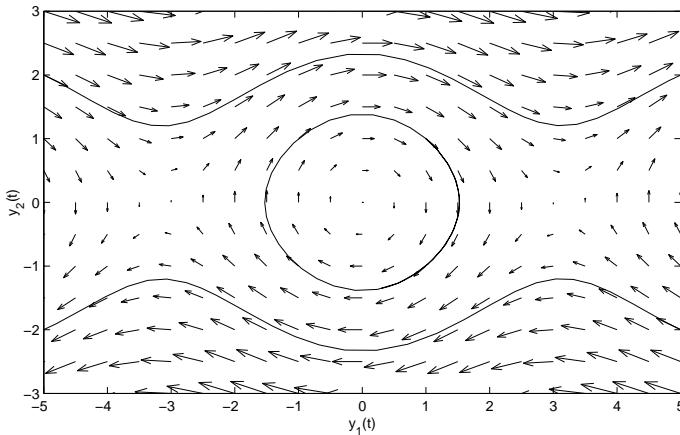


Figura 10.5: Grafic de fază pentru ecuația pendulului

### 10.8.3. Opțiuni

Funcția `odeset` crează o structură de opțiuni care poate fi transmisă unui rezolvitor. Argumentele lui `odeset` sunt perechi nume proprietate/valoare proprietate. Sintaxa este

```
optiuni=odeset('nume1', valoare1, 'nume2', valoare2, ...)
```

Aceasta crează o structură de opțiuni în care proprietățile cu numele dat primesc o valoare specificată. Proprietățile nespecificate primesc valori implicate. Pentru toate proprietățile este suficient să dăm doar caracterele de la început care identifică unic numele proprietății. `odeset` fără argumente afișează toate numele de proprietăți și valorile lor posibile; valorile implicate apar între acolade:

```
>> odeset
    AbsTol: [ positive scalar or vector {1e-6} ]
    RelTol: [ positive scalar {1e-3} ]
    NormControl: [ on | {off} ]
    OutputFcn: [ function ]
    OutputSel: [ vector of integers ]
    Refine: [ positive integer ]
    Stats: [ on | {off} ]
    InitialStep: [ positive scalar ]
    MaxStep: [ positive scalar ]
    BDF: [ on | {off} ]
    MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
    Jacobian: [ matrix | function ]
    JPATTERN: [ sparse matrix ]
```

Categorie	Numele proprietăți
Controlul erorii	RelTol, AbsTol, NormControl
Ieșire rezolvitor	OutputFcn, OutputSel, Refine, Stats
Matrice jacobiană	Jacobian, JPattern, Vectorized
Controlul pasului	InitialStep, MaxStep
Matrice de masă și DAE	Mass, MStateDependence, MvPattern, MassSingular, InitialSlope
Evenimente	Events
specifice ode15s	MaxOrder, BDF

Tabela 10.7: Proprietăți ale rezolvatorilor

```

Vectorized: [ on | {off} ]
Mass: [ matrix | function ]
MStateDependence: [ none | weak | strong ]
MvPattern: [ sparse matrix ]
MassSingular: [ yes | no | {maybe} ]
InitialSlope: [ vector ]
Events: [ function ]

```

Modificarea unei structuri de opțiuni se poate realiza cu

```
optiuni=odeset(optiune-veche,'nume1', valoare1,...)
```

care poziționează valorile lui opțiuni pe valorile existente în optiune-veche, iar pe cele precizate de perechile nume/valoare le actualizează sau cu

```
options=odeset(optiune-veche, optiune-noua)
```

care combină structurile optiune-veche și optiune-noua. Valorile din optiune-noua diferite de [] le înlocuiesc pe cele din optiune-veche. O structură de opțiuni poate fi interogată cu comanda:

```
o=odeget(optiuni,'name')
```

Aceasta returnează valoarea proprietății specificate sau o matrice nulă dacă valoarea proprietății nu este specificată în structura optiuni. Tabela 10.7 dă tipurile de proprietăți și numele proprietăților.

Exemplul următor rezolvă sistemul lui Rössler [32, secțiunea 12.2],

$$\begin{aligned}\frac{d}{dt}y_1(t) &= -y_2(t) - y_3(t), \\ \frac{d}{dt}y_2(t) &= y_1(t) + \alpha y_2(t), \\ \frac{d}{dt}y_3(t) &= b + y_3(t)(y_1(t) - c),\end{aligned}$$

unde  $a$ ,  $b$  și  $c$  sunt parametri reali. Funcția care definește ecuația diferențială este:

```

function yd=Roessler(t,y,a,b,c)
%ROESSLER sistemul Roessler parametrizat

yd = [-y(2)-y(3); y(1)+a*y(2); b+y(3)*(y(1)-c)];
```

Vom modifica eroarea absolută și cea relativă cu

```
options = odeset('AbsTol',1e-7,'RelTol',1e-4);
```

Script-ul `Roessler.m` (sursa 10.4) rezolvă sistemul lui Rössler pe intervalul  $t \in [0, 100]$  cu valoarea inițială  $y(0) = [1, 1, 1]^T$  și cu seturile de parametrii  $(a, b, c) = (0.2, 0.2, 2.5)$  și  $(a, b, c) = (0.2, 0.2, 5)$ . Rezultatele apar în figura 10.6. Subplot-ul 221 dă graficul soluției în spațiul tridimensional.

#### **Sursa MATLAB 10.4 Sistemul lui Rössler**

```
tspan = [0,100]; y0 = [1;1;1];
options = odeset('AbsTol',1e-7,'RelTol',1e-4);
a=0.2; b=0.2; c1=2.5; c2=5;
[t,y] = ode45(@Roessler,tspan,y0,options,a,b,c1);
[t2,y2] = ode45(@Roessler,tspan,y0,options,a,b,c2);
subplot(2,2,1), plot3(y(:,1),y(:,2),y(:,3))
title('c=2.5'), grid
xlabel('y_1(t)'), ylabel('y_2(t)'), zlabel('y_3(t)');
subplot(2,2,2), plot3(y2(:,1),y2(:,2),y2(:,3))
title('c=5'), grid
xlabel('y_1(t)'), ylabel('y_2(t)'), zlabel('y_3(t)');
subplot(2,2,3); plot(y(:,1),y(:,2))
title('c=2.5')
xlabel('y_1(t)'), ylabel('y_2(t)')
subplot(2,2,4); plot(y2(:,1),y2(:,2))
title('c=5')
xlabel('y_1(t)'), ylabel('y_2(t)')
```

sional al fazelor pentru  $c = 2.5$  și subplot-ul 223 dă proiecția ei pe planul  $y_1y_2$ . Subplot-urile 222 și 224 dau graficele corespunzătoare pentru  $c = 5$ . Vom mai discuta și exemplifica opțiunile și în subsecțiunile următoare. Pentru detalii a se vedea `help odest` sau `doc odeset`.

#### **10.8.4. Ecuații stiff**

”Stiff” (țepăniș, rigid, dificil, anevoiești) este un concept subtil, dificil și important în rezolvarea numerică a ecuațiilor diferențiale ordinare. El depinde de ecuația diferențială, de condițiile initiale și de metoda numerică. În [46] se dă următoarea caracterizare computațională a acestui termen:

„O problemă este stiff dacă soluția căutată variază lent, dar există soluții apropriate care variază rapid, astfel că metoda numerică trebuie să utilizeze pași foarte mici pentru a obține rezultate satisfăcătoare.”

Conceptul de ”stiffness” este o chestiune de eficiență. Metodele nonstiff pot rezolva problemele stiff, dar într-un timp foarte lung.

Exemplul care urmează provine din [46] și este un model al propagării unei flăcări. Când se aprinde un chibrit, mingea (sfera) de foc crește rapid până când atinge dimensiunea critică. Apoi rămâne la această dimensiune, deoarece volumul de oxigen consumat de combustie în interiorul mingii echilibrează volumul disponibil la suprafață. Un model simplu este dat de problema cu valori inițiale:

$$\begin{aligned} y' &= y^2 - y^3, \\ y(0) &= \delta, \quad 0 \leq t \leq 2/\delta \end{aligned} \tag{10.8.1}$$

Funcția reală  $y(t)$  reprezintă raza sferei. Termenii  $y^2$  și  $y^3$  provin din suprafață și volum. Soluția se caută pe un interval de lungime invers proporțională cu  $\delta$ . Vom încerca să rezolvăm problema cu `ode45`. Dacă nu este foarte mic problema nu este foarte stiff. Alegem  $\delta = 0.01$  și eroarea relativă  $10^{-4}$ .

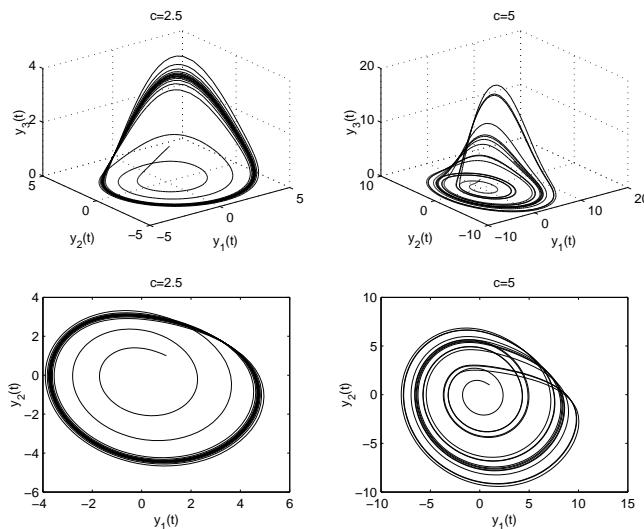
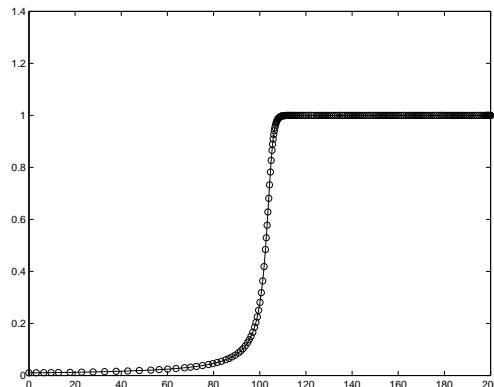


Figura 10.6: Soluțiile sistemului lui Rössler în spațiul fazelor

```
delta=0.01;
F = inline('y^2-y^3','t','y');
opts = odeset('RelTol',1e-4);
ode45(F, [0, 2/delta], delta, opts);
```

Neavând parametrii de ieșire, rezolvitorul reprezintă grafic soluția (vezi figura 10.7). Se observă că ea pornește de la 0.1, crește lent până când  $t$  se apropie de  $1/\delta$ , adică 100 și apoi crește rapid până la valoarea 1, ajungându-se într-o stare de echilibru. Se utilizează 185 de puncte. Dacă luăm  $\delta$  mai mic,

Figura 10.7: Propagarea unei flăcări,  $\delta = 0.1$ 

de exemplu 0.0001, caracterul stiff devine mai pregnant. Se generează 12161 de puncte. Graficul apare în figura 10.8, în partea de sus. În partea de jos este un zoom obținut în vecinătatea stării de echilibru.

Figura a fost obținută cu script-ul chibrit2.m dat mai jos. Opțiunea Stats setată pe on permite obținerea unor statistici ale rezolvitorului.

```
delta=1e-4; er=1e-4;
F = inline('y^2-y^3','t','y');
opts = odeset('RelTol',er,'Stats','on');
[t,y]=ode45(F,[0,2/delta],delta,opts);
subplot(2,1,1)
plot(t,y,'c-'); hold on
h=plot(t,y,'bo');
set(h,'MarkerFaceColor','b','Markersize',4);
hold off
title ode45
subplot(2,1,2)
plot(t,y,'c-'); hold on
h=plot(t,y,'bo');
set(h,'MarkerFaceColor','b','Markersize',4);
axis([0.99e4,1.12e4,0.9999,1.0001])
hold off
```

De notat că rezolvitorul menține soluția în limita de eroare cerută, dar cu prețul unui efort foarte mare. Situația este și mai dramatică pentru valori mai mici ale erorii relative, de exemplu  $10^{-5}$  sau  $10^{-6}$ . Inițial problema nu este stiff. Ea devine astfel pe măsură ce se apropiie de starea de echilibru,

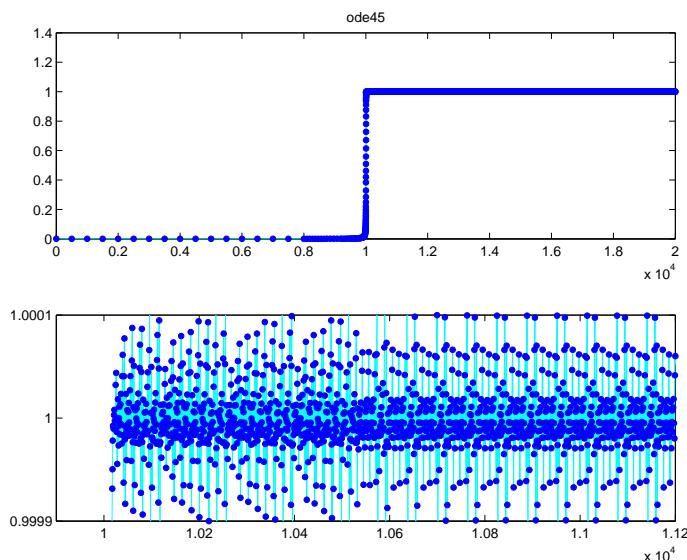


Figura 10.8: Propagarea unei flăcări,  $\delta = 0.0001$ , eroarea relativă  $1e-4$  (sus) și un zoom pe soluție (jos)

adică de valoare  $y(t) = 1$ , unde are loc o creștere rapidă (comparativ cu scara de timp foarte lungă).

Vom utiliza un rezolvitor pentru probleme de tip stiff. Metodele pe care se bazează astfel de rezolvitori sunt metode *implicite*. La fiecare pas rezolvitorii utilizează operații matriciale și rezolvă sisteme de

ecuații liniare care îi ajută să prevadă evoluția soluției (desigur că pentru probleme scalare matricea are dimensiunea 1 pe 1). Să calculăm soluția folosind acum rezolvitorul `ode23s`. Vom modifica în script-ul precedent doar numele rezolvitorului: `ode23s` în loc de `ode45`. Graficul obținut apare în figura 10.9. Efortul depus de rezolvitor este de această dată mult mai mic, după cum se poate vedea

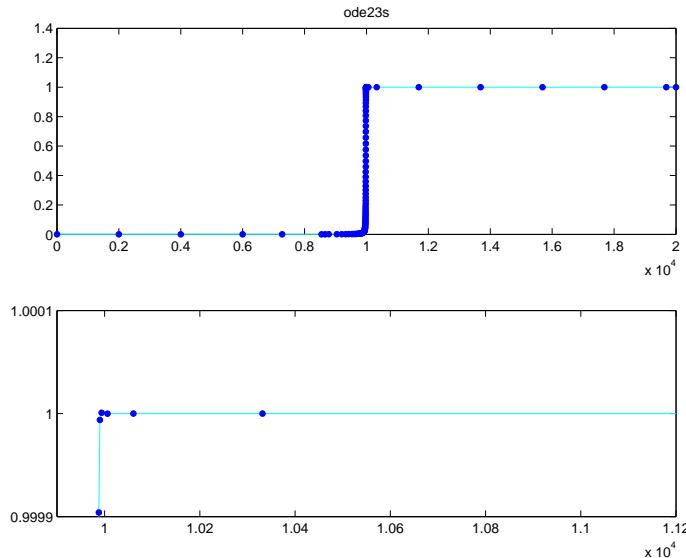


Figura 10.9: Propagarea unei flăcări,  $\delta = 0.0001$ , eroarea relativă  $1e-4$  (sus) și un zoom pe soluție (jos), obținute cu `ode23s`

examinând statisticile generate de rezolvitor. Statistica generată de `ode23s` este în acest caz:

```
99 successful steps
7 failed attempts
412 function evaluations
99 partial derivatives
106 LU decompositions
318 solutions of linear systems
```

A se compara cu cea generată de `ode45`:

```
3040 successful steps
323 failed attempts
20179 function evaluations
0 partial derivatives
0 LU decompositions
0 solutions of linear systems
```

Se poate obține soluția exactă a problemei (10.8.1). Ecuația este cu variabile separabile. Integrând se obține o ecuație implicită care-l dă pe  $y$ :

$$\frac{1}{y} + \ln\left(\frac{1}{y} - 1\right) = \frac{1}{\delta} + \ln\left(\frac{1}{\delta} - 1\right) - t.$$

Expresia soluției exacte este

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}$$

unde  $a = 1/\delta - 1$ , iar  $W$  este funcția lui Lambert. Aceasta din urmă este soluția ecuației funcționale

$$W(z)e^{W(z)} = z.$$

Cleve Moler [46] dă o comparație foarte plastică între rezolvitorii expliciti și cei implictii.

„Imaginați-vă că vă întoarceți dintr-o plimbare pe munte. Sunteți într-un canion îngust cu pante abrupte de ambele părți. Un algoritm explicit va selecta gradientul local pentru a găsi direcția de coborâre. Dar urmarea gradientului în ambele părți ale drumului vă va trimite viguros înainte și înapoi prin canion, la fel ca `ode45`. Veți ajunge în final acasă, dar înțunericul se va lăsa cu mult timp înainte de a ajunge. Un algoritm implicit vă va ține cu ochii la drum și va anticipa unde vă va duce fiecare pas. Concentrarea suplimentară merită.”

Vom considera acum un exemplu datorat lui Robertson și tratat amănunțit în [32, 17]. Sistemul lui Robertson

$$\begin{aligned} \frac{dy_1}{dt} &= -\alpha y_1(t) + \beta y_2(t)y_3(t), \\ \frac{dy_2}{dt} &= \alpha y_1(t) - \beta y_2(t)y_3(t) - \gamma y_2^2(t), \\ \frac{dy_3}{dt} &= \gamma y_2^2(t) \end{aligned}$$

modelează reacția între trei specii chimice. Sistemul este codificat în funcția `chem.m`, dată mai jos.

```
function yprime=chem(t,y,alpha,beta,gamma)
%CHEM - modelul lui Robertson pentru reactii chimice

yprime = [-alpha*y(1)+beta*y(2)*y(3);
           alpha*y(1)-beta*y(2)*y(3)-gamma*y(2)^2;
           gamma*y(2)^2];
```

Script-ul `robertson.m` rezolvă sistemul lui Robertson pentru  $\alpha = 0.04$ ,  $\beta = 10^4$ ,  $\gamma = 3 \times 10^7$ ,  $t \in [0, 3]$  și condițiile inițiale  $y(0) = [1, 0, 0]^T$ . Se utilizează rezolvitorii `ode45` și `ode15s`, generându-se și statistici. Din motive de scară s-a reprezentat numai  $y_2$ .

```
alpha = 0.04; beta = 1e4; gamma = 3e7;
tspan = [0,3]; y0 = [1;0;0];
opts=odeset('Stats','on');
[ta,ya] = ode45(@chem,tspan,y0,opts,alpha,beta,gamma);
subplot(1,2,1), plot(ta,ya(:,2),'-*')
ax = axis; ax(1) = -0.2; axis(ax);
xlabel('t'), ylabel('y_2(t)')
title('ode45','FontSize',14)
[tb,yb] = ode15s(@chem,tspan,y0,opts,alpha,beta,gamma);
subplot(1,2,2), plot(tb,yb(:,2),'-*')
axis(ax)
xlabel('t'), ylabel('y_2(t)')
title('ode15s','FontSize',14)
```

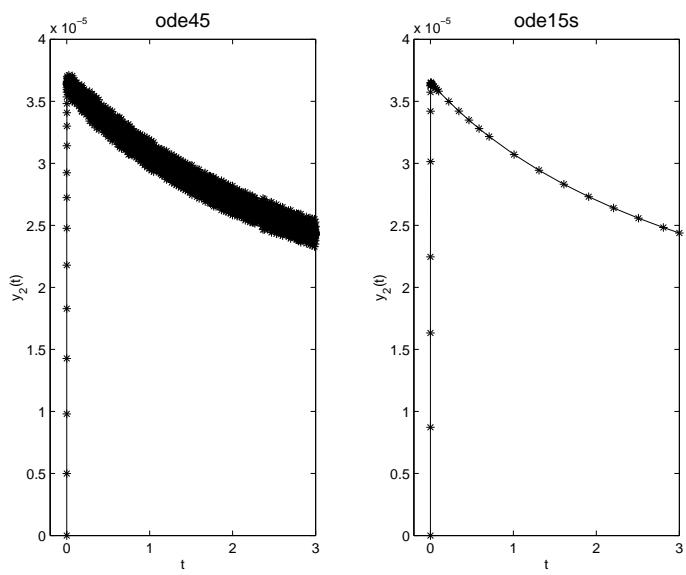


Figura 10.10: Soluția  $y_2$  a sistemului lui Robertson, obținută cu `ode45` (stânga) și `ode15s`

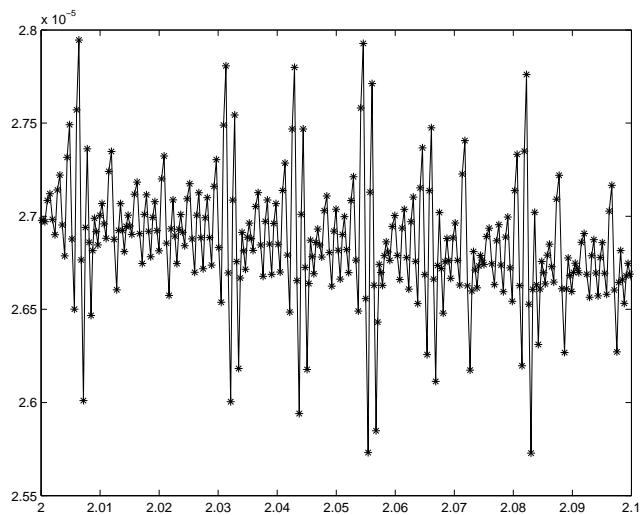


Figura 10.11: Zoom pe soluția  $y_2$  a sistemului lui Robertson, obținută cu `ode45`

Graficele lui  $y_2$  apar în figura 10.10. Figura 10.11 conține un zoom pe graficul obținut cu `ode45`. Statisticile generate de rezolvatorii sunt

```
2052 successful steps
440 failed attempts
14953 function evaluations
0 partial derivatives
0 LU decompositions
0 solutions of linear systems
```

pentru `ode45` și

```
33 successful steps
5 failed attempts
73 function evaluations
2 partial derivatives
13 LU decompositions
63 solutions of linear systems
```

pentru `ode15s`. De notat că în calculele de mai sus avem

```
disp([length(ta), length(tb)])
8209           34
```

ceea ce ne arată că `ode45` returnează cam de 250 de ori mai multe puncte decât `ode15s`.

Rezolvatorii stiff utilizează informații despre matricea jacobiană,  $\partial f_i / \partial y_j$ , în diferite puncte ale soluției. Implicit, se generează automat aproximări ale jacobianului utilizând diferențe finite. Totuși, dacă jacobianul se dă explicit, se obține o eficiență și o robustețe mai mare a codului. Sună disponibile opțiuni care permit să se specifică o funcție care evaluatează jacobianul sau o matrice constantă (Jacobian), dacă jacobianul este rar și şablonul de raritate (`Jspattern`), dacă este scris în formă vectorială (`Vectorized`). Pentru a ilustra modul de codificare a jacobianului, vom considera sistemul

$$\frac{d}{dt}y(t) = Ay(t) + y(t) \cdot (1 - y(t)) + v, \quad (10.8.2)$$

unde  $A$  este o matrice  $N \times N$  și  $v$  este un vector  $N \times 1$  cu

$$A = r_1 \begin{bmatrix} 0 & 1 & & \\ -1 & 0 & 1 & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \end{bmatrix} + r_2 \begin{bmatrix} -2 & 1 & & \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \end{bmatrix},$$

$v = [r_1 - r_2, 0, \dots, 0, r_1 + r_2]^T$ ,  $r_1 = -a/(2\Delta x)$  și  $r_2 = b/\Delta x^2$ . Aici  $a$ ,  $b$  și  $\Delta x$  sunt parametrii cu valorile  $a = 1$ ,  $b = 5 \times 10^{-2}$  și  $\Delta x = 1/(N + 1)$ . Acest sistem de ecuații diferențiale ordinare apare la semidiscretizarea prin metoda liniilor a ecuației cu derive parțiale

$$\frac{\partial}{\partial t}u(x, t) + a\frac{\partial}{\partial x}u(x, t) = b\frac{\partial^2}{\partial x^2}u(x, t) + u(x, t)(1 - u(x, t)), \quad 0 \leq x \leq 1,$$

cu condițiile pe frontieră de tip Dirichlet  $u(0, t) = u(1, t) = 1$ . Această ecuație cu derive parțiale poate fi rezolvată și direct cu `pdepe`. Componenta  $y_j(t)$  a sistemului de ecuații diferențiale ordinare

aproximează  $u(j\Delta x, t)$ . Presupunem că ecuația cu derivate parțiale verifică condițiile  $u(x, 0) = (1 + \cos 2\pi x)/2$ , pentru care se poate arăta că  $u(x, t)$  trebuie către starea de echilibru  $u(x, t) \equiv 1$  când  $t \rightarrow \infty$ . Condiția inițială corespunzătoare pentru ecuația diferențială ordinată este  $(y_0)_j = (1 + \cos(2\pi j/(N+1))/2)$ . Jacobianul pentru ecuația diferențială ordinată are forma  $A + I - 2\text{diag}(y(t))$ , unde  $I$  este matricea identică. Sursa MATLAB 10.5 conține o funcție care implementează și rezolvă sistemul (10.8.2) utilizând `ode15s`. Utilizarea subfuncțiilor și a tipurilor function handle face posibil ca întreg codul să fie conținut într-un singur fișier, numit `rccd.m`. S-a luat  $N = 40$  și  $t \in [0, 2]$ . Subfuncția `jacobian` evaluează jacobianul, `jpattern` dă şablonul de raritate al jacobianului sub formă unei matrice rare cu elemente 0 și 1 și `f` codifică membrul drept al ecuației diferențiale. A  $j$ -a coloană a matricei de ieșire  $y$  conține aproximarea lui  $y_j(t)$ ; matricea  $U$  a fost creată adăugând o coloană de 1 la fiecare capăt al lui  $y$ , conform condițiilor pe frontieră pentru ecuația cu derivate parțiale. Graficul produs de `rccd` apare în figura 10.12.

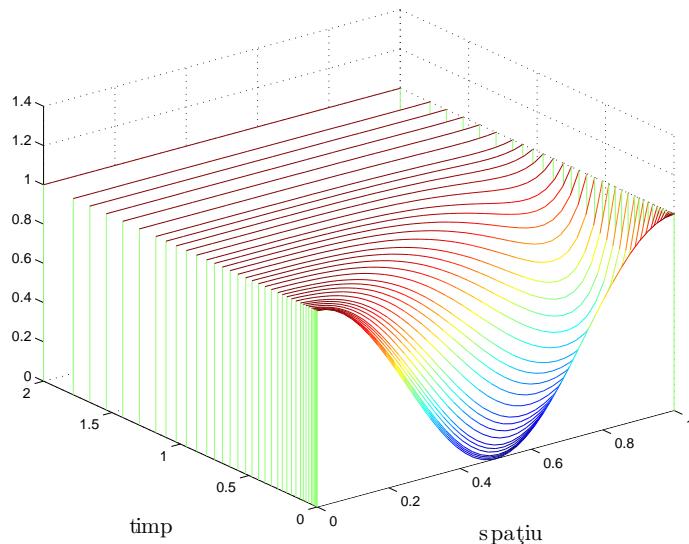


Figura 10.12: Exemplu stiff, cu informații despre jacobian

### 10.8.5. Tratarea evenimentelor

În multe situații, determinarea ultimei valori  $t_{final}$  a lui `tspan` este un aspect important al problemei. Un exemplu este căderea unui corp asupra căruia acționează forța gravitațională și rezistența aerului. Când atinge el pământ? Un alt exemplu este problema celor două corpuri, adică determinarea orbitei unui corp supus atracției gravitaționale a unui alt corp mult mai greu. Care este perioada orbitei? Facilitatea de prelucrare a evenimentelor a rezolvatorilor din MATLAB furnizează răspunsuri la astfel de întrebări.

Detectia evenimentelor în MATLAB presupune două funcții  $f(t, y)$  și  $g(t, y)$  și o condiție inițială  $(t_0, y_0)$ . Problema este de a găsi o funcție  $y(t)$  și o valoare finală  $t_*$  astfel încât

$$\begin{aligned} y' &= f(t, y) \\ y(t_0) &= y_0 \end{aligned}$$

**Sursa MATLAB 10.5** Problemă stiff cu informații despre jacobian

```

function rcd
%RCD EDO Stiff pentru problema reactie-convecție-difuzie.
% obținuta prin semidiscretizare cu metoda liniilor

N = 40; a = 1; b = 5e-2;
tspan = [0;2]; space = [1:N]/(N+1);
y0 = 0.5*(1+cos(2*pi*space));
y0 = y0(:);
options = odeset('Jacobian',@jacobian,'Jpattern',...
    jpattern(N),'RelTol',1e-3,'AbsTol',1e-3);
[t,y] = ode15s(@f,tspan,y0,options,N,a,b);
e = ones(size(t)); U = [e y e];
waterfall([0:1/(N+1):1],t,U)
xlabel('spa\c{t}iu','FontSize',16,'Interpreter','LaTeX')
ylabel('timp','FontSize',16,'Interpreter','LaTeX')

% -----
% Subfunctii.
% -----
function dydt = f(t,y,N,a,b)
%F      ecuația diferențială.

r1 = -a*(N+1)/2; r2 = b*(N+1)^2;
up = [y(2:N);0]; down = [0;y(1:N-1)];
e1 = [1;zeros(N-1,1)]; eN = [zeros(N-1,1);1];
dydt = r1*(up-down) + r2*(-2*y+up+down) + (r2-r1)*e1 +...
    (r2+r1)*eN + y.*(1-y);

% -----
function dfdy = jacobian(t,y,N,a,b)
%JACOBIAN  Jacobianul.

r1 = -a*(N+1)/2; r2 = b*(N+1)^2;
u = (r2-r1)*ones(N,1);
v = (-2*r2+1)*ones(N,1) - 2*y;
w = (r2+r1)*ones(N,1);
dfdy = spdiags([u v w], [-1 0 1], N, N);
% -----
function S = jpattern(N)
%JPATTERN sablonul de raritate al jacobianului.
e = ones(N,1);
S = spdiags([e e e], [-1 0 1], N, N);

```

și

$$g(t_*, y(t_*)) = 0.$$

Un model simplu al unui corp în cădere este

$$y'' = -1 + y'^2,$$

cu o condiție inițială care dă valori pentru  $y(0)$  și  $y'(0)$ . Problema este pentru ce valori a lui  $t$  avem  $y(t) = 0$ ? Codul pentru funcția  $f(t, y)$  este

```
function ydot=f(t,y)
ydot = [y(2); -1+y(2)^2];
```

Ecuația a fost scrisă ca un sistem de două ecuații de ordinul I, deci  $g(t, y) = y_1$ . Codul pentru  $g(t, y)$  este

```
function [gstop,isterminal,direction] = g(t,y)
gstop = y(1);
isterminal = 1;
direction = 0;
```

Primul argument de ieșire,  $gstop$ , este valoarea pe care dorim să-o anulăm. Dacă al doilea argument de ieșire,  $isterminal$ , are valoarea 1, rezolvitorul va termina execuția dacă  $gstop$  este zero. Dacă  $isterminal = 0$ , evenimentul este înregistrat și rezolvarea continuă.  $direction$  poate fi -1, 1 sau 0, după cum zeroul se atinge dacă funcția este descrescătoare, crescătoare sau nemonotonă. Calculul și reprezentarea traectoriei se poate face cu

```
function falling_body(y0)
opts = odeset('events',@g);
[t,y,tfinal] = ode45(@f, [0,Inf],y0,opts);
tfinal
plot(t,y(:,1),'-', [0,tfinal],[1,0],'o')
axis([-0.1, tfinal+0.1, -0.1, max(y(:,1)+0.1)]);
xlabel t
ylabel y
title('Corp in cadere')
text(tfinal-0.8, 0, ['tfinal = ' num2str(tfinal)])
```

Pentru valoarea inițială  $y0=[1; 0]$  se obține

```
>> falling_body([1;0])
tfinal =
1.65745691995813
```

și graficul din figura 10.13.

Detectia evenimentelor este utilă în probleme ce presupun fenomene periodice. Problema celor două corpuri este un exemplu bun. Ea descrie orbita unui corp asupra căruia acționează forța gravitațională a unui corp mult mai greu. Utilizând coordonate carteziene,  $u(t)$  și  $v(t)$  cu originea în corpul mai greu, ecuațiile sunt:

$$\begin{aligned} u''(t) &= -\frac{u(t)}{r(t)^3} \\ v''(t) &= -\frac{v(t)}{r(t)^3}, \end{aligned}$$

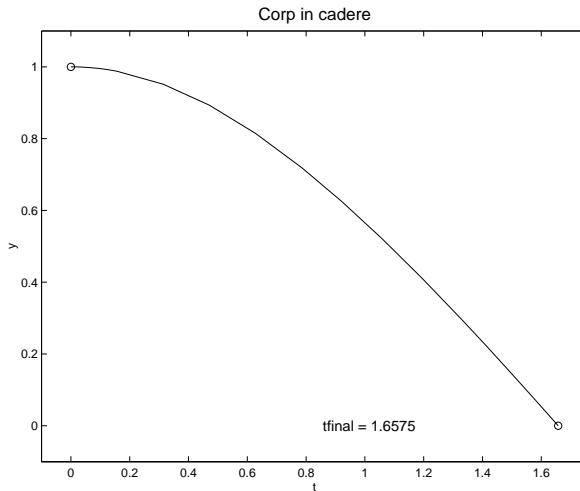


Figura 10.13: Traiectoria unui corp în cădere

unde  $r(t) = \sqrt{u(t)^2 + v(t)^2}$ . Întreaga rezolvare este conținută într-un singur fișier de tip funcție, `orbit.m` (sursa MATLAB 10.6). Parametrul de intrare, `reltol`, este eroarea (precizia) relativă dorită. Codificarea problemei `twobody` și funcția de tratare a evenimentelor `gstop` sunt date prin subfuncții (ele pot fi păstrate la fel de bine în fișiere separate). Calculul orbitei se realizează cu `ode45`. Argumentul de intrare `y0` este un vector cu 4 elemente, care dă poziția inițială și viteza. Corpul ușor pornește din poziția  $(1, 0)$  și are viteza inițială  $(0, 0.3)$ , care este perpendiculară pe vectorul poziției inițiale. Argumentul `opts` este o structură creată cu `odeset`, care specifică eroarea relativă (egală cu `reltol`) și funcția de tratare a evenimentelor `gstop`. Ultimul argument al lui `ode45` este o copie `y0`, transmisă atât lui `twobody` cât și lui `gstop`. Vectorul bidimensional `d` din `gstop` este diferența dintre poziția curentă și punctul de pornire. Viteza în poziția curentă este dată de vectorul bidimensional `v`, iar cantitatea `val` este produsul scalar al lui `d` și `v`. Expresia funcției de oprire este

$$g(t, y) = d'(t)^T d(t),$$

unde

$$d = (y_1(t) - y_1(0), y_2(t) - y_2(0))^T.$$

Punctele în care  $g(t, y(t)) = 0$  sunt extreme locale ale lui  $d(t)$ . Punând `dir = 1`, vom indica că zerourile lui  $g(t, y)$  sunt atinse de sus, ceea ce corespunde minimelor. Setând `isterm = 1`, vom indica faptul că procesul de calcul trebuie oprit la întâlnirea primului minim. Dacă orbita este periodică, atunci orice minim al lui  $d$  apare când corpul se întoarce în punctul inițial.

Apelând `orbit` cu o precizie mică  
`orbit(2e-3)`  
 se obține

```
tfinal =
2.35087197761946
yfinal =
0.98107659901112 -0.00012519138558
```

**Sursa MATLAB 10.6** Problema celor două corpuri

```

function orbit(reltol)
y0 = [1; 0; 0; 0.3];
opts = odeset('events', @gstop,'RelTol',reltol);
[t,y,te,ye] = ode45(@twobody,[0,2*pi], y0, opts, y0);
tfinal = te(end)
yfinal = ye(end,1:2)
plot(y(:,1),y(:,2),'-',0,0,'ro')
axis([-0.1 1.05 -0.35 0.35])

%-----
function ydot = twobody(t,y,y0)
r = sqrt(y(1)^2 + y(2)^2);
ydot = [y(3); y(4); -y(1)/r^3; -y(2)/r^3];

%-----
function [val,isterm,dir] = gstop(t,y,y0)
d = y(1:2)-y0(1:2);
v = y(3:4);
val = d'*v;
isterm = 1;
dir = 1;

```

și graficul din figura 10.14(a). Se poate observa din valoarea lui  $y_{\text{final}}$  și din grafic o abatere de la periodicitate. Avem nevoie de o precizie mai bună. Cu comanda

```

orbit(1e-6)
se obține
```

```

tfinal =
2.38025846171798
yfinal =
0.99998593905520 0.00000000032239
```

Valoarea  $y_{\text{final}}$  este acum suficient de apropiată de  $y_0$ , iar graficul arată mult mai bine (figura 10.14(b)).

Vom considera acum o problemă de urmărire [32, secțiunea 12.2]. Presupunem că un iepure urmează un drum predefinit  $(r_1(t), r_2(t))$  din plan și că o vulpe urmărește iepurile astfel ca (a) în fiecare moment tangenta la drumul vulpii indică întotdeauna spre iepure și (b) viteza vulpii este de  $k$  ori viteza iepurelui. Atunci drumul  $(y_1(t), y_2(t))$  al vulpii este determinat de sistemul de ecuații diferențiale

$$\begin{aligned}\frac{dy_1}{dt} &= s(t)(r_1(t) - y_1(t)), \\ \frac{dy_2}{dt} &= s(t)(r_2(t) - y_2(t)),\end{aligned}$$

unde

$$s(t) = \frac{k\sqrt{\left(\frac{dr_1}{dt}(t)\right)^2 + \left(\frac{dr_2}{dt}(t)\right)^2}}{\sqrt{(r_1(t) - y_1(t))^2 + (r_2(t) - y_2(t))^2}}.$$

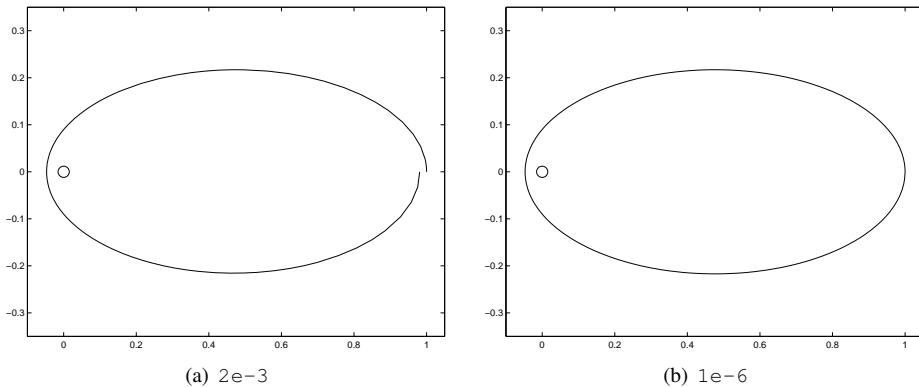


Figura 10.14: Orbitele pentru pecizia  $2e-3$  (stânga) și  $1e-6$

De notat că acest sistem pune probleme atunci când vulpea se apropie de iepure. Presupunem că iepurile urmează spirala de ecuație

$$\begin{bmatrix} r_1(t) \\ r_2(t) \end{bmatrix} = \sqrt{1+t} \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix},$$

și că vulpea pornește din poziția  $y_1(0) = 3$ ,  $y_2(0) = 0$ . Membrul drept este implementat prin funcția `fox1`:

```

function yprime = fox1(t,y,k)
%FOX1    urmarire vulpe-iepure.
%           YPRIME = FOX1(T,Y,K).

r = sqrt(1+t)*[cos(t); sin(t)];
r_p = (0.5/sqrt(1+t)) * [cos(t)-2*(1+t)*sin(t); ...
    sin(t)+2*(1+t)*cos(t)];
dist = max(norm(r-y),1e-6);
if dist > 1e-4
    factor = k*norm(r_p)/dist;
    yprime = factor*(r-y);
else
    error('Model ODE prost definit')
end

```

Dacă distanța dintre vulpe și iepure devine prea mică se apeleză `error`. Script-ul de mai jos apeleză `fox1` și produce figura 10.15. Pozițiile inițiale sunt marcate prin cercuri.

```

tspan = [0,10]; y0=[3,0]; k=0.75;
[tfox,yfox] = ode45(@foxl,tspan,y0,[],k);
plot(yfox(:,1),yfox(:,2)), hold on
plot(sqrt(1+tfox).*cos(tfox),sqrt(1+tfox).*...
      sin(tfox),'--')
plot([3,1],[0,0],'o')
axis equal, axis([-3.5,3.5,-2.5,3.1])
legend('Vulpe','Iepure',0), hold off

```

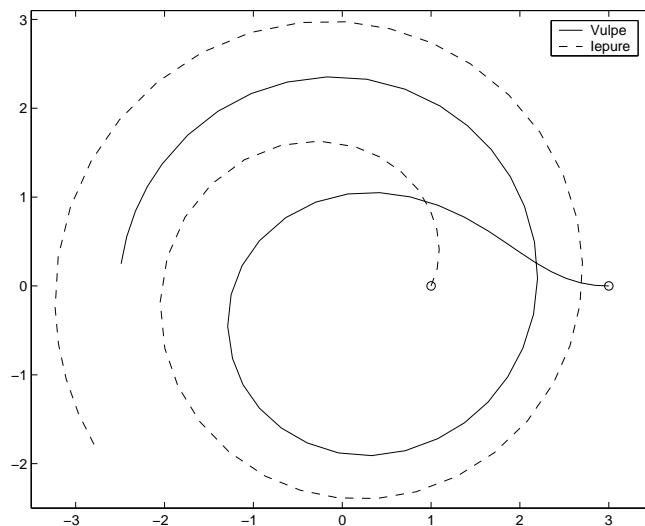


Figura 10.15: Exemplu de urmărire

Implementarea de mai sus este nesatisfăcătoare pentru  $k > 1$ , adică atunci când vulpea este mai rapidă decât iepurele. În acest caz, dacă iepurele este prins în intervalul de timp specificat, nici o soluție nu este afișată. Situația se poate evita utilizând facilitatea de localizare a evenimentelor, așa cum se poate vedea în script-ul următor. Pozițiile inițiale, drumurile celor două animale și poziția finală apar în figura 10.16.

```
tspan = [0,10]; y0=[3,0];
k=1.1;
opt = odeset('RelTol',1e-6,'AbsTol',1e-6,'Events',@events);
[tfox,yfox,te,ye,ie] = ode45(@fox2,tspan,y0,opt,k);
plot(yfox(:,1),yfox(:,2)), hold on
plot(sqrt(1+tfox).*cos(tfox),sqrt(1+tfox).*...
    sin(tfox),'--')
plot([3,1],[0,0],'o')
plot(yfox(end,1),yfox(end,2),'*')
axis equal, axis([-3.5,3.5,-2.5,3.1])
legend('Vulpe','Iepure',0), hold off
```

Aici s-a utilizat funcția `odeset` pentru a seta erorile și a specifica funcția de localizare a evenimentelor. Membrul drept și funcția de localizare a evenimentelor se dau în sursa MATLAB 10.7. Ele sunt memorate în fișiere distincte. În funcția de localizare `events` se calculează distanța dintre animale minus pragul de eroare  $1e-4$ . S-a ales `direction=-1` pentru că distanța trebuie să descrească. Parametrii de ieșire ne spun momentul (`te`), poziția (`ye`) și care componentă a evenimentului apare (`ie`). Pentru exemplul nostru se obține

```
>> te,ye,ie
te =
5.0710
```

**Sursa MATLAB 10.7 Funcțiile fox2 și events pentru problema de urmărire**

```

function yprime = fox2(t,y,k)
%FOX2    simulare urmarire vulpe iepure
%           YPRIME = FOX2(T,Y,K) .

r = sqrt(1+t)*[cos(t); sin(t)];
r_p = (0.5/sqrt(1+t)) * [cos(t)-2*(1+t)*sin(t);...
                           sin(t)+2*(1+t)*cos(t)];
dist = max(norm(r-y),1e-6);
factor = k*norm(r_p)/dist;
yprime = factor*(r-y);

function [value,isterminal,direction] = events(t,y,k)
%EVENTS    Functie eveniment pentru FOX2.
%           Localizare cand vulpea este aproape de iepure.

r = sqrt(1+t)*[cos(t); sin(t)];
value = norm(r-y) - 1e-4;      % vulpea aproape de iepure.
isterminal = 1;                % oprire integrare.
direction = -1;                % valoarea descreste catre 0.

```

```

ye =
    0.8646   -2.3073
ie =
    1

```

ceea ce ne spune că iepurele a fost capturat la momentul 5.071 în punctul de coordonate  $(0.8646, -2.3073)$ .

### 10.8.6. Ecuații implice

Rezolvitorul `ode15i`, introdus începând cu versiunea 7, rezolvă ecuații implice de forma

$$f(t, y, y') = 0$$

utilizând o metodă BDF de ordin variabil. Sintaxa minimală este

```
[T, Y] = ode15i(@odefun, tspan, y0, yp0)
```

dar sintaxa generală suportă toate facilitățile celorlalți rezolvitori. Noutatea este apariția parametrului  $yp0$  care este valoarea  $y'(t_0)$ . Trebuie îndeplinită condiția de consistență  $f(t_0, y(t_0), y'(t_0)) = 0$ . Se pot obține valori consistente cu ajutorul funcției `decic`. Să rezolvăm ecuația

$$y'^2 + y^2 - 1 = 0, \quad t \in [\pi/6, \pi/4],$$

cu condiția inițială  $y(\pi/6) = 1/2$ . Soluția exactă este  $y(t) = \sin(t)$ , iar valoarea de pornire pentru derivată  $y'(\pi/6) = \sqrt{3}/2$ . Iată codul pentru rezolvare

```
tspan = [pi/6,pi/4];
[T, Y] = ode15i(@implic,tspan,1/2,sqrt(3)/2);
```

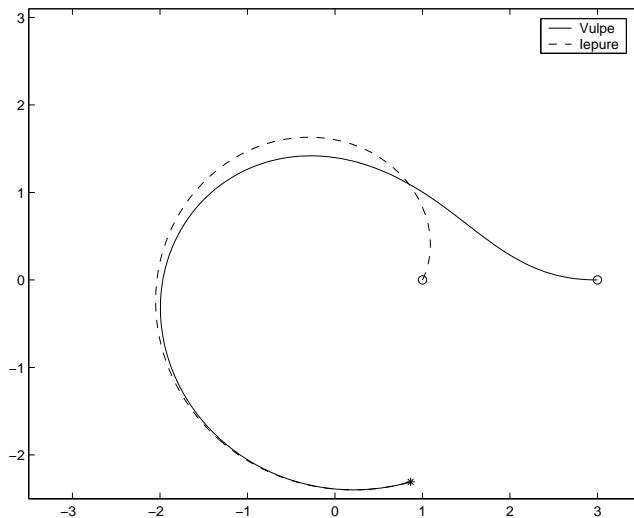


Figura 10.16: Exemplu de urmărire cu capturare

și pentru ecuația diferențială

```
function z=implic(t,y,yp)
z=yp^2+y^2-1;
```

## Probleme

**Problema 10.1.** Rezolvați problema:

$$y' = 1 - y^2, \quad y(0) = 0.$$

folosind diferite metode ale căror tabele Butcher au fost date în acest capitol, precum și `ode23` și `ode45`. Calculați eroarea globală, știind că soluția exactă este

$$y(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

și verificați că este  $O(h^p)$ .

**Problema 10.2.** Rezolvați ecuațiile și comparați cu soluțiile exacte:

(a)

$$y' = \frac{1}{4}y(1 - \frac{1}{20}y), \quad x \in [0, 20], \quad y(0) = 1;$$

cu soluția exactă

$$y(x) = \frac{20}{1 + 19e^{-x/4}};$$

(b)

$$y'' = 0.032 - 0.4(y')^2, \quad x \in [0, 20], \quad y(0) = 30, \quad y'(0) = 0;$$

cu soluția exactă

$$\begin{aligned} y(x) &= \frac{5}{2} \log \left( \cosh \left( \frac{2\sqrt{2}x}{25} \right) \right) + 30, \\ y'(x) &= \frac{\sqrt{2}}{5} \tanh \left( \frac{2\sqrt{2}x}{25} \right). \end{aligned}$$

**Problema 10.3.** Ecuația atratorului Lorenz

$$\begin{aligned} \frac{dx}{dt} &= -ax + ay, \\ \frac{dy}{dt} &= bx - y - xz, \\ \frac{dz}{dt} &= -cz + xy \end{aligned}$$

are soluții haotice care sunt sensibil dependente de condițiile inițiale. Rezolvați numeric pentru  $a = 5$ ,  $b = 15$ ,  $c = 1$  cu condițiile inițiale

$$x(0) = 2, \quad y(0) = 6, \quad z(0) = 4, \quad t \in [0, 20],$$

cu toleranța  $T = 10^{-4}$ . Repetați pentru

- (a)  $T = 10^{-5}$ ;
- (b)  $x(0) = 2.1$ .

Comparați rezultatele cu cele obținute anterior. În fiecare caz reprezentați grafic.

**Problema 10.4.** Evoluția unei epidemii de gripă într-o populație de  $N$  indivizi este modelată de sistemul de ecuații diferențiale

$$\begin{aligned} \frac{dx}{dt} &= -\beta xy + \gamma, \\ \frac{dy}{dt} &= \beta xy - \alpha y \\ \frac{dz}{dt} &= \alpha y - \gamma, \end{aligned}$$

unde  $x$  este numărul de indivizi susceptibili de a face infecția,  $y$  este numărul de infectați, iar  $z$  este numărul de imuni, care include și numărul de bolnavi refăcuți după boală, la momentul  $t$ . Parametrii  $\alpha$ ,  $\beta$ ,  $\gamma$  sunt ratele de recuperare, transmisie și respectiv recontaminare (replenishment) (pe zi). Se presupune că populația este fixă, astfel că noile nașteri sunt compensate de morți.

Utilizați funcțiile `oderk` și `ode45` pentru a rezolva ecuațiile cu condițiile inițiale  $x(0) = 980$ ,  $y(0) = 20$ ,  $z(0)=0$ , dându-se parametrii  $\alpha = 0.05$ ,  $\beta = 0.0002$ ,  $\gamma = 0$ . Simularea se va termina când  $y(t) > 0.9N$ . Determinați aproximativ numărul maxim de persoane infectate și momentul când apare.

Investigați efectul (a) variației numărului inițial de indivizi infectați asupra evoluției epidemiei și (b) introducerea unei rate de recontaminare nenule.

**Problema 10.5.** [17] Căpitanul Kirk<sup>4</sup> și echipajul său de pe nava spațială *Enterprise* au eşuat fără energie pe orbita din jurul unei planetei de tip Pământ Capella III, la o altitudine de 127 de km. Frânarea atmosferică cauzează căderea orbitei, și dacă nava atinge straturile dense ale atmosferei, frânarea excesivă și încălzirea datorată frecării va cauza daune irreparabile sistemului de menținere a vieții. Ofițerul științific, Mr. Spock, estimează că reparațiile temporare la motoarele de impuls vor necesita 29 de minute cu condiția ca ele să fie terminate înainte ca frânarea să crească la  $5g$  ( $1g = 9.81\text{ms}^{-1}$ ). Deoarece Mr. Spock este un geniu matematic, el a decis să simuleze degradarea orbitei prin rezolvarea numerică a ecuațiilor de mișcare cu perechea de metode incluse DOPRIS. Ecuațiile de mișcare ale navei, în condițiile frânării atmosferice, sunt date de

$$\begin{aligned}\frac{dv}{dt} &= \frac{GM \sin \gamma}{r^2} - c\rho v^2 \\ \frac{d\gamma}{dt} &= \left( \frac{GM}{rv} - v \right) \frac{\cos \gamma}{r} \\ \frac{dz}{dt} &= -v \sin \gamma \\ \frac{d\theta}{dt} &= \frac{v \cos \gamma}{r},\end{aligned}$$

unde

$v$  este viteza tangențială (m/s);

$\gamma$  este unghiul de reintrare (între viteza și orizontală);

$z$  este altitudinea (m);

$M$  este masa planetară ( $6 \times 10^{24}$  kg);

$G$  este constanta gravitației ( $6,67 \times 10^{-11}$  SI);

$c$  este constanta de frânare ( $c = 0.004$ );

$r$  este distanța până la centrul planetei ( $z + 6.37 \times 10^6$  m);

$\rho$  este densitatea atmosferei ( $1.3 \exp(-z/7600)$ );

$\theta$  este longitudinea;

$t$  este timpul (s).

La momentul  $t = 0$ , valorile inițiale sunt  $\gamma = 0$ ,  $\theta = 0$  și  $v = \sqrt{GM/r}$ . Mr. Spock a rezolvat ecuațiile numeric pentru a găsi istoricul decelerării și momentul și locul impactului în care căderea orbitei nu mai poate fi prevenită. Repetați simularea utilizând o metodă Runge-Kutta cu pas variabil și estimați decelerarea maximă încercată în timpul coborârii și înălțimea la care apare. Va da căpitanul Kirk ordinul de abandonare a navei?

**Problema 10.6.** Cometa Halley și-a atins ultima dată periheliul (apropierea maximă de soare) la 9 februarie 1986. Poziția și componentele vitezei în acel moment erau

$$\begin{aligned}(x, y, z) &= (0.325514, -0.459460, 0.166229) \\ \left( \frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt} \right) &= (-9.096111, -6.916686, -1.305721).\end{aligned}$$

---

<sup>4</sup>Personaje din Star Trek, seria I

Pozitia este măsurată în unități astronomice (distanța medie de la pământ la soare), iar timpul în ani. Ecuațiile mișcării sunt

$$\begin{aligned}\frac{d^2x}{dt^2} &= -\frac{\mu x}{r^3}, \\ \frac{d^2y}{dt^2} &= -\frac{\mu y}{r^3}, \\ \frac{d^2z}{dt^2} &= -\frac{\mu z}{r^3},\end{aligned}$$

unde  $r = \sqrt{x^2 + y^2 + z^2}$ ,  $\mu = 4\pi^2$ , iar perturbațiile planetare au fost neglijate. Rezolvați aceste ecuații numeric pentru a determina aproximativ momentul următorului periheliu.

**Problema 10.7.** Considerăm din nou problema orbitei (celor două corpuri) scrisă sub forma

$$\begin{aligned}y'_1 &= y_3 \\ y'_2 &= y_4 \\ y'_3 &= -y_1/r^3 \\ y'_4 &= -y_2/r^3,\end{aligned}$$

cu condițiile inițiale

$$y(0) = \left[ 1 - e, 0, 0, \sqrt{\frac{1+e}{1-e}} \right]^T,$$

unde  $r = \sqrt{y_1^2 + y_2^2}$ . Soluția reprezintă mișcare pe orbită eliptică cu excentricitatea  $e \in (0, 1)$ , cu perioada  $2\pi$ .

(a) Arătați că soluțiile pot fi scrise sub forma

$$\begin{aligned}y_1 &= \cos E - e \\ y_2 &= \sqrt{1-e^2} \sin E \\ y_3 &= \sin E / (e \cos E - 1) \\ y_4 &= \sqrt{1-e^2} \cos E / (1 + e \cos E),\end{aligned}$$

unde  $E$  este soluția ecuației lui Kepler

$$E - e \sin E = x.$$

- (b) Rezolvați problema cu `oderk` și `ode45` și reprezentați grafic soluția, variația lungimii pasului pentru  $x \in [0, 20]$  și o precizie de  $10^{-5}$ .
- (c) Calculați erorile globale, numărul de evaluări de funcții și numărul de responziuni pe pas pentru toleranțele  $10^{-4}, 10^{-5}, \dots, 10^{-11}$ .

**Problema 10.8.** [46] La jocurile olimpice de la Ciudad de Mexico din 1968 Bob Beamon a stabilit un record mondial la săritura în lungime de 8.90 metri. Acest record a depășit cu 0.80 metri recordul precedent și a rezistat până în 1991. După săritura remarcabilă a lui Beamon, mulți au sugerat că rezistența redusă a aerului la altitudinea de peste 2000 de metri a fost un factor important.

Considerăm un sistem fixat de coordonate carteziene, cu axa orizontală  $x$ , axa verticală  $y$  și originea pe pragul de bătăie. Singurele forțe care acționează după bătăie sunt gravitația și rezistența aerodinamică

$D$ , care este proporțională cu pătratul vitezei. Nu avem vânt, viteza inițială este  $v_0$  și unghiul cu axa  $Ox$  este de  $\theta_0$  radiani. Ecuările care descriu mișcarea săritorului sunt

$$\begin{aligned} x' &= v \cos \theta, & y' &= v \sin \theta, \\ \theta' &= -\frac{g}{v} \cos \theta, & v' &= -\frac{D}{m} - g \sin \theta. \end{aligned}$$

Rezistența este

$$D = \frac{c\rho s}{2} (x'^2 + y'^2).$$

Constantele pentru această problemă sunt accelerația gravitațională,  $g = 9.81 \text{ m/s}^2$ , masa  $m = 80 \text{ kg}$ , coeficientul de frânare  $c = 0.72$ , aria secțiunii săritorului,  $s = 0.50 \text{ m}^2$  și unghiul inițial  $\theta_0 = 22.5^\circ = \pi/8$  radiani. Calculați pentru patru sărituri, cu diferite valori ale vitezei inițiale  $v_0$  și ale densității aerului,  $\rho$ . Lungimea fiecărei sărituri este  $x(t_f)$ , unde timpul în aer,  $t_f$ , este determinat de condiția  $y(t_f) = 0$ .

- (a) Săritură „nominală” la altitudine mare,  $v_0 = 10 \text{ m/s}$  și  $\rho = 0.94 \text{ kg/m}^3$ .
- (b) Săritură „nominală” la nivelul mării,  $v_0 = 10 \text{ m/s}$  și  $\rho = 1.29 \text{ kg/m}^3$ .
- (c) Abordarea sprinterului la altitudine mare,  $\rho = 0.94 \text{ kg/m}^3$ . Determinați  $v_0$  astfel ca lungimea săriturii să fie egală cu recordul lui Beamon, 8.90 m.
- (d) Abordarea sprinterului la nivelul mării,  $\rho = 1.29 \text{ kg/m}^3$  și  $v_0$  valoarea determinată la punctul (c).

Prezentări rezultatele completând tabela următoare:

$v_0$	$\theta_0$	$\rho$	distanță
10	22.5	0.94	???
10	22.5	1.29	???
???	22.5	0.94	8.90
???	22.5	1.29	???

Care factor este mai important, densitatea aerului sau viteza inițială a săritorului?

**Problema 10.9.** Rezolvați problema stiff

$$\begin{aligned} y'_1 &= \frac{1}{y_1} - x^2 - \frac{2}{x^3}, \\ y'_2 &= \frac{y_1}{y_2^2} - \frac{1}{x} - \frac{1}{2x^{3/2}}, \end{aligned}$$

$x \in [1, 10]$ , cu condițiile inițiale  $y_1(1) = 1$ ,  $y_2 = 1$ , utilizând un rezolvitor nonstiff și apoi unul stiff. Sistemul are soluțiile exacte  $y_1 = 1/x^2$ ,  $y_2 = 1/\sqrt{x}$ .

**Problema 10.10.** Ecuăția lui van der Pol are forma

$$y'' - \mu (1 - y_1^2) y'_1 + y_1 = 0, \quad (10.8.3)$$

unde  $\mu > 0$  este un parametru scalar.

1. Să se rezolve ecuația în cazul când  $\mu = 1$  pe intervalul  $[0, 20]$  și condițiile inițiale  $y(0) = 2$  și  $y'(0) = 0$  (nonstiff). Să se reprezinte grafic  $y$  și  $y'$ .
2. Să se rezolve ecuația pentru  $\mu = 1000$  (stiff), intervalul de timp  $[0, 3000]$  și vectorul valorilor inițiale  $[2; 0]$ . Să se reprezinte grafic  $y$ .

**Problema 10.11.** Un dop de lungime  $L$  este pe punctul de a fi expulzat dintr-o sticlă ce conține un lichid în fermentație. Ecuațiile de mișcare a dopului sunt

$$\begin{aligned}\frac{dv}{dt} &= \begin{cases} g(1+q) \left[ \left(1 + \frac{x}{d}\right)^{-\gamma} + \frac{RT}{100} - 1 + \frac{qx}{L(1+q)} \right], & x < L; \\ 0, & x \geq L \end{cases} \\ \frac{dx}{dt} &= v,\end{aligned}$$

unde

$g$  accelerația gravitațională;

$q$  raportul frecare-greutate al dopului;

$x$  deplasarea dopului în gâtul sticlei;

$t$  timpul;

$d$  lungimea gâtului sticlei

$R$  rata procentuală cu care presiunea crește;

$\gamma$  constanta adiabatică pentru gazul din sticlă ( $\gamma = 1.4$ ).

Condițiile inițiale sunt  $x(0) = x'(0) = 0$ . Atât timp cât  $x < L$  dopul este încă în sticlă, dar el părăsește sticla când  $x = L$ . Integrați ecuațiile de mișcare cu DOPRIS (tabela 10.5) și toleranța 0.000001 pentru a găsi momentul la care dopul este aruncat. Determinați de asemenea viteza de expulzare când

$$q = 20, \quad L = 3.75\text{cm}, \quad d = 5\text{cm}, \quad R = 4.$$

**Problema 10.12.** Un model simplu al bătăilor inimii umane este dat de

$$\begin{aligned}\varepsilon x' &= -(x^3 - Ax + c), \\ c' &= x,\end{aligned}$$

unde  $x$  este deplasarea de la echilibru a fibrei musculare,  $c(t)$  este concentrația unui control chimic, iar  $\varepsilon$  și  $A$  sunt constante pozitive. Se așteaptă ca soluțiile să fie periodice. Aceasta se poate vedea reprezentând soluția în planul fazelor ( $x$  pe abscisă,  $c$  pe ordonată), trebuind să se obțină o curbă închisă. Presupunem că  $\varepsilon = 1$  și  $A = 3$ .

- (a) Calculați  $x(t)$  și  $c(t)$ , pentru  $0 \leq t \leq 12$  și valorile inițiale  $x(0) = 0.1$ ,  $c(0) = 0.1$ . Reprezentați ieșirea în planul fazelor. Cam cât este perioada?
- (b) Repetați punctul (a) cu  $x(0) = 0.87$ ,  $c(0) = 2.1$ .

**Problema 10.13.** Concepți și implementați o strategie de control al pasului pentru metoda lui Euler cu un estimator al erorii bazat pe metoda lui Heun. Testați pe două probleme din acest capitol.

**Problema 10.14.** [46] Determinați traiectoria unei ghiulele de tun sferice într-un sistem staționar de coordonate carteziene, care are o axă orizontală  $x$ , o axă verticală  $y$  și originea în punctul de lansare. Viteza inițială a proiectilului în acest sistem de coordonate are mărimea  $v_0$  și face un unghi de  $\theta_0$  radiani cu axa  $x$ . Singurele forțe care acționează asupra proiectilului sunt gravitația și rezistența aerodinamică,  $D$ , care depinde de viteza proiectilului relativă la orice vânt care ar putea fi prezent. Ecuațiile care descriu mișcarea proiectilului sunt

$$\begin{aligned}x' &= v \cos \theta, & y' &= v \sin \theta, \\ \theta' &= -\frac{g}{v} \cos \theta, & v' &= -\frac{D}{m} - g \sin \theta.\end{aligned}$$

Constantele pentru această problemă sunt: accelerări gravitațională,  $g = 9.81 \text{ m/s}^2$ , masa  $m = 15 \text{ kg}$  și viteza inițială  $v_0 = 50 \text{ m/s}$ . Se presupune că vântul este orizontal și viteza sa este o funcție dată de timp,  $w(t)$ . Rezistența aerodinamică este proporțională cu pătratul vitezei relative la vânt a proiectilului

$$D(t) = \frac{c\rho s}{2} ((x' - w(t))^2 + y'^2),$$

unde  $c = 0.2$  este coeficientul de frânare,  $\rho = 1.29 \text{ kg/m}^3$  este densitatea aerului, iar  $s = 0.25 \text{ m}^2$  este aria secțiunii transversale a proiectilului.

Considerăm patru tipuri de vânt.

- Nici un vânt:  $w(t) = 0$  pentru orice  $t$ .
- Vânt staționar din față:  $w(t) = -10 \text{ m/s}$ , pentru orice  $t$ .
- Vânt intermitent din spate:  $w(t) = 10 \text{ m/s}$ , dacă partea întreagă a lui  $t$  este pară și zero în caz contrar.
- Vijelie:  $w(t)$  este o variabilă aleatoare normală cu media zero și abaterea medie pătratică  $10 \text{ m/s}$ .

Partea întreagă a unui număr se poate calcula cu funcția MATLAB `floor`, iar o variabilă aleatoare cu media 0 și abaterea medie pătratică  $\sigma$  se poate genera cu `sigma*randn`.

Pentru fiecare din aceste tipuri de vânt realizăți următoarele calcule. Găsiți cele 17 traectorii ale căror unghiuri inițiale sunt multiplii de 5 grade, adică  $\theta_0 = k\pi/36$  radiani,  $k = \overline{1, 17}$ . Desenați toate cele 17 traectorii pe o figură. Determinați care dintre aceste traectorii are cea mai mare bătaie orizontală. Pentru acea traectorie, determinați unghiul inițial în grade, timpul de zbor, bătaia orizontală, viteza la impact și numărul de pași execuți de rezolvitor.

Care dintre cele patru tipuri de vânt necesită mai multe calcule. De ce?

**Problema 10.15.** [17] Un parașutist sare dintr-un avion ce se deplasează cu viteza  $v \text{ m/s}$  la altitudinea de  $y \text{ m}$ . După o perioadă de cădere liberă, parașuta se deschide la înălțimea  $y_p$ . Ecuatiile de mișcare ale parașutistului sunt

$$\begin{aligned} x' &= v \cos \theta, \\ y' &= v \sin \theta, \\ v' &= -D/M - g \sin \theta, \quad D = \frac{1}{2} \rho C_D A v^2, \\ \theta' &= -g \cos \theta / v, \end{aligned}$$

unde  $x$  este coordonata orizontală,  $\theta$  este unghiul de coborâre,  $D$  este rezistența aerului, iar  $A$  este aria de referință pentru forța de tracțiune, dată de

$$A = \begin{cases} \sigma, & \text{dacă } y \geq y_p; \\ S, & \text{dacă } y < y_p. \end{cases}$$

Constantele sunt

$$\begin{aligned} g &= 9.81 \text{ m/s}^2, & M &= 80 \text{ kg}, & \rho &= 1.2 \text{ kg/m}^3, \\ \sigma &= 0.5 \text{ m}^2, & S &= 30 \text{ m}^2, & C_D &= 1. \end{aligned}$$

Utilizați un rezolvitor adecvat pentru a simula coborârea parașutistului. Folosiți facilități de interpolare pentru a determina înălțimea critică  $y_p$ . Determinați momentul impactului și viteza minimă de coborâre înainte de deschiderea parașutei.



# CAPITOLUL 11

## Aproximări în mai multe variabile

Problema aproximării funcțiilor de mai multe variabile și cea a aproximării integralelor multiple sunt deosebit de importante atât din punct de vedere teoretic cât și practic, ele intervenind atât în probleme matematice abstractive cât și în modele ale diverselor fenomene sau procese din natură și societate.

Vom nota cu  $\mathbb{P}_m^n$  mulțimea polinoamelor în  $n$  variabile și de grad global cel mult  $m$ . Fie  $D \in \mathbb{R}^n$ ,  $f : D \rightarrow \mathbb{R}$ . Fie, de asemenea,  $F_1 f, F_2 f, \dots, F_p f$  informații despre  $f$  (de regulă valori ale funcției sau ale unor derivate parțiale ale acestora). Se pune problema ca pe baza informațiilor  $F_k$ ,  $k = \overline{1, p}$ , să se determine o funcție  $F$ , astfel încât  $f \approx F$ , într-un sens precizat, în domeniul  $D$ .

De exemplu, dacă  $P_i \in D$ ,  $i = \overline{1, s}$ , iar  $F_i f = f(P_i)$ ,  $i = \overline{1, s}$  se ajunge la o problemă de interpolare polinomială.

Dacă  $F_i f = f(P_i)$ ,  $i = \overline{1, s}$ , iar  $f \in \mathbb{P}_m^n$  se determină astfel încât

$$\sum_{i=1}^s [F(p_i) - f(P_i)]^2 \rightarrow \min$$

se obține o problemă de aproximare discretă în sensul celor mai mici pătrate.

### 11.1. Aproximarea funcțiilor de mai multe variabile pe un domeniu rectangular

Ne vom ocupa în continuare de aproximarea funcțiilor definite pe un domeniu rectangular și de extinderea unor procedee unidimensionale. Expunerea urmează liniile directoare din [12].

Fie  $D = \prod_{k=1}^n [a_k, b_k] \subset \mathbb{R}^n$  și  $\mathcal{F}_n$  o mulțime de funcții de  $n$  variabile independente definite pe  $D$ . Notăm prin  $P_i : \mathcal{F}_n \rightarrow \mathcal{G}_i$ ,  $i = \overline{1, n}$   $n$  proiectori ce acționează asupra funcției  $f \in \mathcal{F}_n$ , fiecare în raport cu variabila  $x_i$ , iar prin  $R_i = I - P_i$ , unde  $I$  este operatorul identic, operatorii rest corespunzători. Operatorul  $R_i : \mathcal{F}_n \rightarrow \mathcal{H}_i$  este de asemenea proiector.

Cu ajutorul operației de *produs tensorial* (computere) se pot construi operatorii  $P_i P_j$ ,  $i, j = \overline{1, n}$ , care, în caz că oricare doi comută sunt de asemenea proiectori, deoarece  $(P_i P_j)^2 = P_i P_j$ . Deoarece

suma a doi projectorii  $P_i + P_j$  nu este projector ( $(P_i + P_j)^2 \neq P_i + P_j$ ), se lucrează cu *suma booleană*  $P_i \oplus P_j = P_i + P_j - P_i P_j$ , care este projector.

Datorită proprietății de asociativitate a produsului și a sumei booleene, aceste operații se pot extinde la trei sau mai mulți operatori. Putem defini projectorii

$$P := P_1 P_2 \dots P_n,$$

$$S := P_1 \oplus P_2 \oplus \dots \oplus P_n = \sum_{i=1}^n \tilde{P}_i,$$

cu

$$\tilde{P}_1 = P_1, \quad \tilde{P}_i = P_i - \left( \sum_{k=1}^{i-1} \tilde{P}_k \right) P_i, \quad i = \overline{2, n},$$

adică produsul și suma booleană a celor  $n$  projectorii considerați. De exemplu, pentru trei projectorii avem

$$P_1 P_2 P_3 = P_1 (P_2 P_3),$$

$$P_1 \oplus P_2 \oplus P_3 = P_1 + P_2 + P_3 - P_1 P_2 - P_1 P_3 - P_2 P_3 + P_1 P_2 P_3.$$

Fie  $\mathcal{P}$  mulțimea formată din projectorii  $P_1, \dots, P_n$  și toți projectorii obținuți din aceștia cu ajutorul operațiilor de produs și sumă booleană.

Imaginea  $Qf$  a unei funcții  $f \in \mathcal{F}_n$  prin orice element  $Q \in \mathcal{P}$  poate fi considerată o aproximantă a lui  $f$ . De exemplu, dacă  $P_i$  este operatorul de interpolare Lagrange  $L_{m_i}^{x_i}$  relativ la nodurile  $x_{i_0}, \dots, x_{i_{m_i}}$  ce acționează asupra variabilei  $x_i$  atunci

$$(L_{m_i}^{x_i} f) = \sum_{k=0}^{m_i} \ell_k(x_i) f(x_1, \dots, x_{i-1}, x_{i_k}, x_{i+1}, \dots, x_{i_{m_i}}),$$

unde

$$\ell_k(x_i) = \frac{(x_i - x_{i_0}) \dots (x_i - x_{i_{k-1}})(x_i - x_{i_{k+1}}) \dots (x_i - x_{i_{m_i}})}{(x_{i_k} - x_{i_0}) \dots (x_{i_k} - x_{i_{k-1}})(x_{i_k} - x_{i_{k+1}}) \dots (x_{i_k} - x_{i_{m_i}})},$$

este o aproximantă a funcției  $f$ . Mai mult, putem scrie una din expresiile termenului rest din această aproximare. Astfel, utilizând forma cu diferențe divizate pentru rest, avem

$$(R_{m_i}^{x_i} f) = u(x_i)[x_i, x_{i_0}, \dots, x_{i_{m_i}}, f(x_1, \dots, x_{i-1}, \cdot, x_{i+1}, \dots, x_{i_{m_i}})].$$

Operatorul de diferență divizată acționează asupra funcției  $f$  în raport cu variabila  $x_i$ , iar  $u(x_i) = (x_i - x_{i_0}) \dots (x_i - x_{i_{m_i}})$ .

Pentru a compara aproximantele generate de elementele lui  $\mathcal{P}$  se definește pe  $\mathcal{P}$  o relație de ordine parțială.

**Definiția 11.1.1.** Pentru  $X, Y \in \mathcal{P}$ ,  $X \leq Y \Leftrightarrow XY = X$ .

Dacă produsul a oricare doi projectorii este comutativ, adică  $P_i P_j = P_j P_i$ ,  $i \neq j$ , atunci  $(\mathcal{P}, \leq)$  este o latică distributivă. Din relațiile

$$P = P_1 \dots P_n \leq X, \quad X \in \mathcal{P}$$

și

$$Y \leq S = P_1 \oplus \cdots \oplus P_n, \quad Y \in \mathcal{P},$$

rezultă că  $P$  este element minimal, iar  $S$  element maximal al laticei  $(\mathcal{P}, \leq)$ .

**Definiția 11.1.2.** Fie  $f \in \mathcal{F}_n$ . Aproximanta  $Pf$  a funcției  $f$  se numește algebric minimală, iar aproximanta  $Sf$  se numește algebric maximală.

Pentru exemplificare, fie  $n = 2$  și  $P_1 = L_r^x$ ,  $P_2 = L_s^y$  operatorii de interpolare Lagrange unidimensionali relativ la nodurile  $x_0, \dots, x_r$ , respectiv  $y_0, \dots, y_s$ . Dacă  $f \in \mathcal{F}_2$ , atunci aproximanta algebric minimală este

$$(L_r^x L_s^y f)(x, y) = \sum_{i=0}^r \sum_{j=0}^s \ell_i(x) \tilde{\ell}_j(y) f(x_i, y_j), \quad (11.1.1)$$

iar cea algebric maximală este

$$\begin{aligned} (L_r^x \oplus L_s^y f)(x, y) &= \sum_{i=0}^r \ell_i(x) f(x_i, y) + \sum_{j=0}^s \tilde{\ell}_j(y) f(x, y_j) \\ &\quad - \sum_{i=0}^r \sum_{j=0}^s \ell_i(x) \tilde{\ell}_j(y) f(x_i, y_j), \end{aligned} \quad (11.1.2)$$

unde  $\ell_i(x)$  și  $\tilde{\ell}_j(y)$  sunt polinoamele fundamentale Lagrange corespunzătoare.

Prin verificare directă pentru produs se obține

$$(L_r^x L_s^y f)(x_i, y_j) = f(x_i, y_j), \quad i = \overline{0, r}, \quad j = \overline{0, s},$$

iar pentru suma booleană

$$\begin{aligned} (L_r^x \oplus L_s^y f)(x_i, y) &= f(x_i, y), & i = \overline{0, r}, \quad y \in [a_2, b_2], \\ (L_r^x \oplus L_s^y f)(x, y_j) &= f(x, y_j), & j = \overline{0, s}, \quad x \in [a_1, b_1]. \end{aligned}$$

Să considerăm grila  $\{(x_i, y_j) : i = \overline{0, r}, j = \overline{0, s}\}$ . Interpolarea algebric minimală este o interpolare discretă (punctuală); ea reproduce valorile funcției numai pe noduri. Interpolarea algebric maximală este o *interpolare transfiniță* sau *interpolare blending*; funcția interpolatoare reproduce valorile lui  $f$  pe segmentele  $\{(x_i, y) \in \mathbb{R}^2 : a_2 \leq y \leq b_2\}$ ,  $i = \overline{0, r}$  și  $\{(x, y_j) \in \mathbb{R}^2 : a_1 \leq x \leq b_1\}$ ,  $j = \overline{0, s}$  (figura 11.1).

Revenind la operatorii rest  $R_i = I - P_i$ ,  $i = \overline{1, n}$  se observă că mulțimea formată din  $R_i$  și operatorii obținuți din aceștia prin produs tensorial și sumă booleană, formează și ei în raport cu relația „ $\leq$ ” (dată în definiția 11.1.1) tot o latice (dacă produsul este comutativ). În această latice  $R_1 R_2 \dots R_n$  este element minimal, iar  $R_1 \oplus R_2 \oplus \dots \oplus R_n$  este element maximal. Avem de asemenea următoarele descompuneri pentru operatorul identic

$$\begin{aligned} I &= P_1 \dots P_n + R_1 \oplus \dots \oplus R_n \\ I &= P_1 \oplus \dots \oplus P_n + R_1 \dots R_n, \end{aligned} \quad (11.1.3)$$

numite *descompunere minimală* și respectiv *maximală*. Dacă  $f \in \mathcal{F}_n$ , pe baza descompunerilor anterioare avem formulele de aproximare

$$f = P_1 \dots P_n f + R_1 \oplus \dots \oplus R_n f \quad (11.1.4)$$

$$f = P_1 \oplus \dots \oplus P_n f + R_1 \dots R_n f, \quad (11.1.5)$$

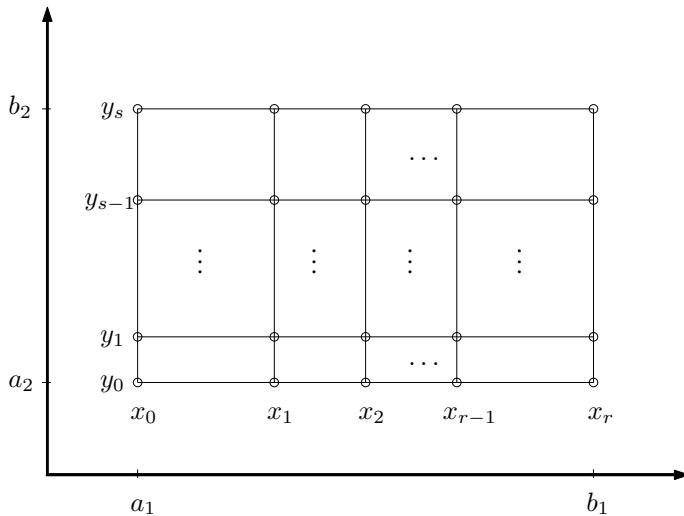


Figura 11.1: Interpretarea geometrică a interpolanților produs tensorial și sumă booleană

numite *formulă de aproximare algebric minimală* și respectiv *formulă de aproximare algebric maximală*.

Pentru exemplificare să considerăm din nou cazul bidimensional și operatorii de interpolare Lagrange  $L_r^x$  și  $L_s^y$ . Operatorii rest corespunzători vor fi  $R_r^x = I - L_r^x$  și  $R_s^y = I - L_s^y$ . Pentru formula algebric minimală se va obține

$$f = L_r^x L_s^y + R_r^x \oplus R_s^y, \quad (11.1.6)$$

iar pentru cea algebric maximală

$$f = L_r^x \oplus L_s^y + R_r^x R_s^y. \quad (11.1.7)$$

Cunoscând expresiile termenului rest al formulei de interpolare Lagrange și presupunând că există toate derivatele parțiale ale lui  $f$  necesare și că acestea satisfac condițiile de continuitate cerute, obținem

$$\begin{aligned} (R_r^x \oplus R_s^y f)(x, y) &= \frac{u_r(x)}{(r+1)!} \frac{\partial^{r+1}}{\partial x^{r+1}} f(\xi_1, y) + \frac{u_s(y)}{(s+1)!} \frac{\partial^{s+1}}{\partial y^{s+1}} f(x, \eta_1) \\ &\quad - \frac{u_r(x)u_s(y)}{(r+1)!(s+1)!} \frac{\partial^{r+s+2}}{\partial x^{r+1}\partial y^{s+1}} f(\xi_2, \eta_2), \end{aligned}$$

cu  $\xi_1, \xi_2 \in [\alpha_1, \alpha_2]$ ,  $\eta_1, \eta_2 \in [\beta_1, \beta_2]$ ,

$$(R_r^x R_s^y f)(x, y) = \frac{u_r(x)u_s(y)}{(r+1)!(s+1)!} \frac{\partial^{r+s+2}}{\partial x^{r+1}\partial y^{s+1}} f(\xi_3, \eta_3),$$

cu  $\xi_3 \in [\alpha_1, \alpha_2]$ ,  $\eta_3 \in [\beta_1, \beta_2]$ ,  $\alpha_1 = \min\{x, x_0, \dots, x_r\}$ ,  $\alpha_2 = \max\{x, x_0, \dots, x_r\}$ ,  $\beta_1 = \min\{y, y_0, \dots, y_s\}$ ,  $\beta_2 = \max\{y, y_0, \dots, y_s\}$ . De notat că termenul rest se poate exprima și sub formă integrală și cu ajutorul diferențelor divizate.

**Exemplul 11.1.3.** Dacă luăm  $r = s = 1$  și nodurile 0 și 1 după fiecare coordonată, operatorii Lagrange unidimensionali sunt:

$$\begin{aligned} L_1^x(x, y) &= (1-x)f(0, y) + xf(1, y) \\ L_1^y(x, y) &= (1-y)f(x, 0) + yf(x, 1), \end{aligned}$$

iar pentru operatorii produs tensorial și sumă booleană se obțin expresiile

$$(L_1^x L_1^y)(x, y) = (1-x)(1-y)f(0, 0) + (1-x)yf(0, 1) + x(1-y)f(1, 0) + xyf(1, 1)$$

și respectiv

$$\begin{aligned} (L_1^x \oplus L_1^y)(x, y) &= (1-x)f(0, y) + xf(1, y) + (1-y)f(x, 0) + yf(x, 1) - \\ &\quad (1-x)(1-y)f(0, 0) - (1-x)yf(0, 1) - x(1-y)f(1, 0) + xyf(1, 1). \end{aligned}$$

Prima aproximantă reproduce valorile funcției doar în vîrfurile pătratului  $[0, 1] \times [0, 1]$ , pe când cea de-a doua reproduce valorile funcției pe frontiera pătratului. Dacă există derivatele parțiale până la ordinea necesare, expresiile corespunzătoare pentru rest sunt

$$\begin{aligned} (R_P f)(x, y) &= (R_1^x \oplus R_1^y)(x, y) = \frac{x(x-1)}{2} \frac{\partial^2}{\partial x^2} f(\xi_1, y) + \\ &\quad \frac{y(y-1)}{2} \frac{\partial^2}{\partial y^2} f(x, \eta_1) - \frac{x(x-1)}{2} \frac{y(y-1)}{2} \frac{\partial^4}{\partial x^2 \partial y^2} f(\xi_2, \eta_2). \end{aligned}$$

și respectiv

$$(R_S f)(x, y) = (R_1^x R_1^y)(x, y) = \frac{x(x-1)}{2} \frac{y(y-1)}{2} \frac{\partial^4}{\partial x^2 \partial y^2} f(\xi, \eta),$$

unde  $\xi, \eta, \xi_1, \eta_1, \xi_2, \eta_2 \in (0, 1)$ . Dacă  $f$  este de clasă  $C^{2,2}$ , au loc delimitările

$$\begin{aligned} \|R_P f\|_\infty &\leq \frac{1}{8} \left[ \left\| \frac{\partial^2}{\partial x^2} f \right\|_\infty + \left\| \frac{\partial^2}{\partial y^2} f \right\|_\infty + \frac{1}{8} \left\| \frac{\partial^4}{\partial x^2 \partial y^2} f \right\|_\infty \right], \\ \|R_S f\|_\infty &\leq \frac{1}{64} \left\| \frac{\partial^4}{\partial x^2 \partial y^2} f \right\|_\infty. \end{aligned} \quad \diamond$$

Implementările MATLAB pentru interpolanții Lagrange de tip produs tensorial și sumă booleană sunt date în sursele 11.1 și respectiv 11.2.

**Exemplul 11.1.4.** Să considerăm funcția  $f : [-2, 2] \times [-2, 2] \rightarrow \mathbb{R}$ ,  $f(x, y) = xe^{-x^2-y^2}$ . Graficul ei, graficele aproximantelor algebric minimală, respectiv algebric maximală de tip Lagrange și ale resturilor corespunzătoare apar în figura 11.2. S-au considerat cinci noduri echidistante după fiecare coordonată,  $x_k, y_k = -2 + k$ ,  $k = \overline{0, 4}$ .

În concluzie, formula de aproximare generată de produsul operatorilor  $P_1, \dots, P_n$  este algebric minimală, iar formula generată de suma booleană a acestor operatori este algebric maximală, proprietăți datorate calității operatorilor produs și sumă booleană de a fi element minimal, respectiv maximal în laticea  $(\mathcal{P}, \leq)$ .

Se va da în continuare și o altă interpretare a celor două formule de aproximare extremală, având în vedere ordinul de aproximare al operatorilor unidimensionali  $P_i$ ,  $i = \overline{1, n}$  de la care se pornește. Această interpretare va permite extinderea procedeelor de mai sus și la alți operatori care nu sunt proiecitori.

**Sursa MATLAB 11.1** Interpolant bidimensional produs tensorial de tip Lagrange

---

```
function Z=prodtens(u,v,x,y,f)
%PRODTENS - interpolant Lagrange produs tensorial
%apel [Z,X,Y]=prodtens(u,v,x,y,f)
%u - abscise pt evaluare
%v - ordonate pt evaluare
%x - abscise noduri
%y - ordonate noduri
%f - functia
[X,Y]=meshgrid(x,y);
F=f(X,Y);
lu=pfl2b(x,u)';
lv=pfl2b(y,v)';
Z=lu*F*lv;
```

---

**Sursa MATLAB 11.2** Interpolant bidimensional sumă booleană de tip Lagrange

---

```
function Z=sumbool(u,v,x,y,f)
%SUMBOOL - interpolant Lagrange suma booleana
%apel [Z,X,Y]=sumbool(u,v,x,y,f)
%u - abscise pt evaluare
%v - ordonate pt evaluare
%x - abscise noduri
%y - ordonate noduri
%f - functia
[X,Y]=meshgrid(x,y);
F=f(X,Y);
[X1,V1]=meshgrid(x,v);
F1=f(X1,V1);
[U2,Y2]=meshgrid(u,y);
F2=f(U2,Y2);
lu=pfl2b(x,u);
lv=pfl2b(y,v);
Z=F1*lu+lv'*F2-lu'*F*lv;
```

---

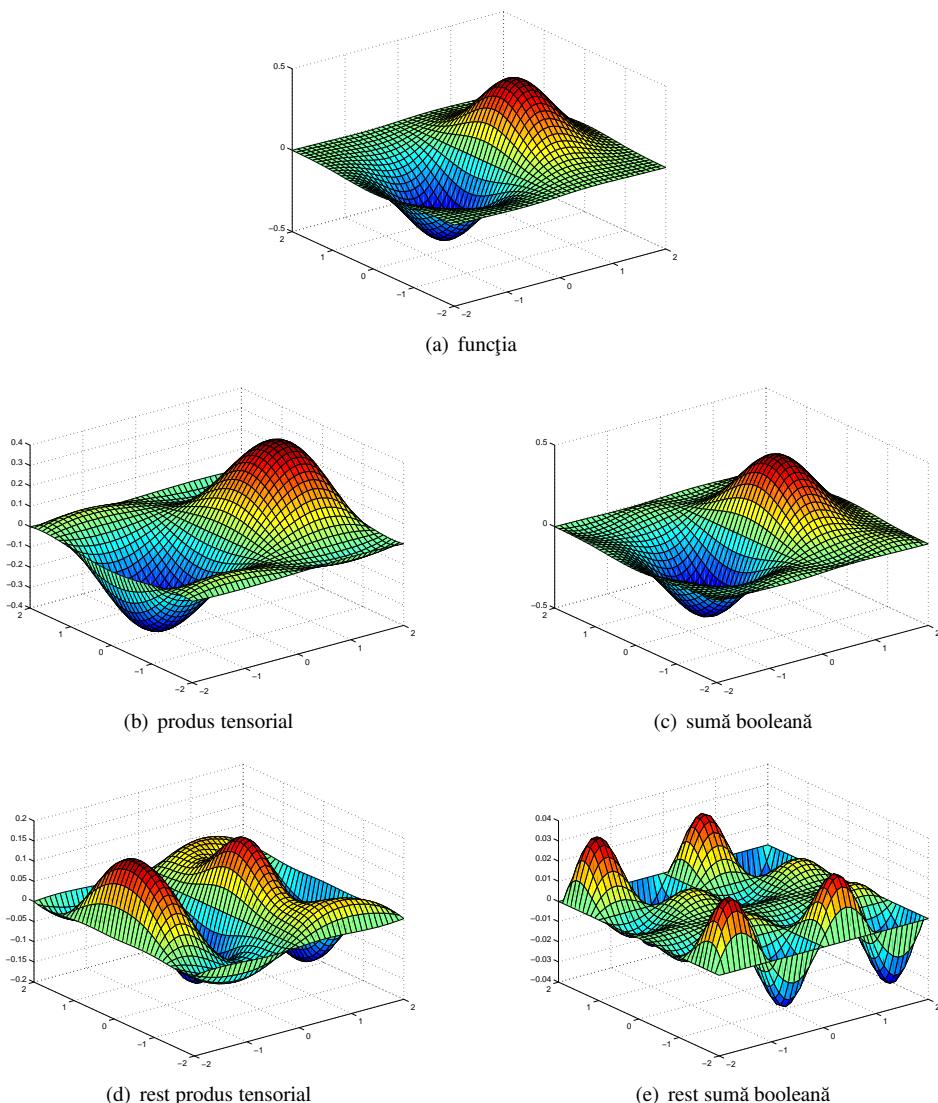


Figura 11.2: Graficul funcției  $f : [-2, 2] \times [-2, 2] \rightarrow \mathbb{R}$ ,  $f(x, y) = xe^{-x^2-y^2}$  din exemplul 11.1.4 (figura 11.2(a)), aproximarea algebrică minimală (figura 11.2(b)), cea maximală (figura 11.2(c)) și resturile corespunzătoare (figura 11.2(d) și respectiv 11.2(e))

Dacă notăm prin  $m_i$  ordinul de aproximare al operatorului  $P_i$ ,  $i = \overline{1, n}$ , atunci din (11.1.4) rezultă că ordinul de aproximare al operatorului produs tensorial este

$$\text{ord}(P) = \min\{\text{ord}(P_1), \dots, \text{ord}(m_n)\}, \quad (11.1.8)$$

iar din (11.1.5) rezultă că ordinul de aproximare al operatorului sumă booleană este

$$\text{ord}(S) = \text{ord}(P_1) + \dots + \text{ord}(P_n). \quad (11.1.9)$$

Pentru două variabile formula de aproximare algebric minimală (11.1.6) are ordinul de aproximare  $\min(r, s) + 1$ , în timp ce ordinul formulei de interpolare algebric maximale (11.1.7) este  $r + s + 2$ .

Prin urmare, proprietățile de extremalitate ale celor doi operatori sunt caracterizate și prin ordinul de aproximare, operatorul produs având ordinul de aproximare minim, iar operatorul sumă booleană având ordinul de aproximare maxim în mulțimea  $\mathcal{P}$  a tuturor operatorilor generați din  $P_1, \dots, P_n$  cu ajutorul celor două operații.

Deși operatorul sumă booleană are un ordin mare de aproximare, în expresia sa

$$Sf = (P_1 + \dots + P_n - P_1 P_2 - \dots - P_{n-1} P_n + \dots + P_1 P_2 \dots P_n)f, \quad (11.1.10)$$

vor interveni termeni care sunt funcții de  $n-1, n-2, \dots, 1$  variabile. O astfel de formulă este aplicabilă doar atunci când se cunosc valorile funcției  $f$  sau ale unor derivele parțiale ale ei pe hiperfețele lui  $D$  sau pe secțiuni ale domeniului  $D$  paralele cu aceste hiperfețe. Acesta este un dezavantaj major al aproximării blending. El poate fi înălțat aplicând funcției  $f$  pe hipersuprafețele respective ale lui  $D$  alți operatori liniari de aproximare. Continuând acest prodeu din aproape în aproape se ajunge după cel mult  $n-1$  etape la o aproximare scalară (numericală), adică la o aproximare ce conține numai valori ale lui  $f$  pe puncte ale lui  $D$ . Evident că restul unei astfel de formule de aproximare, obținută din formula blending conține mai mulți termeni. Pentru ilustrare, să considerăm cazul unei funcții de două variabile, definită pe  $[a_1, b_1] \times [a_2, b_2]$ . Considerând operatorii  $P_1^1, P_2^1$ , în care indicele superior indică numărul nivelului (etapei) de aproximare, obținem la prima etapă

$$f = (P_1^1 + P_2^1 - P_1^1 P_2^1)f + R_1^1 R_2^1 f, \quad (11.1.11)$$

unde  $P_1^1 f$  conține pe  $x_2$  ca variabilă liberă, iar  $P_2^1$  pe  $x_1$ . Aplicând pe al doilea nivel lui  $P_1 f$  operatorul  $P_2^2$ , iar lui  $P_2^1 f$  operatorul  $P_1^2$ , obținem formula de aproximare scalară:

$$f = (P_1^1 P_2^2 + P_1^2 P_2^1 - P_1^1 P_2^1)f + (P_2^1 R_1^2 + P_1^1 R_2^2 + R_1^1 R_2^1)f. \quad (11.1.12)$$

Restul acestei formule conține trei termeni. Singura proprietate a tuturor operatorilor folosiți este comutativitatea produsului. Avem mai multe posibilități de alegere a operatorilor de pe al doilea nivel. Ei depind în primul rând de informațiile disponibile asupra lui  $f$ . O cale naturală de alegere este aceea ca ordinul de aproximare al formulei originale (11.1.11) să fie păstrat. O astfel de formulă de aproximare se va numi *formulă consistentă*. De exemplu, alegând în (11.1.12) pe post de  $P_1^1$  și  $P_2^1$  operatorii de interpolare Lagrange  $L_1^x$  și  $L_2^y$  relativi la nodurile  $a_1, b_1$  și  $a_2, b_2$ , iar în locul lui  $P_1^2$  și  $P_2^2$  operatorii de interpolare Hermite  $H_3^x$  și  $H_4^y$  relativi la nodurile duble  $a_1, b_1$  și respectiv  $a_2$  triplu și  $b_2$  dublu se obține

$$f = (L_1^x H_4^y + H_3^x L_1^y - L_1^x L_1^y)f + (L_1^y R_3^x + L_1^x R_4^y + R_1^x R_1^y)f. \quad (11.1.13)$$

Cum ordinea de aproximare ale operatorilor  $L_1, H_3$  și  $H_4$  sunt 2, 4 și 5, rezultă că ordinul formulei de aproximare blending

$$f = L_1^x \oplus L_2^y + R_1^x R_2^y f \quad (11.1.14)$$

este 4, iar ordinul de aproximare al lui (11.1.13) este tot 4, căci

$$\begin{aligned} \|(L_1^y R_3^x + L_3^x R_4^y + R_1^x R_1^y)f\| &\leq \|R_3^x f\| + \|R_4^y f\| + \|R_1^x R_1^y f\| = \\ &= c_1(b_1 - a_1)^4 + c_2(b_2 - a_2)^5 + c_3(b_1 - a_1)^2(b_2 - a_2)^2 \\ &= (c_1 + c_2 h + c_3)h^4, \quad \text{unde } h = b_1 - a_1 = b_2 - a_2. \end{aligned}$$

Deci (11.1.13) este consistentă.

Deoarece ordinul de aproximare al formulei inițiale nu poate fi mărit, este preferabil ca operatorii folosiți în următoarele nivele de aproximare să fie astfel aleși încât termenii din expresia restului formulei finale să aibă același ordin de mărime. Formula numerică astfel obținută se va numi *omogenă*.

Formula (11.1.13) nu este omogenă deoarece  $\|R_3^x f\| = O(h^4)$ ,  $\|R_1^x R_1^y f\| = O(h^4)$ , dar  $\|R_4^y f\| = O(h^5)$ . Pentru a deveni omogenă trebuie ca operatorul  $H_4^y$  folosit pe al doilea nivel să fie înlocuit prin un operator cu ordinul de aproximare 4, de exemplu  $H_3^y$ . Astfel, din formula originală (11.1.14), am dedus următoarea formulă numerică omogenă

$$f = (L_1^x H_3^y + H_3^x L_1^y - L_1^x L_1^y)f + (L_1^y R_3^x + L_1^x R_3^y + R_1^x R_1^y)f.$$

## 11.2. Integrarea numerică a funcțiilor de mai multe variabile

Fie  $D \subseteq \mathbb{R}^n$ , funcția  $f : D \rightarrow \mathbb{R}$ , punctele  $P_i \in D$ ,  $i = \overline{0, m}$  și  $w$  o funcție pondere nenegativă, definită pe  $D$ .

**Definiția 11.2.1.** *Formula*

$$\int \cdots \int_D w(x_1, \dots, x_n) f(x_1, \dots, x_n) dx_1 \dots dx_n = \sum_{i=0}^m A_i f(P_i) + R_m f \quad (11.2.1)$$

se numește formulă de integrare numerică a funcției  $f$  sau formulă de cubatură. Parametrii  $A_i$  se numesc coeficienții formulei, punctele  $P_i$  se numesc nodurile ei, iar  $R_m$  termenul rest.

Problema construirii unei formule de cubatură constă în determinarea coeficienților și a nodurilor ei, folosind condiții corespunzătoare. În funcție de condițiile folosite, formulele de cubatură pot fi clasificate în deterministe și nedeterministe, iar cele deterministe în formule de tip interpolator, de tip Gauss, optimale, etc.

O metodă eficientă de construire a formulelor de cubatură în cazul în care  $D$  este un domeniu rectangular, constă în exprimarea parametrilor acesteia cu ajutorul coeficienților, respectiv a nodurilor unei formule de cuadratură unidimensionale. Pentru simplificare ne vom limita la cazul bidimensional. Fie  $D = [a, b] \times [c, d]$ ,  $\Delta_x$  diviziunea  $a = x_0 < x_1 < \dots < x_m = b$ ,  $\Delta_y$  diviziunea  $c = y_0 < y_1 < \dots < y_n = d$ ,  $w(x, y) = 1$ ,  $\forall (x, y) \in D$ .

În acest caz formula (11.2.1) devine

$$\int_a^b \int_c^d f(x, y) dx dy = \sum_{i=0}^m \sum_{j=0}^n A_{ij} f(x_i, y_j) + R_{m,n}(f). \quad (11.2.2)$$

Se poate obține o astfel de formulă pornind de la formula de interpolare Lagrange (bidimensională)

$$f = L_m^x L_n^y f + R_m^x \oplus R_n^y f.$$

Dacă  $f \in C^{m+1,n+1}(D)$ , prin integrare termen cu termen se obține

$$\int_a^b \int_c^d f(x, y) dx dy = \sum_{i=0}^m \sum_{j=0}^n A_i B_j f(x_i, y_j) + R_{mn}(f) \quad (11.2.3)$$

unde

$$A_i = \int_a^b \ell_i(x) dx, \quad B_j = \int_c^d \tilde{\ell}_j(y) dy,$$

iar

$$\begin{aligned} R_{m,n}(f) &= \int_a^b \int_c^d (R_m^x \oplus R_n^y) f(x, y) dx dy \\ &= \frac{1}{(m+1)!} \int_a^b \int_c^d u_m(x) f^{(m+1,0)}(\xi_x, y) dx dy \\ &\quad + \frac{1}{(n+1)!} \int_a^b \int_c^d u_n(y) f^{(0,n+1)}(x, \eta_y) dx dy \\ &\quad - \frac{1}{(m+1)!} \frac{1}{(n+1)!} \int_a^b \int_c^d u_m(x) u_n(y) f^{(m+1,n+1)}(\tilde{\xi}_x, \tilde{\eta}_y) dx dy. \end{aligned}$$

Dacă  $\Delta_x$  și  $\Delta_y$  sunt diviziori uniforme ale intervalului  $[a, b]$  și respectiv  $[c, d]$ , atunci formula de cubatură (11.2.3) se numește de tip *Newton-Cotes*.

**Cazuri particulare.** Pentru  $m = n = 1$  se obține formula de cubatură a trapezului.

$$\begin{aligned} \int_a^b \int_c^d f(x, y) dx dy &= \frac{(b-a)(d-c)}{4} [f(a, c) + f(a, d) + f(b, c) + f(b, d)] \\ &\quad + R_{11}(f), \end{aligned}$$

unde

$$\begin{aligned} R_{11}(f) &= -\frac{(b-a)^3(d-c)}{12} f^{(2,0)}(\xi_1, \eta_1) - \frac{(b-a)(d-c)^3}{12} f^{(0,2)}(\xi_2, \eta_2) \\ &\quad - \frac{(b-a)^3(d-c)^3}{144} f^{(2,2)}(\xi_3, \eta_3). \end{aligned}$$

Pentru  $m = n = 2$  se obține formula de cubatură a lui Simpson.

$$\begin{aligned} \int_a^b \int_c^d f(x, y) dx dy &= \frac{(b-a)(d-c)}{36} \left\{ f(a, c) + f(a, d) + f(b, c) + f(b, d) \right. \\ &\quad + 4 \left[ f\left(\frac{a+b}{2}, c\right) + f\left(\frac{a+b}{2}, d\right) + f\left(a, \frac{c+d}{2}\right) + f\left(b, \frac{c+d}{2}\right) \right] \\ &\quad \left. + 16f\left(\frac{a+b}{2}, \frac{b+c}{2}\right) \right\} + R_{22}(f), \end{aligned}$$

unde

$$\begin{aligned} R_{22}(f) &= -\frac{(b-a)^5(d-c)}{2880} f^{(4,0)}(\xi_1, \eta_1) - \frac{(b-a)(d-c)^5}{2880} f^{(0,4)}(\xi_2, \eta_2) \\ &\quad - \frac{(b-a)^5(d-c)^5}{2880^2} f^{(4,4)}(\xi_3, \eta_3). \end{aligned}$$

Partiționând intervalele  $[a, b]$  și  $[c, d]$  se pot obține formule de cubatură repede. Vom ilustra pentru formula lui Simpson. Să presupunem că  $[a, b]$  este împărțit în  $m$  părți egale, iar  $[c, d]$  în  $n$  părți egale, obținându-se o diviziune cu  $mn$  dreptunghiuri. Vom împărți fiecare dreptunghi în patru părți egale ca în figura 11.3 (vârfurile dreptunghiului sunt indicate prin cercuri negre, iar punctele intermediare cu cercuri mai albe).

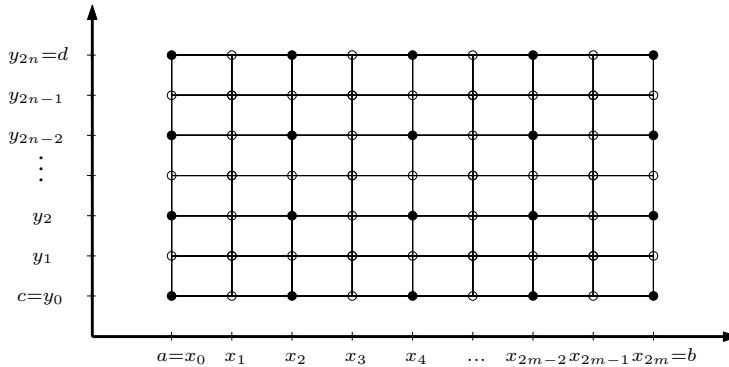


Figura 11.3: Diviziunea pentru formula repetată a lui Simpson

Fie

$$h = \frac{b-a}{2m}, \quad k = \frac{d-c}{2n}.$$

Punctele diviziunii vor avea coordonatele

$$\begin{aligned} x_i &= x_0 + ih, & x_0 &= a, & i &= \overline{0, 2m} \\ y_j &= y_0 + jk, & y_0 &= b, & j &= \overline{0, 2n}. \end{aligned}$$

Introducem notația  $f(x_i, y_j) = f_{ij}$ . Aplicând formula lui Simpson fiecarui dreptunghi al diviziunii avem

$$\begin{aligned} \int_a^b \int_c^d f(x, y) dx dy &= \frac{hk}{9} \sum_{i=0}^m \sum_{j=0}^n [f_{2i, 2j} + f_{2i+2, j} + f_{2i+2, 2j+2} \\ &\quad + f_{2i, 2j+2} + 4(f_{2i+1, 2j} + f_{2i+2, 2j+1} + f_{2i+1, 2j+2} + f_{2i, 2j+1}) \\ &\quad + 16f_{2i+1, 2j+1}] + R_{m,n}(f). \end{aligned}$$

Reducând termenii asemenea se obține

$$\int_a^b \int_c^d f(x, y) dx dy = \frac{hk}{9} \sum_{i=0}^m \sum_{j=0}^n \lambda_{ij} f_{ij} + R_{m,n}(f),$$

unde  $\lambda_{ij}$  sunt date de matricea

$$\Lambda = \begin{bmatrix} 1 & 4 & 2 & 4 & 2 & \dots & 4 & 2 & 4 & 1 \\ 4 & 16 & 8 & 16 & 8 & \dots & 16 & 8 & 16 & 4 \\ 2 & 8 & 4 & 8 & 4 & \dots & 8 & 4 & 8 & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 2 & 8 & 4 & 8 & 4 & \dots & 8 & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & 8 & \dots & 16 & 8 & 16 & 4 \\ 1 & 4 & 2 & 4 & 2 & \dots & 4 & 2 & 4 & 1 \end{bmatrix}.$$

În [10, 66], pentru  $f \in C^{4,4}(D)$  se dă următoarea formă a restului

$$R_{mn}(f) = -\frac{(b-a)(d-c)}{180} \left[ h^4 f^{(4,0)}(\xi_1, \eta_1) + k^4 f^{(0,4)}(\xi_2, \eta_2) \right],$$

cu  $\xi_1, \eta_1, \xi_2, \eta_2 \in D$ .

Se pot da formule de cubatură de tip Gauss bidimensionale. De exemplu, dacă  $x_i, i = \overline{0, m}$  și  $y_j, j = \overline{0, n}$  sunt rădăcinile polinomului Legendre relativ la intervalul  $[a, b]$  și respectiv  $[c, d]$ , se obține formula de cubatură Gauss-Legendre, în care

$$\begin{aligned} A_i &= \frac{[(m+1)!]^4(b-a)^{2m+3}}{[(2m+2)!]^2(x_i-a)(b-x_i)[u'(x_i)]^2}, & i &= \overline{0, m} \\ B_j &= \frac{[(n+1)!]^4(b-a)^{2n+3}}{[(2n+2)!]^2(y_j-c)(d-y_j)[u'(y_j)]^2}, & j &= \overline{0, n}, \end{aligned}$$

iar dacă  $f \in C^{2m+2, 2n+2}(D)$

$$\begin{aligned} R_{mn}(f) &= (d-c)\lambda_m f^{(2m+2,0)}(\xi_1, \eta_1) + (b-a)\lambda_n f^{(0,2n+2)}(\xi_2, \eta_2) \\ &\quad - \lambda_m \lambda_n f^{(2m+2, 2n+2)}(\xi_3, \eta_3), \end{aligned}$$

unde

$$\lambda_m = \frac{[(m+1)!]^4(b-a)^{2m+3}}{[(2m+2)!]^3(2m+3)}, \quad \lambda_n = \frac{[(n+1)!]^4(b-a)^{2n+3}}{[(2n+2)!]^3(2n+3)}.$$

Pentru detalii a se vedea [12].

În cazul  $m = n = 0$  se obține o formulă de cubatură cu un singur nod, analoagă formulei dreptunghiului

$$\int_a^b \int_c^d f(x, y) dx dy = (b-a)(d-c)f\left(\frac{a+b}{2}, \frac{c+d}{2}\right) + R_{00}(f),$$

unde

$$\begin{aligned} R_{00}(f) &= \frac{(b-a)^3(d-c)}{24} f^{(2,0)}(\xi_1, \eta_1) + \frac{(b-a)(d-c)^3}{24} f^{(0,2)}(\xi_2, \eta_2) \\ &\quad - \frac{(b-a)^3(d-c)^3}{576} f^{(2,2)}(\xi_3, \eta_3). \end{aligned}$$

Utilizarea metodelor de aproximare de mai sus nu este limitată la domenii rectangulare. De exemplu, tehnica de la formula de cubatură a lui Simpson se poate modifica pentru a fi aplicabilă la aproximarea unor unor integrale de forma

$$\int_a^b \int_{c(x)}^{d(x)} f(x, y) dx dy$$

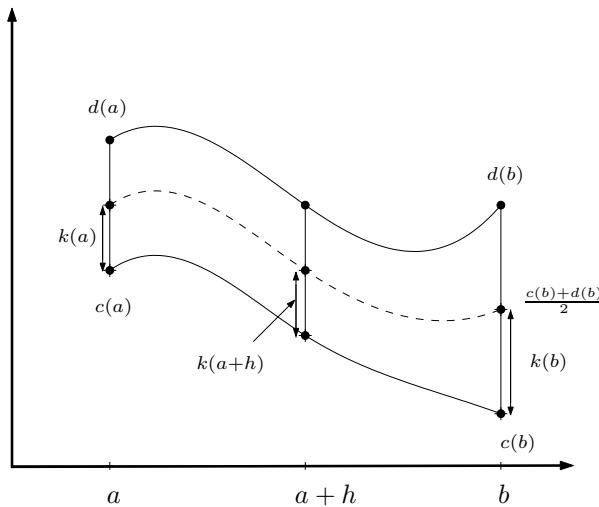
sau

$$\int_c^d \int_{a(y)}^{b(y)} f(x, y) dx dy,$$

cu condiția ca domeniul de integrare să fie simplu în raport cu  $x$ , respectiv  $y$ .

Pentru o integrală de primul tip pasul după  $x$  va fi  $h = \frac{b-a}{2}$ , dar cel după  $y$  va varia odată cu  $x$  (vezi figura 11.4):

$$k(x) = \frac{d(x) + c(x)}{2}.$$

Figura 11.4: Formula lui Simpson pentru un domeniu simplu în raport cu  $x$ 

Se obține

$$\begin{aligned} \int_a^b \int_{c(x)}^{d(x)} f(x, y) dx dy &\approx \int_a^b \frac{k(x)}{3} [f(x, c(x)) + 4f(x, c(x) + k(x)) + f(x, d(x))] dx \\ &\approx \frac{h}{3} \left\{ \frac{k(a)}{3} [f(a, c(a)) + 4f(a, c(a) + k(a)) + f(a, d(a))] \right. \\ &\quad + 4 \frac{k(a+h)}{3} [f(a+h, c(a+h)) + 4f(a+h, c(a+h) + k(a+h)) \\ &\quad + f(a+h, d(a+h))] \\ &\quad \left. + \frac{k(b)}{3} [f(b, c(b)) + 4f(b, c(b) + k(b)) + f(b, d(b))] \right\}. \end{aligned}$$

Dacă domeniul de integrare  $D$  este curbiliniu, construim un dreptunghi  $R \supset D$ , ale căruia laturi să fie paralele cu axele de coordonate (figura 11.5). Se consideră funcția auxiliară

$$f^*(x, y) = \begin{cases} f(x, y), & \text{dacă } (x, y) \in D, \\ 0, & \text{dacă } (x, y) \in R \setminus D. \end{cases}$$

Evident

$$\iint_D f(x, y) dx dy = \iint_R f^*(x, y) dx dy$$

Ultima integrală se poate aproxima printr-o tehnică cunoscută.

### 11.2.1. Considerații de implementare

Fie domeniul de integrare

$$D = [a, b] \times [c, d].$$

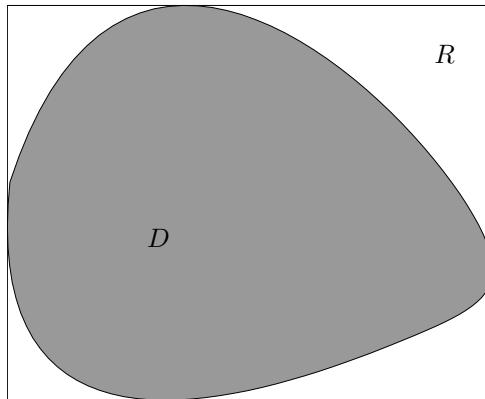


Figura 11.5: Încadrarea unui domeniu curbiliniu într-un dreptunghi

Integrala de aproximat se poate scrie sub forma

$$\int_a^b \int_c^d f(x, y) dx dy = \int_a^b \left( \int_c^d f(x, y) dy \right) dx = \int_a^b F(x) dx,$$

unde

$$F(x) = \int_c^d f(x, y) dy.$$

Să presupunem că `quadquad` este o rutină de cuadratură unidimensională adaptivă. Ideea este de a folosi această această rutină pentru a calcula valorile lui  $F$  definite mai sus și de a folosi din nou rutina pentru a integra  $F$ . Un exemplu de implementare este dat în sursa MATLAB 11.3.

## 11.3. Aproximări în mai multe variabile în MATLAB

### 11.3.1. Interpolarea funcțiilor de mai multe variabile în MATLAB

Există două funcții pentru interpolarea bidimensională în MATLAB: `interp2` și `griddedInterpolant`. Sintaxa cea mai generală a lui `interp2` este

`ZI = interp2(x, y, z, XI, YI, metoda)`

Aici vectorii  $x$  și  $y$  conțin coordonatele nodurilor de interpolare,  $z$  conține valorile funcției în noduri, iar  $XI$  și  $YI$  sunt matrice care conțin coordonatele punctelor în care dorim să facem evaluarea.  $ZI$  conține valorile interpolantului în punctele  $XI$ ,  $YI$ . Parametrul `metoda` poate avea valoarea:

- 'linear' – interpolare bilineară (implicită);
- 'cubic' – interpolare bicubică;
- 'nearest' – interpolare bazată pe vecinul cel mai apropiat;
- 'spline' – interpolare spline.

Toate metodele de interpolare cer ca  $x$  și  $y$  să fie monotone, și să aibă formatul ca și cum ar fi produse de `meshgrid`. Dacă valorile date în  $x$ ,  $y$ ,  $XI$  și  $YI$  nu sunt echidistante, ele sunt transformate într-un domeniu echidistant înainte de interpolare. Pentru eficiență, dacă  $x$  și  $y$  sunt deja echidistante și

**Sursa MATLAB 11.3 Aproximarea unei integrale duble pe dreptunghi**

```

function Q = quaddbl(F,xmin,xmax,ymin,ymax,tol,...  

    quadm,varargin)  

%QUADDBL - aproximeaza o integrala dubla  

%Parametrii  

%F - functia de integrat  

%xmin,xmax,ymin,ymax - limitele de integrare  

%tol -precizia, implicit 1e-6  

%iintm - metoda de integrare interioara  

%quadm - metoda de integrare, implicit adquad  

if nargin < 5, error('Necesita minim 5 argumente'); end  

if nargin < 6 | isempty(tol), tol = 1.e-6; end  

if nargin < 7 | isempty(quadm), quadm = @adquad; end  

F = fcncchk(F);  
  

Q = quadm(@innerint, ymin, ymax, tol, F, ...  

    xmin, xmax, tol, quadm, varargin{:});  
  

%-----  

function Q = innerint(y, F, xmin, xmax, tol, quadm, varargin)  

%INNERINT - utilizata de QUADDBL pentru integrala interioara.  

%  

% quadm determina formula de cuadratura ce va fi utilizata  

% Evalueaza integrala interioara pentru fiecare valoare  

% a variabilei exterioare  
  

Q = zeros(size(y));  

for i = 1:length(y)  

    Q(i) = quadm(F, xmin, xmax, tol, y(i), varargin{:});  

end

```

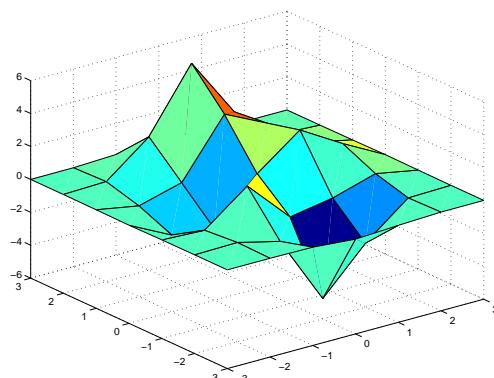


Figura 11.6: Graficul lui peaks pe o grilă grosieră

monotone, metoda se poate da într-o din formele '`*linear`', '`*cubic`', '`*spline`', sau '`*nearest`'.

Dăm un exemplu care încearcă să interpoleze funcția `peaks` pe o grilă de  $7 \times 7$ . Generăm grila, calculăm valorile funcției și o reprezentăm grafic cu

```
[X, Y]=meshgrid(-3:1:3);
Z=peaks(X, Y);
surf(X, Y, Z)
```

Graficul apare în figura 11.6. Calculăm apoi interpolanții pe o grilă mai fină și îi reprezentăm:

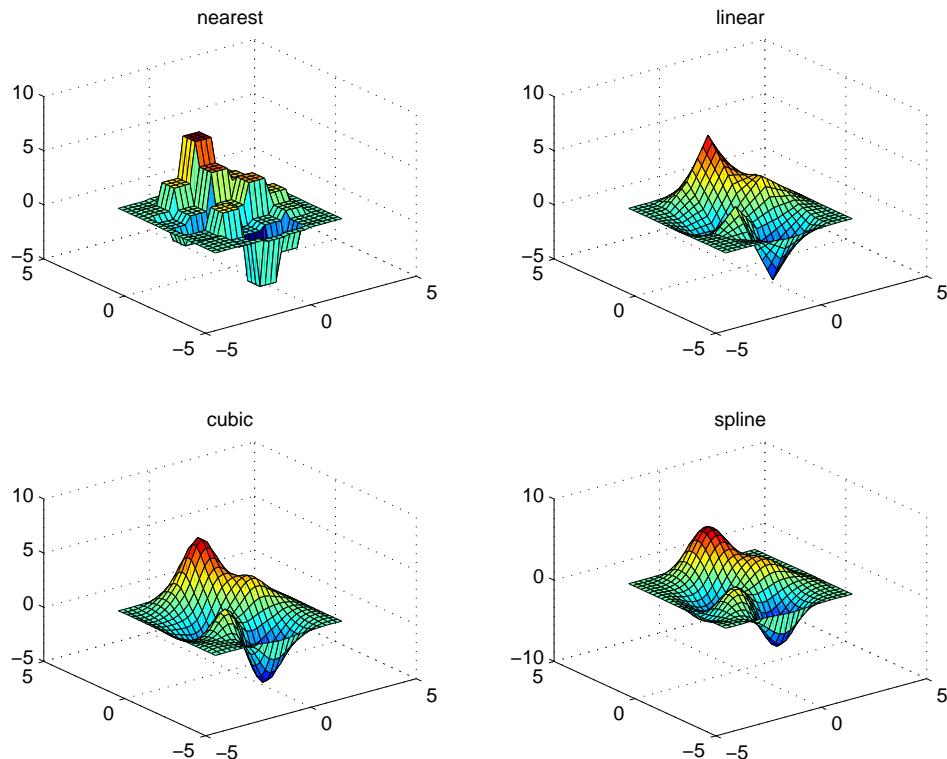
```
[XI, YI]=meshgrid(-3:0.25:3);
ZI1=interp2(X, Y, Z, XI, YI, 'nearest');
ZI2=interp2(X, Y, Z, XI, YI, 'linear');
ZI3=interp2(X, Y, Z, XI, YI, 'cubic');
ZI4=interp2(X, Y, Z, XI, YI, 'spline');
subplot(2,2,1), surf(XI, YI, ZI1)
title('nearest')
subplot(2,2,2), surf(XI, YI, ZI2)
title('linear')
subplot(2,2,3), surf(XI, YI, ZI3)
title('cubic')
subplot(2,2,4), surf(XI, YI, ZI4)
title('spline')
```

Graficele lor apar în figura 11.7. Dacă înlocuim peste tot `surf` cu `contour` obținem graficele din figura 11.8.

Funcția `griddata` are aceeași sintaxă ca și `interp2`. Datele de intrare sunt nodurile `x` și `y`, care nu mai trebuie să fie monotone și valorile `z` în noduri. Prin interpolare se calculează valorile `ZI` corespunzătoare nodurilor `XI` și `YI`, care de obicei sunt obținute cu `meshgrid`. Argumentul metoda poate avea valorile '`linear`', '`cubic`', `nearest` și '`v4`', ultima semnificând o metodă de interpolare specifică MATLAB 4. Toate metodele exceptând `v4` se bazează pe triangulație Delaunay (o triangulație a unei mulțimi de puncte care maximizează unghiul minim). Metoda este utilă pentru a interpoala valori pe o suprafață. Exemplul următor interpolează puncte generate aleator, situate pe suprafață  $z = \frac{\sin(x^2+y^2)}{x^2+y^2}$ . Pentru a nu avea probleme în origine s-a adăugat un `eps` la numitor.

```
x=rand(100,1)*16-8; y=rand(100,1)*16-8;
R=sqrt(x.^2+y.^2)+eps;
z=sin(R)./R;
xp=-8:0.5:8;
[XI, YI]=meshgrid(xp, xp);
ZI=griddata(x, y, z, XI, YI);
mesh(XI, YI, ZI); hold on
plot3(x, y, z, 'ko'); hold off
```

Rezultatul apare în figura 11.9, în care punctele generate aleator sunt marcate prin cercuri, iar interpolantul a fost reprezentat cu `mesh`.

Figura 11.7: Interpolanți construiți cu `interp2`

### 11.3.2. Calculul integralelor duble în MATLAB

Integralele duble pe un dreptunghi pot fi evaluate cu `dblquad`. Pentru ilustrare, să presupunem că dorim să aproximăm integrala

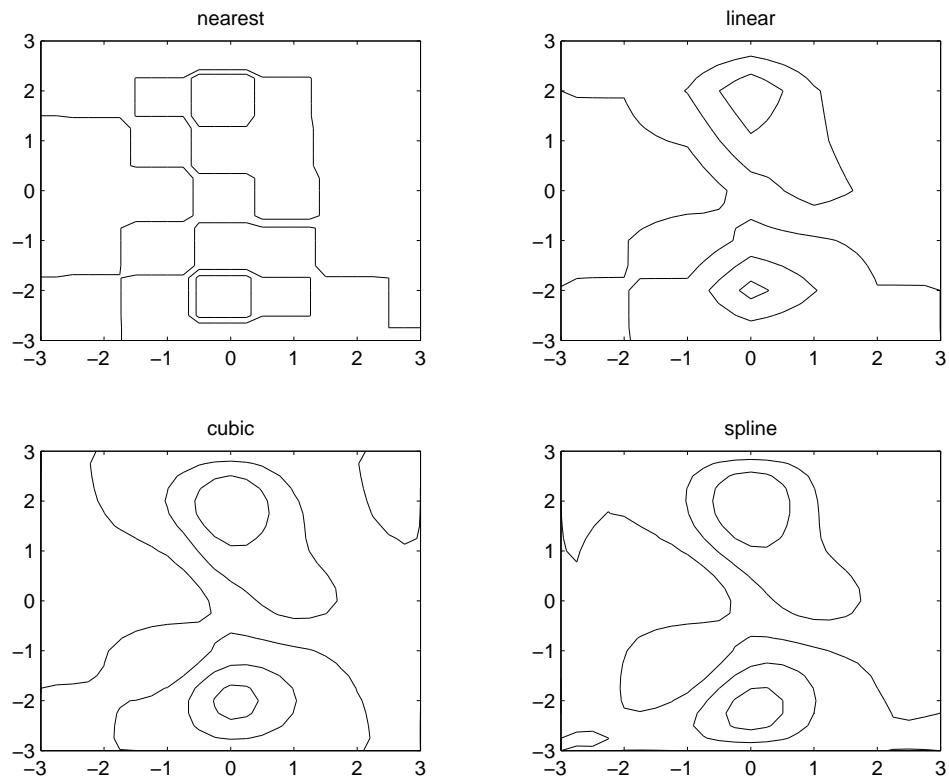
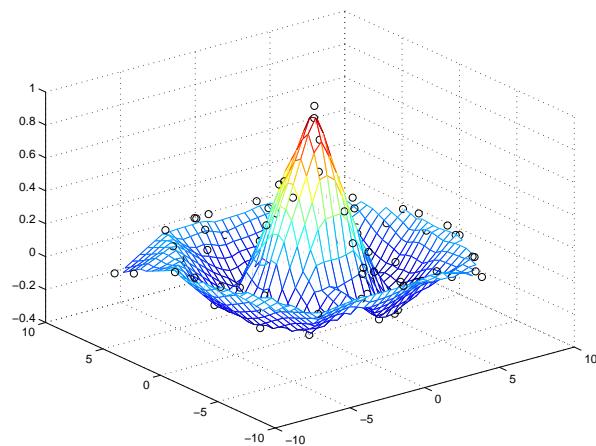
$$\int_0^\pi \int_\pi^{2\pi} (y \sin x + x \cos y) dx dy$$

Valoarea exactă a integralei este  $-\pi^2$ , după cum se poate verifica cu toolbox-ul Symbolic:

```
>> syms x y
>> z=y*sin(x)+x*cos(y);
>> int(int(z,x,pi,2*pi),y,0,pi)
ans =
-pi^2
```

Integrandul se poate da ca obiect inline, expresie sir de caractere, sau function handle. Să presupunem că integrandul este dat în fișierul `integrand.m`:

```
function z = integrand(x, y)
z = y*sin(x)+x*cos(y);
```

Figura 11.8: Interpolanți construiți cu `interp2`Figura 11.9: Interpolare cu `griddata`

Vom utiliza `dblquad` și vom face și verificarea:

```
>> Q = dblquad(@integrand, pi, 2*pi, 0, pi)
Q =
-9.8696
>> -pi^2
ans =
-9.8696
```

Calculăm integrala și cu `quaddbl` (sursa MATLAB 11.3):

```
>> Q2=quaddbl(@integrand,pi,2*pi,0,pi)
Q2 =
-9.8696
```

Integrandul transmis lui `dblquad` trebuie să accepte un vector  $x$  și un scalar  $y$  și să returneze un vector. Se pot transmite argumente suplimentare lui `dblquad` pentru a specifica precizia și metoda de integrare unidimensională (implicit `quad`). Să presupunem că vrem să calculăm

$$\int_4^6 \int_0^1 (y^2 e^x + x \cos y) dx dy$$

cu precizia  $1e-8$  și să folosim `quadl` în loc de `quad`:

```
>> fi=inline('y.^2.*exp(x)+x.*cos(y)');
>> dblquad(fi,0,1,4,6,1e-8,@quadl)
ans =
87.2983
```

Valoarea exactă a integralei calculată cu Maple sau cu toolbox-ul Symbolic este

$$\frac{152}{3}(e-1) + \frac{1}{2}(\sin 6 - \sin 4).$$

Verificare:

```
>> 152/3*(exp(1)-1)+1/2*(sin(6)-sin(4))
ans =
87.2983
```

## Probleme

**Problema 11.1.** Scrieți o funcție MATLAB care să reprezinte grafic o suprafață  $f(x, y)$ ,  $f : [0, 1] \times [0, 1] \rightarrow \mathbb{R}$  ce verifică condițiile

$$\begin{aligned} f(0, y) &= g_1(y) & f(1, y) &= g_2(y) \\ f(x, 0) &= g_3(x) & f(x, 1) &= g_4(y), \end{aligned}$$

unde  $g_i$ ,  $i = \overline{1, 4}$  sunt funcții date definite pe  $[0, 1]$ .

**Problema 11.2.** Determinați interpolanții bidimensionali produs tensorial și sumă booleană core-spunzători unui interpolant Hermite unidimensional cu nodurile duble 0 și 1. Reprezentați grafic acest interpolant, dacă se alege  $f(x, y) = x \exp(x^2 + y^2)$ .

**Problema 11.3.** Extindeți algoritmul de Casteljau la cazul bidimensional. Dați o implementare MATLAB.

**Problema 11.4.** Extindeți algoritmul Cox - de Boor la cazul bidimensional. Dați o implementare MATLAB.

**Problema 11.5.** Adaptați rutina `quaddbl` pentru a aproxima integrale duble de forma

$$\int_a^b \int_{c(x)}^{d(x)} f(x, y) dy dx$$

sau

$$\int_c^d \int_{a(y)}^{b(y)} f(x, y) dx dy,$$

când domeniul de integrare este simplu în raport cu  $x$  sau  $y$ .

**Problema 11.6.** Se consideră integrala dublă a funcției  $f(x, y) = x^2 + y^2$  pe domeniul eliptic  $R$  dat de  $-5 < x < 5$ ,  $y^2 < \frac{3}{5}(25 - x^2)$ .

- (a) Să se reprezinte grafic funcția pe domeniul  $R$ .
- (b) Să se determine valoarea exactă a integralei utilizând Maple sau toolbox-ul Symbolic.
- (c) Să se aproximeze valoarea integralei transformând domeniul într-unul rectangular.
- (d) Să se aproximeze valoarea integralei folosind funcțiile din problema 11.5.

**Problema 11.7.** Se consideră funcția  $f(x, y) = y \cos x^2$  și domeniul triunghiular definit de  $T = \{x \geq 0, y \geq 0, x + y \leq 1\}$  și

$$\int \int_T f(x, y) dx dy.$$

- (a) Să se reprezinte grafic funcția pe  $T$  folosind `trimesh` sau `trisurf`.
- (b) Să se aproximeze valoarea lui  $I$  transformând integrala într-o integrală pe pătratul unitate a unei funcții care este nulă înafara lui  $T$ .
- (c) Să se aproximeze integrala folosind funcțiile din problema 11.5.

---

## Bibliografie

---

- [1] O. Agratini, *Aproximare prin operatori liniari*, Presa Universitară Clujeană, Cluj-Napoca, 2000.
- [2] O. Agratini, *Positive Approximation Processes*, Hiperboreea Press, Turda, 2001.
- [3] Octavian Agratini, Ioana Chiorean, Gheorghe Coman, Radu Trîmbițaș, *Analiză numerică și teoria aproximării*, vol. III, Presa Universitară Clujeană, 2002, D. D. Stancu, Gh. Coman, (coord.).
- [4] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed., SIAM, Philadelphia, PA, 1994, disponibila prin www, <http://www.netlib.org/templates>.
- [5] Jean-Paul Berrut, Lloyd N. Trefethen, *Barycentric lagrange interpolation*, SIAM Review **46** (2004), no. 3, 501–517.
- [6] Å. Björk, *Numerical Methods for Least Squares Problem*, SIAM, Philadelphia, 1996.
- [7] E. Blum, *Numerical Computing: Theory and Practice*, Addison-Wesley, 1972.
- [8] P. Bogacki, L. F. Shampine, *A 3(2) pair of Runge-Kutta formulas*, Appl. Math. Lett. **2** (1989), no. 4, 321–325.
- [9] C. G. Broyden, *A Class of Methods for Solving Nonlinear Simultaneous Equations*, Math. Comp. **19** (1965), 577–593.
- [10] L. Burden, J. D. Faires, *Numerical Analysis*, PWS Kent, Boston, 1986.
- [11] P. G. Ciarlet, *Introduction à l'analyse numérique matricielle et à l'optimisation*, Masson, Paris, Milan, Barcelone, Mexico, 1990.
- [12] Gheorghe Coman, *Analiză numerică*, Editura Libris, Cluj-Napoca, 1995.
- [13] I. Cuculescu, *Analiză numerică*, Editura Tehnică, Bucureşti, 1967.
- [14] P. J. Davis, P. Rabinowitz, *Numerical Integration*, Blaisdell, Waltham, Massachusetts, 1967.
- [15] James Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
- [16] J. E. Dennis, J. J. Moré, *Quasi-Newton Methods, Motivation and Theory*, SIAM Review **19** (1977), 46–89.

- [17] J. Dormand, *Numerical Methods for Differential Equations. A Computational Approach*, CRC Press, Boca Raton New York, 1996.
- [18] T. A. Driscoll, N. Hale, L. N. Trefethen, *Chebfun guide*, Pafnuty Publications, Oxford, 2014.
- [19] Gerald Farin, *Curves and Surfaces for Computer-Aided Geometric Design: a Practical Guide*, fourth ed., Academic Press, 1996.
- [20] E. Fehlberg, *Klassische Runge-Kutta-Formeln fünfter und siebenter Ordnung mit Schrittweiten-Kontrolle*, Computing **4** (1969), 93–106, Corrigendum: *ibid.* **5**, 184.
- [21] E. Fehlberg, *Klassische Runge-Kutta-Formeln vierter und niedriger Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme*, Computing **6** (1970), 61–71, Corrigendum: *ibid.* **5**, 184.
- [22] J. G. F. Francis, *The QR transformation: A unitary analogue to the LR transformation*, Computer J. **4** (1961), 256–272, 332–345, parts I and II.
- [23] W. Gander, W. Gautschi, *Adaptive quadrature - revisited*, BIT **40** (2000), 84–101.
- [24] W. Gautschi, *Numerical Analysis, An Introduction*, Birkhäuser, Basel, 1997.
- [25] Walther Gautschi, *Orthogonal polynomials: applications and computation*, Acta Numerica **5** (1996), 45–119.
- [26] J. Gilbert, C. Moler, R. Schreiber, *Sparse matrices in MATLAB: Design and implementation.*, SIAM J. Matrix Anal. Appl. **13** (1992), no. 1, 333–356, disponibil în kit-ul MATLAB.
- [27] D. Goldberg, *What every computer scientist should know about floating-point arithmetic*, Computing Surveys **23** (1991), no. 1, 5–48.
- [28] H. H. Goldstine, J. von Neumann, *Numerical inverting of matrices of high order*, Amer. Math. Soc. Bull. **53** (1947), 1021–1099.
- [29] Gene H. Golub, Charles van Loan, *Matrix Computations*, 3rd ed., John Hopkins University Press, Baltimore and London, 1996.
- [30] Jens Gravesen, *Differential Geometry and Design of Shape and Motion*, Technical University of Denmark, Lingby, Denmark, 2002.
- [31] P. R. Halmos, *Finite-Dimensional Vector Spaces*, Springer Verlag, New York, 1958.
- [32] D. J. Higham, N. J. Higham, *MATLAB Guide*, SIAM, Philadelphia, 2000.
- [33] N. J. Higham, F. Tisseur, *A Block Algorithm for Matrix 1-Norm Estimation, with an Application to 1-Norm Pseudospectra*, SIAM Journal Matrix Anal. Appl. **21** (2000), no. 4, 1185–1201.
- [34] Nicholas J. Higham, *The Test Matrix Toolbox for MATLAB*, Tech. report, Manchester Centre for Computational Mathematics, 1995, disponibil via WWW la adresa <http://www.ma.man.ac.uk/MCCM/MCCM.html>.
- [35] Nicholas J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.
- [36] E. Isaacson, H. B. Keller, *Analysis of Numerical Methods*, John Wiley, New York, 1966.
- [37] D. Kincaid, W. Cheney, *Numerical Analysis: Mathematics of Scientific Computing*, Brooks/Cole Publishing Company, Belmont, CA, 1991.
- [38] Mirela Kohr, *Capitole speciale de mecanică*, Presa Univ. Clujeană, 2005.
- [39] V. N. Kublanovskaya, *On some algorithms for the solution of the complete eigenvalue problem*, USSR Comp. Math. Phys. **3** (1961), 637–657.
- [40] The Mathworks Inc., Natick, Ma, *Using MATLAB*, 2002.

- [41] The Mathworks Inc., Natick, Ma, *MATLAB. Getting Started*, 2004, Version 7.
- [42] The Mathworks Inc., Natick, Ma, *MATLAB. Symbolic Math Toolbox*, 2004, Version 7.
- [43] The Mathworks Inc., Natick, Ma, *MATLAB. The Language of Technical Computing. Mathematics*, 2004, Version 7.
- [44] The Mathworks Inc., Natick, Ma, *MATLAB. The Language of Technical Computing. Programming*, 2004, Version 7.
- [45] The Mathworks Inc., Natick, Ma, *Using MATLAB Graphics*, 2004, Version 7.
- [46] Cleve Moler, *Numerical Computing in MATLAB*, SIAM, 2004, disponibil via www la adresa <http://www.mathworks.com/moler>.
- [47] J. J. Moré, M. Y. Cosnard, *Numerical Solutions of Nonlinear Equations*, ACM Trans. Math. Softw. **5** (1979), 64–85.
- [48] Shoichiro Nakamura, *Numerical Computing and Graphic Vizualization in MATLAB*, Prentice Hall, Englewood Cliffs, NJ, 1996.
- [49] Dana Petcu, *Matematică asistată de calculator*, Eubeea, Timișoara, 2000.
- [50] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, Cambridge, New York, Port Chester, Melbourne, Sidney, 1996, disponibila prin www, <http://www.nr.com/>.
- [51] J. J. Rissler, *Méthodes mathématiques pour la CAO*, Masson, Paris, Milan, Barcelone, Bonn, 1991.
- [52] I. A. Rus, *Ecuatii diferențiale, ecuații integrale și sisteme dinamice*, Transilvania Press, Cluj-Napoca, 1996.
- [53] I. A. Rus, P. Pavel, *Ecuatii diferențiale*, Editura Didactică și Pedagogică, București, 1982, ediția a doua.
- [54] H. Rutishauser, *Solution of the eigenvalue problems with the LR transformation*, Nat. Bur. Stand. App. Math. Ser. **49** (1958), 47–81.
- [55] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996, disponibilă via www la adresa <http://www-users.cs.umn.edu/~saad/books.html>.
- [56] H. E. Salzer, *Lagrangian interpolation at the chebyshev points  $\cos(\nu\pi/n)$ ,  $\nu = 0(1)n$ ; some unnoted advantages*, Computer Journal **15** (1974), 156–159.
- [57] A. Sard, *Linear Approximation*, American Mathematical Society, Providence, RI, 1963.
- [58] Thomas Sauer, *Numerische Mathematik I*, Universität Erlangen-Nürnberg, Erlangen, 2000, Vorlesungskript.
- [59] Thomas Sauer, *Numerische Mathematik II*, Universität Erlangen-Nürnberg, Erlangen, 2000, Vorlesungskript.
- [60] R. Schwarz, H., *Numerische Mathematik*, B. G. Teubner, Stuttgart, 1988.
- [61] L. F. Shampine, R. C. Allen, S. Pruess, *Fundamentals of Numerical Computing*, John Wiley & Sons, Inc, 1997.
- [62] L. F. Shampine, I. Gladwell, S Thompson, *Solving ODEs with matlab*, Cambridge University Press, Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo, 2003.
- [63] L. F. Shampine, Reichelt R. W., *The MATLAB ODE suite*, SIAM J. Sci. Comput. **18** (1997), no. 1, 1–22.
- [64] D. D. Stancu, *Asupra formulei de interpolare a lui Hermite și a unor aplicații ale acesteia*, Acad. R. P. Rom. Studii și Cercetări Matematice **8** (1957), 339–355, Filiala Cluj a Academiei.

- [65] D. D. Stancu, *Analiză numerică – Curs și culegere de probleme*, Lito UBB, Cluj-Napoca, 1977.
- [66] D. D. Stancu, G. Coman, P. Blaga, *Analiză numerică și teoria aproximării*, vol. II, Presa Universitară Clujeană, Cluj-Napoca, 2002, D. D. Stancu, Gh. Coman, (coord.).
- [67] D. D. Stancu, Gh. Coman, O. Agratini, R. Trîmbițaș, *Analiză numerică și Teoria aproximării*, vol. I, Presa Universitară Clujeană, Cluj-Napoca, 2001, D. D. Stancu, Gh. Coman, (coord.).
- [68] J. Stoer, R. Burlisch, *Einführung in die Numerische Mathematik*, vol. II, Springer Verlag, Berlin, Heidelberg, 1978.
- [69] J. Stoer, R. Burlisch, *Introduction to Numerical Analysis*, 2nd ed., Springer Verlag, 1992.
- [70] A. H. Stroud, *Approximate Calculation of Multiple Integrals*, Prentice Hall Inc., Englewood Cliffs, NJ, 1971.
- [71] Lloyd N. Trefethen, David Bau III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1996.
- [72] E. E. Tyrtyshnikov, *A Brief Introduction to Numerical Analysis*, Birkhäuser, Boston, Basel, Berlin, 1997.
- [73] C. Überhuber, *Computer-Numerik*, vol. 1, 2, Springer Verlag, Berlin, Heidelberg, New-York, 1995.
- [74] C. Ueberhuber, *Numerical Computation. Methods, Software and Analysis*, vol. I, II, Springer Verlag, Berlin, Heidelberg, New York, 1997.
- [75] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

---

## Indice

---

\ (operator), 12, 131  
: (operator), 10  
% (comentariu), 25

algoritm  
    precis, 91  
    regresiv stabil, 92  
    stabil, 92

algoritmul  
    Cox-deBoor, 249  
    de Casteljau, 250

algoritmul lui Strassen, 127

analiza progresivă a erorilor, 93  
analiza regresivă a erorilor, 93  
aproximantă, 73  
aproximantă algebric maximală, 413  
aproximantă algebric minimală, 413  
axioma fundamentală a aritmeticii în virgulă flotantă, 77

axis, 54

blkdiag, 7  
break, 23

chol, 134  
clf, 53  
close, 53  
colorbar, 64  
colormap, 64

cometarii MATLAB, 25

cond, 106  
condeig, 346  
condest, 106  
condiționare

    a unui sistem liniar, 104  
condiționare  
    a unei probleme, 84  
    a unui algoritm, 87  
    număr de condiționare, 85

continue, 23  
contour, 61  
conv, 181  
convergență liniară, 298  
cross, 14  
cuadraturi adaptive, 284

dblquad, 427  
deconv, 181  
demo, 2  
derivare numerică, 268  
descompunere maximală, 413  
descompunere minimală, 413  
det, 13  
diag, 8  
diary, 5  
diferență divizată cu noduri multiple, 222  
diferență finită  
    regresivă, 214

diferență regresivă, 214  
diferențe divizate, 209  
diferențe finite  
    progresive, Vezi diferențe progresive  
diferențe progresive, 212  
diff(Symbolic), 41  
digits, 45  
doc, 2  
dot, 14

- double**, 46  
**ecuații normale**, 158  
**eig**, 345  
 element de cea mai bună aproximare, 153  
 element pivot, 111  
 eliminare gaussiană, 109  
**eps**, 3  
 eroare, 74  
 eroare absolută, 74  
 eroare asimptotică, 298  
 eroare de trunchiere, 353  
 eroare relativă, 74  
**error**, 24, 39  
**eye**, 6  
**ezcontour**, 61  
  
**fcnchk**, 31  
**feval**, 31  
**figure**, 53  
**fill**, 53  
**find**, 19  
 fișier M, 24
  - funcție, 25, 26
  - script, 25**flops**, 112  
**for**, 21, 22  
**format**, 3  
 formulă de derivare numerică, 266  
 formulă de integrare numerică, 270  
 formulă de tip Gauss-Christoffel, *Vezi* formulă de cuadratură de tip Gauss  
 formulă de aproximare
  - algebric maximală, 414
  - algebric minimală, 414
  - consistentă, 418
  - omogenă, 419
 formulă de cuadratură, *Vezi* formulă de integrare numerică  
 formulă de cubatură, 419
  - a lui Simpson, 420
  - a trapezului, 420
  - Newton-Cotes, 420
 formula de aproximare
  - a lui Bernstein, 243
 formula Euler-MacLaurin, 288  
 formula lui Newton progresivă, 213  
 formula lui Newton regresivă, 214  
 formula lui Simpson, 272  
  
 formula lui Simpson repetată, 272  
 formula repetată a trapezului, 271  
 formula trapezului, *Vezi* regula trapezului  
 formule de cuadratură de tip Gauss, 276  
 formule Newton-Cotes, 276  
**fplot**, 58  
**fsolve**, 45  
 funcție
  - B-spline, 244
 funcție grilă, 363  
 function handle, 29  
 funcții anonte, 30  
 funcții imbricate, 29  
**fzero**, 318  
  
**gallery**, 9  
**global**, 34  
 grad de exactitate, 266, 270  
 grilă, 363  
  
**help**, 1  
**hess**, 347  
  
 identitatea lui Marsden, 245  
 IEEE 754 (standardul), 79  
**if**, 21  
 indice de eficiență, 299  
 inegalitatea de stabilitate, 364  
**Inf**, 2  
**inline**, 30  
**int**, 41  
**int16**, 37  
**int32**, 37  
**int8**, 37  
 integrare numerică, 269  
**interp1**, 231  
 interpolare blending, *Vezi* interpolare transfiniță  
 interpolare Hermite, 192  
 interpolare Lagrange, 189
  - metoda lui Aitken, 208
  - metoda lui Neville, 207
 interpolare polinomială, 186  
 interpolare spline, 222  
 interpolare transfiniță, 413  
**inv**, 13, 14  
 iterație vectorială, 328  
  
**lasterr**, 39

- lasterror, 39  
legend, 56  
length, 6  
limit, 43  
load, 4  
logical, 20  
loglog, 51  
lookfor, 2  
lu, 134
- maple, 45  
matrice  
    companion, 324  
    complement Schur, 114  
    descompunere cu valori singulare, 347  
    descompunere LU a unei ~, 112  
    descompunere LUP a unei ~, 116  
    descompunere QR a unei ~, 121  
    descompunere Schur a unei ~, 325  
    descompunere Schur reală a unei ~, 327  
    diagonalizabilă, 325  
    factorizare Cholesky a unei ~, 119  
    forma normală Jordan a unei ~, 325  
    hermitiană, 98  
    Hessenberg superioară, 327  
    Jacobi, 279  
    nederogatorie, 325  
    normală, 98  
    ortogonală, 98  
    polinom caracteristic al unei ~, 323  
    simetrică, 98  
    similară, 325  
    spectrul unei ~, 98  
    transformarea RQ a unei ~, 334  
    unitară, 98  
    urma unei ~, 102  
    valoare proprie a unei ~, 323  
    vector propriu al unei ~, 323
- mesh, 62  
meshc, 62  
meshgrid, 61
- metodă cu un pas  
    consistentă, 353  
    convergentă, 366  
    funcția de eroare principală, 354  
    stabilă, 364
- metodă cu un pas  
    ordin, 353  
    ordin exact, 353
- metoda  
    aproximațiilor succesive, 307  
    dezvoltări Taylor, 355  
    eliminării a lui Gauss, *Vezi* eliminare gaussiană  
    falsei poziții, 301  
    Gauss-Seidel, 141  
    lui Broyden, 316  
    lui Euler, 354  
    lui Euler modificată, 357  
    lui Heun, 357  
    lui Jacobi, 141  
    lui Newton, 305  
    lui Romberg, 285  
    lui Sturm, 299  
    puterii, *Vezi* iterație vectorială  
    QR, 330  
        cu deplasare spectrală, 338  
        cu pas dublu, 343  
        simplă, 336  
        rafinării iterative, 127  
        relaxării, 142  
        secantei, 303  
        SOR, 142
- metode  
    quasi-Newton, 314  
    Runge-Kutta, 357
- NaN, 3  
nargin, 27  
nargout, 27  
norm, 97, 104  
normă matricială, 98  
normă Frobenius, 103  
normă matricială  
    indusă, 99  
    subordonată, 99
- notația  
     $O$ , 90  
     $\Omega$ , 91  
     $\Theta$ , 90
- număr de condiționare, 105  
număr de condiționare al unei matrice, 105
- obiect inline, 30  
ode15i, 402  
ode113, 383  
ode15s, 383  
ode23s, 383

- ode23tb**, 383  
**ode23t**, 383  
**ode23**, 383  
**ode45**, 383  
**ones**, 6  
**operator**  
 liniar și pozitiv, 257  
 spline cu variație diminuată, *Vezi* opera-  
 torul lui Schoenberg  
**operatorul**  
 Bernstein, 240, 260  
 Hermite-Fejér, 260  
 lui Schoenberg, 255, 260  
 ordin de convergență, 298  
**partiția unității**, 245  
**pchip**, 232, 234  
**pi**, 3  
**pinv**, 182  
 pivotare maximală pe coloana, 110  
 pivotare parțială, *Vezi* pivotare maximală pe  
 coloană  
 pivotare scalată pe coloană, 111  
**plot**, 49  
**plot3**, 59  
**polar**, 52  
 polinoame ortogonale, 164  
 Cebîșev de speță a două, 176  
 Cebîșev de speță I, 169  
 Hermite, 177  
 Jacobi, 177  
 Laguerre , 176  
 Legendre, 165  
**polinom**  
 Bernstein, 240  
 de cea mai bună aproximare uniformă,  
 262  
**poly**, 180  
**polyder**, 180  
**polyfit**, 183, 231  
**polyval**, 180  
**polyvalm**, 180  
**ppval**, 234  
 principiul efectelor egale, 75  
**print**, 66, 67  
 problemă de interpolare, 154  
 problemă de cea mai bună aproximare, 153  
 problemă incorrect pusă, 89  
 problemă numerică, 72  
 problema prost condiționată, 88  
 produs tensorial, 411  
 puncte de alternanță Cebîșev, 262  
**qr**, 135  
**quad**, 292  
**quadl**, 292  
 rafinare iterativa, *Vezi* metoda rafinării iterative  
**rand**, 6  
**randn**, 6  
**rcond**, 106  
 regula trapezelor, *Vezi* formula repetată a  
 trapezului  
 regula trapezului, 271  
**repmat**, 7  
**reshape**, 8  
 rezolvarea numerică a ecuațiilor diferențiale  
 metode cu un pas, 353  
**roots**, 180  
 rotație Givens, 123  
**save**, 4  
**semilogx**, 52  
**semilogy**, 52  
**simple**, 42  
**simplify**, 42  
**single**, 37, 82  
**size**, 5  
**solve**, 44  
**sort**, 16  
**spline**  
 Not-a-knot, 227  
**spline**, 232  
 spline complete, 227  
 spline cubice, 224  
 spline naturale, 227  
 subfuncții, 27  
**subplot**, 58  
**subs**, 42  
 suma booleană, 412  
**surf**, 63  
**surfc**, 63  
**svd**, 348  
**switch**, 21, 23  
**sym**, 40  
**syms**, 41  
 tabelă Butcher, 361

tabela Butcher, 376  
tablou de celule, 32  
taylor, 42  
Teorema lui Peano, 187  
text, 56  
tic, 34  
title, 52  
toc, 34  
transformare  
    Householder, 121  
trapz, 293  
tril, 9  
triu, 9  
try-catch, 39  
  
uint16, 37  
uint32, 37  
uint8, 37  
  
valori proprii generalizate, 347  
varargin, 32  
varargout, 32  
vectorize, 31  
view, 64  
virgulă flotantă  
    anulare, 77  
    exponent, 75  
    normalizarea unui număr, 75  
    reprezentare, 75  
    semnificant, 75  
vpa, 45  
  
waterfall, 64  
while, 21, 23  
whos, 4  
  
 xlabel, 52, 61  
 xlim, 55  
  
 ylabel, 52  
 ylim, 55  
  
zeros, 6