Software matematic

Radu Tiberiu Trîmbiţaş

Prefaţă

Pentru a descărca sursele din această carte şi soluţiile problemelor trimitem cititorul la pagina de web a autorului: `http://www.math.ubbcluj.ro/~tradu`.

Radu Tiberiu Trîmbiţaş
Cluj-Napoca, februarie 2005

Cuprins

Lista surselor MATLAB

CAPITOLUL 1

Introducere în MATLAB

Cuprins

MATLAB[1] este un sistem interactiv destinat calculelor numerice. Prima versiune MATLAB a fost scrisă în anii '70 de Cleve Moler. MATLAB uşurează sarcina utilizatorului de a rezolva problemele numerice. Aceasta permite concentrarea asupra părţii creatoare a rezolvării problemei şi încurajează experimentele. MATLAB utilizează algoritmi cunoscuţi şi testaţi, în care utilizatorul poate avea încredere. Operaţiile puternice se pot realiza uşor cu un număr mic de comenzi (de multe ori una sau două). Vă puteţi programa propriul set de funcţii pentru aplicaţia dumneavoastră. De asemenea, sunt disponibile facilităţi grafice excelente, iar imaginile pot fi inserate în documente LaTeX sau Word. Pentru o introducere mai detaliată în MATLAB a se vedea [5, 15, 10].

1.1. Lansarea MATLAB şi sistemul de help

Sub sistemul de operare Windows, MATLAB se lansează dând un click dublu pe icon-ul 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 comenzile sunt introduse în linia de comandă de la tastatură), ca un limbaj de programare de nivel înalt şi sub formă de rutine apelate dintr-un limbaj de programare, de exemplu C.

Informaţiile de help pot fi obţinute 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.

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 obţine informaţii de help despre funcţiile elementare se tastează

[1] MATLAB® este o marcă înregistrată a Mathworks Inc., Natick MA

```
>> help elfun
```

Pentru a obţine 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 caută în fişierele help un cuvânt cheie. Propunem cititorului să testeze `lookfor factorization`, care dă informaţii despre rutinele de factorizare a matricelor, deosebit de utile în algebra liniară.

Pentru începători şi cei care predau MATLAB demonstraţiile sunt foarte utile. Un set cuprinzător se poate lansa prin comanda

```
>> demo
```

Atenţie, ea şterge toate variabilele!

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

1.2. Modul calculator

Operaţiile aritmetice de bază sunt `+ - * /` şi ridicarea la putere `^`. Ordinea implicită a operaţiilor se poate schimba cu ajutorul parantezelor.

MATLAB recunoşte mai multe tipuri de numere:

- întregi, cum are 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 obţine ca rezultat al unei operaţii ilegale sau al unei nedeterminări din analiza matematică (0/0, ∞/∞, $\infty - \infty$, etc.).

Notaţia 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 *afişează* numerele este controlat de comanda `format`. Tastaţi `help format` pentru o listă completă. Tabela următoare dă câteva exemple.

Comanda	Exemple de ieşiri
`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 ieşire şi permite să se afişeze mai multă informaţie.

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

- `eps = 2.2204e-16 =` 2^{-54} este epsilon-ul maşinii 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` şi `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
>>ans*5
ans =
   -65
```

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

Tabela 1.1: Funcţii elementare şi funcţii matematice speciale (`"fun*"` indică existenţa mai multor funcţii al căror nume începe cu ”`fun`”

Dacă dorim să suprimăm afişarea ultimei expresii evaluate, vom pune caracterul „;” la sfârşitul 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 afişată, sau cu „;”, caz în care valoarea expresiei nu va fi afişată.

```
>> 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 construcţii de tip wildcard, desemnate prin *. Rezultatul salvării se păstrează în fişierul `nume-fisier` de tip .mat, în format binar, specific MATLAB. Variabilele salvate pot fi încărcate prin

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

Se pot face salvări şi încărcări şi în format ascii, în dublă precizie sau prin adăugare la un fişier existent. Pentru detalii a se vedea `help save` şi `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 comenzile şi rezultatele afişate pe ecran (cu excepţia celor ale comenzilor grafice) în fişierul `nume-fisier`. Acest proces de ,,jurnalizare" se termină prin

```
>>diary off
```

MATLAB dispune de un set bogat de funcţii elementare, care apar organizate pe clase în tabela 1.1.

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 şi n coloane. Vectorii linie ($m = 1$) şi 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 şi punctul şi virgula sau ,,newline" pentru a separa liniile:

```
>> A = [5 7 9
1 -3 -7]
A =
     5     7     9
     1    -3    -7
```

`zeros`	Matricea nulă
`ones`	Matrice formată din elemente 1
`eye`	Matricea identică
`repmat`	Replicarea şi pavarea tablourilor
`rand`	Numere aleatoare distribuite uniform
`randn`	Numere aleatoare distribuite normal
`linspace`	Vector de elemente echidistante
`logspace`	Vector de elemente spaţiate logaritmic

Tabela 1.2: Funcţii pentru generarea de matrice

```
>> 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 obţine cu comanda `size`:

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

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

MATLAB are un set util de funcţii pentru construirea unor matrice speciale, vezi tabela 1.2. Matricele de zerouri, de elemente 1 şi matricele identice se obţin cu funcţiile `zeros`, `ones` şi respectiv `eye`. Toate au aceeaşi 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 situaţie comună se întâlneşte atunci când se doreşte 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 funcţie înrudită cu `size` este funcţia `length`: `length(A)` este cea mai mare dintre dimensiunile lui `A`. Astfel, pentru un vector $\mathtt{n} \times 1$ sau $1 \times \mathtt{n}$, `x`, `length(x)` returnează `n`.

Funcţiile `rand` şi `randn` generează matrice de numere (pseudo-)aleatoare, utilizând aceeaşi sintaxă ca şi `eye`. Funcţia `rand` produce o matrice de numere aleatoare având distribuţia uniformă pe intervalul [0,1]. Funcţia `randn` generează o matrice de numere aleatoare având distribuţia normală standard. Apelate fără argumente, ambele funcţii 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 şi experimentele cu numere aleatoare este important ca secvenţele 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 iniţială (starea de la lansarea MATLAB). Pentru întregi `j` nenuli, generatorul este setat pe a `j`-a stare. Starea lui `randn` se setează în acelaşi mod. Perioadele lui `rand` şi `randn`, adică numărul de termeni generaţi înainte ca secvenţele să înceapă să se repete este mai mare decât $2^{1492} \approx 10^{449}$.

Matricele se pot construi şi 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 funcţia `blkdiag`, care este mai uşor de utilizat decât notaţia 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
```

Funcţia `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` lipseşte, 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
```

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

`reshape`	Schimbarea dimensiunii
`diag`	Matrice diagonale şi diagonale ale matricelor
`blkdiag`	Matrice diagonală pe blocuri
`tril`	Extragerea părţii triunghiulare inferior
`triu`	Extragerea părţii triunghiulare inferior
`fliplr`	Rotire matrice în jurul axei de simetrie verticale
`flipud`	Rotire matrice în jurul axei de simetrie orizontale
`rot90`	Rotaţia unei matrice cu 90 de grade

Tabela 1.3: Funcţii de manipulare a matricelor

Funcţia `reshape` schimbă dimensiunile unei matrice: `reshape(A,m,n)` produce o matrice `m` pe `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
```

Funcţia `diag` lucrează cu diagonalele unei matrice şi 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 $\texttt{k} = 0$ înseamnă diagonala principală, $\texttt{k} > 0$ specifică diagonale situate deasupra diagonalei principale, iar $\texttt{k} < 0$ 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`, `diag(A)` este vectorul coloană format din elementele de pe diagonala principală a lui `A`. Pentru a produce o matrice diagonală având aceaşi diagonală ca `A` se va utiliza `diag(diag(A))`. Analog cazului vectorial, `diag(A,k)` produce un vector coloană construit din a `k`-a diagonală 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
```

`tril(A)` obţine partea triunghiulară inferior a lui `A` (elementele situate pe diagonală principală şi dedesubtul ei şi în rest zero). Analog lucrează `triu(A)` pentru partea triunghiulară superior. Mai general, `tril(A,k)` dă elementele situate pe diagonala a `k`-a a lui `A` şi dedesubtul ei, în timp ce `triu(A,k)` dă elementele situate pe a `k`-a diagonală a lui `A` şi 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 funcţii pentru generarea unor matrice speciale. Aceste matrice au proprietăţi interesante care le fac utile pentru construirea de exemple şi testarea algoritmilor. Ele sunt date în tabela 1.4.

Funcţia `gallery` asigură accesul la o colecţie bogată de matrice de test creată de Nicholas J. Higham [6]. Pentru detalii vezi `help gallery`.

1.3.2. Indexarea şi notaţia „:”

Pentru a permite accesul şi atribuirea la nivel de *submatrice*, MATLAB are o notaţie puternică bazată pe caracterul „:”. Ea este utilizată pentru a defini vectori care acţionează ca indici. Pentru scalarii `i` şi `j`, `i:j` desemnează vectorul linie cu elementele `i`, `i+1`, . . . `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 =
```

`compan`	matrice companion
`gallery`	colecţie de matrice de test
`hadamard`	matrice Hadamard
`hankel`	matrice Hankel
`hilb`	matrice Hilbert
`invhilb`	inversa matricei Hilbert
`magic`	pătrat magic
`pascal`	matricea Pascal (coeficienţi 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

```
    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` ≥ 1 şi `j` ≥ 1 (indicii zero sau negativi nu sunt admişi în MATLAB). Notaţia `A(p:q,r:s)` desemnează submatricea constând din intersecţia liniilor de la `p` la `q` şi coloanelor de la `r` la `s` a lui `A`. Ca un caz special, caracterul „:” singur, ca specificator de linie şi 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 şi coloană individuali. De exemplu, `A([i,j,k],[p,q])` produce submatricea dată de intersecţia liniilor `i`, `j` şi `k` şi coloanelor `p` şi `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 conţine toate elementele lui `A`, aşezate 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 atriburi, `A(:)` completează `A`, păstrându-i forma. Cu astfel de notaţie, matricea de numere prime 3×3, `A`, se poate defini prin

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

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

Legată de notaţia „:” este funcţia `linspace`, care în loc de increment acceptă număr de puncte: `linspace(a,b,n)` generează `n` puncte echidistante între `a` şi `b`. Dacă `n` lipseşte, 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
```

Notaţia `[]` înseamnă matricea vidă, 0×0. Atribuirea lui `[]` unei linii sau unei matrice este un mod de a şterge o linie sau o coloană a matricei:

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

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

Operaţia	Sens matricial	Sens tablou
Adunare	+	+
Scădere	-	-
Înmulţire	*	.*
Împărţire stângă	\	.\
Împărţire uzuală	/	./
Ridicare la putere	^	.^

Tabela 1.5: Operaţii pe matrice şi tablouri

1.3.3. Operaţii în sens matricial şi în sens tablou

Pentru scalarii `a` şi `b`, operaţiile `+`, `-`, `/` and `^` produc rezultate evidente. Pe lângă operatorul uzual de împărţire, cu semnificaţia $\frac{a}{b}$, MATLAB are operatorul de împărţire stângă (backslash \), cu semnificaţia $\frac{b}{a}$. Pentru matrice, toate aceste operaţii 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.

Operaţiile de adunare şi scădere sunt identice atât în sens matricial cât şi în sens tablou. Produsul `A*B` este înmulţirea matricială uzuală. Operatorii de împărţire `/` şi \ definesc soluţii ale sistemelor liniare: `A\B` este soluţia `X` a lui `A*X = B`, în timp ce `A/B` este soluţia 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
```

Înmulţirea şi împărţirea în sens tablou, sau pe elemente, se specifică precedând operatorul cu un punct. Dacă `A` şi `B` sunt matrice de aceeaşi 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` şi `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 obţine cu funcţia `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 obţine ridicând la pătrat fiecare element al lui `A`:

```
>> A^2, A.^2
ans =
     7    10
    15    22
```

```
ans =
     1     4
     9    16
```

Operaţia .^ permite ca exponentul să fie un tablou când dimensiunile bazei şi 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 întregi pozitivi. Dacă `n<0` este un întreg, atunci `A^n` este definit prin `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 condiţionată din punct de vedere al valorilor proprii.

Transpusa conjugată a matricei `A` se obţine cu `A'`. Dacă `A` este reală, atunci aceasta este transpusa obişnuită. Transpusa fără conjugare se obţine cu `A.'`. Alternativele funcţionale `ctranspose(A)` şi `transpose(A)` sunt uneori mai convenabile.

În cazul particular al vectorilor coloană `x` şi `y`, `x'*y` este produsul scalar, care se poate obţine şi cu `dot(x,y)`. Produsul vectorial a doi vectori se poate obţine cu `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 şi 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
```

Totuşi, 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 `ones(1,2)`. Dacă o matrice este înmulţită sau împărţită cu un scalar, operaţia se realizează element cu element:

```
>> [3 4 5; 4 5 6]/12
ans =
    0.2500    0.3333    0.4167
    0.3333    0.4167    0.5000
```

Funcţiile 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 funcţiile de bază pentru analiza datelor. Cel mai simplu mod de utilizare a 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)]
```

`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`	Diferenţa elementelor

Tabela 1.6: Funcţii de bază pentru analiza datelor

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

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

Dacă argumentele sunt matrice, aceste funcţii acţionează pe coloane. Astfel, `max` şi `min` returnează un vector ce conţine elementul maxim şi respectiv cel minim al fiecărei coloane, `sum` returnează un vector ce conţine sumele coloanelor, iar `sort` sortează elementele din fiecare coloană a unei matrice în ordine crescătoare. Funcţiile `min` şi `max` pot returna un al doilea argument care specifică în care componente sunt situate elementul minim şi 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
```

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

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

sau utilizând

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

Funcţiile `max` şi `min` pot fi făcute să acţioneze 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
```

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

Funcţia `diff` calculează diferenţe. Aplicată unui vector `x` de lungime `n` produce vectorul `[x(2)-x(1) x(3)-x(2) ... 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 relaţionali şi logici

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

Comparaţia între scalari produce 1 dacă relaţia este adevărată şi 0 în caz contrar. Comparaţiile sunt definite între matrice de aceeaşi dimensiune şi între o matrice şi un scalar, rezultatul fiind în ambele cazuri o matrice de 0 şi 1. La comparaţia matrice-matrice se compară perechile corespunzătoare de elemente, pe când la comparaţia 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` şi `B` sunt identice, se poate utiliza expresia `isequal(A,B)`:

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

Mai există şi alte funcţii logice înrudite cu `isequal` şi al căror nume începe cu `is`. O selecţie a lor apare în tabela 1.7; pentru o listă completă a se tasta `doc is`. Funcţia `isnan` este utilă deoarece testul `x == NaN` produce întotdeauna 0 (false), chiar dacă `x` este `NaN`! (Un `NaN` este prin definiţie diferit de orice şi nu are o relaţie de ordine cu nimic.)

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

```
>> x = [-1 1 1]; y = [1 2 -3];
>> x>0 & y>0
```

`ischar`	Testează dacă argumentul este şir de caractere(string)
`isempty`	Testează dacă argumentul este vid
`isequal`	Testează dacă tablourile sunt identice
`isfinite`	Testează dacă elementele unui tablou sunt finite
`isieee`	Testeză dacă maşina utilizează aritmetica IEEE
`isinf`	Testează dacă elementele unui tablou sunt `inf`
`islogical`	Testează dacă argumentul este un tablou logic
`isnan`	Test de NaN
`isnumeric`	Testează dacă argumentul este numeric
`isreal`	Testează dacă argumentul este tablou real
`issparse`	Testează dacă argumentul este tablou rar

Tabela 1.7: O selecţie de funcţii logice `is*`

```
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 funcţie: `xor(a,b)`. Operatorii logici `and`, `or`, `not` şi cei relaţionali pot fi apelaţi şi în formă funcţională: `and(a,b)`, ..., `eq(a,b)`, ... (vezi `help ops`).

Precedenţa operatorilor este rezumată în tabela 1.8 (vezi `help precedence`). MATLAB evaluează operatorii de precedenţă egală de la stânga la dreapta. Precedenţa se poate modifica cu ajutorul parantezelor.

De notat că versiunile MATLAB anterioare lui MATLAB 6 aveau aceeaşi precedenţă pentru `and` şi `or` (spre deosebire de majoritatea limbajelor de programare). MathWorks recomandă folosirea parantezelor pentru a garanta obţinerea rezultatelor identice în toate versiunile MATLAB.

Pentru matrice `all` returnează un vector linie ce conţine rezultatul lui `all` aplicat fiecărei coloane. De aceea `all(all(A=B))` este un alt mod de a testa egalitatea matricelor `A` şi `B`. Funcţia `any` lucrează analog; de exemplu, `any(any(A==B))` re-

Nivel de precedenţă	Operator
1 (cea mai mare)	transpusa (.´), putere(.^), transpusa conjugată complexă(’), putere matricială(^)
2	plus unar (+), minus unar (-), negaţie (∼)
3	înmulţire (.*), împărţire dreaptă (./), împărţire stângă (.\), înmulţire matricială (*), împărţire dreaptă matricială (/), împărţire stângă matricială (\)
4	adunare (+), scădere (-)
5	două puncte (:)
6	mai mic (<), mai mic sau egal (<=), mai mare (>), mai mare sau egal (>=), egal (==), diferit (˜=)
7	şi logic (&)
8 (cea mai mică)	sau logic (|)

Tabela 1.8: Precedenţa operatorilor

turnează 1 dacă A şi B au cel puţin un element egal şi 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 obţine elementele finite ale lui `x`,

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

şi să înlocuim componentele 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 obţinut din aşezarea coloanelor una peste alta (adică `A(:)`), şi 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` şi `j` ce conţin indicii de linie şi coloană ale elementelor nenule.

Rezultatele operatorilor logici şi ale funcţiilor logice din MATLAB sunt tablouri de elemente 0 şi 1, care sunt exemple de tablouri logice. Astfel de tablouri pot fi create şi prin aplicarea funcţiei `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 aceeaşi dimensiune ca şi `A`, extrage elementele lui `A` corespunzând elementelor lui `M` cu partea reală nenulă. Chiar dacă `i2` are aceleaşi elemente ca `i3` (şi la comparaţie 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: instrucţiunea `if`, instrucţiunea de ciclare `for`, instrucţiunea de ciclare `while` şi instrucţiunea `switch`. Cea mai simplă formă a instrucţiunii `if` este

`if` *expresie*
    instrucţiuni
`end`

unde secvenţa *instrucţiuni* este executată dacă părţile reale ale elementelor lui `expresie` sunt toate nenule. Secvenţa de mai jos interschimbă `x` şi `y` dacă `x` este mai mare decât `y`:

```
if x > y
   temp = y;
```

```
   y = x;
   x = temp;
end
```

Atunci când o instrucţiune `if` este urmată în aceeaşi linie de alte instrucţiuni, este nevoie de o virgulă pentru a separa `if`-ul de instrucţiunea 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 spaţiu între `else` şi `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` *condiţie1* & *condiţie2*”, *condiţie2* nu este evaluată dacă *condiţie1* este falsă (aşa-numită evaluare prin scurtcircuit). Acest lucru este util când evaluarea lui *condiţie2* ar putea da o eroare — probabil din cauza unei variabile nedefinite sau a unei depăşiri de indice.

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

`for` *variabilă = expresie*
instrucţiuni
`end`

De obicei, *expresie* este un vector de forma `i:s:j`. Instrucţiunile 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 notaţiei 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 creşterea lizibilităţii. Editorul-debuger-ul MATLAB poate realiza indentarea automată. Codul următor construieşte 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* i 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*
instrucţiuni
`end`
Secvenţa *instrucţiuni* 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
```

Execuţia unui ciclu `while` sau `for` poate fi terminată cu o instrucţiune `break`, care dă controlul primei instrucţiuni de după `end`-ul corespunzător. Construcţia `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 limbaje, 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.

Instrucţiunea `continue` cauzează trecerea controlului la execuţia unui ciclu `for` sau `while` următoarei iteraţii, sărind instrucţiunile rămase din ciclu. Un exemplu trivial este:

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

care afişează întregii de la 5 la 10.

Structura de control cu care încheiem este instrucţiunea `switch`. Ea constă din „`switch` *expresie*” urmată de o listă de instrucţiuni „`case` *expresie instrucţiuni*”, terminată opţional cu „`otherwise` *instrucţiuni*” şi 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
```

Funcţia `error` generează un mesaj de eroare şi opreşte execuţia curentă. Expresia ce urmează după `case` poate fi o listă de valori delimitate de acolade. Expresia 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
```

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

1.4.2. Fişiere M

Fişierele M din MATLAB sunt echivalentele programelor, funcţiilor, subrutinelor şi 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 utilitare, care pot fi utilizate repetat;
- schimbul de fişiere M.

Multe fişiere M scrise de entuziaşti pot fi obţinute de pe Internet, pornind de la pagina de web `http://www.mathworks.com`. Există şi grupul de ştiri `comp.soft-sys.matlab`, dedicat MATLAB. (Grupurile de ştiri pot fi citite în mai multe moduri, inclusiv printr-un web browser.) Se pot obţine detalii tastând `info` la prompterul MATLAB.

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

Fişiere M de tip script (sau fişiere de comenzi) — nu au nici un argument de intrare sau ieşire şi operează asupra variabilelor din spaţiul de lucru.

Fişiere M de tip funcţie — conţin o linie de definiţie `function` şi pot accepta argumente de intrare şi returna argumente de ieşire, iar variabilele lor interne sunt locale funcţiei (înafară de cazul când sunt declarate `global`).

Un fişier script permite memorarea unei secvenţe 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 cărţi de pică care sunt bine amestecate. Probabilitatea de a alege o carte particulară din pachet este 1/13. Acţiunea de extragere a unei cărţi se implementează prin generarea unui număr aleator. Jocul continuă prin punerea cărţii înapoi în pachet şi reamestecare până când utilizatorul apasă o tastă diferită de `r` sau s-a atins numărul de repetări (20).

```
%JOCCARTI
%Simularea unui joc de carti
```

```
rand('state',sum(100*clock));
for k=1:20
   n=ceil(13*rand);
   fprintf('Cartea extrasa: %3.0f\n',n)
   disp('  ')
   disp('Apasati r si Return pentru a continua')
   r=input('sau orice litera 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 fişier script încep cu simbolul % şi deci sunt linii de comentariu. Ori de câte ori MATLAB întâlneşte un % va ignora restul liniei. Aceasta ne permite să inserăm texte explicative care vor face fişierele M mai uşor de înţeles. Începând cu versiunea 7 se admit blocuri de comentarii, adică comentarii care să se întindă pe mai multe linii. Ele sunt delimitate prin operatorii %{ şi %}. Ei trebuie să fie singuri pe linie, ca în exemplul:

```
%{
Comentariu bloc
pe doua linii
%}
```

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

```
>> joccarti
Cartea extrasa:   7

Apasati r si Return pentru a continua
sau orice litera pentru a termina: r
Cartea extrasa:   3

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

Fişierele M de tip funcţie permit extinderea limbajului MATLAB prin scrierea de funcţii proprii care acceptă şi returnează argumente. Ele se pot utiliza în acelaşi mod ca funcţiile MATLAB existente, cum ar fi `sin`, `eye`, `size`, etc.

Sursa MATLAB 1.1 dă o funcţie simplă care calculează media şi abaterea medie pătratică a unei selecţii (vector). Acest exemplu ilustrează unele facilităţi ale

Sursa MATLAB 1.1 Funcţia `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);
```

funcţiilor. Prima linie începe cu cuvântul cheie `function` urmat de argumentele de ieşire, `[med,abmp]` şi de simbolul `=`. În dreapta `=` urmează numele funcţiei, `stat`, urmat de argumentele de intrare, în cazul nostru `x`, între paranteze. (În general, putem avea orice număr de argumente de intrare şi de ieşire.) Numele de funcţie trebuie să fie la fel ca al fişierului `.m` în care funcţia este memorată – în cazul nostru `stat.m`.

A doua linie a unui fişier funcţie se numeşte linie H1 sau help 1. Se recomandă ca ea să aibă următoarea formă: să înceapă cu un %, urmat fără nici un spaţiu de numele funcţiei cu litere mari, urmat de unul sau mai multe spaţii şi 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” şi “a”. Când se tastează `help nume_functie`, toate liniile, de la prima linie de comentariu pâna la prima linie care nu este de comentariu (de obicei o linie goală, pentru lizibilitatea codului sursă) sunt afişate pe ecran. Deci, aceste linii descriu funcţia şi argumentele sale. Se convine ca numele de funcţie şi 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* funcţiilor utilizator în acest mod, oricât de scurte ar fi. Este util ca în liniile de comentariu din text sa apară data scrierii funcţiei şi datele când s-au făcut modificări. Comanda `help` lucrează similar şi pe fişiere script.

Funcţia `stat` se apelează la fel ca orice funcţie 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 funcţie mai complicată este `sqrtn`, ilustrată în sursa 1.2. Dându-se $a > 0$, ea calculează $\sqrt{a}$ cu metoda lui Newton,

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

afişând şi iteraţiile. Dăm exemple de utilizare:

```
>> [x,it]=sqrtn(2)
 k             x_k            er. relativa
 1:  1.5000000000000000e+000  3.33e-001
 2:  1.4166666666666665e+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. relativa
 1:  1.5000000000000000e+000  3.33e-001
 2:  1.4166666666666665e+000  5.88e-002
 3:  1.4142156862745097e+000  1.73e-003
 4:  1.4142135623746899e+000  1.50e-006
x =
    1.4142
```

Acest fişier M utilizează comanda `return`, care dă controlul apelantului. Spre deosebire de alte limbaje de programare, nu este necesar să se pună `return` la sfârşitul unei funcţii sau al unui script. Funcţia `nargin` returnează numărul de argumente de intrare cu care funcţia a fost apelată şi permite atribuirea de valori implicite argumentelor nespecificate. Dacă apelul lui `sqrtn` nu a furnizat o valoare pentru `tol`, atunci `tol` primeşte valoarea `eps`. Numărul de argumente la ieşire este returnat de funcţia `nargout`.

Un fişier M de tip funcţie poate conţine alte funcţii, numite subfuncţii, care pot să apară în orice ordine după funcţia principală (sau primară). Subfuncţiile sunt vizibile numai din funcţia principală sau din alte subfuncţii. Ele realizează calcule care

Sursa MATLAB 1.2 Funcţia `sqrtn`

```
function [x,iter] = sqrtn(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             x_k            er. relativa\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\', iter, x, xdiff)
    if xdiff <= tol, return, end
end
error('Nu s-a atins precizia dupa 50 de iteratii.')
```

trebuie separate de funcţia principală, dar nu sunt necesare în alte fişiere M, sau supraîncarcă funcţii cu acelaşi nume (subfuncţiile au prioritate mai mare). Help-ul pentru o subfuncţie se poate specifica punând numele funcţiei urmat de "/" şi numele subfuncţiei.

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

Cele mai multe funcţii MATLAB sunt fişiere M păstrate pe disc, dar există şi funcţii predefinite conţinute în interpretorul MATLAB. Calea MATLAB (MATLAB

path) este o listă de directori care specifică unde caută MATLAB fişierele M. Un fişier M este disponibil numai dacă este pe calea MATLAB. Drumul poate fi setat şi modificat prin comenzile `path` şi `addpath`, sau prin utilitarul (fereastra) path Browser, care se apelează din opţiunea de meniu File-Set Path sau tastând `pathtool`. Un script (dar nu şi o funcţie) care nu este pe calea de căutare se poate executa cu `run` urmat de calea completă până la fişierul M. Un fişier M se poate afişa pe ecran cu comanda `type`.

Un aspect important al MATLAB este dualitatea comenzi-funcţii. Înafară de forma clasică, nume, urmat de argumente între paranteze, funcţiile pot fi apelate şi sub forma nume, urmat de argumente separate prin spaţii. MATLAB presupune în al doilea caz că argumentele sunt şiruri de caractere. De exemplu apelurile `format long` şi `format('long')` sunt echivalente. Dar, în exemplul,

```
>> sqrt 2
ans =
    7.0711
```

MATLAB interpretează 2 ca un şir şi `sqrt` se aplică valorii ASCII a lui 2, şi anume 50.

MATLAB 7 permite definirea de funcţii imbricate , adică funcţii conţinute în corpul altor funcţii. În exemplul care urmează, funcţia `F2` este imbricată în funcţia `F1`:

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

Ca orice altă funcţie, o funcţie imbricată are propriul său spaţiu de lucru în care se memorează variabilele pe care le utilizează. Ea are de asemenea acces la spaţiul de lucru al tuturor funcţiilor în care este imbricată. Astfel, de exemplu, o variabilă care are o valoare atribuită ei de funcţia exterioară poate fi citită şi modificată de o funcţie imbricată la orice nivel în funcţia exterioară. Variabilele create într-o funcţie imbricată pot fi citite sau modificate în orice funcţie care conţine funcţia imbricată.

1.4.3. Argumente funcţie

În multe probleme, cum ar fi integrarea numerică, rezolvarea unor ecuaţii operatoriale, minimizarea unei funcţii, este nevoie ca o funcţie să fie transmisă ca argument unei alte funcţii. Aceasta se poate realiza în mai multe feluri, depinzând de modul în

care funcţia apelată a fost scrisă. Vom ilustra aceasta cu funcţia `ezplot`, care reprezintă grafic funcţia $f(x)$ peste domeniul implicit $[-2\pi, 2\pi]$. Un prim mod este transmiterea funcţiei printr-o construcţie numită *function handle*. Acesta este un tip de date MATLAB care conţine toate informaţiile necesare pentru a evalua o funcţie. Un function handle poate fi creat punând caracterul @ în faţa numelui de funcţie. Astfel, dacă `fun` este un fişier M de tip funcţie de forma cerută de `ezplot`, atunci putem tasta

```
ezplot(@fun)
```

`fun` poate fi numele unei funcţii predefinite:

```
ezplot(@sin)
```

Numele unei funcţii poate fi transmis ca un şir de caractere:

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

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

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

sau ca obiect inline

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

Un obiect inline este o funcţie definită printr-un şir şi care poate fi atribuită unei variabile şi apoi evaluată:

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

MATLAB determină şi ordonează argumentele unei funcţii inline. Dacă acest lucru nu este satisfăcător, argumentele se pot defini şi ordona explicit, transmiţând 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)
```

Sursa MATLAB 1.3 Funcţia `fd_deriv`

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

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

Începând cu versiunea 7, MATLAB permite funcţii anonime. Ele pot fi definite în linii de comandă, fişiere M de tip funcţie sau script şi nu necesită un fişier M. Sintaxa pentru crearea unei funcţii anonime este

```
f = @(listaarg)expresie
```

Instrucţiunea de mai jos crează o funcţie anonimă care ridică argumentul ei la pătrat:

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

Dăm şi un exemplu de utilizare

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

Pentru a evalua în corpul unei funcţii o funcţie transmisă ca parametru se utilizează funcţia `feval`. Sintaxa ei este `feval(fun,x1,x2,...,xn)`, unde `fun` este funcţia, iar `x1,x2,...,xn` sunt argumentele sale. Să considerăm funcţia `fd_deriv` din sursa 1.3. Această funcţie aproximează derivata funcţiei dată ca prim argument cu ajutorul diferenţei 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 funcţia `sqrtn` (sursa MATLAB 1.2) în locul funcţie predefinite `sqrt`:

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

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

```
>> 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 şir vom insera

```
f=fcnchk(f);
```

la începutul funcţiei (în acest mod lucrează `ezplot` şi alte funcţii MATLAB, vezi [15] pentru exemple).

Este uneori necesar să se „vectorizeze" un obiect inline sau o expresie de tip şir, adică să se convertească înmulţirile, ridicările la putere şi împărţirile în operaţii în sens tablou, astfel ca să se poată utiliza argumente vectori şi matrice. Acest lucru se poate realiza cu funcţia `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 şirul `f`.

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

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

Dacă funcţia nu are nici un argument apelul ei are forma

```
fhandle()
```

Linia a doua a funcţiei `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 situaţii o funcţie trebuie să accepte sau să returneze un număr variabil, posibil nelimitat, de argumente. Aceasta se poate realiza utilizând funcţiile `varargin` şi `varargout`. Să presupunem că dorim să scriem o funcţie `companb` ce construieşte matricea companion pe blocuri, de dimensiune $mn \times mn$, a matricelor $n \times n$ A_1, A_2, ..., A_m:

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

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

Sursa MATLAB 1.4 Funcţia `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 şi 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 conţine 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 1×3 ale cărui elemente sunt matrice 2×2 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 ieşire 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 ieşire se determină cu `nargout` şi apoi se crează tabloul de celule `varargout` ce conţine ieşirea dorită. Ilustrăm cu apelurile funcţiei `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 Funcţia `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 funcţii sunt locale spaţiului de lucru al acelei funcţii. Uneori este convenabil să creăm variabile care există în mai multe spaţii de

lucru, eventual chiar cel principal. Aceasta se poate realiza cu ajutorul instrucţiunii `global`. Ca exemplu dăm codurile pentru funcţiile `tic` şi `toc` (cu unele comentarii prescurtate). Aceste funcţii pot contoriza timpul, gestionând un cronometru. Variabila globală TICTOC este vizibilă în ambele funcţii, dar este invizibilă în spaţiul de lucru de bază (nivel linie de comandă sau script) sau în orice altă funcţie 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 funcţii, variabilele globale vor apare înaintea primei apariţii a unei variabile locale, ideal la începutul fişierului. Se convine ca numele de variabile globale să fie scrise cu litere mari, să fie lungi şi sugestive.

1.4.6. Recursivitate

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

Funcţia `koch` din sursa MATLAB 1.6 utilizează recursivitatea pentru a desena o curbă Koch şi este inspirată din [5]. Construcţia 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 construcţii unei linii orizontale. Imaginea din dreapta jos ne arată ce se întâmplă când fiecare din aceste linii este prelucrată. Imaginile din stânga şi dreapta jos ne arată următoarele două niveluri de recursivitate.

Funcţia `koch` are trei argumente de intrare. Primele două, `pl` şi `pr` dau coordonatele (x, y) ale capetelor segmentului curent şi al treilea, `level`, indică nivelul de

Sursa MATLAB 1.6 Funcţia `koch`

```
function koch(pl,pr,level)
%KOCH   Curba Koch generata recursiv.
%       Apel KOCH(PL, PR, LEVEL) unde punctele PL şi PR
%       sunt extremitatea stângă şi 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(pmidl,ptop,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
```

recursivitate cerut. Dacă `level` = 0 se desenează un segment; altfel `koch` se autoapelează de patru ori cu `level` decrementat cu 1 şi cu puncte care definesc capetele celor patru segmente mai scurte.

Figura 1.1 a fost obţinută cu următorul cod:

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

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 obţine o curbă numită fulgul de zăpadă al lui Koch (Koch's snowflake). Codul este

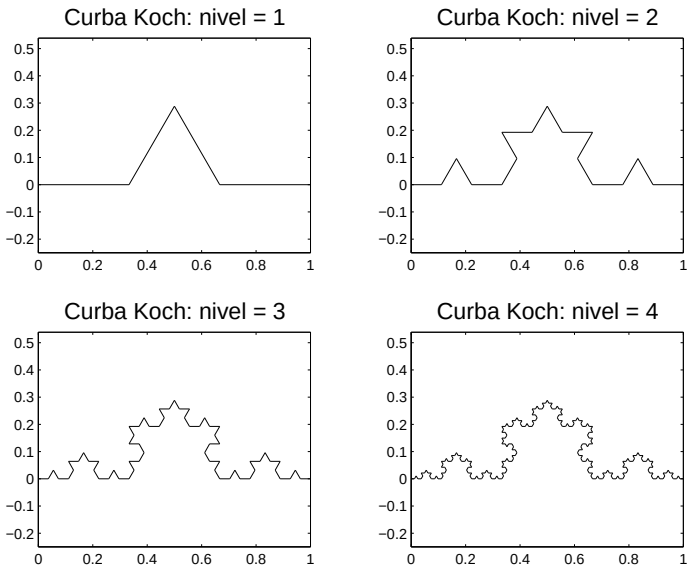


Figura 1.1: Curbe Koch create cu funcţia `koch`.

dat în sursa 1.7. Funcţia `snowflake` acceptă la intrare numărul de laturi `edges` şi nivelul de recursivitate `level`. Valorile implicite ale acestor parametrii sunt 7 şi respectiv 4. În figura 1.2 se dau două exemple.

Pentru alte exemple de recursivitate, a se vedea funcţiile MATLAB `quad` şi `quadl` şi sursele din secţiunile **??** şi **??**, utilizate pentru integrarea numerică.

1.4.7. Alte tipuri numerice

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

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

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

```
>> ones(2,2,'int8')
```

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
```

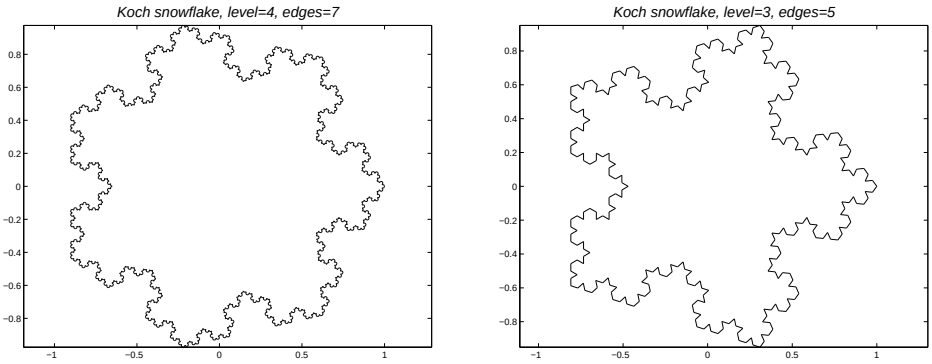


Figura 1.2: Fulgi Koch obţinuţi cu `snowflakes`

returnează o matrice 2×2 cu elemente de tipul `int8`

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

Funcţiile care definesc tipuri de date întregi au acelaşi nume ca şi tipul. De exemplu

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

atribuie lui `x` valoarea 5 reprezentată sub forma unui întreg pe 8 biţi. Funcţia `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
```

Funcţiile `intmax` şi `intmin`, având ca argument un nume de tip întreg, returnează cea mai mare şi 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 operaţii aritmetice între întregi de acelaşi 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 situaţii se dă un mesaj de avertisment care se poate inhiba (sau reactiva) cu funcţia `intwarning`.

Dacă `A` şi `B` sunt tablouri de tip întreg, împărţirile în sens tablou, `A./B` şi `A.\B`, se realizează în aritmetica în dublă precizie, iar rezultatul se converteşte 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 operaţiile binare în care un operand este un tablou de tip întreg iar celălat este un scalar de tip double, MATLAB realizează operaţia element cu element în dublă precizie şi converteşte 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 şi apoi converteşte rezultatul în `int8`; al doilea şi al treilea element din tablou după înmulţirea în dublă precizie sunt `1.6` şi `2.4`, care sunt routunjite la cel mai apropiat întreg, 2.

Tipul `single` va fi descris în §**??**.

Pentru detalii asupra tipurilor nondouble vezi [12, 13].

1.4.8. Controlul erorilor

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

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

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

În MATLAB 7 funcţia `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 conţine mesajul de eroare şi identificatorul acestuia.

Exemplul următor determină cauza erorii la înmulţirea a două matrice utilizând `try-catch` şi `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 afişază 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 funcţia 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` şi [13].

1.5. Toolbox-urile Symbolic

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

Toolboxul Symbolic Math defineşte 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 şir a simbolului. Toolbox-ul Symbolic Math utilizează obiectele simbolice pentru a reprezenta variabile simbolice, expresii şi matrice. Aritmetica cu care se operează asupra obiectelor simbolice este implicit cea raţională.

Obiectele simbolice se construiesc cu ajutorul declaraţiei `sym`. De exemplu, instrucţiunea

```
x = sym('x');
```

produce o variabilă simbolică numită `x`. Se pot combina mai multe declaraţii de acest tip folosind forma `syms`:

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

Exemplele din această secţiune presupun 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 obţine 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 obţine `g` ci o expresie echivalentă, dar după simplificare cu comanda `simple(diff(int_g,x))` se obţine un şir de mesaje care informează utilizatorul asupra regulilor utilizate şi în final

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

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

```
>> 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`.

Substituţii şi 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` şi `c=4`:

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

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

Funcţia `simplify` este o funcţie puternică care aplică diverse tipuri de identităţi pentru a aduce o expresie la o formă mai „simplă”. Exemplu:

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

Funcţia `simple` este o funcţie neortodoxă care are drept scop obţinerea 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 ieşire are rolul de a inhiba mesajele lungi referitoare la procesul de simplificare.

A se vedea `help sym/simplify` şi `help sym/simple` sau `doc sym/simplify` şi `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 înjurul 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)
                          2        3         4
                1 + x + 1/2 x  + 1/6 x  + 1/24 x
```

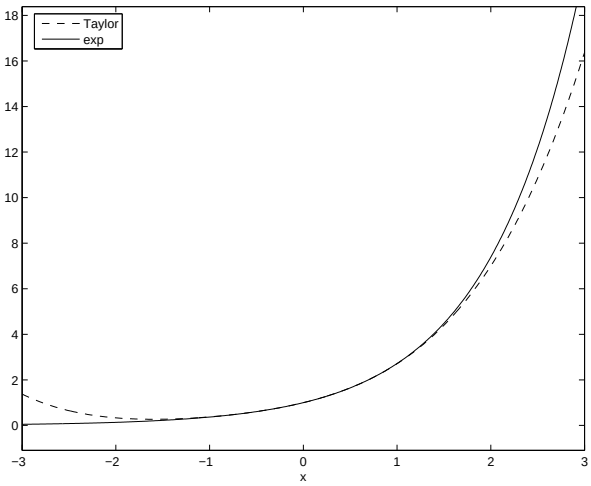


Figura 1.3: Comparaţie între exponenţială şi dezvoltarea sa Taylor

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 şi grafic cele două aproximante cu `ezplot` (a se vedea capitolul 2 pentru o descriere a facilităţilor 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. Funcţia `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 şi modul de utilizare a comenzii `limit` a se vedea `help sym/limit` sau `doc sym/limit`. Ne vom limita la a da două exemple simple. Primul calculează $\lim_{x\to 0} \frac{\sin x}{x}$:

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

Al doilea exemplu calculează limitele laterale ale funcţiei 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 ecuaţiilor. Toolbox-ul Symbolic Math poate rezolva ecuaţii şi sisteme de ecuaţii, inclusiv neliniare. A se vedea `help sym/solve` sau `doc sym/solve`. Vom da câteva exemple. Înainte de rezolvarea unei ecuaţii vom şterge memoria, vom defini simbolurile şi ecuaţiile. (Este o bună practică de a şterge memoria înainte de rezolvarea unei noi probleme pentru a evita efectele provocate de valorile precedente ale unor variabile.)

Vom începe cu rezolvarea ecuaţiei 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 ecuaţii să se obţină mai multe soluţii (în exemplul de mai sus s-au obţinut 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 ieşire este importantă.

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

```
>> clear, syms x y
>> eq1='y=2*exp(x)'; eq2='y=3-x^2';
>> [x,y]=solve(eq1,eq2,x,y)
x =
.36104234240225080888501262630700
y =
2.86964842699269588876157155521484
```

Ultimul exemplu rezolvă ecuaţia trigonometrică $\sin x = \frac{1}{2}$. Aceasta are o infinitate de soluţii. Secvenţa de comenzi

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

dă doar soluţia

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

Pentru soluţiile 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 =
2.617993877991494365385536152732 9
```

Rezultatul este un şir 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
```

Funcţia `maple` trimite comenzi Maple nucleului Maple. A se consulta help-urile corespunzătoare şi documentaţia.

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

- numeric – operaţiile MATLAB în virgulă flotantă;
- raţional – aritmetica simbolică exactă Maple;
- VPA – aritmetica cu precizie variabilă Maple (variable precision arithmetic).

Aritmetica de precizie variabilă se realizează cu ajutorul funcţiei `vpa`. Numărul de cifre este controlat de variabila Maple Digits. Funcţia `digits` afişează 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, instrucţiunile MATLAB

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

folosesc modul de calcul numeric pentru a produce

```
ans =
   0.83333333333333
```

Cu toolbox-ul Symbolic Math, intrucţiunea

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

va produce, folosind calculul simbolic

```
ans =
 5/6
```

Tot în toolbox, cu aritmetica cu precizie variabilă, instrucţiunile

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

au ca rezultat

```
>> ans =
 .83333333333333333333333
```

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

În exemplul următor

```
>> digits(32)
>> clear, phi1=vpa((1+sqrt(5))/2)
phi1 =
1.6180339887498949025257388711907
>> phi2=vpa('(1+sqrt(5))/2'), diff=phi1-phi2
phi2 =
1.6180339887498948482045868343656
diff =
.543211520368251e-16
```

discrepanţa dintre `phi1` şi `phi2` se explică prin aceea că prima atribuire face calculele în dublă precizie şi converteşte rezultatul în vpa, iar a doua utilizează şirul şi face toate calculele în vpa.

Pentru detalii suplimentare asupra Symbolic Math Toolbox trimitem cititorul la [11].

Probleme

Problema 1.1. Calculaţi 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. Scrieţi un fişier M de tip funcţie care evaluează dezvoltarea MacLaurin a funcţiei $\ln(x+1)$:

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

Convergenţa are loc pentru $x \in [-1, 1]$. Testaţi funcţia MATLAB pentru valori ale lui x din $[-0.5, 0.5]$ şi verificaţi ce se întâmplă când x se apropie de -1 sau 1.

Problema 1.3. Scrieţi un script MATLAB care citeşte un întreg şi determină scrierea sa cu cifre romane.

Problema 1.4. Implementaţi algoritmul lui Euclid în MATLAB.

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

Problema 1.6. Scrieţi 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? Rezolvaţi prin însumare şi utilizând formula care dă pe S_n.

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

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

folosind toolbox-ul Symbolic.

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

Problema 1.10. Scrieţi o funcţie MATLAB care primeşte la intrare coordonatele vârfurilor unui triunghi şi subdivizează recursiv triunghiul în patru triunghiuri, după mijloacele laturilor. Subdivizarea continuă până când se atinge un nivel dat.

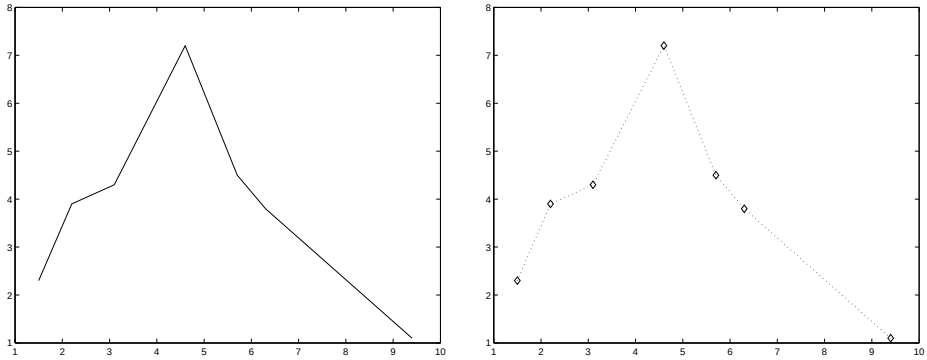
Problema 1.11. Scrieţi o funcţie MATLAB care să extragă dintr-o matrice dată A o parte care să fie diagonală pe blocuri, unde blocurile sunt de dimensiune dată, iar colţul din stânga sus al fiecărui bloc este situat pe diagonala principală.

CAPITOLUL 2

Grafică în MATLAB

Cuprins

MATLAB are facilităţi grafice puternice şi versatile. Se pot genera grafice şi figuri relativ uşor, iar atributele lor se pot modifica cu uşurinţă. Nu ne propunem să fim exhaustivi, ci dorim doar să introducem cititorul în facilităţile grafice MATLAB care vor fi necesare în continuare. Figurile existente pot fi modificate uşor cu utilitarul Plot Editor. Pentru utilizarea sa a se vedea `help plotedit` şi meniul Tools sau bara din ferestrele figurilor. Celor care doresc detalii sau să aprofundeze subiectul le recomandăm [5, 9, 14, 16].



(a) Implicit (b) Neimplicit

Figura 2.1: Grafice x-y simple

2.1. Grafice bidimensionale

2.1.1. Grafice de bază

Funcţia 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 obţine imaginea din stânga figurii 2.1(a), în care punctele `x(i)`, `y(i)` sunt unite în secvenţă. 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 implicite ale unor facilităţi cum ar fi domeniul pentru axele x şi y, spaţiile dintre diviziunile de pe axe, culoarea şi tipul liniei.

Mai general, în loc de `plot(x,y)`, putem utiliza `plot(x,y,`*şir*`)`, unde *şir* este un şir de caractere ce controlează culoarea, marcajul şi stilul de linie. De exemplu, `plot(x,y,'r*--')` ne spune că în fiecare punct `x(i)`, `y(i)` se va plasa un asterisc roşu, punctele fiind unite cu o linie roşie întreruptă. `plot(x,y,'+y')` marchează punctele cu un plus galben, fără a le uni cu nici o linie. Tabela 2.1 dă toate opţiunile disponibile. Imaginea din partea dreaptă a figurii 2.1 s-a obţinut cu `plot(x,y,'kd:')`, care desenează o linie punctată neagră marcată cu romburi. Cele trei elemente din *şir* pot apare în orice ordine; de exemplu, `plot(x,y,'ms--')` şi `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 aceeaşi figură graficele pentru `x(i)`, `y(i)` şi `b(i)`, `c(i)` cu linie continuă verde şi respectiv cu linie întreruptă roşie.

	Culoare
r	roşu
g	verde
b	albastru
c	cian
m	magenta
y	galben
k	negru
w	alb

	Marcaj
o	cerc
*	asteric
.	punct
+	plus
x	ori
s	pătrat
d	romb
^	triunghi în sus
v	triunghi în jos
>	triunghi dreapta
<	triunghi stânga
p	pentagramă (stea cu 5 colţuri)
h	hexagramă (stea cu 6 colţuri)

	Stil de linie
-	continuă (implicit)
--	întreruptă
:	punctată
-.	linie-punct

Tabela 2.1: Opţiuni pentru comanda `plot`

Comanda `plot` acceptă şi argumente matriciale. Dacă `x` este un vector de dimensiune m şi `Y` este o matrice $m \times n$, `plot(x,Y)` suprapune graficele obţinute din `x` şi fiecare coloană a lui `Y`. Similar, dacă `X` şi `Y` sunt matrice de aceeaşi dimensiune, `plot(X,Y)` suprapune graficele obţinute din coloanele corespunzătoare ale lui `X` şi `Y`. Dacă argumentele lui `plot` nu sunt reale, atunci părţile imaginare sunt în general ignorate. Singura excepţie este atunci când `plot` este apelat cu un singur argument. Dacă `Y` este complex, `plot(Y)` este echivalent cu `plot(real(Y),imag(Y))`. în cazul când `Y` este real, `plot(Y)` desenează graficul obţinut luând pe abscisă indicii punctelor şi pe ordonată `Y`.

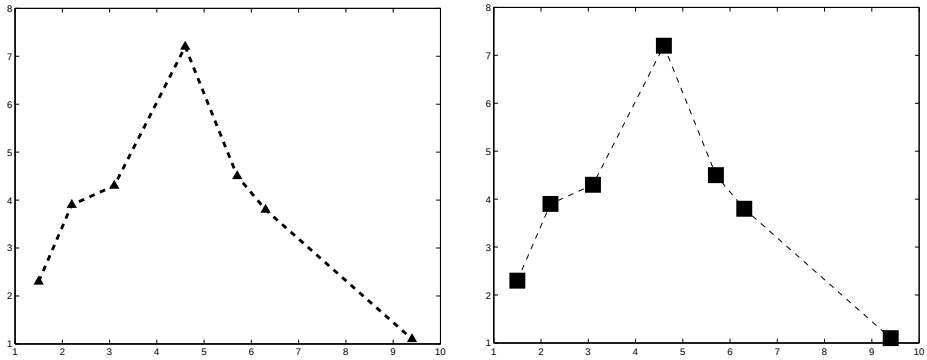
Atributele se pot controla furnizând argumente suplimentare lui `plot`. Proprietăţile `Linewidth` (implicit 0.5 puncte) şi `MarkerSize` (implicit 6 puncte) pot fi specificate în puncte, unde un punct este 1/72 inch. De exemplu, comanda

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

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

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

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



(a) (b)

Figura 2.2: Două grafice x-y neimplicite

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

iar cel din dreapta cu comanda

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

Funcţia `loglog`, spre deosebire de `plot`, scalează axele logaritmic. Această facilitate este utilă pentru a reprezenta relaţii 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 şi deci pe o scară log-log valorile vor fi situate pe o dreaptă cu panta 3. Vom verifica aceasta reprezentând restul şi dreapta de referinţă cu panta 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
```

În acest exemplu s-au utilizat comenzile `title`, `xlabel` şi `ylabel`. Aceste funcţii afişează şirul parametru de intrare deasupra imaginii, axei x şi respectiv axei y. Comanda `box off` elimină caseta de pe marginea graficului curent, lăsând doar axele de coordonate. Dacă `loglog` primeşte şi valori nepozitive, MATLAB va da un avertisment şi va afişa doar datele pozitive. Funcţiile înrudite `semilogx` şi `semilogy`, scalează doar una din axe.

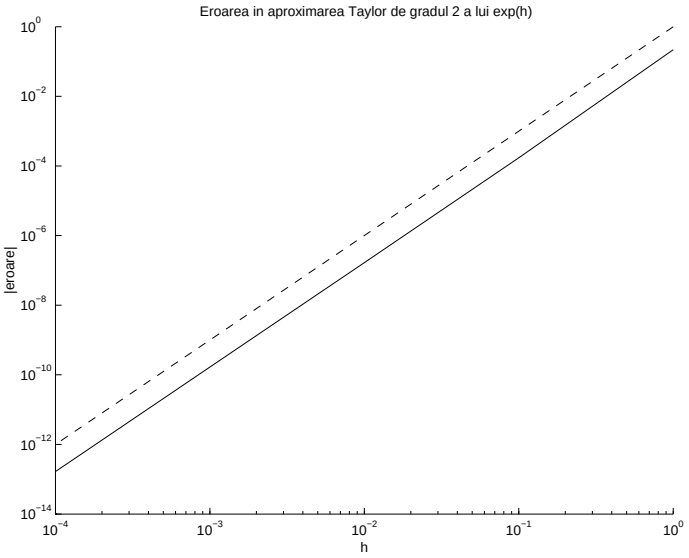


Figura 2.3: Exemplu cu `loglog`

Dacă o comandă de afişare 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 comenzii `polar(t,r)`, unde `t` este unghiul polar, iar `r` este raza polară. Se poate folosi şi un parametru suplimentar `s`, cu aceeaşi semnificaţie ca la `plot`. Graficul unei curbe în coordonate polare, numită cardioidă, şi care are ecuaţia

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

unde a este o constantă reală dată, apare în figura 2.4 şi se obţine cu secvenţa:

```
t=0:pi/50:2*pi;
a=2; r=a*(1+cos(t));
polar(t,r)
title('cardioida')
```

Funcţia `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 şi ultimul se uneşte cu primul. Culoarea `c` se poate da şi sub forma unui triplet RGB, `[r g b]`. Elementele `r`, `g` şi `b`, care trebuie să fie scalari din [0,1], determină nivelul de

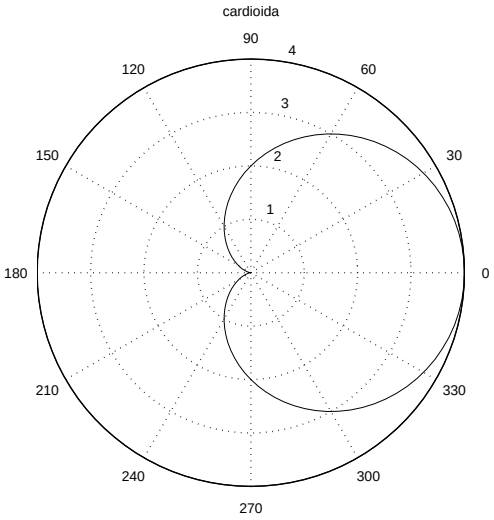


Figura 2.4: Grafic în coordonate polare — cardioidă

roşu, verde şi 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 proporţii egale de roşu, verde şi albastru se obţin nuanţe 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` şterge 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 atribute ale unei figuri pot fi modificate interactiv, după afişarea 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`).

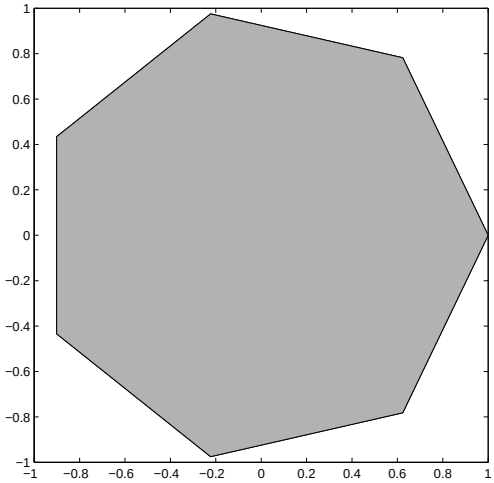


Figura 2.5: Exemplu de utilizare `fill`

`axis([xmin xmax ymin ymax])`	Setează limitele axelor x şi y
`axis auto`	Returnează limitele implicite
`axis equal`	Egalează unităţile pe axele de coordonate
`axis off`	Elimină axele
`axis square`	Face caseta axelor pătrată (cubică)
`axis tight`	Setează limitele axelor egale cu limitele datelor
`xlim([xmin xmax])`	Setează limitele pe axa x
`ylim([ymin,ymax])`	Setează limitele pe axa y

Tabela 2.2: Unele comenzi pentru controlul axelor

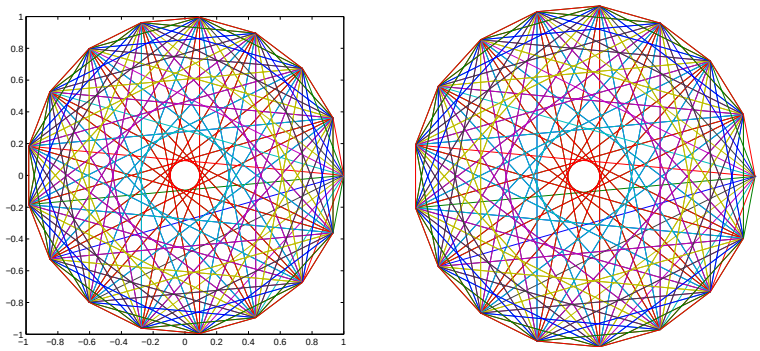


Figura 2.6: Utilizarea `axis off`

2.1.2. Axe şi adnotarea

Diversele aspecte ale unui grafic pot fi controlate cu comanda `axis`. Unele opţiuni se dau în tabela 2.2. Axele pot fi eliminate cu `axis off`. Raportul dintre unitatea pe x şi 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 obţinut 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 şi respectiv y. Pentru a reveni la setările implicite, pe care MATLAB le alege automat în funcţie de datele care urmează a fi reprezentate, se utilizează `axis auto`. Dacă se doreşte 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 cu `xlim([xmin xmax])` şi `ylim([ymin ymax])`.

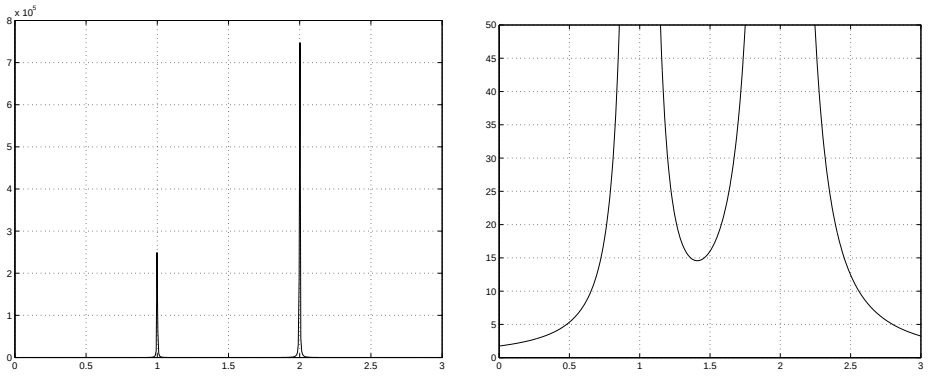
Exemplul următor reprezintă funcţia $1/(x-1)^2 + 3/(x-2)^2$ pe intervalul [0,3]:

```
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 şi verticale care pornesc de la diviziunile axelor. Rezultatul se poate vedea în figura 2.7(a). Datorită singularităţilor din $x = 1, 2$ graficul nu dă prea multă informaţie. Totuşi, executând comanda

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

se obţine figura 2.7(b), care se focalizează asupra părţii interesante a primului grafic.



(a) (b)

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

Exemplul următor reprezintă epicicloida

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

pentru $a = 12$ şi $b = 5$.

```
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)')
```

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

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

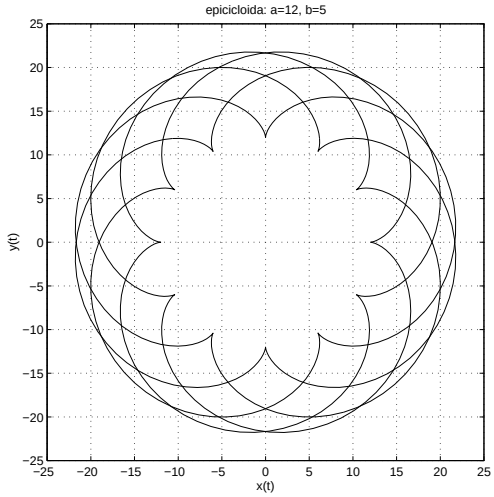


Figura 2.8: Epicicloidă

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

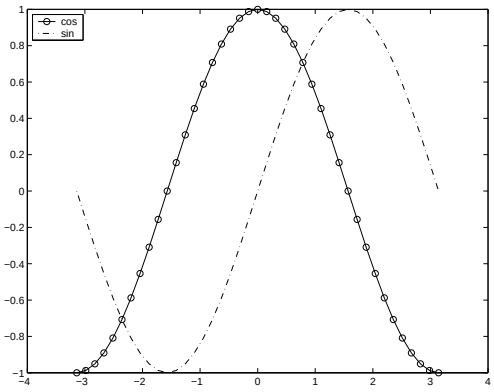


Figura 2.9: Grafic cu legendă

Textele se pot include în grafice cu ajutorul comenzii `text(x, y, s)`, unde `x` şi `y` sunt coordonatele textului, iar `s` este un şir de caractere sau o variabilă de tip şir. Începând cu versiunea 5, MATLAB permite introducerea în interiorul parametrului `s` a unor construcţii TEX, de exemplu _ pentru indice, ^ pentru exponent, sau litere greceşti (\alpha, \beta, \gamma, etc.). De asemenea, anumite atribute ale textului,

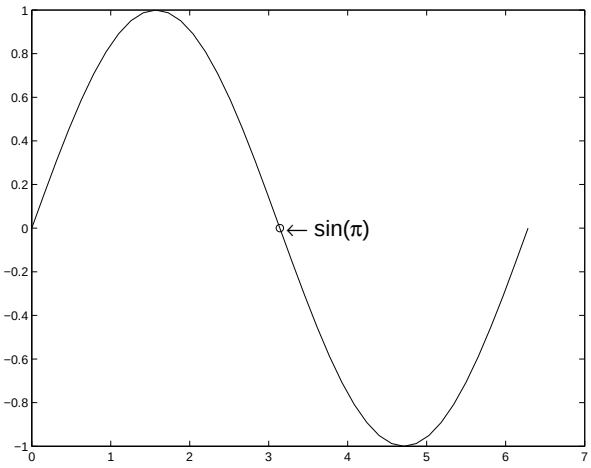


Figura 2.10: Exemplu de utilizare `text`

cum ar fi tipul font-ului, dimensiunea şi 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 şirul $\sin(\pi)$. Rezultatul apare în figura 2.10. Aceste facilităţi se pot utiliza şi în titluri, legende sau etichete ale axelor, care sunt obiecte de tip text. Începând cu MATLAB 7 primitivele text suportă un subset puternic LaTeX. Proprietatea corespunzătoare se numeşte `Interpreter` şi poate avea valorile `TeX`, `LaTeX` sau `none`. Pentru un exemplu de utilizare a macrourilor LaTeX a se vedea script-ul `graphLegendre.m`, pagina **??**.

2.1.3. Mai multe grafice pe aceeaşi figură

Funcţia MATLAB `subplot` permite plasarea mai multor imagini pe o grilă în aceeaşi figură. Utilizând `subplot(mnp)`, sau echivalent, `subplot(m,n,p)`, fereastra figurii se împarte într-un tablou `m`×`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 doua ş.a.m.d. De exemplu, `subplot(425)` împarte fereastra figurii într-o matrice 4×2 de regiuni şi ne spune că toate comenzile 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 poziţia (4,1) devine activă. Vom da în continuare mai multe exemple.

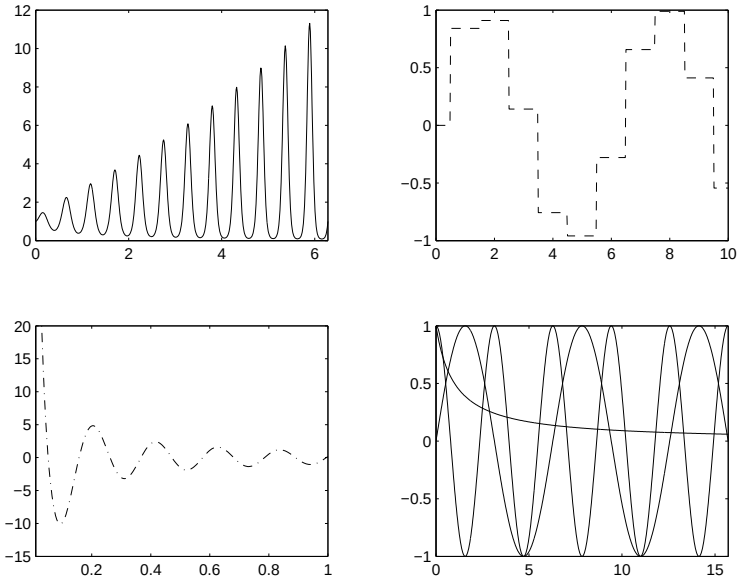


Figura 2.11: Exemple cu `subplot` şi `fplot`

Pentru cei nefamiliarizaţi cu grafica MATLAB, comanda de reprezentare grafică a unei funcţii, `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 funcţiei $\exp(\sqrt{x}\sin 12x)$ pe intervalul $0 \leq x \leq 2\pi$. În al doilea apel se selectează stilul de linie întreruptă cu `'--'`, în locul celei implicite, continue. Argumentul `[0.01 1 -15 20]` din cel de-al treilea apel setează limitele pe axele x şi y la $0.01 \leq x \leq 1$ şi respectiv $-15 \leq y \leq 20$, iar `'-.'` utilizează stilul de linie linie-punct. Ultimul exemplu arată cum se pot reprezenta mai multe funcţii la un singur apel.

Sintaxa generală a lui `fplot` este

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

cu semnificaţia:

- `fun` este funcţia de reprezentat (function handle, obiect inline sau expresie);

- `lims` dă limitele pe axele x şi/sau y;
- `tol` este eroarea relativă (inplicit 2×10^{-3});
- se folosesc cel puţin `N+1` puncte la generarea graficului;
- `LineSpec` determină tipul de linie;
- `p1, p2, ...` sunt parametrii adiţionali transmişi lui `fun`, care trebuie să aibă parametrii de intrare `x`, `p1`, `p2`, ...

Este posibil să se obţină grile neregulate de imagini apelând `subplot` cu şabloane 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 secţiuni, tabela 2.3 dă cele mai populare funcţii MATLAB pentru grafice 2D. Funcţiile cu numele de forma `ezfun` sunt variante uşor de utilizat (easy) ale funcţiilor `fun` corespunzătoare.

2.2. Grafice tridimensionale

Fucţia `plot3` este un analog tridimensional al lui `plot`. Figura 2.13 a fost obţinută 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('{\itExemplu} plot3','FontSize',14)
```

Acest exemplu utilizează funcţiile `xlabel`, `ylabel` şi `title`, precum şi analogul lor `zlabel`. De notat şi utilizarea notaţiei TEX în titlu, pentru a produce text

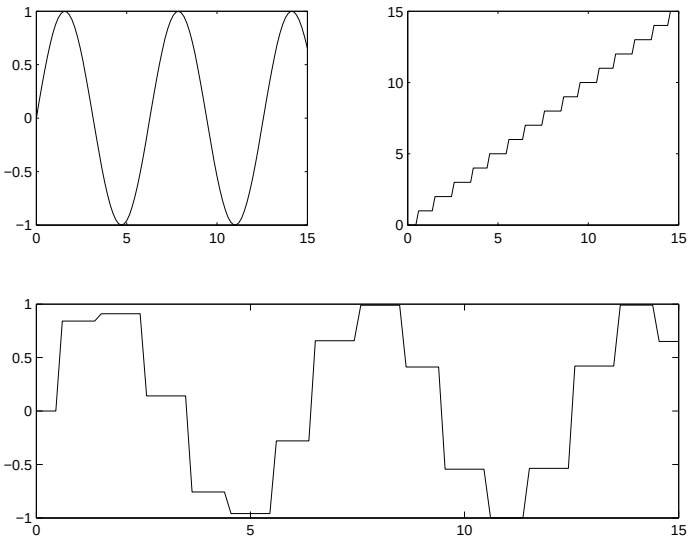


Figura 2.12: Grile neregulate produse cu `subplot`

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

Tabela 2.3: Funcţii pentru grafice 2D

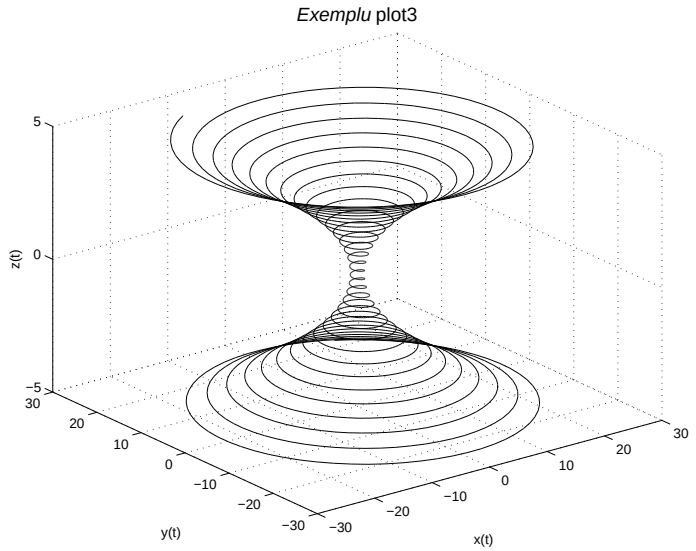


Figura 2.13: Grafic 3D creat cu `plot3`

italic. Culoarea, marcajul şi stilul de linie pentru `plot3` se controlează la fel ca pentru `plot`. Limitele de axe în spaţiul tridimensional se determină automat, dar ele pot fi schimbate cu

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

Înafară de `xlim` şi `ylim`, există şi `zlim`, prin care se pot schimba limitele pe axa z.

O facilitate uşor de utilizat de desenare a contururilor este oferită de `ezcontour`. Apelul lui `ezcontour` în exemplul următor produce contururi pentru funcţia $\sin(3y - x^2 + 1) + \cos(2y^2 - 2x)$ pe domeniul dat de $-2 \leq x \leq 2$ şi $-1 \leq y \leq 1$; 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 funcţia `contour`. Întâi se fac iniţializările `x = -2:.01:2` şi `y = -1:.01:1` pentru a obţine puncte mai apropiate în domeniul respectiv. Apoi se execută `[X,Y] = meshgrid(x,y)`, care obţine matricele `X` şi `Y` astfel încât

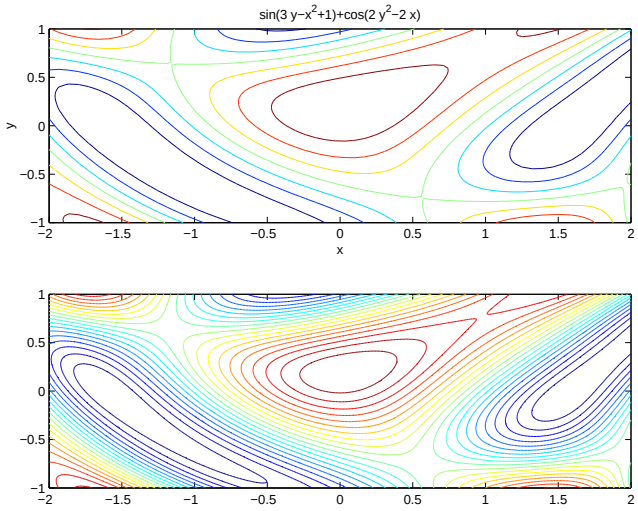


Figura 2.14: Contururi obţinute cu `ezcontour` (sus) şi `contour` (jos).

fiecare linie a lui `X` să fie o copie a lui `x` şi fiecare coloană a lui `Y` să fie o copie a vectorului `y`. (Funcţia `meshgrid` este foarte utilă la pregătirea datelor pentru multe funcţii MATLAB de grafică 3D.) Matricea `Z` este apoi generată prin operaţii de tip tablou din `X` şi `Y`; `Z(i,j)` memorează valoarea funcţiei corespunzând lui `x(j)` şi `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 spaţierea dată de `x` şi `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.

Funcţia `contour` se poate utiliza şi la reprezentarea funcţiilor implicite cum ar fi

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

Pentru a o reprezenta grafic, rescriem ecuaţia sub forma

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

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

Pentru aplicaţii în mecanică ale funcţiei `contour` vezi [8].

Funcţia `mesh` acceptă date în aceeaşi formă ca şi `contour` şi produce o reprezentare de suprafaţă de tip cadru de sârmă (wire-frame). Funcţia `meshc` se deosebeşte de `mesh` prin aceea că adaugă un grafic de tip contur dedesubtul suprafeţei. Exemplul de mai jos, care produce figura 2.16, lucrează cu suprafaţa definită de $\sin(y^2 + x) - \cos(y - x^2)$ pentru $0 \leq x, y \leq \pi$. Primul grafic este generat

Sursa MATLAB 2.1 Reprezentarea grafică a unei funcţii implicite

```
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)
```

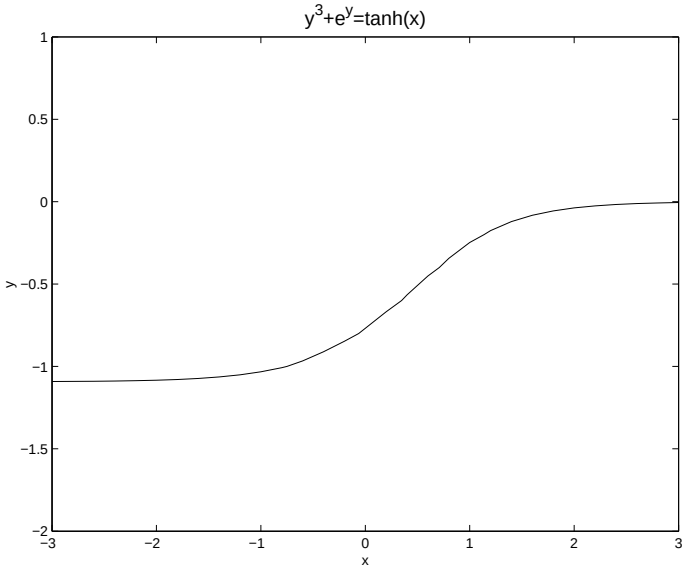


Figura 2.15: Curbă dată printr-o funcţie implicită

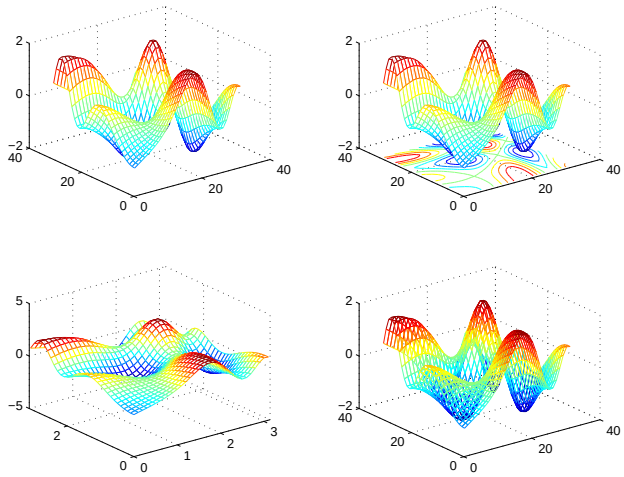


Figura 2.16: Grafice obţinute cu `mesh` şi `meshc`

cu `subplot` şi `mesh(Z)`. Deoarece nu se dă nici o informaţie pentru abscisă şi ordonată, `mesh` utilizează în locul lor indicii de linie şi de coloană. A doua imagine arată efectul lui `meshc(Z)`. Pentru cea de-a treia, s-a utilizat `mesh(x,y,Z)`, şi deci gradaţiile de pe axele x şi y corespund valorilor x şi 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 afişarea 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
```

Funcţia `surf` diferă de `mesh` prin aceea că produce un grafic de suprafaţa cu celulele umplute (colorate), iar `surfc` adaugă contururi dedesubt. În exemplul următor am vizualizat un paraboloid hiperbolic. Imaginile de pe prima linie a figurii 2.17 arată efectele lui `surf` şi `surfc`. În ele apare o scară de culori, realizată utilizând `colorbar`. Harta de culori curentă se poate seta cu `colormap`; vezi `doc`

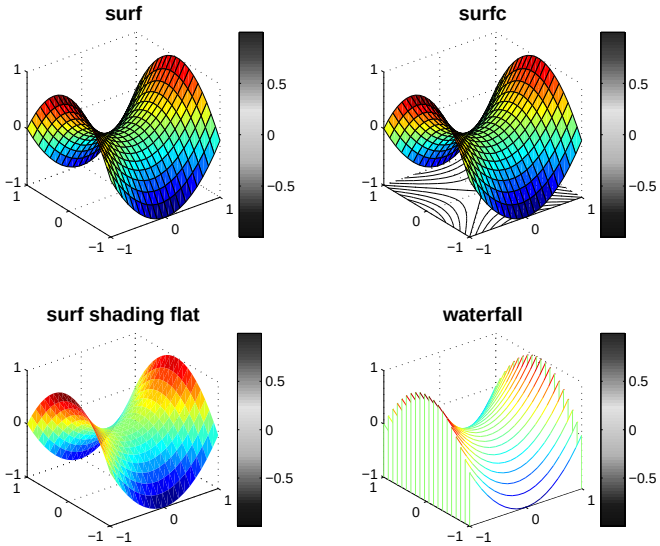


Figura 2.17: Suprafeţe desenate cu `surf`, `surfc` şi `waterfall`

`colormap`. A treia imagine (poziţia (2,1)) utilizează funcţia `shading` cu opţiunea `flat` pentru a elimina liniile grilei de pe suprafaţă; o altă opţiune este `interp`, care selectează culorile prin interpolare. A patra imagine (poziţia (2,2)) utilizează funcţia înrudită `waterfall`, care este similară lui `mesh`, dar fără cadrul de sârmă pe direcţia coloanelor.

```
[X,Y]=meshgrid(linspace(-1,1,20)); Z=X.^2-Y.^2;
FS = 'FontSize';
subplot(2,2,1), surf(X,Y,Z),
        title('\bf{surf}',FS,14), colorbar
subplot(2,2,2), surfc(X,Y,Z),
        title('\bf{surfc}',FS,14), colorbar
subplot(2,2,3), surf(X,Y,Z), shading flat
        title('\bf{surf} shading flat',FS,14), colorbar
subplot(2,2,4), waterfall(X,Y,Z)
        title('\bf{waterfall}',FS,14), colorbar
```

Graficele tridimensionale din figurile 2.13, 2.16 şi 2.17 utilizează unghiurile de vizualizare implicite ale MATLAB. Acestea pot fi modificate cu `view`. Apelul `view(a,b)` alege unghiul de rotaţie în sens invers acelor de ceasornic în jurul axei z (azimutul) de `a` grade şi unghiul faţă de planul xOy (elevaţia) 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.

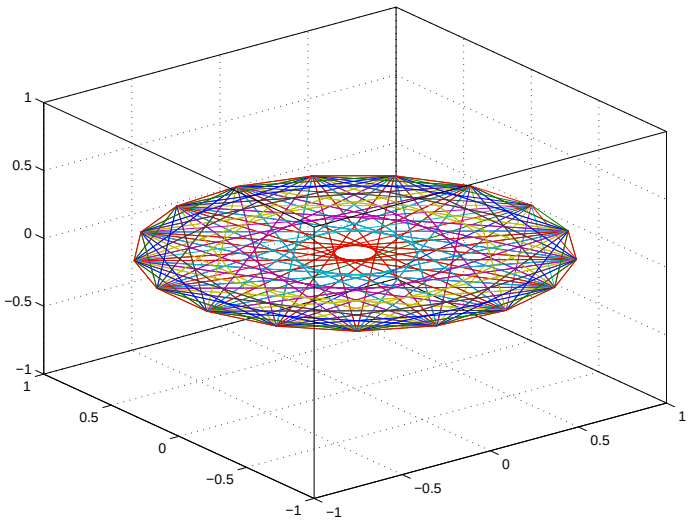


Figura 2.18: Vedere 3D a unei imagini 2D.

Este posibil sa 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 obţinută tastând

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

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

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

```
plot([1 2 NaN 3 4])
```

desenează două linii disjuncte şi nu uneşte punctele 2 şi 3, în timp ce

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

produce graficul `surfc` cu gaură din figura 2.19. (Funcţia `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)
```

şi generează o matrice de cote utilă pentru a testa şi demonstra facilităţile grafice 3D.)

`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`	suprafaţă wire-frame cu cortină
`surf`*	suprafaţă plină
`surfc`*	suprafaţă plină plus contururi
`waterfall`	wire-frame unidirecţional
`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 funcţii `fun` au corespondent `ezfun`

Tabela 2.4: Funcţii grafice 3D

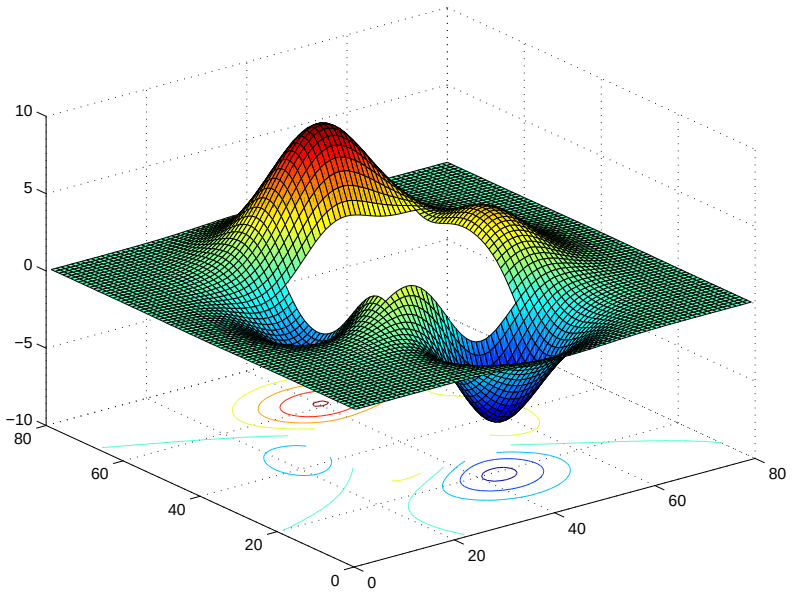


Figura 2.19: Grafic `surfc` al unei matrice ce conţine NaN-uri.

2.3. Salvarea şi imprimarea graficelor

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

```
print -dperiferic -optiuni numefisier
```

Ea are mai multe opţiuni, care pot fi vizualizate cu `help print`. Dintre tipurile de periferice admise amintim:

- dps - Postscript pentru imprimante alb-negru;
- dpsc - Postscript pentru imprimante color;
- 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 nivelul de calitate nn (implicit nn=75).

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

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

crează un fişier Postscript încapsulat alb şi negru, nivelul 2, numit `myfig.eps`, care poate fi listat pe o imprimantă PostScript sau inclus într-un document. Acest fişier poate fi încorporat într-un document LaTeX, aşa cum se schiţează 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 şi în formă funcţională (vezi secţiunea 1.4.2, pagina 33 pentru dualitatea comenzi/funcţii). Pentru a ilustra utilitatea formei funcţionale, exemplul următor generează o secvenţă de cinci figuri şi le salvează în fişierele `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 comenzii `print` este format prin concatenare, utilizând funcţia `int2str`, care converteşte un întreg în şir. Astfel, de exemplu, pentru `i=1`, instrucţiunea `print` este echivalentă cu `print('-deps2', 'fig1.eps')`.

Comanda `saveas` salvează o figură într-un fişier 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 şi imprima figuri din meniul `File` al ferestrei figurii.

2.4. Facilităţi grafice noi în MATLAB 7

În MATLAB 7 s-au introdus instrumente grafice noi şi s-au perfecţionat unele facilităţi:

- noi interfeţe grafice interactive pentru crearea şi modificarea graficelor fără a introduce cod M;
- generare de cod M dintr-o figură, permiţând reutilizarea graficelor din program;
- facilităţi îmbunătăţite 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 rotaţii, translaţii şi scalări asupra grupurilor de obiecte grafice;
- panouri de interfeţe utilizator şi controale ActiveX accesibile din GUIDE;
- suport Handle Graphics® îmbunătăţit şi suport pentru afişarea unor ecuaţii cu interfaţă completă TEX sau LATEX.

Probleme

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

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

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) $a = 3$, $b = 1$, $c = 1/2$. Experimentaţi 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 suprafaţa 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

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

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

Problema 2.6 (Elicoid). Să se reprezinte grafic suprafaţa 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

Algebră liniară în MATLAB

3.1. Rezolvarea sistemelor de ecuaţii liniare în MATLAB

Fie m numărul de ecuaţii şi n numărul de necunoscute. Instrumentul fundamental de rezolvare a sistemelor de ecuaţii liniare este operatorul $\backslash$ (vezi secţiunea 1.3.3).

El tratează trei tipuri de sisteme de ecuaţii liniare, pătratice ($m = n$), supradeterminate ($m > n$) şi 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 diferiţi în funcţie de matricea coeficienţilor. Diversele cazuri, care sunt diagnosticate automat prin examinarea matricei sistemului includ:

- matrice triunghiulare sau permutări de matrice triunghiulare;
- matrice simetrice, pozitiv definite;
- matrice pătratice nesingulare;
- sisteme dreptunghiulare supradeterminate;
- sisteme dreptunghiulare subdeterminate.

3.1.1. Sisteme pătratice

Dacă A este o matrice pătratică nesingulară de ordinul n, atunci `A\b` este soluţia sistemului `Ax=b`, calculată prin factorizare LU cu pivotare parţială. În timpul rezolvării, MATLAB calculează `rcond(A)` şi tipăreşte 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.
```

3.1.2. Sisteme supradeterminate

Dacă $m > n$, în general sistemul $Ax = b$ nu are nici o soluţie. Expresia MATLAB `A\b` calculează soluţia sistemului în sensul celor mai mici pătrate, adică minimizează norma euclidiană a reziduului (adică `norm(A*x-b)`) peste toţi vectorii `x`. Dacă A are rang maxim m, atunci soluţia este unică. Dacă A are rangul k mai mic decât m (în acest caz spunem că A este deficientă de rang), `A\b` calculează o soluţie de bază cu cel mult k elemente nenule (k este determinat şi `x` este calculat utilizând factorizarea QR cu pivotare pe coloană). În ultimul caz MATLAB dă un mesaj de avertisment.

Soluţia se mai poate calcula şi cu `x_min=pinv(A)*b`, unde `pinv(A)` este pseudo-inversa lui `A`. Dacă A este deficientă de rang, `x_min` este soluţia 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 noţiunea de inversă pentru matrice dreptunghiulare şi deficiente de rang. Pseudo-inversa A^+ a lui A este matricea unică care satisface condiţiile

$$AA^+A = A, \qquad A^+AA^+ = A^+, \qquad (A^+A)^+ = A^+A, \qquad (AA^+)^+ = AA^+.$$

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
```

3.1.3. Sisteme subdeterminate

În cazul unui sistem subdeterminat putem fie să nu avem nici o soluţie fie să avem o infinitate. În ultimul caz, `A\b` produce o soluţie de bază cu cel mult k elemente nenule, unde k este rangul lui A. În general această soluţie nu are norma euclidiană minimă; soluţia cu norma minimă se poate calcula cu `pinv(A)*b`. Dacă sistemul nu are nici o soluţie (este incompatibil), atunci $A\backslash b$ este o soluţie în sensul celor mai mici pătrate. Exemplul următor ilustrează diferenţa dintre soluţia obţinută cu $\backslash$ şi `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 foloseşte 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 ecuaţii şi 4 necunoscute. Deoarece matricea coeficienţilor conţine întregi mici, este recomandabil să afişăm soluţia în format raţional. Soluţia particulară se obţine 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)`.

Soluţia completă a unui sistem supradeterminat poate fi caracterizată prin adăugarea unui vector arbitrar din spaţiul nul al matricei sistemului, care poate fi găsit cu funcţia `null` cu o opţiune care cere o bază raţională

```
>> Z=null(R,'r')
   Z =
        5/12        -2/3
      -47/24         1/3
         1            0
         0            1
```

Se poate verifica că `R*Z` este zero şi orice vector de forma `x=p+Z*q`, unde `q` este un vector arbitrar, satisface `R*x=b`.

3.1.4. Factorizarea LU şi Cholesky

Funcţia `lu` calculează o factorizare LUP cu pivotare parţială. Apelul `[L,U,P]=lu(A)` returnează factorii triunghiulari şi matricea de permutare. Forma `[L,U]=lu(A)` returnează $\mathtt{L} = \mathtt{P}^T\mathtt{L}$, deci `L` este o matrice triunghiulară cu liniile permutate.

```
>> 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
```

Deşi factorizarea LU este definită şi pentru matrice dreptunghiulare, `lu` acceptă la intrare numai matrice pătratice.

Rezolvarea sistemului `Ax=b` cu `x=A\b` cu `A` pătratică este echivalentă cu secvenţa MATLAB:

```
[L,U] = lu(A);  x = U\(L\b);
```

Determinantul şi 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 şi pozitiv definită calculează R astfel încât $A = R^*R$. Factorizarea Cholesky permite înlocuirea sistemului `A*x=b` cu `R'*R*x=b`. Deoarece operatorul \ recunoaşte sisteme triunghiulare, sistemul se poate rezolva rapid cu `x=R\(R'\R\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
```

Funcţia `chol` examinează doar partea triunghiulară superior a lui `A`. Dacă `A` nu este pozitiv definită se dă un mesaj de eroare. În `[R,p]=chol(A)`, dacă `p=0` factorizarea s-a terminat cu succes, iar în caz de eşec `p` este un număr natural nenul. Pentru detalii a se vedea `help chol` sau `doc chol`.

Funcţia `cholupdate` modifică factorizarea Cholesky atunci când matricea originală a fost afectată de o perturbaţie de rang 1 (adică cu o matrice de forma $+xx^*$ sau $-xx^*$, unde x este un vector).

3.1.5. Factorizarea QR

În MATLAB există patru variante de factorizare QR – completă sau economică şi 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 şi o matrice superior triunghiulară `R` de dimensiune $m \times n$. Forma de apel este `[Q,R]=qr(C)`. În multe cazuri ultimele $m - n$ coloane nu sunt necesare, deoarece ele sunt înmulţite cu zerouri în porţiunea 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 şi 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 =
      -1.7321   -3.4641
            0   -1.4142
```

Pentru matrice dreptunghiulare mari, cu $m \gg n$, câştigul de timp şi 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 $|r_{11}| \geq |r_{22}| \geq \cdots \geq |r_{nn}|$. Pivotarea pe coloane este utilă pentru detectarea singularităţilor sau deficienţelor 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 ieşire, 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
```

Funcţiile `qrdelete`, `qrinsert` şi `qrupdate` modifică factorizarea QR când se şterge sau se inserează o coloană din matricea originală sau când matricea este afectată de o perturbaţie de rang 1.

Să considerăm acum un sistem $Ax = b$, pătratic, unde A este o matrice pătratică de ordinul n de forma:

$$A = (a_{i,j}), \qquad a_{i,j} = \begin{cases} 1, & \text{pentru } i = j \text{ sau } j = n; \\ -1, & \text{pentru } i > j; \\ 0, & \text{în rest.} \end{cases}$$

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 secvenţa

```
A=[-tril(ones(n,n-1),-1)+eye(n,n-1),ones(n,1)]
```

Să presupunem că `b` s-a iniţializat cu `b=A*ones(n,1)`. Un astfel de sistem are soluţia $x = [1, 1, \ldots, 1]^T$. 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     0     0     0     0     0
     1     1     1     1     1     0     0     0     0     0
     1     1     1     1     1     0     0     0     0     0
     1     1     1     1     1     0     0     0     0     0
     1     1     1     1     1     0     0     0     0     0
     1     1     1     1     1     0     0     0     0     0
     1     1     1     1     1     0     0     0     0     1
>> norm(b-A*x)/norm(b)
ans =
    0.3191
```

rezultat total eronat, deşi `A` este bine condiţionată

```
>> cond(A)
ans =
   44.8023
```

Dacă rezolvăm folosind metoda QR, se obţine

```
>> [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
```

Funcţia `linsolve` permite rezolvarea mai rapidă a sistemelor de ecuaţii liniare prin specificarea matricei sistemului. Apelată sub forma

```
x = linsolve(A,b,opts)
```

rezolvă sistemul liniar `A*x=b`, selectând un rezolvitor adecvat în funcţie de proprietăţile matricei `A`, descrise de structura `opts`. Nu se face nici un test pentru a

verifica dacă A are astfel de proprietăţi. Dăm lista numelor de câmpuri şi a valorilor lor posibile.

nume câmp	proprietatea matricei
LT	triunghiulară inferior
UT	triunghiulară superior
UHESS	Hessenberg superioară
SYM	simetrică sau hermitiană
POSDEF	positiv definită
RECT	dreptunghiulară
TRANSA	transpusa (conjugată) a lui A – specifică dacă se rezolvă A*x=b sau A'x=b

Dacă `opts` lipseşte, sistemul se rezolvă cu factorizare LUP, dacă A este pătratică şi cu factorizare QR cu pivotare în caz contrar. Dacă A este prost condiţionată în cazul pătratic sau deficientă 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
```

3.2. Polinoame şi potrivirea datelor în MATLAB

MATLAB reprezintă un polinom

$$p(x) = p_1 x^n + p_2 x_{n-1} + p_n x + p_{n+1}$$

printr-un vector linie `p=[p(1) p(2) ...p(n+1)]` al coeficienţilor, ordonaţi descrescător după puterile variabilei.

Să considerăm trei probleme legate de polinoame:

- *evaluarea* – dându-se coeficienţii să se calculeze valoarea polinomului în unul sau mai multe puncte;
- *determinarea rădăcinilor* – dându-se coeficienţii să se determine rădăcinile polinomului;
- *potrivirea datelor (data fitting)* – dându-se o mulţime de date $(x_i, y_i)_{i=1}^m$ să se determine un polinom (sau o altă combinaţie de funcţii de bază) care „se potriveşte" cu aceste date.

Evaluarea se face cu ajutorul schemei lui Horner, implementată în MATLAB prin funcţia `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 obţinerea matricei

$$p(X) = p_1X^n + p_2X_{n-1} + p_nX + p_{n+1},$$

unde X este o matrice pătratică se poate face cu comanda `Y = polyvalm(p,X)`.

Rădăcinile (reale şi complexe) ale polinomului `p` se pot obţine cu `z = roots(p)`. Funcţia `poly` realizează operaţia inversă, adică construieşte polinomul cunoscând rădăcinile. Ea acceptă ca argument şi o matrice pătratică `A`, caz în care `p=poly(A)` calculează polinomul caracteristic al lui `A`, adică $\det(xI - A)$.

Funcţia `polyder` calculează coeficienţii derivatei unui polinom, fără a-l evalua.

Ca exemplu, să considerăm polinomul $p(x) = x^2 - x - 1$. Rădăcinile lui le obţinem 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 2×2

```
>> 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
```

Înmulţirea şi împărţirea polinoamelor se realizează cu `conv` şi `deconv`. Sintaxa lui `deconv` este `[q,r]=deconv(g,h)`, unde `g` este deîmpărţitul, `h` împărţitorul, `q` câtul şi `r` restul. În exemplul următor vom împărţi $x^2-2x-x+2$ la $x-2$, obţinând câtul x^2-1 şi restul 0. Polinomul iniţial se va obţine 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 observaţii (y_i) măsurate în valorile specificate (t_i):

$$y_i = y(t_i), \qquad i = \overline{1,m}.$$

Modelul nostru este o combinaţie de n funcţii de bază (π_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 α. În notaţie matricială, modelul se poate exprima ca:

$$y \approx A(\alpha)c.$$

Reziduurile sunt diferenţele dintre valorile observate şi cele date de model

$$r_i = y_i - \sum_{j=1}^{n} c_j\pi_j(t_i,\alpha)$$

sau în notaţie 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 explicaţie intuitivă, fizică, a celei de-a doua alegeri ar fi aceea că anumite observaţii sunt mai importante decât altele şi le vom asocia ponderi, w_i. De exemplu, dacă la observaţia 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 α şi neliniară în caz contrar.

Orice algoritm de rezolvare a unei probleme de aproximare în sensul celei mai mici pătrate fără ponderi poate fi utilizat la rezolvarea unei probleme cu ponderi prin scalarea observaţiilor şi a matricei de proiectare. În MATLAB aceasta se poate realiza prin

```
A=diag(w)*A
y=diag(w)*y
```

Dacă problema este liniară şi avem mai multe observaţii decât funcţii de bază, suntem conduşi la rezolvarea sistemului supradeterminat (vezi secţiunea 3.1.2)

$$Ac \approx y,$$

pe care îl vom rezolva în sensul celor mai mici pătrate

$$c = A \backslash y.$$

Abordarea teoretică se bazează pe rezolvarea ecuaţiilor normale

$$A^T A c = A^T y.$$

Dacă funcţiile de bază sunt liniar independente şi deci $A^T A$ nesingulară, soluţia 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 funcţia MATLAB `pinv`.

Fie sistemul $Ax = b$ arbitrar. Dacă A este o matrice $m \times n$ cu $m > n$ şi are rangul n, atunci fiecare din următoarele trei instrucţiuni

```
x=A\b
x=pinv(A)*b
x=inv(A'*A)*A'*b
```

calculează aceeaşi soluţie în sensul celor mai mici pătrate, deşi operatorul \ o face cel mai repede.

Totuşi, dacă A nu are rangul complet, soluţia în sensul celor mai mici pătrate nu este unică. Există mai mulţi vectori care minimizează norma $\|Ax-b\|_2$. Soluţia calculată cu `x=A\b` este o *soluţie de bază*; ea are cel mult r componente nenule, unde r este rangul lui `A`. Soluţia calculată cu `x=pinv(A)*b` este soluţia cu normă minimă (ea minimizează `norm(x)`). Încercarea de a calcula o soluţie cu `x=inv(A'*A)*A'*b` eşuează dacă `A'*A` este singulară. Iată un exemplu care ilustrează diversele soluţii. Matricea

```
A=[1,2,3; 4,5,6; 7,8,9; 10,11,12];
```

este deficientă de rang. Dacă `b=A(:,2)`, atunci o soluţie evidentă a lui `A*x=b` este `x=[0,1,0]'`. Nici una dintre abordările de mai sus nu calculează pe `x`. Operatorul \ ne dă

```
>> x=A\b
   Warning: Rank deficient, rank = 2  tol =   1.4594e-014.
   x =
       0.5000
            0
       0.5000
```

Această soluţie are două componente nenule. Varianta cu pseudoinversă ne dă

```
>> y=pinv(A)*b
   y =
       0.3333
       0.3333
       0.3333
```

Se observă ca `norm(y)=0.5774<norm(x)=0.7071`. A treia variantă eşuează complet:

```
>> z=inv(A'*A)*A'*b
   Warning: Matrix is singular to working precision.
   z =
     Inf
     Inf
     Inf
```

Abordarea bazată pe ecuaţii normale are mai multe dezavantaje. Ecuaţiile normale sunt întotdeauna mai prost condiţionate decât sistemul supradeterminat iniţial.

Numărul de condiţionare se ridică de fapt la pătrat[1]:

$$\mathrm{cond}(A^TA) = \mathrm{cond}(A)^2.$$

În reprezentarea în virgulă flotantă, chiar dacă coloanele lui A sunt liniar independente, $(A^TA)^{-1}$ ar putea fi aproape singulară.

MATLAB evită ecuaţiile normale. Operatorul \ foloseşte intern factorizarea QR. Soluţia se poate exprima prin `c=R\(Q'*y)`.

Dacă baza în care se face aproximarea este $1, t, \ldots, t^n$, se poate folosi funcţia `polyfit`. Comanda `p=polyfit(x,y,n)` calculează coeficienţii polinomului de aproximare discretă de grad `n` în sensul celor mai mici pătrate pentru datele `x` şi `y`. Dacă $n \geq m$, se returnează coeficienţii polinomului de interpolare.

Vom considera două exemple.

O cantitate `y` este măsurată în diferite momente de timp, `t`, pentru a produce următoarele observaţii:

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

Aceste 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 funcţii de forma

$$y(t) = c_1 + c_2e^{-t}.$$

Coeficienţii necunoscuţi se vor calcula prin metoda celor mai mici pătrate. Avem 6 ecuaţii şi două necunoscute, reprezentate printr-o matrice 6×2

```
>> 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
```

[1]Pentru o matrice dreptunghiulară X, numărul de condiţionare ar putea fi definit prin $\mathrm{cond}(X) = \|X\|\|X^+\|$

Soluţia în sensul celor mai mici pă trate 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 diferenţa este minimă în sensul celor mai mici pătrate (figura 3.1). Dacă matricea A este deficientă de rang (adică nu are coloane liniar

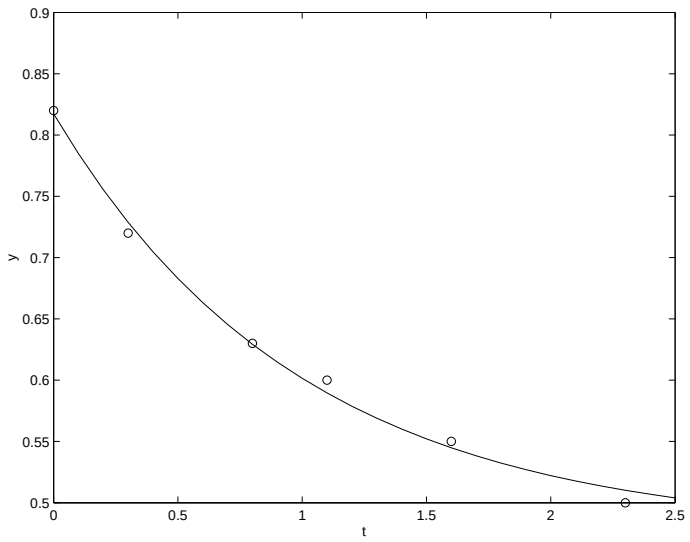


Figura 3.1: Ilustrare a potrivirii datelor

independente), atunci soluţia în sensul celor mai mici pătrate a sistemului $Ax = b$ nu este unică. În acest caz operatorul \ dă un mesaj de avertizare şi produce o soluţie de bază cu cel mai mic număr posibil de elemente nenule.

Al doilea exemplu are ca date de intrare rezultatele de recensământelor obţinute de U. S. Census pentru anii 1900–2000, din zece în zece ani, exprimate în milioane de oameni:

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 doreşte modelarea creşterii populaţiei printr-un polinom de gradul al treilea

$$y(t) = c_1 t^3 + c_2 t^2 + c_3 t + c_4$$

şi predicţia populaţiei din 2010.

Dacă încercăm să calculăm coeficienţii cu `c=polyfit(t,y,3)`, matricea sistemului va fi prost condiţionată, coloanele ei vor fi aproape liniar dependente şi vom obţine 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:

$$s = (t - s)/50.$$

Noua variabilă este în intervalul $[-1, 1]$, iar sistemul va fi bine condiţionat. Script-ul MATLAB 3.1, `census.m`, calculează coeficienţii, reprezintă datele şi polinomul şi estimează populaţia în 2010. Estimaţia apare şi în clar şi marcată cu un asterisc (vezi figura 3.2).

3.3. Valori şi vectori proprii în MATLAB

MATLAB utilizează rutine LAPACK pentru a calcula valori şi vectori proprii. Valorile proprii ale unei matrice se calculează cu funcţia `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` şi în matricea diagonală `D` valorile proprii ale lui `A`. Are loc relaţia `A*V=V*D`. Nu orice matrice are n vectori proprii liniari independenţi, deci matricea `V` returnată de `eig` poate fi singulară (sau datorită erorilor de rotunjire nesingulară, dar foarte prost condiţionată). Matricea din exemplul următor are o valoare proprie dublă 1, dar numai un vector propriu liniar independent:

Sursa MATLAB 3.1 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)';   % anii de recensamant
x = (1890:1:2019)';    % anii de evaluare
w = 2010;              % anul de predictie

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)
```

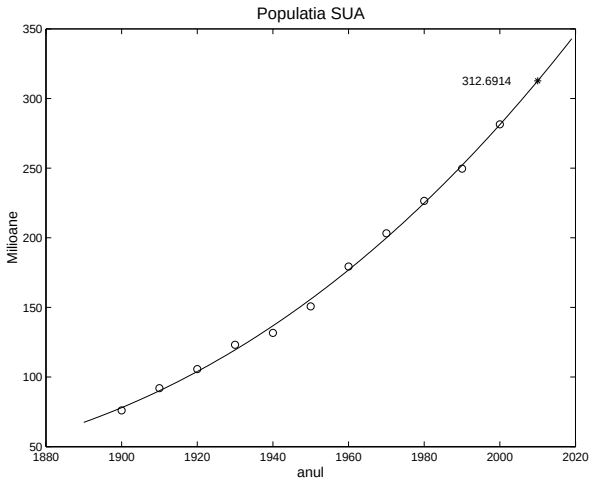


Figura 3.2: Ilustrarea exemplului cu recensământul

```
>> [V,D]=eig([2, -1; 1,0])
V =
    0.7071    0.7071
    0.7071    0.7071
D =
     1     0
     0     1
```

Vectorii proprii sunt scalaţi 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 şi 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 λ 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 egal depărtaţi de extrem sunt numere opuse:

```
>> poly(F)
ans =
    1.0000  -15.0000   55.0000  -55.0000   15.0000   -1.0000
```

Astfel, $\det(F-\lambda I) = -\lambda^5 \det(F-\lambda^{-1} I)$.

Funcţia `condeig` calculează numărul de condiţionare pentru valori proprii. Acesta se defineşte prin

$$\Gamma(A) = \inf\{\mathrm{cond}(X) : X^{-1}AX = \mathrm{diag}(\lambda_i)\}.$$

Forma `c=condeig(A)` returnează un vector al numerelor de condiţionare 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 condiţionare mare indică o valoare proprie sensibilă la perturbaţii ale matricei. Exemplul următor afişează în prima linie valorile proprii ale matricei lui Frank de ordinul 6 şi în a doua linie numerele lor de condiţionare:

```
>> 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 informaţii despre modul de lucru al funcţiei `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 şi coloanele lui A de norme aproximativ egale. Motivarea echilibrării este aceea că poate conduce la un calcul mai precis al vectorilor şi valorilor proprii. Totuşi, uneori echilibrarea nu este necesară şi 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 tridiagonală dacă A este hermitiană). Forma Hessenberg se poate calcula cu `H = hess(A)` sau `[Q,H] = hess(A)`. Ultima formă dă şi matricea unitară prin care se face transformarea. Comanda `T = schur(A)` sau `[Q,T] = 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 obţine cu `schur(A,'complex')`.

MATLAB poate rezolva şi probleme de valori proprii generalizate, adică probleme de forma: fiind date două matrice pătratice de ordinul n, A şi B, să se găsească scalarii λ şi vectorii $x \neq 0$ astfel încât $Ax = \lambda Bx$. Valorile proprii generalizate se pot calcula cu `e = eig(A,B)`, iar `[V,D] = eig(A,B)` returnează o matrice

diagonală `D` a valorilor proprii şi o matrice pătratică de ordinul n a vectorilor proprii `V` astfel încât `A*V=B*V*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 obţine 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 numeşte *descompunere cu valori singulare* (singular value decomposition – SVD) descompunerea

$$A = U\Sigma V^*, \tag{3.3.1}$$

unde Σ este o matrice diagonală reală, iar U şi V sunt matrice unitare (ortogonale în cazul real).

Există două variante de SVD: una completă, care se aplică unei matrice dreptunghiulare $m \times n$ şi care returnează matricele U de dimensiune $m \times m$, Σ de dimensiune $m \times n$ şi V de dimensiune $n \times n$ şi una economică sau redusă în care U are dimensiunea $m \times n$, Σ are dimensiunea $n \times n$ şi V are dimensiunea $n \times n$.

SVD este un instrument util de analiză a aplicaţiilor dintr-un spaţiu vectorial cu valori în alt spaţiu, posibil de dimensiune diferită. Dacă A este pătratică, simetrică şi pozitiv definită SVD (3.3.1) şi descompunerea cu valori proprii coincid. Spre deosebire de descompunerea cu valori proprii, SVD există întotdeauna.

Fie matricea

```
A =
     9     4
     6     8
     2     7
```

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*S*V'` este egală cu `A`, în limita erorilor de rotunjire.

Probleme

Problema 3.1. Pentru un sistem având matricea tridiagonală să se implementeze următoarele metode:

(a) eliminarea gaussiană, cu şi fără pivotare;

(b) descompunerea LU;

(c) descompunerea LUP;

(d) descompunerea Cholesky, dacă matricea este simetrică şi pozitiv definită.

(e) metoda Jacobi;

(f) metoda Gauss-Seidel;

(g) metoda SOR.

Problema 3.2. Implementaţi eliminarea gaussiană cu pivotare parţială în două variante: cu permutare de linii logică (prin intermediul unui vector de permutări) şi cu permutare efectivă a liniilor. Comparaţi timpii de execuţie pentru diverse dimensiuni ale matricei sistemului. Faceţi acelaşi lucru şi pentru descompunerea LUP.

Problema 3.3. Modificaţi descompunerea LUP astfel ca să returneze şi valoarea determinantului matricei iniţiale.

Problema 3.4. Se consideră sistemul

$$\begin{aligned} 2x_1 - x_2 &= 1 \\ -x_{j-1} + 2x_j - x_{j+1} &= j, \qquad j = \overline{2, n-1} \\ -x_{n-1} + 2x_n &= n \end{aligned}$$

(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 3.1.

(d) Să se genereze matricea cu `spdiags`, să se rezolve cu $\backslash$, comparând timpul de rezolvare cu timpul necesar pentru rezolvarea aceluiaşi sistem cu matrice densă.

(e) Să se estimeze numărul de condiţionare al matricei coeficienţilor folosind `condest`.

Problema 3.5. Modificaţi eliminarea gaussiană şi descompunerea LUP astfel ca să utilizeze pivotarea totală.

Problema 3.6. Scrieţi o funcţie MATLAB care să genereze matrice bandă aleatoare de dimensiune dată şi care să fie diagonal dominante. Testaţi metoda lui Jacobi şi metoda SOR pentru sisteme având astfel de matrice.

Problema 3.7. Considerăm ecuaţia diferenţială ordinară

$$y''(x) - p(x)y'(x) - q(x)y(x) = r(x), \qquad x \in [a, b]$$

cu condiţiile pe frontieră $y(a) = \alpha$, $y(b) = \beta$. Presupunem că $q(x) \geq \underline{q} > 0$. Pentru a rezolva ecuaţia numeric, o vom discretiza, căutând soluţiile sale pe punctele echidistante $x_i = a + ih$, $i = 0, \ldots, N-1$, unde $h = (b-a)/(N+1)$. Definim $p_i = p(x_i)$, $q_i = q(x_i)$, $r_i = r(x_i)$ şi $y_i \approx y(x_i)$. Utilizând aproximaţiile

$$y'(x_i) \approx \frac{y_{i+1} - y_i}{2h}$$

şi

$$y''(x_i) \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

şi ţinând cont că $y_0 = \alpha$ şi $y_{N+1} = \beta$, se ajunge la un sistem liniar tridiagonal.

(a) Scrieţi sistemul la care se ajunge prin discretizare şi studiaţi proprietăţile sale.

(b) Scrieţi o funcţie MATLAB care rezolvă numeric ecuaţia diferenţială cu condiţii pe frontieră bazată pe ideea de mai sus. Sistemul se va rezolva printr-o metodă directă şi una iterativă (dacă este posibil).

(c) Arătaţi că sistemul poate fi transformat într-un sistem echivalent a cărui matrice este simetrică şi pozitiv definită. Să se implementeze şi această variantă.

(d) Exploataţi raritatea matricei sistemului.

Testaţi pentru problema

$$y'' = -\frac{2}{x}y' + \frac{2}{x^2}y + \frac{\sin(\ln x)}{x^2},\ x \in [1, 2],\ y(1) = 1,\ y(2) = 2,$$

cu soluţia exactă

$$y = c_1 x + \frac{c_2}{x^2} - \frac{3}{10}\sin(\ln x) - \frac{1}{10}\cos(\ln x),$$

unde

$$\begin{aligned} c_2 &= \frac{1}{70}[8 - 12\sin(\ln 2) - 4\cos(\ln 2)], \\ c_1 &= \frac{11}{10} - c2. \end{aligned}$$

Problema 3.8. Aplicaţi ideea din problema precedentă la rezolvarea ecuaţiei lui Poisson unidimensionale

$$-\frac{d^2v(x)}{dx^2} = f, \qquad 0 < x < 1,$$

cu condiţiile pe frontieră $v(0) = v(1) = 0$. Rezolvaţi sistemul la care se ajunge cu metoda Cholesky şi cu metoda SOR.

Problema 3.9. 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 3.10. 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 $\alpha = 482317$, $\beta = 2196.05$ şi $\gamma = 6708.43$. Aici x_1, x_2, x_3 reprezintă deplasări laterale, iar x_4, x_5, x_6 reprezintă deplasări rotaţionale (tridimensionale) corespunzând forţei aplicate (membrul drept).

(a) Determinaţi x.

(b) Cât de precise sunt calculele? Presupunem întâi date exacte, apoi $\|\Delta A\|/\|A\| = 5 \times 10^{-7}$.

Problema 3.11. 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 parţială.

(b) Împărţiţi fiecare linie cu maximul în modul din linia respectivă şi apoi utilizaţi eliminarea gaussiană.

(c) Rezolvaţi sistemul folosind toolbox-ul Symbolic.

Problema 3.12. (a) Dându-se o funcţie $f \in C[a,b]$, să se determine $\hat{s}_1(f;\cdot) \in \mathbb{S}_1^0(\Delta)$ astfel încât

$$\int_a^b [f(x) - \hat{s}_1(f;x)]^2 \,\mathrm{d}x$$

să fie minimă.

(b) Scrieţi o funcţie MATLAB care construieşte şi rezolvă sistemul de ecuaţii normale de la punctul (a).

(c) Testaţi funcţia de mai sus pentru o funcţie şi o diviziune alese de dumneavoastră.

Problema 3.13. Calculaţi aproximaţiile discrete în sensul celor mai mici pătrate ale funcţiei $f(t) = \sin\left(\frac{\pi}{2}t\right)$ pe $0 \le t \le 1$ de forma

$$\varphi_n(t) = t + t(1-t)\sum_{j=1}^{n} c_j t^{j-1}, \qquad n = 1(1)5,$$

utilizând N abscise $t_k = k/(N+1)$, $k = \overline{1,N}$. De notat că $\varphi_n(0) = 0$ şi $\varphi_n(1) = 1$ sunt valorile exacte ale lui f în $t = 0$ şi respectiv $t = 1$.

(*Indicaţie*: Aproximaţi $f(t) - t$ printr-o combinaţie liniară a polinoamelor $\pi_j(t) = t(1-t)t^{j-1}$, $j = \overline{1,n}$.) Sistemul de ecuaţii normale are forma $Ac = b$, $A = [(\pi_i, \pi_j)]$, $b = [(\pi_i, f - t)]$, $c = [c_j]$.

Ieşire (pentru $n = 1, 2, \ldots, 5$) :

- numărul de condiţionare al sistemului;
- valorile coeficienţilor;
- eroarea maximă şi minimă:

$$e_{\min} = \min_{1\le k\le N} |\varphi_n(t_k) - f(t_k)|, \qquad e_{\max} = \max_{1\le k\le N} |\varphi_n(t_k) - f(t_k)|.$$

Executaţi de două ori pentru

(a) $N = 5, 10, 20$,

(b) $N = 4$.

Comentaţi rezultatele. Reprezentaţi pe acelaşi grafic funcţia şi aproximantele.

Problema 3.14. Determinaţi 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

Reprezentaţi grafic punctele şi aproximanta.
Indicaţie: logaritmaţi.

Problema 3.15. Determinaţi o aproximare discretă în sensul celor mai mici pătrate de forma

$$y = c_1 + c_2 x + c_3 \sin(\pi x) + c_4 \sin(2\pi x)$$

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

Reprezentaţi grafic datele şi aproximanta.

Problema 3.16. Calculaţi valorile proprii ale matricei Hilbert pentru $n = 10, 11, \ldots, 20$ şi numerele de condiţionare corespunzătoare.

Problema 3.17. Matricele

```
P=gallery('pascal',12)
F=galery('frank',12)
```

au proprietatea că dacă λ este valoare proprie, atunci şi $1/\lambda$ este de asemenea valoare proprie. Cât de bine conservă valorile proprii această proprietate? Utilizaţi `condeig` pentru a explica comportarea diferită a celor două matrice.

Problema 3.18. Care este cea mai mare valoare proprie a lui `magic(n)`? De ce?

Problema 3.19. Încercaţi secvenţa de comenzi:

```
n=100;
d=ones(n,1);
A=diag(d,1)+diag(d,-1);
e=eig(A);
plot(-n/2:n/2,e,'.')
```

Recunoaşteţi curba rezultată? Aţi putea găsi o formulă pentru valorile proprii ale acestei matrice?

Problema 3.20. Studiaţi valorile şi vectorii proprii corespunzători ai matricei la care se ajunge la problema 3.7. Reprezentaţi grafic valorile şi vectorii proprii pentru un N dat.

Problema 3.21. Fie T_N matricea la care se ajunge la discretizarea cu diferenţe a ecuaţiei lui Poisson (problema 3.8). Valorile ei proprii sunt

$$\lambda_j = 2\left(1 - \cos\frac{\pi j}{N+1}\right),$$

iar vectorii proprii z_j au componentele

$$z_j(k) = \sqrt{\frac{2}{N+1}}\sin\frac{jk\pi}{N+1}.$$

Să se reprezinte grafic valorile şi vectorii proprii ai lui T_{21}.

Problema 3.22. (a) Implementaţi metoda puterii (iteraţia vectorială).

(b) Testaţi funcţia de la punctul precedent pentru matricea şi vectorul iniţial

$$A = \begin{pmatrix} 6 & 5 & -5 \\ 2 & 6 & -2 \\ 2 & 5 & -1 \end{pmatrix}, \qquad x = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.$$

(r) Aproximaţi raza spectrală $\rho(A)$ a matricei

$$A = \begin{pmatrix} 2 & 0 & -1 \\ -2 & -10 & 0 \\ -1 & -1 & 4 \end{pmatrix},$$

utilizând metoda puterii şi vectorul iniţial $(1, 1, 1)^T$.

Problema 3.23. Determinaţi 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 `eig`.

Problema 3.24. Determinaţi descompunerile cu valori singulare ale următoarelor matrice:

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 7 \\ 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} 5 \\ 4 \end{bmatrix}.$$

CAPITOLUL 4

Interpolare în MATLAB

4.1. Interpolare unidimensională

MATLAB are funcţii pentru interpolare în una, două sau mai multe dimensiuni. Funcţia `polyfit` returnează coeficienţii polinomului de interpolare Lagrange dacă gradul n este egal cu numărul de observaţii minus 1.

Funcţia `interp1` acceptă perechi de date `x(i)`, `y(i)` şi un vector `xi` al punctelor în care se face evaluarea. Ea construieşte interpolantul corespunzător datelor `x` şi `y` şi 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 interpolanţi, precizate de parametrul `metoda`, care poate avea una din următoarele valori

- `'nearest'` - interpolare bazată pe vecinul cel mai apropiat;
- `'linear'` - interpolare liniară pe porţiuni (metoda implicită);
- `'spline'` - interpolare cu spline cubice;
- `'cubic'` sau `'pchip'` - interpolare Hermite cubică pe porţiuni.

Exemplul de mai jos ilustrează funcţionarea lui `interp1` (fişierul `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 şase puncte de pe graficul lui $f(x) = x + \sin \pi x^2$ şi calculează interpolanţii `nearest`, `linear` şi `spline`. Nu am inclus în acest exemplu `cubic` deoarece graficul obţinut este foarte apropiat de cel obţinut cu `spline` şi s-ar fi încărcat figura. Graficul apare în figura 4.1.

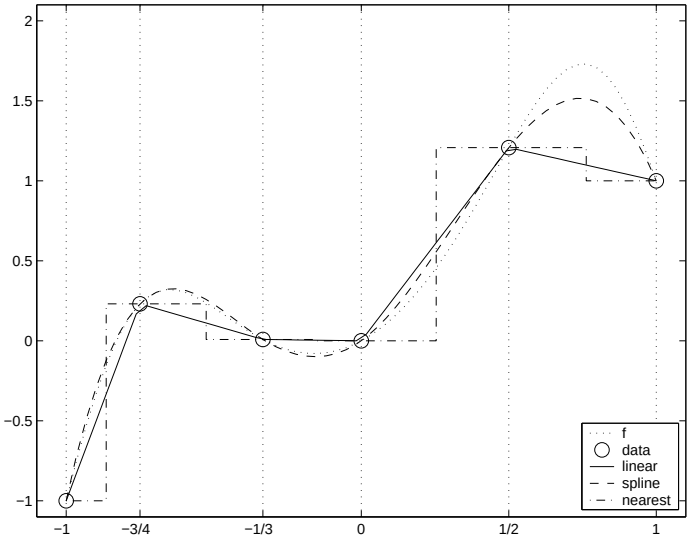


Figura 4.1: Exemplu de interpolare cu `interp1`

Interpolarea `spline` este cea mai netedă, dar interpolarea Hermite pe porţiuni păstrează alura. Vom încerca să ilustram diferenţa dintre interpolarea spline şi interpolarea Hermite pe porţiuni prin exemplul următor (`expsli_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');
h=plot(xi,ys,'-',xi,yc,'-.');
legend(h,'spline','cubic',4)
axis([-1.1,1.6,-0.8,1.6])
```

Figura 4.2 dă graficul astfel obţinut.

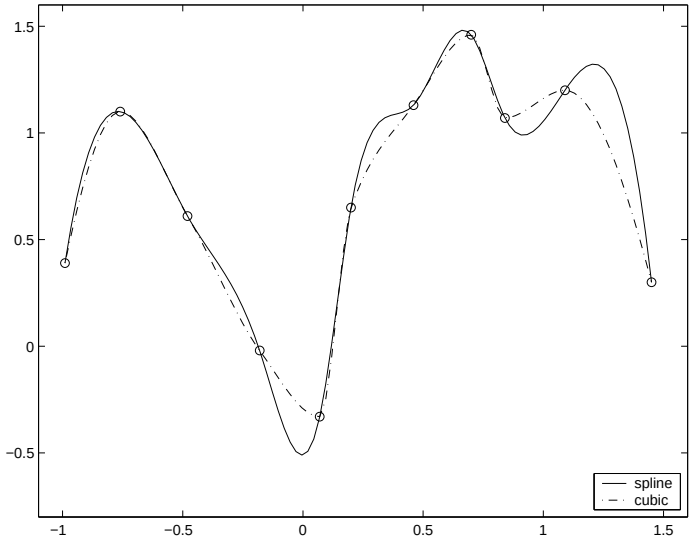


Figura 4.2: Interpolare cubică spline şi Hermite pe porţiuni

Interpolarea spline şi Hermite cubică pe porţiuni se pot realiza şi direct, apelând funcţia `spline` şi respectiv `pchip`..

Dându-se vectorii `x` şi `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 şi interpolarea se face după coloanele lui `y`; dimensiunea lui `yy` este `length(xx)` pe `size(y,2)`. Funcţia `spline` calculează interpolantul spline de tip deBoor. Dacă `y` conţine cu două valori mai multe decât `x`, se calculează interpolantul spline complet, iar prima şi ultima valoare din `y` se consideră a fi derivatele în capete. (Pentru terminologia privind tipurile de spline vezi secţiunea **??**).

Exemplul pe care îl dăm în continuare ia şase puncte de pe graficul lui $y = \sin(x)$, calculează şi reprezintă grafic spline-ul deBoor şi cel complet (vezi figura 4.3).

```
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)
```

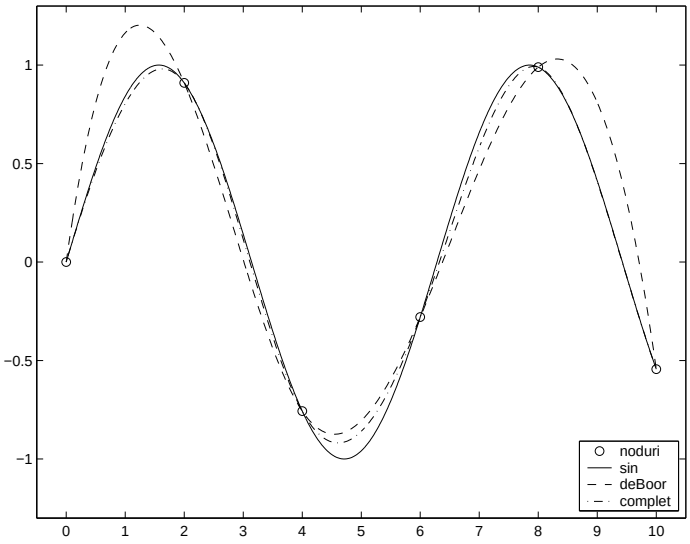


Figura 4.3: Spline deBoor şi complet

Sunt situaţii în care este convenabil să se lucreze cu coeficienţii funcţiei spline (de exemplu dacă nodurile se păstrează şi `xx` se modifică). Comanda `pp=spline(x,y)` memorează coeficienţii într-o structură pp (piecewise polynomial) care conţine forma, nodurile, matricea coeficienţilor (cu 4 coloane pentru spline cubice), numărul de subintervale, ordinul (gradul plus 1) şi dimensiunea. Funcţia `ppval` evaluează spline-ul folosind o astfel de structură. Alte prelucrări de nivel inferior se pot realiza cu `mkpp` (construcţia unei structuri pp) şi `unmkpp` (detalii despre componentele unei structuri pp). De exemplu, comanda `yy = spline(x,y,xx)` se poate înlocui cu scvenţa

```
pp = spline(x,y);
yy = ppval(pp,xx);
```

Vom da acum un exemplu de interpolare spline cu date vectoriale şi utilizare `ppval`. Dorim să citim interactiv mai multe puncte de pe ecran şi să reprezentăm grafic spline-ul parametric care trece prin aceste puncte, cu două rezoluţii diferite (să

zicem cu 20 şi 150 de puncte intermediare pe curbă). Sursa este conţinută în script-ul `splinevect.m` şi 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`. Coeficienţii spline-ului se calculează o singură dată, iar pentru evaluarea spline-ului se foloseşte `ppval`. Propunem cititorului să încerce acest exemplu.

Funcţia `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. Semnificaţia parametrilor este aceeaşi ca în cazul funcţiei `spline`. Exemplul următor calculează interpolantul spline deBoor şi Hermite cubic pe porţiuni pentru un acelaşi 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 4.4. Se observă din nou că interpolantul spline este mai neted, dar interpolantul Hermite cubic pe porţiuni păstrează alura.

4.2. Interpolarea funcţiilor de mai multe variabile în MATLAB

Există două funcţii pentru interpolarea bidimensională în MATLAB: `interp2` şi `griddata`. Sintaxa cea mai generală a lui `interp2` este

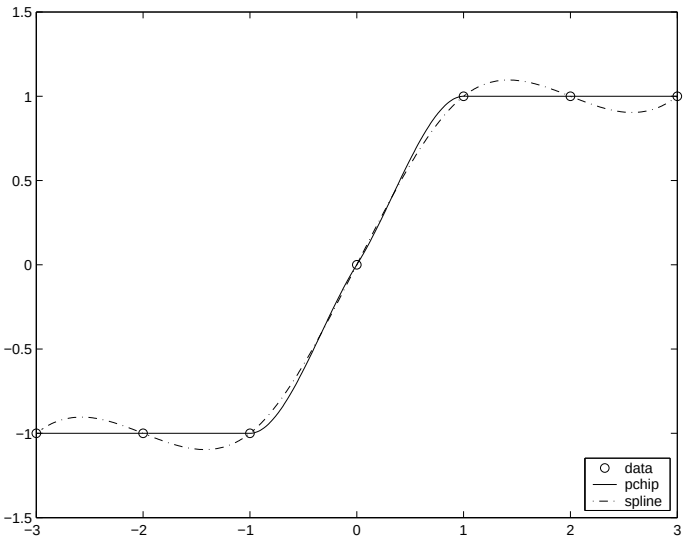


Figura 4.4: Exemplu de utilizare `pchip` şi spline

```
ZI = interp2(x,y,z,XI,YI,metoda)
```

Aici vectorii `x` şi `y` conţin coordonatele nodurilor de interpolare, `z` conţine valorile funcţiei în noduri, iar `XI` şi `YI` sunt matrice ce conţin coordonatele punctelor în care dorim să facem evaluarea. `ZI` conţine 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` şi `y` să fie monotone, şi să aibă formatul ca şi cum ar fi produse de `meshgrid`. Dacă valorile date în `x`, `y`, `XI` şi `YI` nu sunt echidistante, ele sunt transformate într-un domeniu echidistant înainte de interpolare. Pentru eficienţă, dacă `x` şi `y` sunt deja echidistante şi monotone, `metoda` se poate da într-una din formele `'*linear'`, `'*cubic'`, `'*spline'`, sau `'*nearest'`.

Dăm un exemplu care încearcă să interpoleze funcţia `peaks` pe o grilă de 7×7. Generăm grila, calculăm valorile funcţiei şi o reprezentăm grafic cu

```
[X,Y]=meshgrid(-3:1:3); Z=peaks(X,Y); surf(X,Y,Z)
```

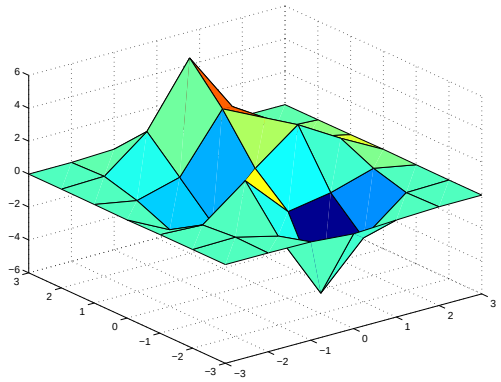


Figura 4.5: Graficul lui `peaks` pe o grilă grosieră

Graficul apare în figura 4.5. Calculăm apoi interpolanţii pe o grilă mai fină şi 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 4.6. Dacă înlocuim peste tot `surf` cu `contour` obţinem graficele din figura 4.7.

Funcţia `griddata` are aceeaşi sintaxă ca şi `interp2`. Datele de intrare sunt nodurile `x` şi `y`, care nu mai trebuie să fie monotone şi valorile `z` în noduri. Prin interpolare se calculează valorile `ZI` corespunzătoare nodurilor `XI` şi `YI`, care de obicei sunt obţinute cu `meshgrid`. Argumentul metoda poate avea valorile `'linear'`, `'cubic'`, `nearest` şi `'v4'`, ultima semnificând o metodă de interpolare specifică MATLAB 4. Toate metodele exceptând `v4` se bazează pe triangulaţie Delaunay (o triangulaţie a unei mulţimi de puncte care maximizează unghiul minim). Metoda este utilă pentru a interpola valori pe o suprafaţă. Exemplul următor interpolează puncte generate aleator, situate pe suprafaţa $z = \frac{\sin(x^2+y^2)}{x^2+y^2}$. 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 4.8, în care punctele generate aleator sunt marcate prin cercuri, iar interpolantul a fost reprezentat cu `mesh`.

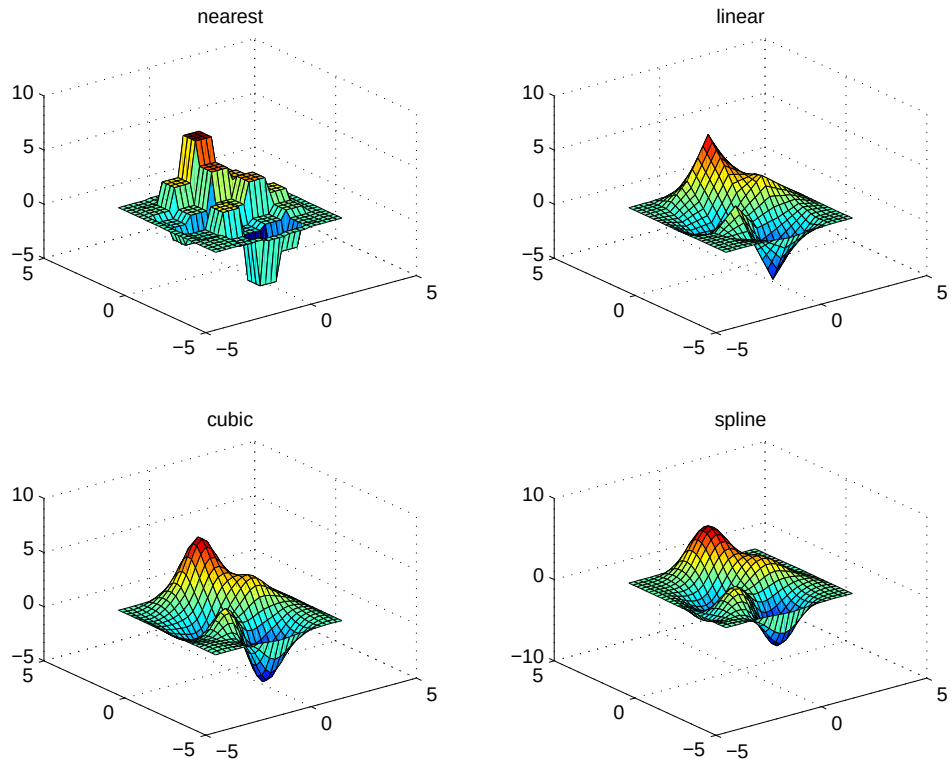


Figura 4.6: Interpolanţi construiţi cu `interp2`

Probleme

Problema 4.1. Să se reprezinte pe acelaşi grafic pentru $[a, b] = [0, 1]$, $n = 11$, funcţia, interpolantul Lagrange şi cel Hermite cu noduri duble în cazurile:

(a) $x_i = \frac{i-1}{n-1}$, $i = \overline{1, n}$, $f(x) = e^{-x}$ şi $f(x) = x^{5/2}$;

(b) $x_i = \left(\frac{i-1}{n-1}\right)^2$, $i = \overline{1, n}$, $f(x) = x^{5/2}$.

Problema 4.2. Aceeaşi problemă, dar pentru cele patru tipuri de interpolanţi spline cubici.

Problema 4.3. Alegând diverse valori ale lui n, pentru fiecare n astfel ales reprezentaţi grafic funcţia lui Lebesgue pentru n noduri echidistante şi respectiv n noduri Cebîşev din intervalul $[0, 1]$.

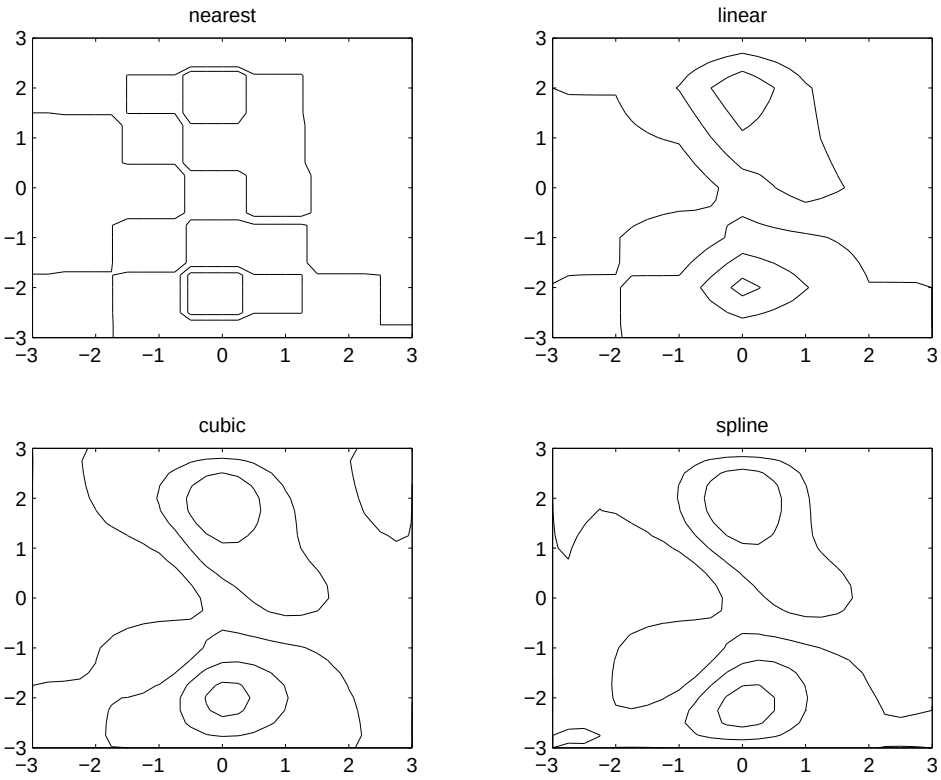


Figura 4.7: Interpolanţi construiţi cu `interp2`

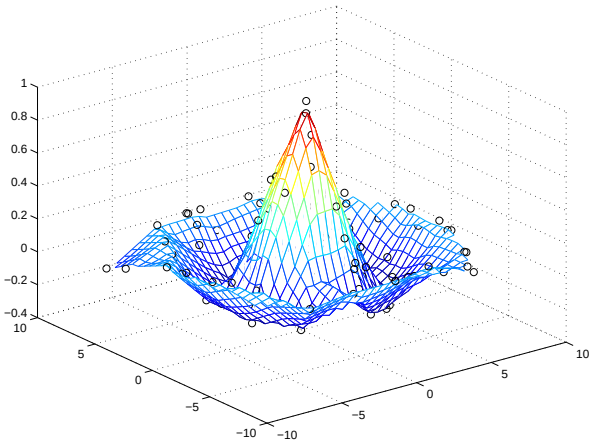


Figura 4.8: Interpolare cu `griddata`

Problema 4.4. Fie punctele $P_i \in \mathbb{R}^2$, $i = 0, n$. Să se scrie:

(a) o funcţie MATLAB care determină o curbă parametrică polinomială de grad n ce trece prin punctele date;

(b) o funcţie MATLAB care determină o curbă parametrică spline cubic ce trece prin punctele date, folosind funcţia pentru spline-ul natural sau cea pentru spline-ul deBoor date în acest capitol.

Testaţi cele două funcţii citind interactiv punctele cu `ginput` şi reprezentând apoi grafic punctele şi cele două curbe astfel determinate.

Problema 4.5. Să se determine o cubică parametrică care trece prin două puncte date şi are în acele puncte vectori tangenţi daţi.

Problema 4.6. Scrieţi o funcţie MATLAB care calculează coeficienţii şi valoarea splinelor cubice de tip Hermite, adică spline cubice de clasă $C^1[a, b]$ care verifică

$$s_3(f, x_i) = f(x_i), \quad s_3'(f, xi) = f'(x_i), \qquad i = \overline{1, n}.$$

Reprezentaţi pe acelaşi grafic funcţia $f(x) = e^{-x^2}$ şi interpolantul corespunzător pentru 5 noduri echidistandte şi 5 noduri Cebîşev pe [0,1].

Problema 4.7. Implementaţi o funcţie MATLAB care calculează inversa matricei Vandermode, folosind rezultatele de la paginile **??**–**??**.

Problema 4.8. [15] Scrieţi o funcţie MATLAB pentru calculul coeficienţilor unui spline periodic de clasă $C^2[a, b]$. Aceasta înseamnă că datele trebuie să verifice $f_n = f_1$ şi că interpolantul rezultat trebuie să fie periodic, de perioadă $x_n - x_1$. Condiţiile de periodicitate de la capete se pot impune mai uşor considerând două puncte suplimentare $x_0 = x_1 - \Delta x_{n-1}$ şi $x_{n+1} = x_n + \Delta x_1$, în care funcţia să ia valorile $f_0 = f_{n-1}$ şi respectiv $f_{n+1} = f_2$.

Problema 4.9. Considerăm datele

```
x = -5:5; y = [0,0,0,1,1,1,0,0,0,0,0];
```

Să se determine coeficienţii aproximantei polinomiale de grad 7 în sensul celor mai mici pătrate corespunzătoare şi să se reprezinte pe acelaşi grafic aproximanta şi polinomul de interpolare Lagrange.

Problema 4.10. Densitatea sodiului sodiului (în kg/m^3) pentru trei temperaturi (în $^\circ C$) este dată în tabela

Temperatura	T_i	94	205	371
Densitatea	ρ_i	929	902	860

(a) Obţineţi polinomul de interpolare Lagrange corespunzător acestor date, folosind toolbox-ul Symbolic.

(b) Determinaţi densitatea pentru $T = 251^\circ$ prin interpolare Lagrange.

Problema 4.11. Aproximaţi

$$y = \frac{1+x}{1+2x+3x^2}$$

pentru $x \in [0, 5]$ folosind interpolarea Lagrange, Hermite şi spline. Alegeţi cinci noduri şi reprezentaţi pe acelaşi grafic funcţia şi interpolanţii. Reprezentaţi apoi erorile de aproximare.

Problema 4.12. Tabela 4.1 dă valorile pentru o proprietate a titanului ca funcţie de temperatura T. Determinaţi şi reprezentaţi grafic o funcţie spline cubică pentru

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 4.1: O proprietate a titanului în funcţie de temperatură

aceste date utilizând 15 noduri. Cât de bine aproximează spline-ul datele în celelalte 6 puncte?

Problema 4.13. Scrieţi o funcţie MATLAB care să reprezinte grafic o suprafaţă $f(x, y)$, $f : [0, 1] \times [0, 1] \to \mathbb{R}$ ce verifică condiţiile

$$f(0, y) = g_1(y) \qquad f(1, y) = g_2(y)$$
$$f(x, 0) = g_3(x) \qquad f(x, 1) = g_4(y),$$

unde g_i, $i = \overline{1, 4}$ sunt funcţii date definite pe $[0, 1]$.

Problema 4.14. Determinaţi interpolanţii bidimensionali produs tensorial şi sumă booleană corespunzători unui interpolant Hermite unidimensional cu nodurile duble 0 şi 1. Reprezentaţi grafic acest interpolant, dacă se alege $f(x, y) = x \exp(x^2 + y^2)$.

CAPITOLUL 5

Funcţii de funcţii

5.1. Integrare numerică în MATLAB

MATLAB are două funcţii de bază pentru integrare numerică, `quad` şi `quadl`. Ambele necesită ca intervalul de integrare $[a, b]$ să fie finit şi integrandul să nu aibă nici o singularitate pe acest interval. Pentru tratarea limitelor infinite sau singularităţilor se pot încerca diverse trucuri cunoscute în analiza numerică cum ar fi schimbarea de variabilă, integrare prin părţi, cuadraturi gaussiene, ş.a.m.d. (vezi [20, 4, 21, 1]).

Cea mai frecventă formă de apel este `q = quad(fun,a,b,tol)` (şi similar pentru `quadl`), unde `fun` este funcţia de integrat. Ea poate fi dată sub formă de şir de caractere, obiect inline sau function handle. Important este să accepte la intrare vectori şi să returneze vectori. Argumentul `tol` este eroarea absolută (implicit 10^{-6}).

Forma `q = quad(fun,a,b,tol,trace)` cu `trace` nenul trasează (urmăreşte) valorile `[fcount a b-a Q]` calculate în timpul aplicării recursive.

Forma `q = quad(fun,a,b,tol,trace,p1,p2,...)` transmite argumentele suplimentare `p1`, `p2`, ..., direct lui `fun`, `fun(x,p1,p2,...)`. În acest caz, pentru a utiliza valori implicite ale lui `tol` sau `trace` în locul lor pe lista de parametri se vor trece matrice vide.

Forma `[q,fcount] = quad(...)` returnează numărul de evaluări de funcţii.

Să presupunem că dorim să aproximăm $\int_0^\pi x \sin x \, \mathrm{d}x$. Integrandul îl putem păstra în fişierul `xsin.m`:

```
function y=xsin(x) y=x.*sin(x);
```

Aproximanta se obţine 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 secţiunea **??** sau în [15]. `quadl` este mai precisă şi se bazează pe o cuadratură de tip Gauss-Lobatto cu 4 puncte (şi grad de exactitate 5) şi o extensie a ei de tip Kronrod cu 7 puncte (şi grad de exactitate 9), ambele descrise în [3]. Cuadratura este adaptivă. Ambele funcţii dau mesaje de avertisment dacă subintervalele devin prea mici sau dacă s-au făcut excesiv de multe evaluări. Astfel de mesaje indică posibile singularităţi.

Pentru a ilustra modul de lucru al lui `quad` şi `quadl` vom aproxima integrala

$$\int_0^1 \left(\frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.09)^2+0.04} - 6 \right) \mathrm{d}x.$$

Integrandul este funcţia MATLAB `humps`, folosită la testarea rutinelor de integrare numerică sau de demo-urile din MATLAB. Vom aplica `quad` acestei funcţii cu `tol=1e-4`. Figura 5.1 reprezintă integrandul şi marchează punctele de pe axa x în care se evaluează integrandul; cercurile corespund valorilor integrandului. Figura arată că subintervalele sunt mai mici acolo unde integrandul variază mai rapid. Ea a fost obţinută modificând funcţia `quad` din MATLAB. Următorul exemplu aproximează integralele lui Fresnel

$$x(t) = \int_0^t \cos(u^2)\,\mathrm{d}u, \qquad y(t) = \int_0^t \sin(u^2)\,\mathrm{d}u.$$

Acestea sunt ecuaţiile parametrice ale unei curbe, numită spirala lui Fresnel. Ea a fost reprezentată în figura 5.2, considerând 1000 de puncte `t` echidistante din intervalul $[-4\pi, 4\pi]$. Din motive de eficienţă, vom exploata simetria şi vom evita integrarea repetată pe $[0, t]$, integrând pe fiecare subinterval şi 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
```

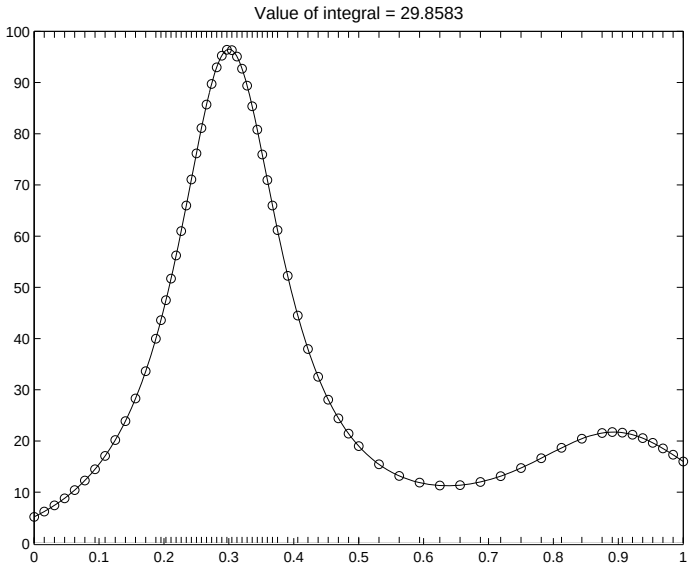


Figura 5.1: Integrarea numerică a lui `humps` prin `quad`

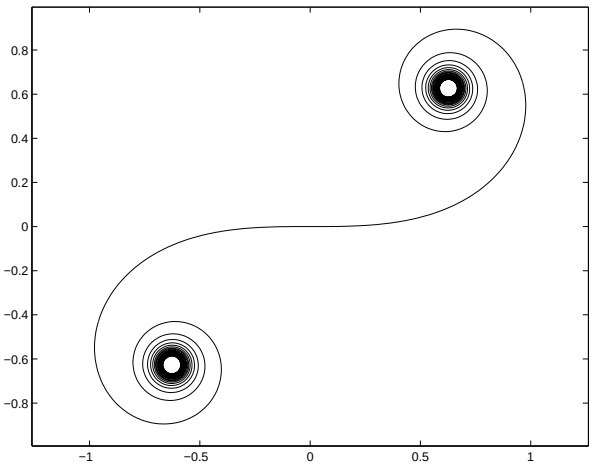


Figura 5.2: Spirala lui Fresnel

Pentru a integra funcţii date prin valori, nu prin expresia lor analitică, se foloseşte funcţia `trapz`. Ea implementează regula trapezelor (nodurile nu trebuie să fie echidistante). Aşa cum am văzut în secţiunea **??**, aceasta dă rezultate bune la integrarea funcţiilor periodice pe intervale a căror lungime este un multiplu întreg al perioadei. Exemplu:

```
>> 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}.

5.2. Calculul integralelor duble în MATLAB

Integralele duble pe un dreptunghi pot fi evaluate cu `dblquad`. Pentru ilustrare, să presupunem că dorim să aproximăm integrala

$$\int_0^{\pi}\int_{\pi}^{2\pi}(y\sin x + x\cos y)\,\mathrm{d}x\,\mathrm{d}y$$

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 şir de caractere, sau function handle. Să presupunem că integrandul este dat în fişierul `integrand.m`:

```
function z = integrand(x, y) z = y*sin(x)+x*cos(y);
```

Vom utiliza `dblquad` şi vom face şi verificarea:

```
>> Q = dblquad(@integrand, pi, 2*pi, 0, pi)
Q =
   -9.8696
>> -pi^2
ans =
   -9.8696
```

Calculăm integrala şi cu `quaddbl` (sursa MATLAB **??**):

```
>> Q2=quaddbl(@integrand,pi,2*pi,0,pi)
Q2 =
   -9.8696
```

Integrandul transmis lui `dblquad` trebuie să accepte un vector `x` şi un scalar `y` şi să returneze un vector. Se pot transmite argumente suplimentare lui `dblquad` pentru a specifica precizia şi metoda de integrare unidimensională (implicit `quad`). Să presupunem că vrem să calculăm

$$\int_4^6 \int_0^1 (y^2 e^x + x \cos y)\, \mathrm{d}x\, \mathrm{d}y$$

cu precizia `1e-8` şi 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
```

5.3. Ecuaţii neliniare şi minimizare

MATLAB are puţine rutine pentru determinarea rădăcinilor unei funcţii de o variabilă. Dacă funcţia este polinomială, am văzut că `roots(p)` returnează rădăcinile lui `p`, unde `p` este vectorul coeficienţilor ordonaţi descrescător după puterile variabilei.

Funcţia `fzero` determină o rădăcină a unei funcţii de o variabilă. Algoritmul folosit de `fzero` este o combinaţie de metode: înjumătăţire, secantă şi interpolare inversă pătratică [15]. 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 şi modul de afişare a rezultatelor sunt controlate de un al treilea argument de intrare, structura `options`, care se poate seta cu ajutorul funcţiei `optimset`. Funcţia `fzero` utilizează două câmpuri ale lui `options`: `TolX` care dă precizia (toleranţa) şi `Display` care specifică nivelul de raportare, cu valorile `off` pentru nici o ieşire, `iter` pentru ieşire la fiecare iteraţie, `final` pentru a afişa doar rezultatul final şi `notify` pentru a afişa rezultatul numai dacă funcţia nu converge (implicit). Pentru exemplul precedent, utilizând `Display` cu `final` obţinem:

```
>> fzero('cos(x)-x',0,optimset('Display','final'))
Zero found in the interval [-0.905097, 0.905097] ans =
    0.7391
```

Argumentul de intrare `x0` poate fi şi un interval la ale cărui capete funcţia să aibă valori de semne contrare. Un astfel de argument este util dacă funcţia are singularităţi. 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 ieşire este valoarea funcţiei în zeroul calculat. Deoarece funcţia $f(x) = \tan x - x$ are o singularitate în $\pi/2$ (vezi figura 5.3) vom da ca argument de pornire un interval ce conţine un zero, dar fără singularităţi:

```
>> [x,fval]=fzero('tan(x)-x',[-1,1],os)
Zero found in the interval: [-1, 1]. x =
     0
fval =
     0
```

Se pot transmite parametrii suplimentari `p1,p2,...` funcţiei `f` cu apeluri de forma

```
x = fzero(f,x0,options,p1,p2,...)
```

Dacă se doreşte ca `options` să aibă valori implicite se poate folosi pe poziţia respectivă matricea vidă `[]`.

MATLAB nu are nici o funcţie pentru rezolvarea sistemelor de ecuaţii neliniare. Totuşi, se poate încerca rezolvarea unor astfel de sisteme prin minimizarea sumei

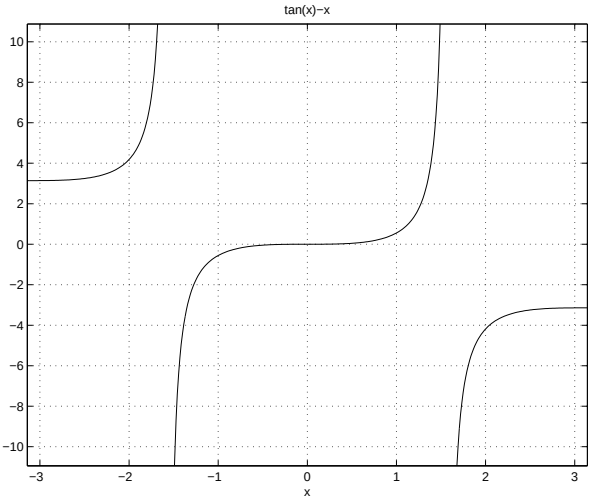


Figura 5.3: Singularitate a funcţiei $f(x) = \tan x - x$, evidenţiată cu `ezplot('tan(x)-x',[-pi,pi]),grid`

pătratelor reziduurilor. Toolbox-ul Optimization conţine un rezolvitor de ecuaţii neliniare.

Funcţia `fminsearch` caută un minim local al unei funcţii reale de n variabile reale. O formă posibilă de apel este `x=fminsearch(f,x0,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 funcţii permise), `MaxIter` (numărul maxim de iteraţii permise), `TolFun` (precizia de terminare pentru valoarea funcţiei). Valoarea implicită pentru `TolX` şi `TolFun` este `1e-4`.

Exemplul 5.3.1. Sistemul neliniar din exemplele **??** şi **??**, adică

$$\begin{aligned} f_1(x_1,x_2,x_3) &:= 3x_1 - \cos(x_1x_2) - \frac{1}{2} = 0, \\ f_2(x_1,x_2,x_3) &:= x_1^2 - 81(x_2+0.1)^2 + \sin x_3 + 1.06 = 0, \\ f_3(x_1,x_2,x_3) &:= e^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{aligned}$$

ar putea fi rezolvat încercând minimizarea sumei pătratelor membrilor stângi:

$$F(x_1,x_2,x_3) = [f_1(x_1,x_2,x_3)]^2 + [f_1(x_1,x_2,x_3)]^2 + [f_1(x_1,x_2,x_3)]^2.$$

Funcţia de minimizat este dată în fişierul `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 $x^{(0)} = [0.5, 0.5, 0.5]^T$ şi precizia 10^{-9} şi pentru x. Comenzile MATLAB şi 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 =
   0.49999999959246
   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 execuţie este mai lung şi precizia nu este la fel de bună (şi datorită aspectului mai complicat al funcţiei obiectiv). Aproximanta obţinută astfel poate fi folosită ca vector de pornire pentru metoda lui Newton. ♢

Funcţia `fminsearch` se bazează pe varianta Nelder-Mead a algoritmului simplex. Algoritmul este lent, dar are avantajul că nu utilizează derivate şi este insensibil la discontinuităţi ale funcţiei. Toolbox-ul Optimization conţine metode mai sofisticate de minimizare.

Probleme

Problema 5.1. Să se aproximeze

$$\int_0^1 \frac{\sin x}{x}\,\mathrm{d}x,$$

folosind o cuadratură adaptivă şi metoda lui Romberg. Ce probleme pot să apară? Să se compare rezultatul cu cel furnizat de `quad` sau `quadl`.

Problema 5.2. Pornind de la o integrală convenabilă, să se aproximeze π cu 8 zecimale exacte, folosind metoda lui Romberg şi o cuadratură adaptivă.

Problema 5.3. Aproximaţi

$$\int_{-1}^1 \frac{2}{1+x^2}\,\mathrm{d}x$$

folosind formula trapezelor şi formula repetată a lui Simpson, pentru diverse valori ale lui n. Cum variază precizia odată cu n? Reprezentaţi grafic.

Problema 5.4. Funcţia eroare, erf, se defineşte prin

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, \mathrm{d}t.$$

Tabelaţi valorile acestei funcţii pentru $x = 0.1, 0.2, \ldots, 1$, utilizând funcţia `adquad`. Să se compare rezultatele cu cele furnizate de funcţiile MATLAB `quad` şi `erf`.

Problema 5.5. (a) Utilizaţi `adquad` şi funcţia `quad` din MATLAB pentru a aproxima

$$\int_{-1}^{2} \frac{1}{\sin \sqrt{|t|}} \, \mathrm{d}t.$$

(b) De ce nu apar probleme de tip împărţire la zero în $t = 0$?

Problema 5.6. Pentru un număr $p \in \mathbb{N}$ considerăm integrala

$$I_p = \int_0^1 (1-t)^p f(t) \, \mathrm{d}t.$$

Să se compare formula trapezelor pentru n subintervale cu formula Gauss-Jacobi cu n noduri şi parametrii $\alpha = p$ şi $\beta = 0$. Luaţi, de exemplu, $f(t) = \mathrm{tg}t$, $p = 5(5)20$ şi $n = 10(10)50$ în cazul formulei trapezelor şi $n = 1(1)5$ pentru formula Gauss-Jacobi.

Problema 5.7. Fie

$$f(x) = \ln(1+x) \ln(1-x).$$

(a) Utilizaţi `ezplot` pentru a reprezenta grafic $f(x)$ pentru $x \in [-1, 1]$.

(b) Utilizaţi Maple sau toolbox-ul Symbolic pentru a obţine valoarea exactă a integralei

$$\int_{-1}^{1} f(x) \, \mathrm{d}x.$$

(c) Găsiţi valoarea numerică a expresiei de la (b).

(d) Ce se întâmplă dacă încercăm să utilizăm

$$\texttt{adquad}(\texttt{'log(1 + x).* log(1 - x)'}, -1, 1).$$

la aproximarea valorii integralei?

(e) Cum evitaţi dificultatea? Justificaţi soluţia.

(f) Utilizaţi `adquad` cu diverse precizii (toleranţe). Reprezentaţi grafic eroarea şi numărul de evaluări în funcţie de toleranţă.

Problema 5.8. Extindeţi algoritmul de Casteljau la cazul bidimensional. Daţi o implementare MATLAB.

Problema 5.9. Extindeţi algoritmul Cox - de Boor la cazul bidimensional. Daţi o implementare MATLAB.

Problema 5.10. Adaptaţi rutina `quaddbl` pentru a aproxima integrale duble de forma

$$\int_a^b \int_{c(x)}^{d(x)} f(x,y)\,\mathrm{d}y\,\mathrm{d}x$$

sau

$$\int_c^d \int_{a(y)}^{b(y)} f(x,y)\,\mathrm{d}x\,\mathrm{d}y,$$

când domeniul de integrare este simplu în raport cu x sau y.

Problema 5.11. Se consideră integrala dublă a funcţiei $f(x,y) = x^2 + y^2$ pe domeniul eliptic R dat de $-5 < x < 5$, $y^2 < \frac{3}{5}(25 - x^2)$.

(a) Să se reprezinte grafic funcţia 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 intr-unul rectangular.

(d) Să se aproximeze valoarea integralei folosind funcţiile din problema 5.10.

Problema 5.12. Se consideră funcţia $f(x,y) = y\cos x^2$ şi domeniul triunghiular definit de $T = \{x \geq 0, y \geq 0, x + y <= 1\}$ şi

$$\int\int_T f(x,y)\,\mathrm{d}x\,\mathrm{d}y.$$

(a) Să se reprezinte grafic funcţia pe T folosind `trimesh` sau `trisurf`.

(b) Să se aproximeze valoarea lui I transformând integrala într-o integrală pe pătratul unitate a unei funcţii care este nulă înafara lui T.

(c) Să se aproximeze integrala folosind funcţiile din problema 5.10.

Problema 5.13. Găsiţi primele 10 valori pozitive pentru care $x = \mathrm{tg}x$.

Problema 5.14. Investigaţi comportarea metodei lui Newton şi a secantei pentru funcţia

$$f(x) = \mathrm{sign}(x-a)\sqrt{|x-a|}.$$

Problema 5.15 (Adaptată după [15]). Considerăm polinomul

$$x^3 - 2x - 5.$$

Wallis a folosit acest exemplu pentru a prezenta metoda lui Newton în faţa academiei franceze. El are o rădăcină reale în intervalul $(2, 3)$ şi o pereche de rădăcini complexe conjugate.

(a) Utilizaţi Maple sau toolbox-ul Symbolic pentru a calcula rădăcinile. Rezultatele sunt urâte. Convertiţi-le în valori numerice.

(b) Determinaţi toate rădăcinile cu funcţia `roots`.

(c) Determinaţi rădăcina reală cu `fzero`.

(d) Determinaţi toate rădăcinile cu metoda lui Newton (pentru cele complexe folosiţi valori de pornire complexe).

(e) Se poate utiliza metoda înjumătăţirii sau a falsei poziţii la determinarea unei rădăcini complexe? De ce sau de ce nu?

Problema 5.16. Să se rezolve numeric sistemele

$$\begin{aligned} f_1(x,y) &= 1 - 4x + 2x^2 - 2y^3 = 0 \\ f_2(x,y) &= -4 + x^4 + 4y + 4y^4 = 0, \end{aligned}$$

$$\begin{aligned} f_1(x_1,x_2) &= x_1^2 - x_2 + 0.25 = 0 \\ f_2(x_1,x_2) &= -x_1 + x_2^2 + 0.25 = 0, \end{aligned}$$

$$\begin{aligned} f_1(x_1,x_2) &= 2x_1 + x_2 - x_1x_2/2 - 2 = 0 \\ f_2(x_1,x_2) &= x_1 + 2x_2^2 - \cos(x_2)/2 - \frac{3}{2} = 0. \end{aligned}$$

Problema 5.17. Să se rezolve numeric sistemul

$$\begin{aligned} &9x^2 + 36y^2 + 4z^2 - 36 = 0, \\ &x^2 - 2y^2 - 20z = 0, \\ &x^2 - y^2 + z^2 = 0 \end{aligned}$$

Indicaţie. Sunt patru soluţii. Valori bune de pornire $[\pm 1, \pm 1, 0]^T$.

Problema 5.18. 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 ecuaţii şi $n + 2$ necunoscute. Vom lua $a_0 = 5$, $a_n = 0.5$ mol/litru. Să se rezolve sistemul pentru $n = 10$ şi valoarea de pornire `x=[1:-0.1:0.1]'`.

CAPITOLUL 6

Ecuaţii diferenţiale în MATLAB

6.1. Rezolvitori

MATLAB are facilităţi foarte puternice de rezolvare a problemelor cu valori iniţiale pentru ecuaţii diferenţiale ordinare:

$$\frac{d}{dt}y(t) = f(t, y(t)), \qquad y(t_0) = y_0.$$

Cel mai simplu mod de a rezolva o astfel de problemă este de a scrie o funcţie care evaluează f şi de a apela unul dintre rezolvitorii MATLAB. Informaţia minimă pe care un rezolvitor trebuie să o primească este numele funcţiei, mulţimea valorilor lui t pe care se cere soluţia şi valoarea iniţială y_0. Rezolvitorii MATLAB acceptă argumente de intrare şi ieşire opţionale care permit să se specifice mai mult despre problema matematică şi modul de rezolvare a ei. Fiecare rezolvitor MATLAB este conceput să fie eficient în anumite situaţii, dar toţi sunt în esenţă interschimbabili. Toţi rezolvitorii au aceeaşi sintaxă, ceea ce ne permite să încercăm diferite metode numerice atunci când nu ştim care ar fi cea mai potrivită. Sintaxa este

```
[t,y]=rezolvitor(@fun,tspan,y0,optiuni,p1,p2,...)
```

unde `rezolvitor` este unul din rezolvitorii daţi în tabela 6.1.
Argumentele de intrare sunt

- `fun` – specifică funcţia din membrul drept. În versiunile 6.x este un handler de funcţie, iar în versiunile 5.x este un nume de funcţie (în acest caz se scrie `'fun'` nu `@fun`);

Rezolvitor	tip problemă	Tip algoritm
`ode45`	Nonstiff	Pereche Runge-Kutta explicită, cu ordinele 4 şi 5
`ode23`	Nonstiff	Pereche Runge-Kutta explicită, cu ordinele 2 şi 3
`ode113`	Nonstiff	Metodă cu mai mulţi paşi explicită, cu ordin variabil, ordinele de la 1 la 13
`ode15s`	Stiff	Metodă cu mai mulţi paşi implicită, cu ordin variabil, ordinele de la 1 la 15
`ode23s`	Stiff	Pereche Rosenbrock modificată (cu un pas), cu ordinele 2 şi 3
`ode23t`	Stiff	Regula implicită a trapezului, cu ordinele 2 şi 3
`ode23tb`	Stiff	Algoritm Runge-Kutta implicit, ordinele 2 şi 3

Tabela 6.1: Rezolvitori MATLAB pentru ecuaţii diferenţiale ordinare

- `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 soluţiile în acele puncte. Abscisele trebuie ordonate crescător sau descrescător. Rezolvitorul nu îşi alege paşii după valorile din `tspan`, ci obţine valorile în aceste puncte prin prelungiri continue ale formulelor de bază care au acelaşi ordin de precizie ca şi soluţiile calculate în puncte.
- `optiuni` – opţiunile permit setarea unor parametrii ai rezolvitorului şi se crează cu `odeset`.

Parametrii de ieşire sunt:

- `t` – vectorul coloană al absciselor;
- `y` – tabloul soluţiilor: o linie corespunde unei abscise, iar o coloană unei componente a soluţiei.

După `optiuni` pot să apară parametrii variabili, `p1`, `p2`, `...` care sunt transmişi funcţiei `fun` la fiecare apel. De exemplu

```
[t,y]=rezolvitor(@fun,tspan,y0,optiuni,p1,p2,...)
```

apelează

```
fun(T,Y,flag,p1,p2,...).
```

6.2. Exemple non-stiff

Să considerăm ecuaţia scalară

$$y'(t) = -y(t) + 5e^{-t}\cos 5t, \qquad y(0) = 0,$$

pentru $t \in [0, 3]$. Membrul drept este conţinut în fişierul `f1scal.m`:

```
function yder=f1scal(t,y)
%F1SCAL Exemplu de EDO scalara
yder = -y+5*exp(-t).*cos(5*t);
```

Vom folosi rezolvitorul `ode45`. Secvenţa 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 6.1. Soluţia exactă este $y(t) = e^{-t}\sin 5t$. Verificăm

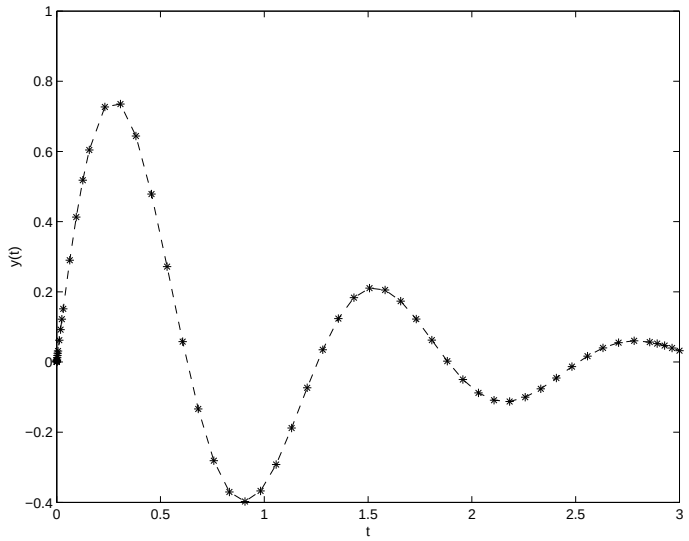


Figura 6.1: Un exemplu de ecuaţie diferenţială ordinară scalară

aceasta calculând maximul modulului diferenţelor dintre valorile furnizate de `ode45` şi valorile soluţiei exacte calculate în abscisele furnizate de `ode45`:

```
>> norm(y-exp(-t).*sin(5*t),inf)
ans =
  3.8416e-004
```

Să considerăm acum ecuaţia pendulului simplu [2, secţiunea 1.4]:

$$\frac{d^2}{dt^2}\theta(t) = -\frac{g}{L}\sin\theta(t),$$

unde g este acceleraţia gravitaţională şi L este lungimea pendulului. Această ecuaţie de ordinul al doilea se poate transforma într-un sistem de ecuaţii de ordinul I, introducând necunoscutele $y_1(t) = \theta(t)$ şi $y_2(t) = d\theta(t)/dt$:

$$\begin{aligned} \frac{d}{dt}y_1(t) &= y_2(t), \\ \frac{d}{dt}y_2(t) &= -\frac{g}{L}\sin y_1(t). \end{aligned}$$

Aceste ecuaţii sunt codificate în fişierul `pend.m`, dat în continuare:

```
function yp=pend(t,y,g,L)
%PEND - pendul simplu
%g - acceleratia gravitationala, L - lungimea
yp=[y(2); -g/L*sin(y(1))];
```

Aici, `g` şi `L` sunt parametrii suplimentari care vor fi furnizaţi lui `pend` de către rezolvitor. Vom calcula soluţia pentru $t \in [0, 10]$ şi trei condiţii iniţiale 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 opţiunile, iar următorii sunt parametrii suplimentari care vor fi transmişi membrului drept. Pentru a obţine grafice de fază vom reprezenta pe $y_2(t)$ în funcţie de $y_1(t)$. Este sugestiv să reprezentăm pe acelaşi grafic şi câmpul de vectori cu `quiver`. Săgeţile generate de `quiver` au direcţia gradientului $[y_2, -\sin y_1]$ şi lungimea proporţională cu norma euclidiană a acestui vector. Imaginea obţinută apare în figura 6.2.

```
[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 soluţie a ecuaţiei 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
```

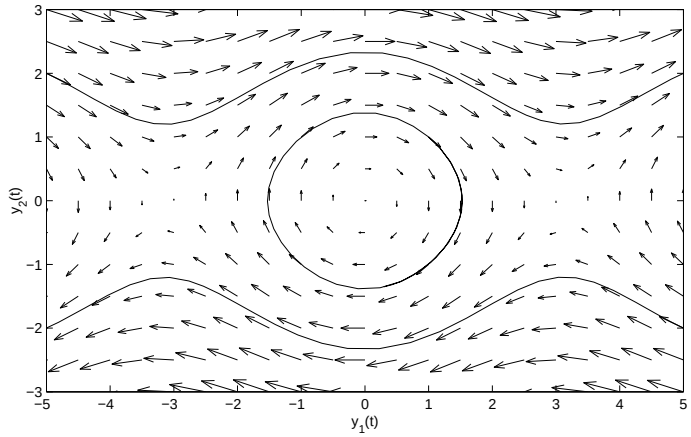


Figura 6.2: Grafic de fază pentru ecuaţia pendulului

6.3. Opţiuni

Funcţia `odeset` crează o structură de opţiuni 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 opţiuni în care proprietăţile cu numele dat primesc o valoare specificată. Proprietăţile nespecificate primesc valori implicite. Pentru toate proprietăţile este suficient să dăm doar caracterele de la început care identifică unic numele proprietăţii. `odeset` fără argumente afişează toate numele de proprietăţi şi valorile lor posibile; valorile implicite apar între acolade:

```
>> odeset
          AbsTol: [ positive scalar or vector {1e-6} ]
          RelTol: [ positive scalar {1e-3} ]
     NormControl: [ on | {off} ]
       OutputFcn: [ function ]
```

Categoria	**Numele proprietăţii**
Controlul erorii	`RelTol`, `AbsTol`, `NormControl`
Ieşire rezolvitor	`OutputFcn`, `OutputSel`, `Refine`, `Stats`
Matrice jacobiană	`Jacobian`, `JPattern`, `Vectorized`
Controlul pasului	`InitialStep`, `MaxStep`
Matrice de masă şi DAE	`Mass`, `MStateDependence`, `MvPattern`, `MassSingular`, `InitialSlope`
Evenimente	`Events`
specifice `ode15s`	`MaxOrder`, `BDF`

Tabela 6.2: Proprietăţi ale rezolvitorilor

```
      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 ]
     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 opţiuni se poate realiza cu

```
optiuni=odeset(optiune-veche,'nume1', valoare1,...)
```

care poziţionează valorile lui opţiuni pe valorile existente in `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` şi `optiune-noua`. Valorile din `optiune-noua` diferite de `[]` le înlocuiesc pe cele din `optiune-veche`. O structură de opţiuni poate fi interogată cu comanda:

```
o=odeget(optiuni,'name')
```

Aceasta returnează valoarea proprietăţii specificate sau o matrice nulă dacă valoarea proprietăţii nu este specificată în structura `optiuni`. Tabela 6.2 dă tipurile de proprietăţi şi numele proprietăţilor.

Exemplul următor rezolvă sistemul lui Rössler [5, secţiunea 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 şi c sunt parametrii reali. Funcţia care defineşte ecuaţia diferenţială 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 şi cea relativă cu

```
options = odeset('AbsTol',1e-7,'RelTol',1e-4);
```

Script-ul `Roessler.m` (sursa 6.1) rezolvă sistemul lui Rössler pe intervalul $t \in [0, 100]$ cu valoarea iniţială $y(0) = [1, 1, 1]^T$ şi cu seturile de parametrii $(a, b, c) = (0.2, 0.2, 2.5)$ şi $(a, b, c) = (0.2, 0.2, 5)$. Rezultatele apar în figura 6.3. Subplot-ul

Sursa MATLAB 6.1 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)')
```

`221` dă graficul soluţiei în subspaţiul tridimensional al fazelor pentru $c = 2.5$ şi subplot-ul `223` dă proiecţia ei pe planul y_1y_2. Subplot-urile `222` şi `224` dau graficele corespunzătoare pentru $c = 5$. Vom mai discuta şi exemplifica opţiunile şi în subsecţiunile următoare. Pentru detalii a se vedea `help odest` sau `doc odeset`.

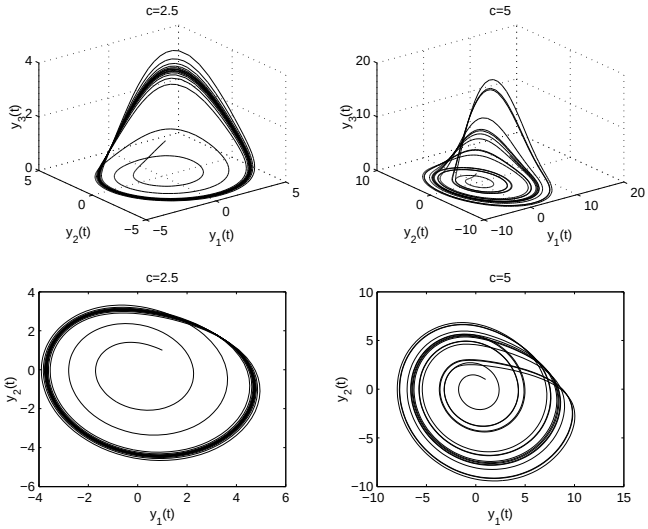


Figura 6.3: Soluţiile sistemului lui Rössler în spaţiul fazelor

6.4. Ecuaţii stiff

”Stiff” (ţeapăn, rigid, dificil, anevoios) este un concept subtil, dificil şi important în rezolvarea numerică a ecuaţiilor diferenţiale ordinare. El depinde de ecuaţia diferenţială, de condiţiile iniţiale şi de metoda numerică. În [15] se dă următoarea caracterizare computaţională a acestui termen:

> ,,O problemă este stiff dacă soluţia căutată variază lent, dar există soluţii apropiate care variază rapid, astfel că metoda numerică trebuie să utilizeze paşi foarte mici pentru a obţine rezultate satisfăcătoare.”

Conceptul de ”stiffness” este o chestiune de eficienţă. Metodele nonstiff pot rezolva problemele stiff, dar într-un timp foarte lung.

Exemplul care urmează provine din [15] şi este un model al propagării unei flăcări. Când se aprinde un chibrit, mingea (sfera) de foc creşte 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 suprafaţă. Un model simplu este dat de problema cu valori iniţiale:

$$\begin{aligned} &y' = y^2 - y^3, \\ &y(0) = \delta, \qquad 0 \le t \le 2/\delta \end{aligned} \tag{6.4.1}$$

Funcţia reală $y(t)$ reprezintă raza sferei. Termenii y^2 şi y^3 provin din suprafaţă şi volum. Soluţia se caută pe un interval de lungime invers proporţională cu δ. Vom

încerca să rezolvăm problema cu `ode45`. Dacă δ nu este foarte mic problema nu este foarte stiff. Alegem $\delta = 0.01$ şi eroarea relativă 10^{-4}.

```
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 ieşire, rezolvitorul reprezintă grafic soluţia (vezi figura 6.4). Se observă că ea porneşte de la 0.1, creşte lent până când t se apropie de $1/\delta$, adică 100 şi apoi creşte rapid până la valoarea 1, ajungându-se într-o stare de echilibru. Se utilizează 185 de puncte. Dacă luăm δ mai mic, de exemplu 0.0001, caracterul

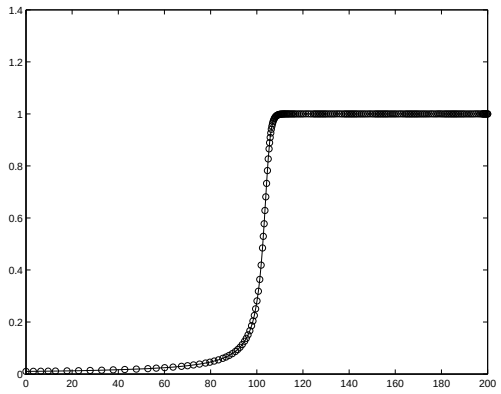


Figura 6.4: Propagarea unei flăcări, $\delta = 0.1$

stiff devine mai pregnat. Se generează 12161 de puncte. Graficul apare în figura 6.5, în partea de sus. În partea de jos este un zoom obţinut în vecinătatea stării de echilibru. Figura a fost obţinută cu script-ul `chibrit2.m` dat mai jos. Opţiunea `Stats` setată pe `on` permite obţinerea 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 menţine soluţia în limita de eroare cerută, dar cu preţul unui efort foarte mare. Situaţia este şi mai dramatică pentru valori mai mici ale erorii relative, de exemplu 10^{-5} sau 10^{-6}. Iniţial problema nu este stiff. Ea devine astfel pe

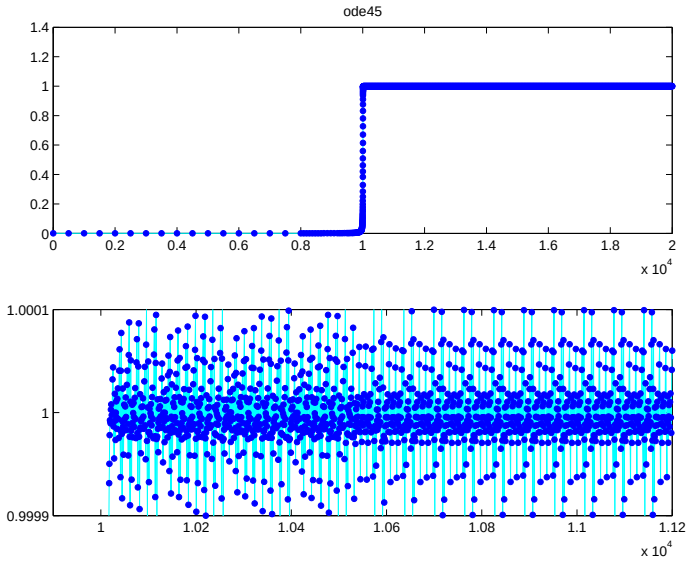


Figura 6.5: Propagarea unei flăcări, $\delta = 0.0001$, eroarea relativă 1e-4 (sus) şi un zoom pe soluţie (jos)

măsură ce se apropie de starea de echilibru, adică de valoare $y(t) = 1$, unde are loc o creştere 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 operaţii matriciale şi rezolvă sisteme de ecuaţii liniare care îi ajută să prevadă evoluţia soluţiei (desigur că pentru probleme scalare matricea are dimensiunea 1 pe 1). Să calculăm soluţia folosind acum rezolvitorul `ode23s`. Vom modifica în script-ul precedent doar numele rezolvitorului: `ode23s` în loc de `ode45`. Graficul obţinut apare în figura 6.6. Efortul depus de rezolvitor este de această dată mult mai mic, după cum se poate vedea examinând statisticile generate de rezolvitor. Statistica generată de `ode23s` este în acest caz:

```
99 successful steps
7 failed attempts
```

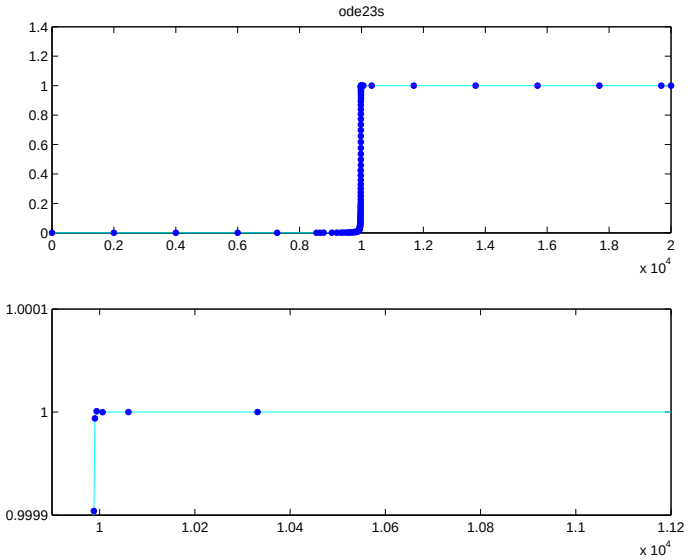


Figura 6.6: Propagarea unei flăcări, $\delta = 0.0001$, eroarea relativă 1e-4 (sus) şi un zoom pe soluţie (jos), obţinute cu `ode23s`

```
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 obţine soluţia exactă a problemei (6.4.1). Ecuaţia este cu variabile separabile. Integrând se obţine o ecuaţie 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 soluţiei exacte este

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}$$

unde $a = 1/\delta - 1$, iar W este funcţia lui Lambert. Aceasta din urmă este soluţia ecuaţiei funcţionale

$$W(z)e^{W(z)} = z.$$

Cleve Moler [15] dă o comparaţie foarte plastică între rezolvitorii expliciţi şi cei impliciţi.

> „Imaginaţi-vă că vă întoarceţi dintr-o plimbare pe munte. Sunteţi într-un canion îngust cu pante abrupte de ambele părţi. Un algoritm explicit va selecta gradientul local pentru a găsi direcţia de coborâre. Dar urmarea gradientului în ambele părţi ale drumului vă va trimite viguros înainte şi înapoi prin canion, la fel ca `ode45`. Veţi ajunge în final acasă, dar întunericul se va lăsa cu mult timp înainte de a ajunge. Un algoritm implicit vă va ţine cu ochii la drum şi va anticipa unde vă va duce fiecare pas. Concentrarea suplimentară merită."

Vom considera acum un exemplu datorat lui Robertson şi tratat amănunţit în [5, 2]. Sistemul lui Robertson

$$\begin{aligned}
\frac{d}{dt}y_1(t) &= -\alpha y_1(t) + \beta y_2(t)y_3(t), \\
\frac{d}{dt}y_2(t) &= \alpha y_1(t) - \beta y_2(t)y_3(t) - \gamma y_2^2(t), \\
\frac{d}{dt}y_3(t) &= \gamma y_2^2(t)
\end{aligned}$$

modelează reacţia între trei specii chimice. Sistemul este codificat în funcţia `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 $\alpha = 0.04$, $\beta = 10^4$, $\gamma = 3 \times 10^7$, $t \in [0, 3]$ şi condiţiile iniţiale $y(0) = [1, 0, 0]^T$. Se utilizează rezolvitorii `ode45` şi `ode15s`, generându-se şi 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)
```

Graficele lui y_2 apar în figura 6.7. Figura 6.8 conţine un zoom pe graficul obţinut cu `ode45`. Statisticile generate de rezolvitori sunt

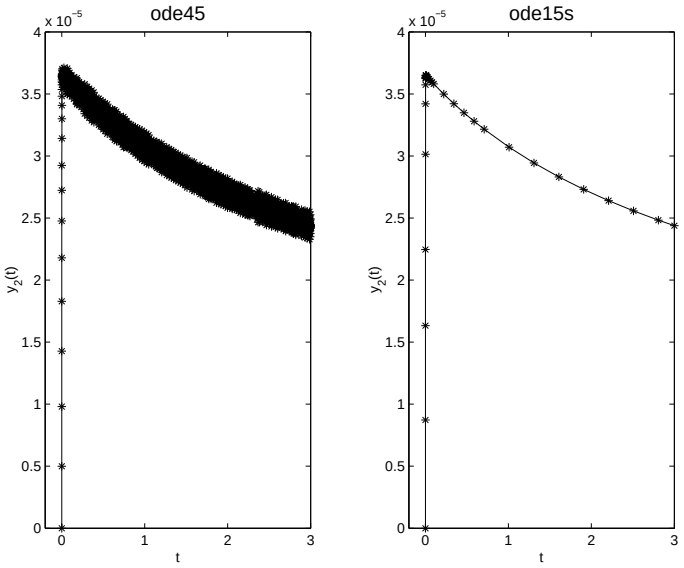


Figura 6.7: Soluţia y_2 a sistemului lui Robertson, obţinută cu `ode45` (stânga) şi `ode15s`

```
2052 successful steps
440 failed attempts
14953 function evaluations
0 partial derivatives
0 LU decompositions
0 solutions of linear systems
```

pentru `ode45` şi

```
33 successful steps
```

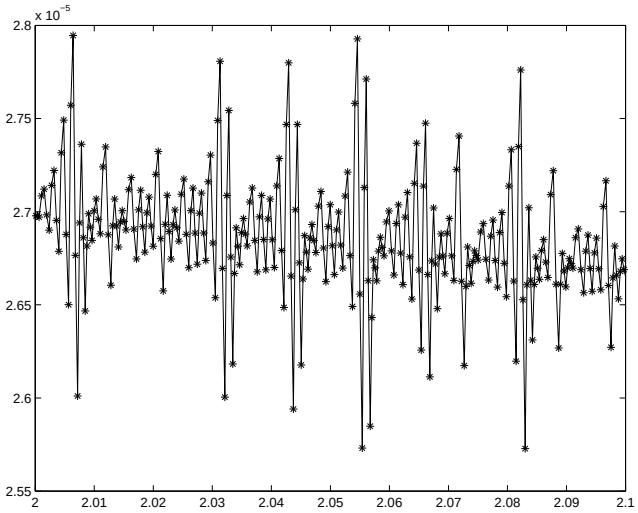


Figura 6.8: Zoom pe soluţia y_2 a sistemului lui Robertson, obţinută cu `ode45`

```
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`.

Rezolvitorii stiff utilizează informaţii despre matricea jacobiană, $\partial f_i/\partial y_j$, în diferite puncte ale soluţiei. Implicit, se generează automat aproximaţii ale jacobianului utilizând diferenţe finite. Totuşi, dacă jacobianul se dă expicit, se obţine o eficienţă şi o robusteţe mai mare a codului. Sund disponibile opţiuni care permit să se specifice o funcţie care evaluează jacobianul sau o matrice constantă (`Jacobian`), dacă jacobianul este rar şi şablonul de raritate (`Jspattern`), dacă este scris în formă vectorială (`Vectorized`). Pentru a ilustra modul de codificare a jacobianului, vom considera sistemul

$$\frac{d}{dt}y(t) = Ay(t) + y(t).*(1 - y(t)) + v, \tag{6.4.2}$$

unde A este o matrice $N \times N$ şi v este un vector $N \times 1$ cu

$$A = r_1 \begin{bmatrix} 0 & 1 & & & \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & -1 & 0 \end{bmatrix} + r_2 \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{bmatrix},$$

$v = [r_1 - r2, 0, \ldots, 0, r_1 + r_2]^T$, $r_1 = -a/(2\Delta x)$ şi $r_2 = b/\Delta x^2$. Aici a, b şi Δx sunt parametrii cu valorile $a = 1$, $b = 5 \times 10^{-2}$ şi $\Delta x = 1/(N+1)$. Acest sistem de ecuaţii diferenţiale ordinare apare la semidiscretizarea prin metoda liniilor a ecuaţiei cu derivate parţiale

$$\frac{\partial}{\partial t}u(x,t) + a\frac{\partial}{\partial x}u(x,t) = b\frac{\partial^2}{\partial x^2}u(x,t) + u(x,t)(1-u(x,t)), \qquad 0 \le x \le 1,$$

cu condiţiile pe frontieră de tip Dirichlet $u(0,t) = u(1,t) = 1$. Această ecuaţie cu derivate parţiale poate fi rezolvată şi direct cu `pdepe`. Componenta $y_j(t)$ a sistemului de ecuaţii diferenţiale ordinare aproximează $u(j\Delta x, t)$. Presupunem că ecuaţia cu derivate parţiale verifică condiţiile $u(x,0) = (1 + \cos 2\pi x)/2$, pentru care se poate arăta că $u(x,t)$ tinde către starea de echilibru $u(x,t) \equiv 1$ când $t \to \infty$. Condiţia iniţială corespunzătoare pentru ecuaţia diferenţială ordinară este $(y_0)_j = (1 + \cos(2\pi j/(N+1)))/2$. Jacobianul pentru ecuaţia diferenţială ordinară are forma $A + I - 2\mathrm{diag}(y(t))$, unde I este matricea identică. Sursa MATLAB 6.2 conţine o funcţie care implementează şi rezolvă sistemul (6.4.2) utilizând `ode15s`. Utilizarea subfuncţiilor şi a tipurilor function handle face posibil ca întreg codul să fie conţinut într-un singur fişier, numit `rcd.m`. S-a luat $N = 40$ şi $t \in [0, 2]$. Subfuncţia `jacobian` evaluează jacobianul, `jpattern` dă şablonul de raritate al jacobianului sub forma unei matrice rare cu elemente 0 şi 1 şi `f` codifică membrul drept al ecuaţiei diferenţiale. A j-a coloană a matricei de ieşire y conţine aproximarea lui $y_j(t)$; matricea `U` a fost creată adăugând o coloană de 1 la fiecare capăt al lui y, conform condiţiilor pe frontieră pentru ecuaţia cu derivate parţiale. Graficul produs de `rcd` apare în figura 6.9.

6.5. Tratarea evenimentelor

În multe situaţii, determinarea ultimei valori t_{final} a lui `tspan` este un aspect important al problemei. Un exemplu este căderea unui corp asupra căruia acţionează forţa gravitaţională şi rezistenţa aerului. Când atinge el pământul? Un alt exemplu este problema celor două corpuri, adică determinarea orbitei unui corp supus atracţiei

Sursa MATLAB 6.2 Problemă stiff cu informaţii despre jacobian

```
function rcd
%RCD EDO Stiff pentru problema reactie-convectie-difuzie.
% obtinuta 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         ecuatia diferentiala.

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);
```

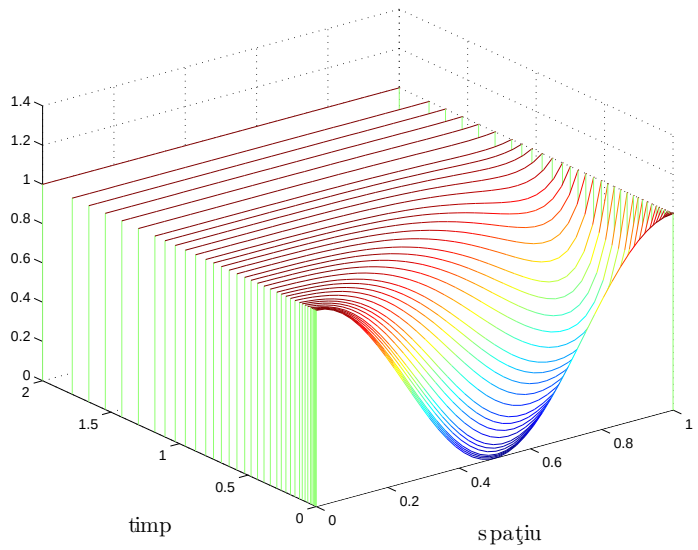


Figura 6.9: Exemplu stiff, cu informaţii despre jacobian

gravitaţionale a unui corp mult mai greu. Care este perioada orbitei? Facilitatea de prelucrare a evenimentelor a rezolvitorilor din MATLAB furnizează răspunsuri la astfel de întrebări.

Detecţia evenimentelor în MATLAB presupune două funcţii $f(t, y)$ şi $g(t, y)$ şi o condiţie iniţială (t_0, y_0). Problema este de a găsi o funcţie $y(t)$ şi o valoare finală t_* astfel încât

$$y' = f(t, y)$$
$$y(t_0) = y_0$$

şi

$$g(t_*, y(t_*)) = 0.$$

Un model simplu al unui corp în cădere este

$$y'' = -1 + y'^2,$$

cu o condiţie iniţială care dă valori pentru $y(0)$ şi $y'(0)$. Problema este pentru ce valori a lui t avem $y(t) = 0$? Codul pentru funcţia $f(t, y)$ este

```
function ydot=f(t,y)
ydot = [y(2); -1+y(2)^2];
```

Ecuaţia a fost scrisă ca un sistem de două ecuaţii 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 ieşire, `gstop`, este valoarea pe care dorim s-o anulăm. Dacă al doilea argument de ieşire, `isterminal`, are valoarea 1, rezolvitorul va termina execuţia dacă `gstop` este zero. Dacă `isterminal = 0`, evenimentul este înregistrat şi rezolvarea continuă. `direction` poate fi -1, 1 sau 0, după cum zeroul se atinge dacă funcţia este descrescătoare, crescătoare sau nemonotonă. Calculul şi reprezentarea traiectoriei 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 iniţială `y0=[1; 0]` se obţine

```
>> falling_body([1;0])
tfinal =
   1.65745691995813
```

şi graficul din figura 6.10.

Detecţia evenimentelor este utilă în probleme ce presupun fenomene periodice. Problema celor două corpuri este un exemplu bun. Ea descrie orbita unui corp asupra căruia acţionează forţa gravitaţională a unui corp mult mai greu. Utilizând coordonate carteziene, $u(t)$ şi $v(t)$ cu originea în corpul mai greu, ecuaţiile sunt:

$$\begin{aligned} u''(t) &= -\frac{u(t)}{r(t)^3} \\ v''(t) &= -\frac{v(t)}{r(t)^3}, \end{aligned}$$

unde $r(t) = \sqrt{u(t)^2 + v(t)^2}$. Întreaga rezolvare este conţinută într-un singur fişier de tip funcţie, `orbit.m` (sursa MATLAB 6.3). Parametrul de intrare, `reltol`, este

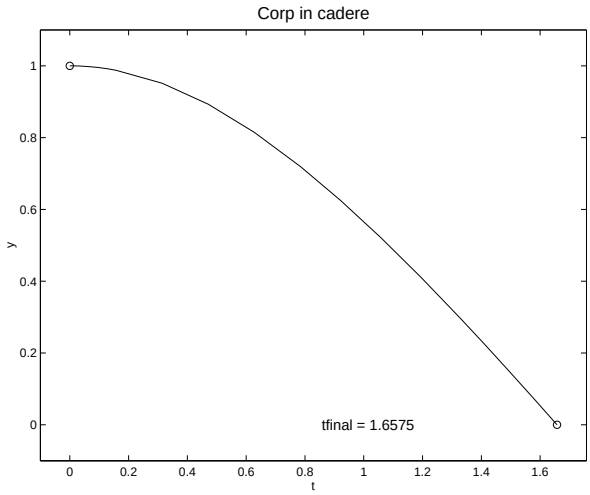


Figura 6.10: Traiectoria unui corp în cădere

Sursa MATLAB 6.3 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;
```

eroarea (precizia) relativă dorită. Codificarea problemei `twobody` şi funcţia de tratare a evenimentelor `gstop` sunt date prin subfuncţii (ele pot fi păstrate la fel de bine în fişiere separate). Calculul orbitei se realizează cu `ode45`. Argumentul de intrare `y0` este un vector cu 4 elemente, care dă poziţia iniţială şi viteza. Corpul uşor porneşte din poziţia $(1, 0)$ şi are viteza iniţială $(0, 0.3)$, care este perpendiculară pe vectorul poziţiei iniţiale. Argumentul `opts` este o structură creată cu `odeset`, care specifică eroarea relativă (egală cu `reltol`) şi funcţia de tratare a evenimentelor `gstop`. Ultimul argument al lui `ode45` este o copie `y0`, transmisă atât lui `twobody` cât şi lui `gstop`. Vectorul bidimensional `d` din `gstop` este diferenţa dintre poziţia curentă şi punctul de pornire. Viteza în poziţia curentă este dată de vectorul bidimensional `v`, iar cantitatea `val` este produsul scalar al lui `d` şi `v`. Expresia funcţiei 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 iniţial.

Apelând `orbit` cu o precizie mică

```
orbit(2e-3)
```

se obţine

```
tfinal =
   2.35087197761946
yfinal =
   0.98107659901112  -0.00012519138558
```

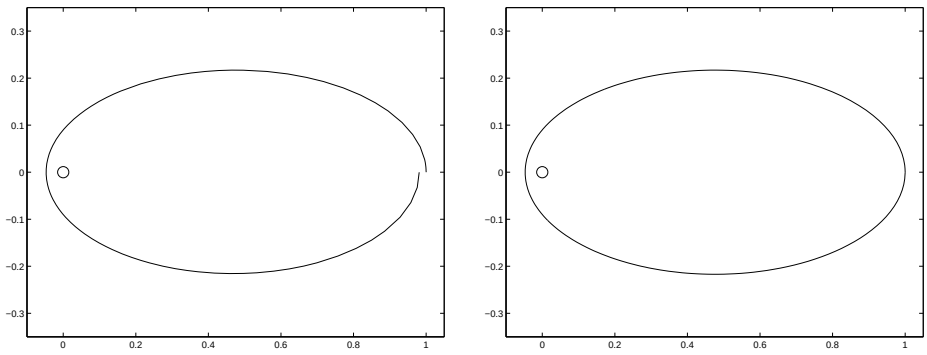
şi graficul din figura 6.11(a). Se poate observa din valoarea lui `yfinal` şi din grafic o abatere de la periodicitate. Avem nevoie de o precizie mai bună. Cu comanda

```
orbit(1e-6)
```

se obţine

```
tfinal =
   2.38025846171798
yfinal =
   0.99998593905520   0.00000000032239
```

Valoarea `yfinal` este acum suficient de apropiată de `y0`, iar graficul arată mult mai bine (figura 6.11(b)).



(a) `2e-3` (b) `1e-6`

Figura 6.11: Orbitele pentru pecizia `2e-3` (stânga) şi `1e-6`

Vom considera acum o problemă de urmărire [5, secţiunea 12.2]. Presupunem că un iepure urmează un drum predefinit $(r_1(t), r_2(t))$ din plan şi că o vulpe urmăreşte iepurele astfel ca (a) în fiecare moment tangenta la drumul vulpii indică întotdeauna spre iepure şi (b) viteza vulpii este de k ori viteza iepurelui. Atunci drumul $(y_1(t), y_2(t))$ al vulpii este determinat de sistemul de ecuaţii diferenţiale

$$\begin{aligned} \frac{d}{dt}y_1(t) &= s(t)(r_1(t) - y_1(t)), \\ \frac{d}{dt}y_2(t) &= s(t)(r_2(t) - y_2(t)), \end{aligned}$$

unde

$$s(t) = \frac{k\sqrt{\left(\frac{d}{dt}r_1(t)\right)^2 + \left(\frac{d}{dt}r_2(t)\right)^2}}{\sqrt{(r_1(t) - y_1(t))^2 + (r_2(t) - y_2(t))^2}}.$$

De notat că acest sistem pune probleme atunci când vulpea se apropie de iepure. Presupunem că iepurele urmează spirala de ecuaţie

$$\begin{bmatrix} r_1(t) \\ r_2(t) \end{bmatrix} = \sqrt{1+t} \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix},$$

şi că vulpea porneşte din poziţia $y_1(0) = 3$, $y_2(0) = 0$. Membrul drept este implementat prin funcţia `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 distanţa dintre vulpe şi iepure devine prea mică se apelează `error`. Script-ul de mai jos apelează `fox1` şi produce figura 6.12. Poziţiile iniţiale sunt marcate prin cercuri.

```
tspan = [0,10]; y0=[3,0]; k=0.75;
[tfox,yfox] = ode45(@fox1,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
```

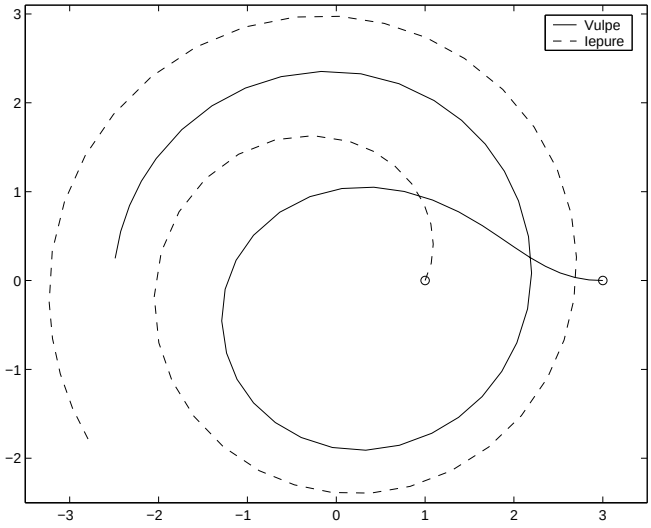


Figura 6.12: Exemplu de urmărire

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 inter-

valul de timp specificat, nici o soluţie nu este afişată. Situaţia se poate evita utilizând facilitatea de localizare a evenimentelor, aşa cum se poate vedea în script-ul următor. Poziţiile iniţiale, drumurile celor două animale şi poziţia finală apar în figura 6.13.

```
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 funcţia `odeset` pentru a seta erorile şi a specifica funcţia de localizare a evenimentelor. Membrul drept şi funcţia de localizare a evenimentelor se dau în sursa MATLAB 6.4. Ele sunt memorate în fişiere distincte. În funcţia de locali-

Sursa MATLAB 6.4 Funcţiile `fox2` şi `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.
```

zare `events` se calculează distanţa dintre animale minus pragul de eroare `1e-4`. S-a ales `direction=-1` pentru că distanţa trebuie să descrească. Parametrii de ieşire

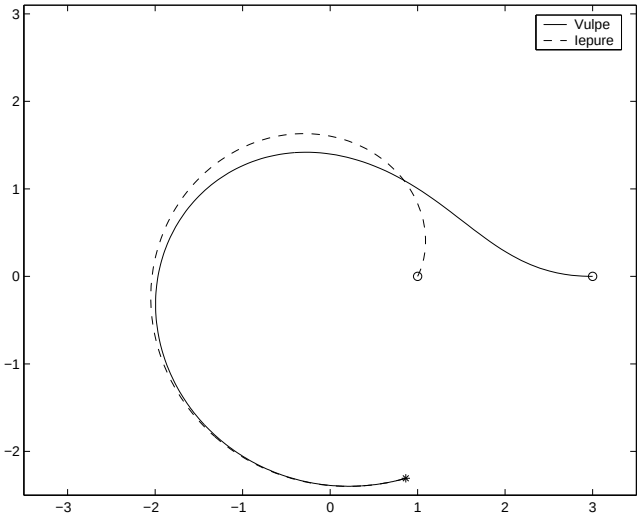


Figura 6.13: Exemplu de urmărire cu capturare

ne spun momentul (`te`), poziţia (`ye`) şi care componentă a evenimentului apare (`ie`). Pentru exemplul nostru se obţine

```
>> te,ye,ie
te =
    5.0710
ye =
    0.8646   -2.3073
ie =
     1
```

ceea ce ne spune că iepurele a fost capturat la momentul 5.071 în punctul de coordonate $(0.8646, -2.3073)$.

6.6. Ecuaţii implicite

Rezolvitorul `ode15i`, introdus începând cu versiunea 7, rezolvă ecuaţii implicite 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 facilităţile celorlalţi rezolvitori. Noutatea este apariţia parametrului `yp0` care este valoarea $y'(t_0)$. Trebuie îndeplinită condiţia de

consistenţă $f(t_0, y(t_0), y'(t_0)) = 0$. Se pot obţine valori consistente cu ajutorul funcţiei `decic`. Să rezolvăm ecuaţia

$$y'^2 + y^2 - 1 = 0, \qquad t \in [\pi/6, \pi/4],$$

cu condiţia iniţială $y(\pi/6) = 1/2$. Soluţia 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);
```

şi pentru ecuaţia diferenţială

```
function z=implic(t,y,yp)
z=yp^2+y^2-1;
```

Probleme

Problema 6.1. Rezolvaţi problema:

$$y' = 1 - y^2, \qquad y(0) = 0.$$

folosind diferite metode ale căror tabele Butcher au fost date în acest capitol, precum şi `ode23` şi `ode45`. Calculaţi eroarea globală, ştiind că soluţia exactă este

$$y(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

şi verificaţi că este $O(h^p)$.

Problema 6.2. Rezolvaţi ecuaţiile şi compraţi cu soluţiile exacte:

(a)
$$y' = \frac{1}{4} y(1 - \frac{1}{20} y), \quad x \in [0, 20], \quad y(0) = 1;$$
cu soluţia 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 soluţia 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 6.3. Ecuaţia atractorului Lorenz

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= -ax + ay, \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= bx - y - xz, \\ \frac{\mathrm{d}z}{\mathrm{d}t} &= -cz + xy \end{aligned}$$

are soluţii haotice care sunt sensibil dependente de condiţiile iniţiale. Rezolvaţi numeric pentru $a = 5$, $b = 15$, $c = 1$ cu condiţiile iniţiale

$$x(0) = 2, \quad y(0) = 6, \quad z(0) = 4, \quad t \in [0, 20],$$

cu toleranţa $T = 10^{-4}$. Repetaţi pentru

(a) $T = 10^{-5}$;

(b) $x(0) = 2.1$.

Comparaţi rezultatele cu cele obţinute anterior. În fiecare caz reprezentaţi grafic.

Problema 6.4. Evoluţia unei epidemii de gripă într-o populaţie de N indivizi este modelată de sistemul de ecuaţii diferenţiale

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= -\beta xy + \gamma, \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= \beta xy - \alpha y \\ \frac{\mathrm{d}z}{\mathrm{d}t} &= \alpha y - \gamma, \end{aligned}$$

unde x este numărul de indivizi susceptibili de a face infecţia, y este numărul de infectaţi, iar z este numărul de imuni, care include şi numărul de bolnavi refăcuţi după boală, la momentul t. Parametrii α, β, γ sunt ratele de recuperare, transmisie şi respectiv recontaminare (replenishment) (pe zi). Se presupune că populaţia este fixă, astfel că noile naşteri sunt compensate de morţi.

Utilizaţi funcţiile `oderk` şi `ode45` pentru a rezolva ecuaţiile cu condiţiile iniţiale $x(0) = 980$, $y(0) = 20$, z(0)=0, dându-se parametrii $\alpha = 0.05$, $\beta = 0.0002$, $\gamma = 0$. Simularea se va termina când $y(t) > 0.9N$. Determinaţi aproximativ numărul maxim de persoane infectate şi momentul când apare.

Investigaţi efectul (a) variaţiei numărului iniţial de indivizi infectaţi asupra evoluţiei epidemiei şi (b) introducerea unei rate de recontaminare nenule.

Problema 6.5. [2] Căpitanul Kirk[1] şi echipajul său de pe nava spaţială *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, şi dacă nava atinge straturile dense ale atmosferei, frânarea excesivă şi încălzirea datorată frecării va cauza daune ireparabile sistemului de menţinere a vieţii. Ofiţerul ştiinţific, Mr. Spock, estimează că reparaţiile temporare la motoarele de impuls vor necesita 29 de minute cu condiţia ca ele să fie terminate înainte ca frânarea să crească la $5g$ ($1g = 9.81\mathrm{ms}^{-1}$). Deoarece Mr. Spock este un geniu matematic, el a decis să simuleze degradarea orbitei prin rezolvarea numerică a ecuaţiilor de mişcare cu perechea de metode incluse DORPRI5. Ecuaţiile de mişcare ale navei, în condiţiile frânării atmosferice, sunt date de

$$\begin{aligned}
\frac{\mathrm{d}v}{\mathrm{d}t} &= \frac{GM\sin\gamma}{r^2} - c\rho v^2 \\
\frac{\mathrm{d}\gamma}{\mathrm{d}t} &= \left(\frac{GM}{rv} - v\right)\frac{\cos\gamma}{r} \\
\frac{\mathrm{d}z}{\mathrm{d}t} &= -v\sin\gamma \\
\frac{\mathrm{d}\theta}{\mathrm{d}t} &= \frac{v\cos\gamma}{r},
\end{aligned}$$

unde

- v este viteza tangenţială (m/s);
- γ este unghiul de reintrare (între viteză şi orizontală);
- z este altitudinea (m);
- M este masa planetară (6×10^{24} kg);
- G este constanta gravitaţiei ($6,67 \times 10^{-11}$ SI);
- c este constanta de frânare ($c = 0.004$);
- r este distanţa până la centrul planetei ($z + 6.37 \times 10^6$ m);
- ρ este densitatea atmosferei ($1.3\exp(-z/7600)$);
- θ este longitudinea;
- t este timpul (s).

[1] Personaje din Star Trek, seria I

La momentul $t = 0$, valorile iniţiale sunt $\gamma = 0$, $\theta = 0$ şi $v = \sqrt{GM/r}$. Mr. Spock a rezolvat ecuaţiile numeric pentru a găsi istoricul decelerării şi momentul şi locul impactului în care căderea orbitei nu mai poate fi prevenită. Repetaţi simularea utilizând o metodă Runge-Kutta cu pas variabil şi estimaţi decelerarea maximă încercată în timpul coborârii şi înălţimea la care apare. Va da căpitanul Kirk ordinul de abandonare a navei?

Problema 6.6. Cometa Halley şi-a atins ultima dată periheliul (apropierea maximă de soare) la 9 februarie 1986. Poziţia şi componentele vitezei în acel moment erau

$$\begin{aligned}(x, y, z) &= (0.325514, -0.459460, 0.166229)\\ \left(\frac{\mathrm{d}x}{\mathrm{d}t}, \frac{\mathrm{d}y}{\mathrm{d}t}, \frac{\mathrm{d}z}{\mathrm{d}t}\right) &= (-9.096111, -6.916686, -1.305721).\end{aligned}$$

Poziţia este măsurată în unităţi astronomice (distanţa medie de la pământ la soare), iar timpul în ani. Ecuaţiile mişcării sunt

$$\begin{aligned}\frac{\mathrm{d}^2x}{\mathrm{d}t^2} &= -\frac{\mu x}{r^3},\\ \frac{\mathrm{d}^2y}{\mathrm{d}t^2} &= -\frac{\mu y}{r^3},\\ \frac{\mathrm{d}^2z}{\mathrm{d}t^2} &= -\frac{\mu z}{r^3},\end{aligned}$$

unde $r = \sqrt{x^2 + y^2 + z^2}$, $\mu = 4\pi^2$, iar perturbaţiile planetare au fost neglijate. Rezolvaţi aceste ecuaţii numeric pentru a determina aproximativ momentul următorului periheliu.

Problema 6.7. 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 condiţiile iniţiale

$$y(0) = \left[1 - e, 0, 0, \sqrt{\frac{1+e}{1-e}}\right]^T,$$

unde $r = \sqrt{y_1^2 + y_2^2}$. Soluţia reprezintă mişcare pe orbită eliptică cu excentricitatea $e \in (0, 1)$, cu perioada 2π.

(a) Arătaţi că soluţiile 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 soluţia ecuaţiei lui Kepler

$$E - e\sin E = x.$$

(b) Rezolvaţi problema cu `oderk` şi `ode45` şi reprezentaţi grafic soluţia, variaţia lungimii pasului pentru $x \in [0, 20]$ şi o precizie de 10^{-5}.

(c) Calculaţi erorile globale, numărul de evaluări de funcţii şi numărul de respingeri pe pas pentru toleranţele 10^{-4}, 10^{-5}, ..., 10^{-11}.

Problema 6.8. [15]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 depăşit cu 0.80 metri recordul precedent şi a rezistat până în 1991. După săritura remarcabilă a lui Beamon, mulţi au sugerat că rezistenţa 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 şi originea pe pragul de bătaie. Singurele forţe care acţionează după bătaie sunt gravitaţia şi rezistenţa aerodinamică D, care este proporţională cu pătratul vitezei. Nu avem vânt, viteza iniţială este v_0 şi unghiul cu axa Ox este de θ_0 radiani. Ecuaţiile care descriu mişcarea săritorului sunt

$$x' = v\cos\theta, \qquad y' = v\sin\theta,$$
$$\theta' = -\frac{g}{v}\cos\theta, \qquad v' = -\frac{D}{m} - g\sin\theta.$$

Rezistenţa este

$$D = \frac{c\rho s}{2}(x'^2 + y'^2).$$

Constantele pentru această problemă sunt acceleraţia gravitaţională, $g = 9.81\ \mathrm{m/s^2}$, masa $m = 80$ kg, coeficientul de frânare $c = 0.72$, aria secţiunii săritorului, $s = 0.50\ \mathrm{m^2}$ şi unghiul iniţial $\theta_0 = 22.5^\circ = \pi/8$ radiani. Calculaţi pentru patru sărituri, cu diferite valori ale vitezei iniţiale v_0 şi ale densităţii aerului, ρ. Lungimea fiecărei sărituri este $x(t_f)$, unde timpul în aer, t_f, este determinat de condiţia $y(t_f) = 0$.

(a) Săritură „nominală” la altitudine mare, $v_0 = 10$ m/s şi $\rho = 0.94\ \mathrm{kg/m^3}$.

(b) Săritură „nominală” la nivelul mării, $v_0 = 10$ m/s şi $\rho = 1.29\ \mathrm{kg/m^3}$.

(c) Abordarea sprinterului la altitudine mare, $\rho = 0.94\ \mathrm{kg/m^3}$. Determinaţi v_0 astfel ca lungimea săriturii să fie egală cu recordul lui Beamon, 8.90 m.

(d) Abordarea sprinterului la nivelul mării, $\rho = 1.29\ \mathrm{kg/m^3}$ şi v_0 valoarea determinată la punctul (c).

Prezentaţi rezultatele completând tabela următoare:

v_0	θ_0	ρ	distanţa
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 iniţială a săritorului?

Problema 6.9. Rezolvaţi 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 condiţiile iniţiale $y_1(1) = 1$, $y_2 = 1$, utilizând un rezolvitor nonstiff şi apoi unul stiff. Sistemul are soluţiile exacte $y_1 = 1/x^2$, $y_2 = 1/\sqrt{x}$.

Problema 6.10. Ecuaţia lui van der Pol are forma

$$y_1'' - \mu\left(1 - y_1^2\right) y_1' + y_1 = 0, \tag{6.6.1}$$

unde $\mu > 0$ este un parametru scalar.

1. Să se rezolve ecuaţia în cazul când $\mu = 1$ pe intervalul [0,20] şi condiţiile iniţiale $y(0) = 2$ şi $y'(0) = 0$ (nonstiff). Să se reprezinte grafic y şi y'.

2. Să se rezolve ecuaţia pentru $\mu = 1000$ (stiff), intervalul de timp [0,3000] şi vectorul valorilor iniţiale [2;0]. Să se reprezinte grafic y.

Problema 6.11. Un dop de lungime L este pe punctul de a fi expulzat dintr-o sticlă ce conţine un lichid în fermentaţie. Ecuaţiile de mişcare a dopului sunt

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \begin{cases} g(1+q)\left[\left(1+\frac{x}{d}\right)^{-\gamma} + \frac{RT}{100} - 1 + \frac{qx}{L(1+q)}\right], & x < L; \\ 0, & x \geq L \end{cases}$$
$$\frac{\mathrm{d}x}{\mathrm{d}t} = v,$$

unde

g acceleraţia gravitaţională;

q raportul frecare-greutate al dopului;

x deplasarea dopului în gâtul sticlei;

t timpul;

d lungimea gâtului sticlei

R rata procentuală cu care presiunea creşte;

γ constanta adiabatică pentru gazul din sticlă ($\gamma = 1.4$).

Condiţiile iniţiale sunt $x(0) = x'(0) = 0$. Atât timp cât $x < L$ dopul este încă în sticlă, dar el părăseşte sticla când $x = L$. Integraţi ecuaţiile de mişcare cu DORPRI5 (tabela **??**) şi toleranţa 0.000001 pentru a găsi momentul la care dopul este aruncat. Determinaţi de asemenea viteza de expulzare când

$$q = 20, \qquad L = 3.75\mathrm{cm}, \qquad d = 5\mathrm{cm}, \qquad R = 4.$$

Problema 6.12. 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 concentraţia unui control chimic, iar ε şi A sunt constante pozitive. Se aşteaptă ca soluţiile să fie periodice. Aceasta se poate vedea reprezentând soluţia în planul fazelor (x pe abscisă, c pe ordonată), trebuind să se obţină o curbă închisă. Presupunem că $\varepsilon = 1$ şi $A = 3$.

(a) Calculaţi $x(t)$ şi $c(t)$, pentru $0 \leq t \leq 12$ şi valorile iniţiale $x(0) = 0.1$, $c(0) = 0.1$. Reprezentaţi ieşirea în planul fazelor. Cam cât este perioada?

(b) Repetaţi punctul (a) cu $x(0) = 0.87$, $c(0) = 2.1$.

Problema 6.13. Concepeţi şi implementaţi o strategie de control al pasului pentru metoda lui Euler cu un estimator al erorii bazat pe metoda lui Heun. Testaţi pe două probleme din acest capitol.

Problema 6.14. [15] Determinaţi traiectoria unei ghiulele de tun sferice într-un sistem staţionar de coordonate carteziene, care are o axă orizontală x, o axă verticală y şi originea în punctul de lansare. Viteza iniţială a proiectilului în acest sistem de coordonate are mărimea v_0 şi face un unghi de θ_0 radiani cu axa x. Singurele forţe care acţionează asupra proiectilului sunt gravitaţia şi rezistenţa aerodinamică, D, care depinde de viteza proiectilului relativă la orice vânt care ar putea fi prezent. Ecuaţiile care descriu mişcarea proiectilului sunt

$$x' = v\cos\theta, \qquad y' = v\sin\theta,$$
$$\theta' = -\frac{g}{v}\cos\theta, \qquad v' = -\frac{D}{m} - g\sin\theta.$$

Constantele pentru această problemă sunt: acceleraţia gravitaţională, $g = 9.81\mathrm{m/s}^2$, masa $m = 15\mathrm{kg}$ şi viteza iniţială $v_0 = 50\mathrm{m/s}$. Se presupune că vântul este orizontal şi viteza sa este o funcţie dată de timp, $w(t)$. Rezistenţa aerodinamică este proporţională cu pătratul vitezei relative la vânt a proiectilului

$$D(t) = \frac{c\rho s}{2}\left((x' - w(t))^2 + y'^2\right),$$

unde $c = 0.2$ este coeficientul de frânare, $\rho = 1.29\mathrm{kg/m}^3$ este densitatea aerului, iar $s = 0.25\mathrm{m}^2$ este aria secţiunii transversale a proiectilului.
Considerăm patru tipuri de vânt.

- Nici un vânt: $w(t) = 0$ pentru orice t.
- Vânt staţionar din faţă: $w(t) = -10\mathrm{m/s}$, pentru orice t.
- Vânt intermitent din spate: $w(t) = 10\mathrm{m/s}$, dacă partea întreagă a lui t este pară şi zero în caz contrar.
- Vijelie: $w(t)$ este o variabilă aleatoare normală cu media zero şi abaterea medie pătratică 10 m/s.

Partea întreagă a unui număr se poate calcula cu funcţia MATLAB `floor`, iar o variabilă aleatoare cu media 0 şi abaterea medie pătratică `sigma` se poate genera cu `sigma*randn`.

Pentru fiecare din aceste tipuri de vânt realizaţi următoarele calcule. Găsiţi cele 17 traiectorii ale căror unghiuri iniţiale sunt multiplii de 5 grade, adică $\theta_0 = k\pi/36$ radiani, $k = \overline{1,17}$. Desenaţi toate cele 17 traiectorii pe o figură. Determinaţi care dintre aceste traiectorii are cea mai mare bătaie orizontală. Pentru acea traiectorie, determinaţi unghiul iniţial în grade, timpul de zbor, bătaia orizontală, viteza la impact şi numărul de paşi executaţi de rezolvitor.
Care dintre cele patru tipuri de vânt necesită mai multe calcule. De ce?

Problema 6.15. [2] Un paraşutist sare dintr-un avion ce se deplasează cu viteza v m/s la altitudinea de y m. După o perioadă de cădere liberă, paraşuta se deschide la înălţimea y_p. Ecuaţiile de mişcare ale paraşutistului sunt

$$\begin{aligned} x' &= v\cos\theta, \\ y' &= v\sin\theta, \\ v' &= -D/M - g\sin\theta, \qquad D = \frac{1}{2}\rho C_D A v^2, \\ \theta' &= -g\cos\theta/v, \end{aligned}$$

unde x este coordonata orizontală, θ este unghiul de coborâre, D este rezistenţa aerului, iar A este aria de referinţă pentru forţa de tracţiune, dată de

$$A = \begin{cases} \sigma, & \text{dacă } y \geq y_p; \\ S, & \text{dacă } y < y_p. \end{cases}$$

Constantele sunt

$$\begin{array}{lll} g = 9.81 \text{ m/s}, & M = 80\text{kg}, & \rho = 1.2\text{kg/m}^3, \\ \sigma = 0.5\text{m}^2, & S = 30\text{m}^2, & C_D = 1. \end{array}$$

Utilizaţi un rezolvitor adecvat pentru a simula coborârea paraşutistului. Folosiţi facilităţi de interpolare pentru a determina înălţimea critică y_p. Determinaţi momentul impactului şi viteza minimă de coborâre înainte de deschiderea paraşutei.

Bibliografie

[1] P. J. Davis, P. Rabinowitz, *Numerical Integration*, Blaisdell, Waltham, Massachusetts, 1967.

[2] J. Dormand, *Numerical Methods for Differential Equations. A Computational Approach*, CRC Press, Boca Raton New York, 1996.

[3] W. Gander, W. Gautschi, *Adaptive quadrature - revisited*, BIT **40** (2000), 84–101.

[4] W. Gautschi, *Numerical Analysis, An Introduction*, Birkhäuser, Basel, 1997.

[5] D. J. Higham, N. J. Higham, *MATLAB Guide*, SIAM, Philadelphia, 2000.

[6] 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`.

[7] D. Kincaid, W. Cheney, *Numerical Analysis: Mathematics of Scientific Computing*, Brooks/Cole Publishing Company, Belmont, CA, 1991.

[8] Mirela Kohr, *Capitole speciale de mecanică*, Presa Univ. Clujeană, 2005.

[9] The Mathworks Inc., Natick, Ma, *Using MATLAB*, 2002.

[10] The Mathworks Inc., Natick, Ma, *MATLAB. Getting Started*, 2004, Version 7.

[11] The Mathworks Inc., Natick, Ma, *MATLAB. Symbolic Math Toolbox*, 2004, Version 7.

[12] The Mathworks Inc., Natick, Ma, *MATLAB. The Language of Technical Computing. Mathematics*, 2004, Version 7.

[13] The Mathworks Inc., Natick, Ma, *MATLAB. The Language of Technical Computing. Programming*, 2004, Version 7.

[14] The Mathworks Inc., Natick, Ma, *Using MATLAB Graphics*, 2004, Version 7.

[15] Cleve Moler, *Numerical Computing in MATLAB*, SIAM, 2004, disponibil via www la adresa `http://www.mathworks.com/moler`.

[16] Shoichiro Nakamura, *Numerical Computing and Graphic Vizualization in MATLAB*, Prentice Hall, Englewood Cliffs, NJ, 1996.

[17] Dana Petcu, *Matematică asistată de calculator*, Eubeea, Timişoara, 2000.

[18] L. F. Shampine, R. C. Allen, S. Pruess, *Fundamentals of Numerical Computing*, John Wiley & Sons, Inc, 1997.

[19] L. F. Shampine, Reichelt R. W., *The MATLAB ODE suite*, SIAM J. Sci. Comput. **18** (1997), no. 1, 1–22.

[20] D. D. Stancu, G. Coman, P. Blaga, *Analiză numerică şi teoria aproximării*, vol. II, Presa Universitară Clujeană, Cluj-Napoca, 2002, D. D. Stancu, Gh. Coman, (coord.).

[21] C. Ueberhuber, *Numerical Computation. Methods, Software and Analysis*, vol. I, II, Springer Verlag, Berlin, Heidelberg, New York, 1997.

Indice