Cuprins

Metode numerice în MATLAB

Rezolvarea sistemelor liniare în MATLAB

Fie *m* numărul de ecuaţii şi *n* numărul de necunoscute. Instrumentul fundamental de rezolvare a sistemelor de ecuaţii liniare este operatorul \.

El tratează trei tipuri de sisteme de ecuaţii liniare, pătratice ($m = n$), supradeterminate ($m > n$) şi subdeterminate ($m < n$). Mai general, operatorul \ 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 / 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.

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 = 1.024999e-018.
x =
  1.0e+008 *
    0.0000
   -0.0000
    0.0007
   -0.0101
    0.0759
   -0.3160
    0.6857
   -0.3825
   -1.7232
```

```
    4.5731
   -4.6502
    1.2423
    1.5968
   -1.4667
    0.3742
```

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.
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

$$AA^+A = A, A^+AA^+ = A^+, (A^+A)^+ = A^+A, (AA^+)^+ = AA^+$$

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]; pinv(A)

ans =
         0         0         0
         0    1.0000         0
         0         0    0.5000
         0         0         0
```

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\b* este o soluţie în sensul celor mai mici pătrate. Exemplul următor ilustrează diferenţa dintre soluţia obţinută cu \ şi `pinv`:

```
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)),  b=fix(10*rand(2,1))
R =
     8     1     6     2
     9     9     0     5
b =
     9
     9
```

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 =
      8/7
     -1/7
      0
```

```
     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 =
      -6/7          -13/63
       6/7          -22/63
       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`.

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 $L = P^T L$, 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
    0.5000   -0.5000    1.0000
    1.0000         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` şi 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 $A = R^T R$. 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).

Vectori şi valori proprii

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:

```
[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
```

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

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

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
```

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

$$A = U\Sigma V^*$$

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 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]
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.

Polinoame şi potrivirea datelor (data fitting)

MATLAB reprezintă un polinom

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

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

$$p(X) = p_1 X^n + p_2 X^{n-1} + \cdots + p_{n+1} I,$$

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 $\det(A - xI)$.

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 $x^3 - 2x^2 - x + 2$ 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):

$$y_i = y(t_i), i = 1, \dots, m.$$

Modelul nostru este o combinaţie de n funcţii de bază (π_i)

$$y(t) \approx c_1\pi_1(t,\alpha) + c_2\pi_2(t,\alpha) + \cdots + c_n\pi_n(t,\alpha).$$

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

$$\|r\|_2^2 = \sum_{i=1}^{m} r_i^2$$

şi

$$\|r\|_{2,w}^2 = \sum_{i=1}^{m} w_i r_i^2.$$

O explicaţie intuitivă, fizică, a celei de-a doua alegeri ar fi aceea că anumite observaţii sunt mai importante decˆat 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 amatricei 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

$$Ac \approx y$$

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 situaţii.
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 = 1.459426e-014.
x =
    0.5000
         0
    0.5000
```

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

```
x=pinv(A)*b
x =
    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 close to singular or badly scaled. Results may
be inaccurate. RCOND = 9.868649e-018.
z =
   -0.8594
    1.3438
   -0.6875
```

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]:

$$cond(A^T A) = cond(A)^2.$$

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

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

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

Dacă baza în care se face aproximarea este $1, t, \dots, t^n$, 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

$$y(t) = c_1 + c_2 e^{-t}.$$

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

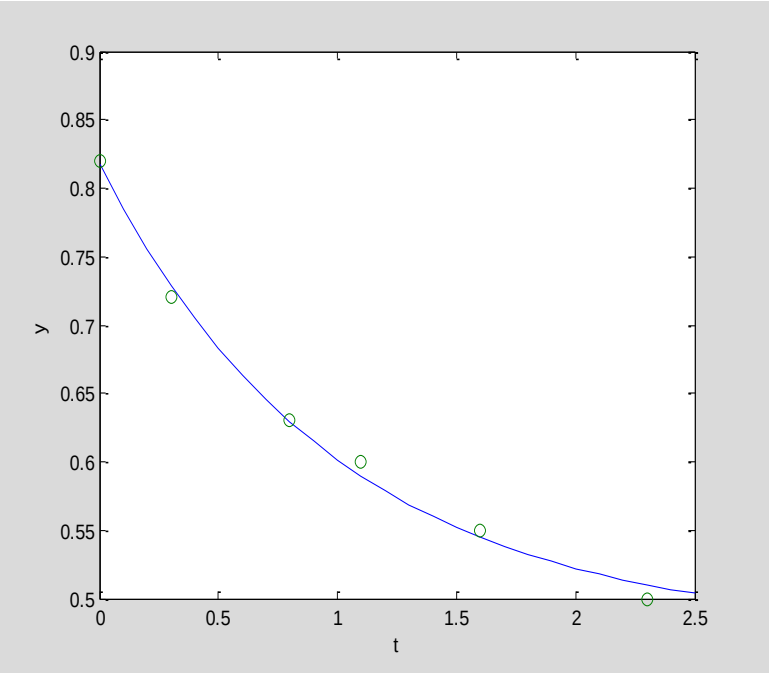
```
E=[ones(size(t)),exp(-t)]
E =
    1.0000    1.0000
    1.0000    0.7408
    1.0000    0.4493
    1.0000    0.3329
    1.0000    0.2019
    1.0000    0.1003
```

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. Dacă

matricea A este deficientă de rang (adică nu are coloane liniar 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

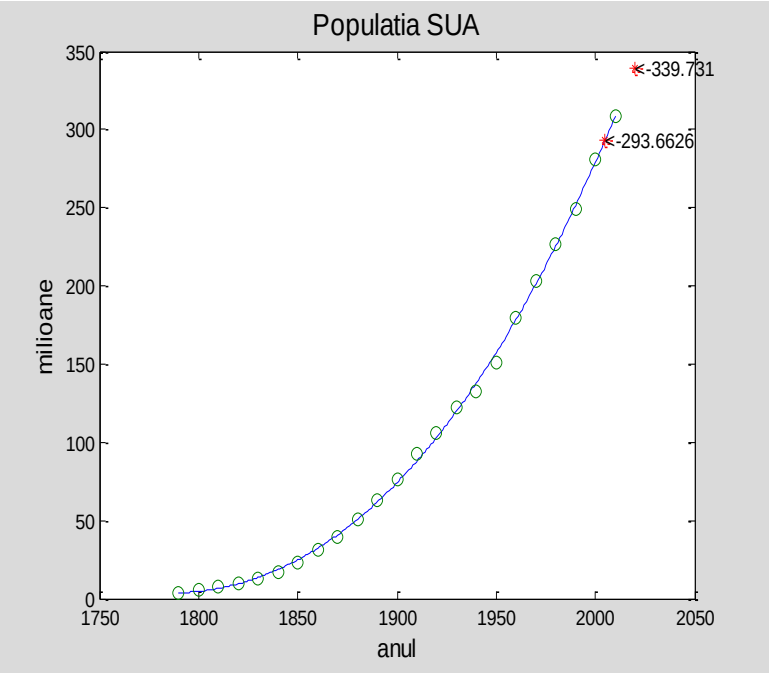
Se doreşte modelarea creşterii populaţiei printr-un polinom de gradul al treilea

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

```
load USCensus
c=polyfit(year,population,3);
x=(year-1900)/110;
c2=polyfit(x,population/1e6,3);
xg=min(year):max(year);
yg=polyval(c2,(xg-1900)/110);
```

```
%predictie
yy=[2005, 2020];
yp=polyval(c2,(yy-1900)/110);
plot(xg,yg,year,population/1e6,'o',yy,yp,'*')
text(yy(1),yp(1),strcat('<-',num2str(yp(1))))
text(yy(2),yp(2),strcat('<-',num2str(yp(2))))
title('Populatia SUA', 'FontSize', 14)
xlabel('anul', 'FontSize', 12)
ylabel('milioane', 'FontSize', 12)
```

```
Warning: Polynomial is badly conditioned. Add points with distinct
X values, reduce the degree of the polynomial, or try centering
and scaling as described in HELP POLYFIT.
{> In polyfit at 76}
```


Populatia SUA
339.731
293.6626
milioane
anul
0
50
100
150
200
250
300
350
1750
1800
1850
1900
1950
2000
2050