Exemple de programe MATLAB

Cuprins

Metoda eliminării a lui Gauss

Constă în două faze

1. Transformarea sistemului într-un sistem triunghiular echivalent
2. Rezolvarea sistemului triunghiular prin substituție inversă

Vom implementa o funcție declarată în următorul mod:

```
function x=Gepp(A,b)
%Gepp - Gaussian elimination with partial pivoting
%call x=Gepp(A,b)
%A - matrix, b- right hand side vector
%x - solution
```

Este mai practic să se lucreze cu matricea extinsă `A=[A,b]`

Etapa 1: triunghiularizare

La pasul i se elimină necunoscuta x_i din ecuația j, facând zerouri sub diagonala principală: se înmulțește ecuația (linia) i cu multiplicatorul $m = \frac{a_{ji}}{a_{ii}}$ și se scade din ecuația (linia i). Deoarece sub diagonala principală elementele vor fi nule, ele se lasă netransformate. Aceasta se poate scrie compact cu codul MATLAB:

```
m=A(j,i)/A(i,i);
A(j,i+1:n+1)=A(j,i+1:n+1)-m*A(i,i+1:n+1);
```

Pentru ca eliminarea să fie posibilă, trebuie ca $a_{ii} \neq 0$. Dacă a_{ii} este zero sau este mic linia i se permută cu linia p astfel ca

$$|a_{\mathrm{ip}}| = \max\{|a_{\mathrm{ji}}| : j = i, \ldots, n\}$$

Operația se numește *pivotare parțială* și a fost introdusă sistematic și studiată de von Neumann și Goldstine în 1947.

```
[u,p]=max(abs(A(i:n,i)));   %pivoting
p=p+i-1;
if u==0, error('no unique solution'), end
if p~=i %line interchange
    A([i,p],i:n+1)=A([p,i],i:n+1);
end
```

După $n-1$ pași, matricea a fost transformată într-o matrice triunghiulară.

Etapa 2: substituția inversă

Din ultima ecuație se obține

$$x_n = \frac{a_{n,n+1}}{a_{n,n}}$$

În general, x_i se obține din ecuația i, $i = n-1, n-2, \ldots, 1$, cu

$$x_i = \frac{1}{a_{\mathrm{ii}}}\left(a_{i,n+1} - \sum_{j=i+1}^{n} a_{\mathrm{ij}} x_j\right)$$

Suma se poate scrie compact cu ajutorul produsului scalar
`A(i,i+1:n)*x(i+1:n)`.

La final, am obținut următoarea sursă:

```
function x=Gepp(A,b)
%Gepp - Gaussian elimination with partial pivoting
%call x=Gepp(A,b)
%A - matrix, b- right hand side vector
%x - solution

%initialization
[l,n]=size(A);
x=zeros(size(b));
%s=max(abs(A),[],2);
```

```
A=[A,b]; %extended matrix
%Elimination
for i=1:n-1
    [u,p]=max(abs(A(i:n,i))); % ./s(i:n)); %pivoting
    p=p+i-1;
    if u==0, error('no unique solution'), end
    if p~=i %line interchange
        A([i,p],i:n+1)=A([p,i],i:n+1);
    end
    j=i+1:n;
    m=A(j,i)/A(i,i);
    A(j,i+1:n+1)=A(j,i+1:n+1)-m*A(i,i+1:n+1);
end
%back substitution
x(n)=A(n,n+1)/A(n,n);
for i=n-1:-1:1
    x(i)=(A(i,n+1)-A(i,i+1:n)*x(i+1:n))/A(i,i);
end
```

Să testăm pe următorul exemplu $Ax = b$, unde

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 3 \\ 1 & 3 & 1 \end{bmatrix}, b = \begin{bmatrix} 6 \\ 12 \\ 10 \end{bmatrix}.$$

```
format long
A=[1,1,1;1,1,3;1,3,1];
b=[6;12;10];
x=Gepp(A,b)
```

```
x = 3x1
     1
     2
     3
```

Metoda lui Newton

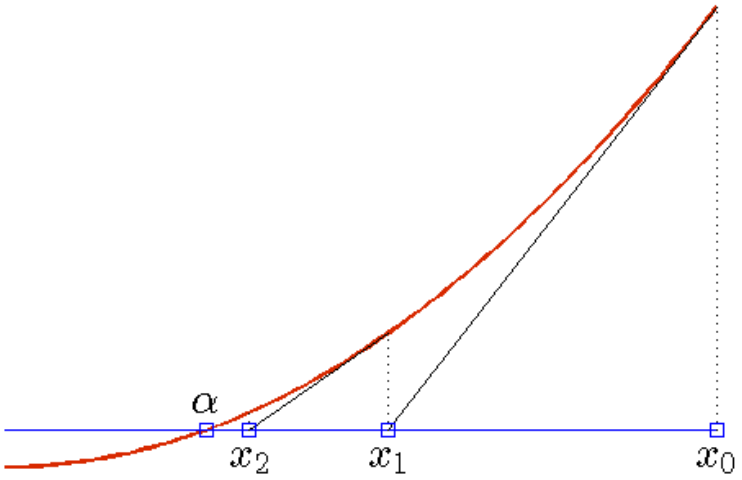
Prezentarea metodei

Dorim să rezolvăm ecuația $f(x) = 0$.

Metoda: se pornește cu un x_0 dat și se execută pasul iterativ

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

până când $|x_{k+1} - x_k| < \varepsilon$, ε precizie dată.



Vom implementa o variantă a metodei care să functioneze și pentru sisteme de ecuații neliniare. Pentru un sistem metoda lui Newton are forma

$$x^{(k+1)} = x^{(k)} - \left[f'\left(x^{(k)}\right)\right]^{-1} f\left(x^{(k)}\right).$$

În acest caz derivata $f'\left(x^{(k)}\right)$ este matricea jacobiană în $x^{(k)}$. Inversarea de matrice este o operație costisitoare și generatoare de erori și de aceea trebuie evitată. Iterația se poate scrie sub forma $x^{(k+1)} = x^{(k)} - w^{(k)}$. Corecția $w^{(k)}$ se poate obține ca soluție a sistemului

$$f'\left(x^{(k)}\right) w^{(k)} = f\left(x^{(k)}\right).$$

Rezolvarea se poate realiza în MATLAB cu operatorul \.

```
function [z,ni]=Newton(f,fd,x0,ea,er,nmax)
%NEWTON - metoda lui Newton pentru ecuatii neliniare in R si R^n
%apel [z,ni]=Newton(f,fd,x0,ea,er,nmax)
%Intrare
%f - functia
%fd - derivata
%x0 - aproximatia initiala
%ea - eroarea absoluta
%er - eroarea relativa
%nmax - numarul maxim de iteratii
%Iesire
%z - aproximatia radacinii
%ni - numarul de iteratii

if nargin < 6, nmax=50; end
if nargin < 5, er=0; end
if nargin < 4, ea=1e-3; end
xp=x0(:);   %x precedent
for k=1:nmax
    xc=xp-fd(xp)\f(xp); %disp(xc);
    if norm(xc-xp,inf)<ea+er*norm(xc,inf)
        z=xc; %succes
        ni=k;
        return
    end
    xp=xc;
end
error('S-a depasit numarul maxim de iteratii');
```

O condiție suficientă de convergență este $f(x_0) f'(x_0) > 0$.

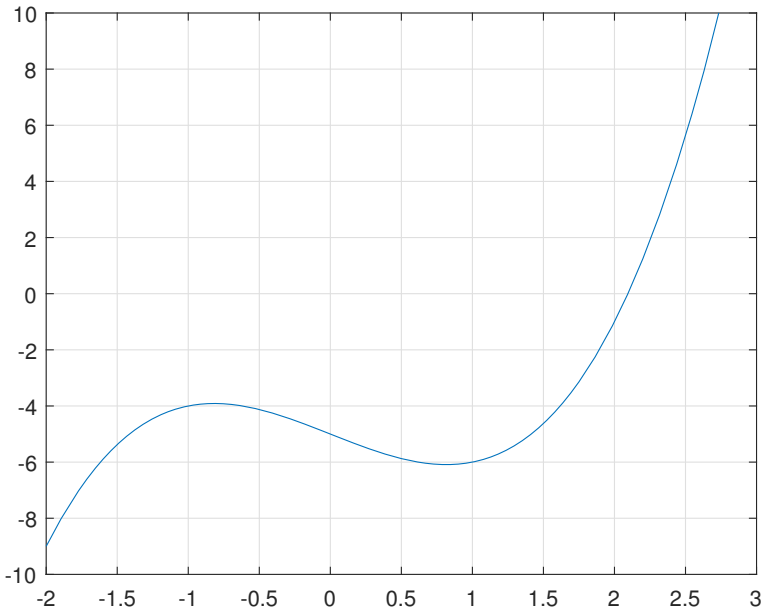
Exemplu: o ecuație scalară

Polinomul $x^3 - 2x - 5$ a fost utilizat de Wallis pentru a prezenta metoda lui Newton în fața Academiei framceze. Să definim polinomul și derivata sa.

```
close all
f=@(x) x.^3-2*x-5;
fd = @(x) 3*x.^2-2;
```

Reprezentăm grafic polinomul:

```
fplot(f, [-2,3])
ylim([-10,10])
grid on
```



El are o rădăcină reală între $x = 2$ și $x = 3$ și o pereche de rădăcini complexe conjugate, cum se poate verifica

```
f(2)
```

```
ans =
    -1
```

```
f(3)
```

```
ans =
    16
```

Determinam rădăcina reală cu metoda lui Newton. Alegem valoarea de pornire $x_0 = 3$ și precizia $\varepsilon = 10^{-8}$

```
x0=3;
[z1,ni1] = Newton(f,fd,x0, 1e-8)
```

```
z1 =
   2.094551481542327

ni1 =
     6
```

Precizia dorită a fost obținută după 6 iterații. Pentru determinarea rădăcinilor complexe vom utiliza valori de pornire complexe.

```
x0=1i;
[z2,ni2] = Newton(f,fd,x0, 1e-8)
```

```
z2 =
 -1.047275740771163 + 1.135939889088928i

ni2 =
     8
```

Verificăm soluțiile

```
f([z1,z2,conj(z2)])
```

```
ans = 1x3 complex
1.0e+-15 *

 -0.888178419700125 + 0.000000000000000i  0.000000000000000 - 0.888178419700125i  0.00000
```

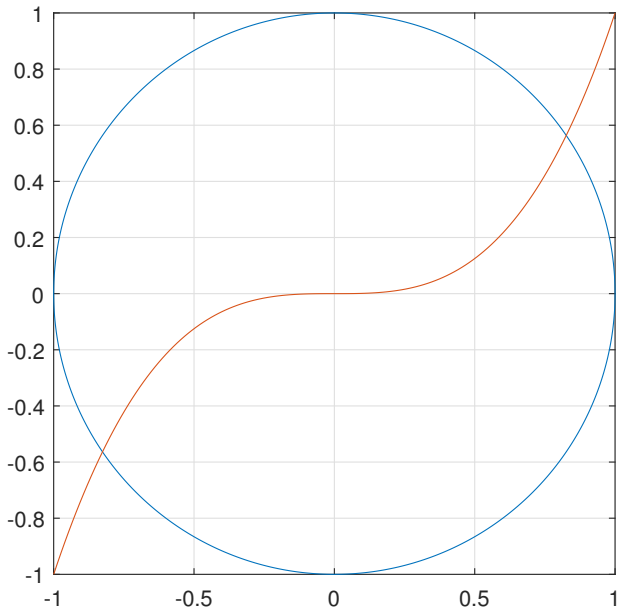
Exemplu: un sistem neliniar

Vom exemplifica acum metoda lui Newton pentru un sistem 2×2.

$$\begin{aligned} x_1^2 + x_2^2 - 1 &= 0 \\ x_1^3 - x_2 &= 0 \end{aligned}$$

Geometric, prima ecuație reprezintă cercul unitate, iar a doua o parabolă cubică.

```
fimplicit(@(x,y) x.^2+y.^2-1,[-1,1]); hold on
fimplicit(@(x,y) x.^3-y,[-1,1])
axis equal; grid on
hold off
```



Definim funcția și jacobianul

```
F = @(x) [x(1)^2+x(2)^2-1; x(1)^3-x(2)];
dF = @(x) [2*x(1), 2*x(2); 3*x(1)^2, -1];
```

Rădăcinile sunt simetrice față de origine. O valoarea de pornire va fi $x_0 = [1, 1]^T$. Cu metoda lui Newton, obținem soluțiile

```
x0=[1;1]
```

```
x0 = 2x1
     1
     1
```

```
[z1,ni1]=Newton(F,dF,x0,1e-8,1e-8,20)
```

```
z1 = 2x1
   0.826031357654187
   0.563624162161259

ni1 =
     5
```

```
[z2,ni2]=Newton(F,dF,-x0,1e-8,1e-8,20)
```

```
z2 = 2x1
  -0.826031357654187
  -0.563624162161259

ni2 =
     5
```

Aplicație la calculul radicalului

Vom folosi metoda lui Newton pentru a calcula $\sqrt{a}$, $a > 0$. Pornim de la ecuația $f(x) = x^2 - a = 0$. Iterația Newton devine

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{x_n^2 + a}{2x_n} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

Ținând cont că $f$este convexă, putem alege ca valoare de pornire orice x_0 pentru care $f(x_0) > 0$, adică $x_0 \geq \sqrt{a}$. O alegere potrivită este sugerată de inegalitatea mediilor

$\sqrt{a} = \sqrt{a \cdot 1} \leq \frac{a+1}{2}$,

deci, vom lua $x_0 = \frac{a+1}{2}$. Șirul astfel obținut este descrescător și mărginit inferior de $\sqrt{a}$, deci convergent.

Vom obține un *algoritm independent de mașină, care exploatează aritmetica finită.* Ținând cont că într-un interval mărginit avem un număr finit de numere în virgulă flotantă, vom termina iterațiile când $x_{n+1} \geq x_n$. Dăm codul pentru această funcție

```
function y=Sqrt(a)
xo=(1+a)/2; xn=(xo+a/xo)/2;
% stopping criterion breaking the monotony
while xn<xo
    xo=xn; xn=(xo+a/xo)/2;
end
y=(xo+xn)/2;
end
```

De menționat că în aritmetica exactă acest algoritm ciclează la infinit.

Să dăm două exemple de utilizare:

```
Sqrt(2)
```

```
ans =
   1.414213562373095
```

```
Sqrt(11)
```

```
ans =
   3.316624790355400
```

Recursivitate

Curba lui Koch

Curba lui Koch pornește de la un segment de dreaptă, pe care îl imparte în trei și înlocuiește segmentul din mijloc cu două segmente congruente care formează laturile unui triunghi echilateral. După aceea, transformarea se aplică recursiv fiecărui segment, până la atingerea unui nivel dorit 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 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.

```
function koch(pl,pr,level)
%KOCH   Curba Koch generata recursiv.
%       Apel KOCH(PL, PR, LEVEL) unde punctele PL si PR
%       sunt extremitatea stanga si dreapta
```

```
%       LEVEL este nivelul de recursivitate.

if level == 0
  plot([pl(1),pr(1)],[pl(2),pr(2)],'k-'); % 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 dreapta-mijloc

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

end
```

Vom testa funcția pentru un segment orizontal de lungime 1 și niveluri de recursivitate de la 1 la 4

```
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 (varfurile unui poligon regulat) se obţine o curbă numită fulgul de zăpadă al lui Koch (Koch's snowflake). Funcţia `snowflake` acceptă la intrare numărul de laturi `edges` şi nivelul de recursivitate `level`. Valorile implicite ale acestor parametrii sunt 7 şi respectiv 4.

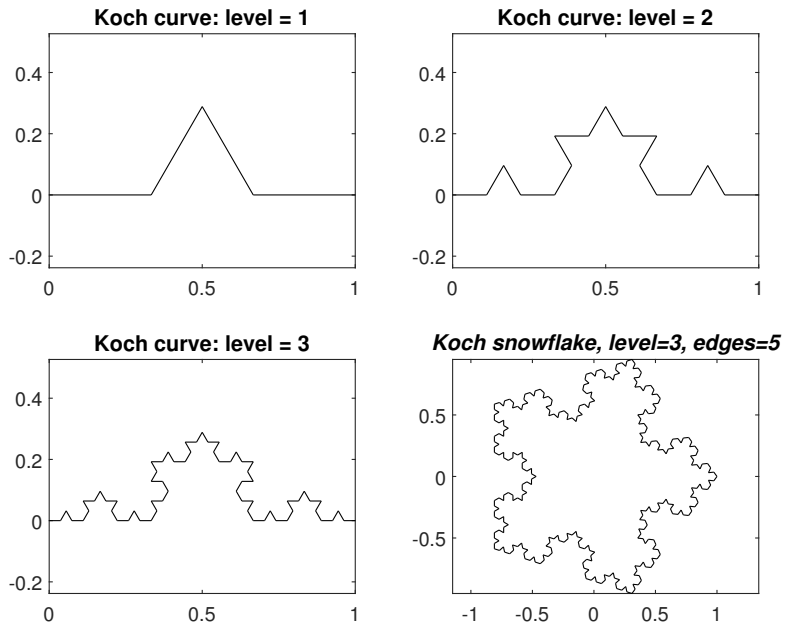
```
function snowflake(edges,level)
if nargin<2, level=4; end
if nargin<1, edges=7; end

%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
```

Dăm două exemple de utilizare

```
snowflake(7,4)
snowflake(5,3)
```



Cuadraturi adaptive

Fie $c = \frac{a+b}{2}$. Formula elementară a lui Simpson este

$$\int_a^b f(x)\,\mathrm{dx} \approx \frac{b-a}{6}\left[f(a) + 4f(c) + f(b)\right] = S_1$$

Pentru două subintervale se obține

$$\int_a^b f(x)\,\mathrm{dx} \approx \frac{b-a}{12}\left[f(a)+4f(d)+2f(c)+4f(e)+f(b)\right] = S_2$$

unde $d = \frac{a+c}{2}$ și $e = \frac{b+c}{2}$. Cantitatea $Q \approx \int_a^b f(x)dx$ se obține combinând cele două aproximante anterioare (extrapolare):

$$Q = S_2 + \frac{S_2 - S_1}{15}.$$

Putem să dăm acum un algoritm recursiv pentru aproximarea integralei. Funcția `adquad` evaluează integrandul aplicând regula lui Simpson. Ea apelează recursiv `quadstep` și aplică extrapolarea.

```
function [Q,fcount] = adquad(F,a,b,tol,varargin)
%ADQUAD  adaptive quadrature
%call [Q,fcount] = adquad(F,a,b,tol,varargin)
% F - integrand
% a,b - interval endpoints
% tol - tolerance, default 1.e-6.
% the aditional arguments are passed to the integrand, F(x,p1,p2,..).

% make F callable by feval.
if ischar(F) & exist(F)~=2
   F = inline(F);
elseif isa(F,'sym')
   F = inline(char(F));
end

if nargin < 4 | isempty(tol), tol = 1.e-6; end

% Initialization
c = (a + b)/2;
fa = F(a,varargin{:}); fc = F(c,varargin{:});
fb = F(b,varargin{:});

% Recursive call
[Q,k] = quadstep(F, a, b, tol, fa, fc, fb, varargin{:});
fcount = k + 3;

% ------------------------------------------------------------
```

```
function [Q,fcount] = quadstep(F,a,b,tol,fa,fc,fb,varargin)

% Recursive subfunction called by adquad

h = b - a;
c = (a + b)/2;
fd = F((a+c)/2,varargin{:});
fe = F((c+b)/2,varargin{:});
Q1 = h/6 * (fa + 4*fc + fb);
Q2 = h/12 * (fa + 4*fd + 2*fc + 4*fe + fb);
if abs(Q2 - Q1) <= tol
   Q  = Q2 + (Q2 - Q1)/15;
   fcount = 2;
else
   [Qa,ka] = quadstep(F, a, c, tol, fa, fd, fc, varargin{:});
   [Qb,kb] = quadstep(F, c, b, tol, fc, fe, fb, varargin{:});
   Q  = Qa + Qb;
   fcount = ka + kb + 2;
end
```

Vom calcula aproximativ integralele cu valoarea exactă cunoscută

$$\int_0^{\pi} \text{sinxdx} = 2, \int_{-1}^{1} \frac{\sin\ x}{1+x^2}\text{dx} = 0$$

```
[vI1,nfe1]=adquad(@sin, 0, pi, 1e-8)
```

```
vI1 =
   1.999999999997953

nfe1 =
   121
```

```
g=@(x)sin(x)./(1+x.^2);
[vi2,nfe2]=adquad(g,-1,1,1e-8)
```

```
vi2 =
    -1.973729821555834e-17

nfe2 =
     5
```