Biblioteca de functii Matlab pentru implementarea metodelor numerice in suportul de luare a deciziilor

B.1. Aproximarea numerică

B.1.1. Determinarea diferenţelor finite

Caseta B.1. Funcţia **difftab** pentru determinarea diferenţelor finite directe

```
function df = difftab(x,f)

n=length(x);
df=zeros(n-1,n-1);
 for i=1:n-1
    if i==1
       for j=1:n-1
          df(1,j)=f(j+1)-f(j);
       end
    else
       for j=1:n-i
          df(i,j)=df(i-1,j+1)-df(i-1,j);
       end
    end
 end
```

Parametrii de intrare:

f – valorile funcţiei;

x - valorile nodurilor în care funcţia *f* este definită.

Parametrii de ieşire:

df – tabelul orizontal cu diferenţe finite.

B.1.2. Interpolarea folosind polinoame Lagrange

Caseta B.2. Funcţia **aprox_lagrange** pentru aproximarea unei funcţii tabelare

```
function [L,yi] = aprox_lagrange(x,  f, xi)

dxi = xi - x;
n = length(x);
L = zeros(size(f));
L(1) = prod(dxi(2:n))/prod(x(1)-x(2:n));
L(n) = prod(dxi(1:n-1))/prod(x(n)-x(1:n-1));
for j=2:n-1
      a = prod(dxi(1:j-1))*prod(dxi(j+1:n));
      b = prod(x(j)-x(1:j-1))*prod(x(j)-x(j+1:n));
      L(j) = a/b;
end
yi = sum(f.*L);
```

Parametrii de intrare:

f – valorile funcţiei care se aproximează;

x - valorile nodurilor de interpolare.

Parametrii de ieşire:

L – coeficienţii polinomului Lagrange calculaţi cu relaţia (3.15);

yi – punctul în care se aproximează funcţia *f*.

B.1.3. Interpolarea folosind polinoame Newton

Caseta B.3. Funcţia **aprox_newton** pentru aproximarea unei funcţii tabelare

```
function [a, D] = aprox_newton(x, f)

n = length(x); D = zeros(n,n); D(:,1) = f';
for j=2:n,
    for k=j:n,
        D(k,j) = (D(k,j-1)-D(k-1,j-1))/(x(k)-x(k-j+1));
    end
```

```
end
a = D(n,n);
for k=(n-1):-1:1,
    a = conv(a,  poly(x(k)));
    m = length(a);
    a(m) = a(m) + D(k, k);
end
```

Parametrii de intrare:

f – valorile funcţiei care se aproximează;

x - valorile nodurilor de interpolare.

Parametrii de ieşire:

a – coeficienţii polinomului de interpolare Newton;

D – tabelul diferenţelor divizate.

B.1.4. Regresia polinomială

Caseta B.4. Funcţia **aprox_regres** pentru aproximarea unei funcţii tabelare

```
function a = aprox_regres (x, f, m)
n = length(x);
B = zeros(1:m+1);
F = zeros(n,m+1);
for k=1:m+1,
   F(:, k) = x'.^(k - 1);
end
A = F'*F;
B = F'*f';
a = A\B;
```

Parametrii de intrare:

f – valorile funcţiei care se aproximează, dată sub formă tabelară prin puncte;

x – valorile nodurilor de interpolare;

m – gradul polinomului de interpolare.

Parametrii de ieşire:

a – coeficienţii polinomului de interpolare.

B.2. Ecuaţii algebrice şi transcendente

B.2.1. Metoda înjumătăţirii intervalului sau metoda bisecţiei

Caseta B.5. Funcţia **met_bisectiei** pentru determinarea rădăcinii unei ecuaţii $f(x)=0$

```
function [d,fd,err,Minf] = met_bisectiei (f,a,b,eps)

Minf = [];
fa = feval(f, a);
fb = feval(f, b);
if fa*fb > 0, return; end
Nmax = round((log2((b-a)/eps)/log2(2)));
for k=1:Nmax,
        d  = (a+b)/2;
        fd = feval(f,d);
        if  fd == 0,
            a = d; b = d;
        elseif  fb*fd > 0,
            b = d;fb = fd;
        else
           a = d;fa = fd;
        end
        Minf  = [Minf;d,fd,a,b];
        if (b-a) < eps,  break,  end
end
d  = (a+b)/2;
fd = feval(f,d);err = abs(b-a)/2;
```

Parametrii de intrare:

f - numele funcţiei care se evaluează;

a, b - valorile limită inferioară respectiv, superioară a intervalului;

eps – precizia de calcul.

Parametrii de ieşire:

d - soluţia aproximativă;

fd - valoarea funcţiei în punctul *d*;

err - eroarea de estimare;

Minf - matricea informaţiilor din interiorul fiecărei iteraţii.

B.2.2. Metoda aproximaţiilor succesive

Caseta B.6. Funcţia **met_aprox_succ** pentru determinarea rădăcinii unei ecuaţii $f(x) = 0$

```
function [x,err,Minf]  = met_aprox_succ(f,x0,eps,Nmax)

Minf(1) = x0;
x = x0;
for k = 1:Nmax,
   x = feval(g, x0);
   err = abs(x-x0);
   if (err<eps), break; end
   x0 = x;
   Minf= [Minf x];
end
```

Parametrii de intrare:

f - numele funcţiei care se evaluează;

*x*0 – aproximaţia iniţială ;

eps – precizia de calcul;

Nmax – numărul maxim de iteraţii.

Parametrii de ieşire:

x - soluţia aproximativă;

err - eroarea de estimare;

Minf - matricea informaţiilor din interiorul fiecărei iteraţii.

B.2.3. Metoda Newton

Caseta B.7. Funcţia **met_newton** pentru determinarea rădăcinii unei ecuaţii $f(x) = 0$

```
function [x,fx,Minf]  = met_newton(f,df,x0,eps,Nmax)

Minf(1) = x0;
f0 = feval(f, x0);
for k=1:Nmax,
      df0 = feval(df,x0);
      if df0 == 0,
            dx = 0;
      else
            dx = f0/df0;
      end
      x = x0 - dx;
      fx = feval(f, x);
      err = abs(dx);
      x0 = x;
      f0 = fx;
      Minf = [Minf;x];
      if  err < eps, break, end
end
```

Parametrii de intrare:

f - numele fişierului care conţine expresia funcţiei;

df – numele fişierului care conţine expresia derivatei funcţiei;

x0 – aproximaţia iniţială;

eps – precizia de calcul;

Nmax – numărul maxim de iteraţii.

Parametrii de ieşire:

x - soluţia aproximativă;

fx – valoarea funcţiei în punctul *x*;

err - eroarea de estimare;

Minf - matricea informaţiilor din interiorul fiecărei iteraţii.

B.2.4. Metoda coardei

Caseta B.8. Funcţia **met_coardei** pentru determinarea rădăcinii unei ecuaţii $f(x) = 0$

```
function [x,Fx,err,Minf] = met_coardei(f, a, b, eps, Nmax)

Minf(1) = a; Minf(2) = b;
f0 = feval(f, a); f1 = feval(f, b);
x0=a;x1=b;
for k=1:Nmax
    df = (f1-f0) / (x1-x0);
    if df == 0,
        dx = 0;
    else
        dx = f1/df;
    end
    x = x1 - dx;
    Fx = feval(f, x);
    err = abs(dx);
    x0 = x1;
    f0 = f1;
    x1 = x;
    f1 = Fx;
    Minf = [Minf, x];
    if (err < eps), break, end
end
```

Parametrii de intrare:

f - numele fişierului care conţine expresia funcţiei;

a, *b* – extremităţile intervalului;

eps – precizia de calcul;

Nmax – numărul maxim de iteraţii.

Parametrii de ieşire:

x - soluţia aproximativă;

Fx – valoarea funcţiei în punctul *x*;

err - eroarea de estimare;

Minf - matricea informaţiilor din interiorul fiecărei iteraţii.

B.3. Rezolvarea numerică a sistemelor de ecuaţii liniare

B.3.1. Metoda eliminărilor Gauss

Caseta B.9. Funcţia **met_elimgauss** pentru soluționarea unui sistem de ecuaţii liniare

```
function  [U, X] = met_elimgauss (A, b)

[n, m] = size(A);
A = [A';b']';
X = zeros(n,1);
L = 1:n;
for i = 1:n-1,
   [Max, j] = max(abs(A(i:n,i)));
   T = L(i);
   L(i) = L(j+i-1);
   L(j+i-1) = T;
   if  A(L(i),i) == 0,
      break,
   end
   for k = i+1:n,
      M = A(L(k),i)/A(L(i),i);
      A(L(k),i+1:n+1) = A(L(k),i+1:n+1)- M*A(L(i),i+1:n+1);
   end
end
X(n) = A(L(n),n+1)/A(L(n),n);
for k = n-1:-1:1,
    X(k) = (A(L(k),n+1)-A(L(k),k+1:n)*X(k+1:n))/A(L(k),k);
end
U=triu(A(:,1:3));
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi.

Parametrii de ieşire:

X – vectorul soluţiei;

U – matricea superior triunghiulară.

B.3.2. Metoda eliminărilor Gauss-Jordan

Caseta B.10. Funcţia **met_gaussjordan** pentru soluționarea unui sistem de ecuaţii liniare

```
function  X = met_gaussjordan(A,b)

[n,n] = size(A);
A = [A b];
X = zeros(n,1);
for j = 1:n,
 for k = 1:n
   if k==j
     A(k,:) = A(k,:) / A(j,j);
   else
     p = A(k,j)/A(j,j);
     A(k,:) = A(k,:) - p*A(j,:);
   end
 end
end
X = A(:,n+1);
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi.

Parametrii de ieşire:

X – vectorul soluţiei.

B.3.3. Factorizarea Doolittle

Caseta B.11. Funcţia **factorizarea_doolittle** pentru soluționarea sistemelor de ecuaţii liniare

```
function [L,U,Y,X]  = factorizarea_doolittle(A,b)

[m, n] = size(A);
L=eye(n,n); U=zeros(n,n);
for j=1:n
  U(1,j)=A(1,j);
```

```
end
for i=2:n
   for j=1:i-1
      if U(j,j)==0   break; end
      t=A(i,j);
      if j~=1
         for k=1:j-1
            t=t-L(i,k)*U(k,j);
         end
      end
      L(i,j)=t/U(j,j);
   end
      for j=i:n
         t=A(i,j);
         for k=1:i-1
            t=t-L(i,k)*U(k,j);
         end
         U(i,j)=t;
      end
end
Y(1,:)=b(1,:)/L(1,1);
for i=2:n
   Y(i,:)=(b(i,:)-L(i,1:i-1)*Y(1:i-1,:))/L(i,i);
end
X(n,:)=Y(n,:)/U(n,n);
for i=(n-1):-1:1
   X(i,:)=(Y(i,:)-U(i,i+1:n)*X(i+1:n,:))/U(i,i);
end
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi.

Parametrii de ieşire:

U - matrice inferior triunghiulară;

L – matrice superior triunghiulară;

Y – vectorul soluţiei sistemului $[L]\cdot[Y] = [b]$;

X – vectorul soluţiei sistemului $[U]\cdot[X] = [Y]$, care reprezintă şi soluţia sistemului $[A]\cdot[X]=[b]$.

B.3.4. Factorizarea Crout

Caseta B.12. Funcţia **factorizarea_crout** pentru soluționarea unui sistem de ecuaţii liniare

```
function [L,U,Y,X] = factorizarea_crout(A,b)

[m, n] = size(A);
U=eye(n,n);L=zeros(n,n);
for i=1:n
   L(i,1)=A(i,1);
end
for j=2:n
   for i=1:j-1
      if L(i,i)==0
         break;
      end
      t=A(i,j);
      if i~=1
         for k=1:i-1
            t=t-L(i,k)*U(k,j);
         end
      end
      U(i,j)=t/L(i,i);
   end
   for i=j:n
      t=A(i,j);
      for k=1:j-1
            t=t-L(i,k)*U(k,j);
      end
      L(i,j)=t;
   end
end
Y(1,:)=b(1,:)/L(1,1);
for i=2:n
   Y(i,:)=(b(i,:)-L(i,1:i-1)*Y(1:i-1,:))/L(i,i);
end
X(n,:)=Y(n,:)/U(n,n);
for i=(n-1):-1:1
   X(i,:)=(Y(i,:)-U(i,i+1:n)*X(i+1:n,:))/U(i,i);
end
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi.

Parametrii de ieşire:

U - matrice inferior triunghiulară;

L – matrice superior triunghiulară;

Y – vectorul soluţiei sistemului $[L]\cdot[Y] = [b]$;

X – vectorul soluţiei sistemului $[U]\cdot[X] = [Y]$, care reprezintă şi soluţia sistemului $[A]\cdot[X]=[b]$.

B.3.5. Factorizarea Cholesky

Caseta B.13. Funcţia **factorizarea_cholesky** pentru soluționarea unui sistem de ecuaţii liniare

```
function [L,Y,X] = factorizarea_cholesky(A,b)

[n,m] = size(A);
for j = 1:n
    s = A(j,j) - A(j,1:j-1)*A(j,1:j-1)';
    if s < 0.0
       error('Matricea nu este pozitiv definita');
    end
    A(j,j) = sqrt(s);
    for i = j+1:n
      A(i,j)=(A(i,j) - (A(i,1:j-1)*A(j,1:j-1)'))/A(j,j);
    end
end
L = tril(A);
Y(1,:)=b(1,:)/L(1,1);
for i=2:n
   Y(i,:)=(b(i,:)-L(i,1:i-1)*Y(1:i-1,:))/L(i,i);
end
Lt=L';
X(n,:)=Y(n,:)/Lt(n,n);
```

```
for i=(n-1):-1:1
   X(i,:)=(Y(i,:)-Lt(i,i+1:n)*X(i+1:n,:))/Lt(i,i);
end
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi.

Parametrii de ieşire:

L - matrice inferior triunghiulară;

Y – vectorul soluţiei sistemului $[L]\cdot[Y] = [b]$;

X – vectorul soluţiei sistemului $[L]^t \cdot [X] = [Y]$, care reprezintă şi soluţia sistemului $[A]\cdot[X]=[b]$.

B.3.6. Metodă bazată pe inversarea matricei coeficienţilor

Caseta B.14. Funcţia **met_inversei** pentru soluționarea unui sistem de ecuaţii liniare

```
function [Ai,X] = met_inversei(A,b)

[m,n] = size(A);
if m ~= n
   disp('Matricea  nu este patratica'); return
end
Ai = eye(m);
for j = 1 : m
   for i = j : m
      if A(i,j) ~= 0
         for k = 1 : m
            s = A(j,k);  A(j,k) = A(i,k);  A(i,k) = s;
            s = Ai(j,k);  Ai(j,k) = Ai(i,k);  Ai(i,k) = s;
         end
         t = 1/A(j,j);
         for k = 1 : m
            A(j,k) = t * A(j,k);
            Ai(j,k) = t * Ai(j,k);
         end
         for L = 1 : m
            if L ~= j
               t = -A(L,j);
               for k = 1 : m
```

```
            A(L,k) = A(L,k) + t * A(j,k);
            Ai(L,k) = Ai(L,k) + t * Ai(j,k);
          end
        end
      end
    end
    break;
  end
  if A(i,j) == 0
    disp('Atentie: Matrice singulara'); return; end
end
X=Ai*b;
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi.

Parametrii de ieşire:

Ai – matricea inversă;

X – vectorul soluţiei sistemului.

B.3.7. Metoda Jacobi

Caseta B.15. Funcţia **met_jacobi** pentru soluționarea unui sistem de ecuaţii liniare

```
function [X,Niter,Minf] = met_jacobi(A,b,X0,Emax,Nmax)

Minf = X0';
[n,m] = size(A);
X = X0;
for k=1:Nmax,
 for i = 1:n,
  s = b(i) - A(i,[1:i-1,i+1:n])*X0([1:i-1,i+1:n]);
  X(i) = s/A(i,i);
 end
 dX = abs(X-X0);
 err = max(dX);
 X0 = X;
 Minf = [Minf;X0'];
 if (err < Emax), break, end
```

```
end
Niter=k;
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi;

X0 – aproximaţia iniţială;

Emax – abaterea maximă între două iteraţii succesive;

Nmax – numărul maxim de iteraţii.

Parametrii de ieşire:

X – vectorul soluţiei sistemului.

Niter – numărul de iteraţii în care s-a determinat soluţia sistemului;

Minf - matricea informaţiilor din interiorul fiecărei iteraţii.

B.3.8. Metoda Seidel - Gauss

Caseta B.16. Funcţia **met_seidelgauss** pentru soluționarea unui sistem de ecuaţii liniare

```
function [X,Niter,Minf] = met_seidelgauss(A,b,X0,Emax,Nmax)

Minf = X0';
[n,m] = size(A);
X = X0;
for k=1:Nmax
 for i = 1:n
  s = b(i) - A(i,[1:i-1,i+1:n])*X0([1:i-1,i+1:n]);
  X0(i) = s/A(i,i);
 end
 dX = abs(X-X0);
 err = max(dX);
 X = X0;
 Minf = [Minf;X0'];
 if (err<Emax), break, end
end
Niter=k;
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi;

X0 – aproximaţia iniţială;

Emax – abaterea maximă între două iteraţii succesive;

Nmax – numărul maxim de iteraţii.

Parametrii de ieşire:

X – vectorul soluţiei sistemului;

Niter – numărul de iteraţii în care s-a determinat soluţia sistemului;

Minf - matricea informaţiilor din interiorul fiecărei iteraţii.

B.3.9. Metoda Seidel – Gauss modificată

Caseta B.17. Funcţia **met_seidelgauss_modif** pentru soluționarea unui sistem de ecuaţii liniare

```
function[X,Niter,Minf]=met_seidelgauss_modif(A,b,X0,Emax,Nmax,acc)


 Minf = X0';
 [n,m] = size(A);
 X = X0;
 for k=1:Nmax,
  Z=X0;
  for i = 1:n,
    s = b(i) - A(i,[1:i-1,i+1:n])*X0([1:i-1,i+1:n]);
    X0(i) = s/A(i,i);
  end
  if k~=1
     X0=Z+acc*(X0-Z);
  end
  dX = abs(X-X0);
  err = max(dX);
  X = X0;
  Minf = [Minf;X0'];
```

```
  if (err<Emax), break, end
end
Niter=k;
```

Parametrii de intrare:

A – matricea coeficienţilor;

b – vectorul termenilor liberi;

X0 – aproximaţia iniţială;

Emax – abaterea maximă între două iteraţii succesive;

Nmax – numărul maxim de iteraţii;

acc – coeficientul de accelerare.

Parametrii de ieşire:

X – vectorul soluţiei sistemului;

Niter – numărul de iteraţii în care s-a determinat soluţia sistemului;

Minf - matricea informaţiilor din interiorul fiecărei iteraţii.

B.4. Rezolvarea numerică a sistemelor de ecuaţii neliniare

B.4.1. Metoda Newton-Raphson

Caseta B.18. Funcţia **met_newraph** pentru determinarea soluţiei unui sistem de ecuaţii neliniare

```
function [X,F,Minf]=met_newraph(Fun,Jac,X0,eps,Nmax)

n=length(X0);
Minf = X0;
F0 = feval(Fun,X0);
for k=1:Nmax,
      dF = feval(Jac,X0);
      if det(dF) == 0,
         dX = zeros(1,n)
      else
```

```
        dX = (dF\F0)';
    end
    X1 = X0 - dX;
    F1 = feval(Fun,X1);
    err = abs(dX);
    X0 = X1;
    F0 = F1;
    Minf = [Minf;X1];
    if (err<eps), break, end
end
X = X0';
F = F0';
```

Parametrii de intrare:

Fun – fişier **funcţie** ce conţine expresiile ecuaţiilor neliniare. Acest fişier are ca variabilă de intrare *X*, iar ca variabilă de ieşire un vector *F*, ale cărui valori reprezintă expresiile ecuaţiilor neliniare evaluate în punctul *X*. Fişierul **Fun** poate fi apelat astfel:

function *F* = **Fun**(*X*)

Jac - fişier funcţie ce conţine expresiile derivatelor parţiale de ordinul 1 ale funcţiilor care definesc sistemul neliniar. Acest fişier are ca variabilă de intrare *X*, iar ca variabilă de ieşire un vector *J*, a cărui valoare reprezintă expresiile derivatelor evaluate în punctul *X*. Fişierul funcţie **Jac** poate fi apelat astfel:

function *J* = **Jac**(*X*)

*X*0 – aproximaţia iniţială;

Nmax – numărul maxim de iteraţii;

eps – precizia de calcul.

Parametrii de ieşire:

X – soluţia aproximativă;

F – valoarea funcţiei în punctul *X*;

Minf – matricea informaţiilor din interiorul fiecărei iteraţii.

B.4.2. Metoda gradientului

Caseta B.19. Funcţia **met_gradientului** pentru soluţionarea unui sistem de ecuaţii neliniare

```
function [X,F,Minf,Niter]=met_gradientului(Fun,Jac,X0,eps,Nmax)


F0 = feval(Fun,X0);
Minf = X0';
for k=1:Nmax
      dF = feval(Jac,X0);
      g0=dF'*F0;
      d0=-g0;
      alfa=(g0'*g0)/((d0'*dF')*(dF*d0));
      X1=X0+alfa*d0;
      dX=X1-X0;
      err = abs(dX);
      F1 = feval(Fun,X1);
      X0 = X1;
      F0 = F1;
      Minf = [Minf;X1'];
      if (err<eps), break, end
end
X = X0;
F = F0;
Niter=k;
```

Parametrii de intrare:

Fun – fişier **funcţie** ce conţine expresiile ecuaţiilor neliniare. Acest fişier are ca variabilă de intrare *X*, iar ca variabilă de ieşire un vector *F*, ale cărui valori reprezintă expresiile ecuaţiilor neliniare evaluate în punctul *X*. Fişierul **Fun** poate fi apelat astfel:

function *F* = **Fun**(*X*)

Jac - fişier funcţie ce conţine expresiile derivatelor parţiale de ordinul 1 ale funcţiilor care definesc sistemul neliniar. Acest fişier are ca variabilă de intrare *X*,

iar ca variabilă de ieşire un vector *J*, a cărui valoare reprezintă expresiile derivatelor evaluate în punctul *X*. Fişierul funcţie **Jac** poate fi apelat astfel:

function *J* = **Jac**(*X*)

*X*0 – aproximaţia iniţială;

Nmax – numărul maxim de iteraţii;

eps – precizia de calcul.

Parametrii de ieşire:

X – soluţia aproximativă;

F – valoarea funcţiei în punctul *X*;

Minf – matricea informaţiilor din interiorul fiecărei iteraţii.

Niter ce simbolizează numărul de iteraţii în care s-a determinat soluţia sistemului.

B.5. Derivarea numerică

B.5.1. Derivarea numerică folosind polinoame de tip Newton

Caseta B.20. Funcţia **deriv_pol_newton** pentru determinarea numerică a valorii derivatei funcţiei *f*(*x*)

```
function [d1f,d2f]=deriv_pol_newton(fun,x1)


x=x1:(x1+10);
h=x(2)-x(1);
y = feval(fun,x);
d1y=diff(y);
d2y=diff(d1y);
d3y=diff(d2y);
d4y=diff(d3y);
d1f=(1/h)*(d1y(1)-d2y(1)/2+d3y(1)/3-d4y(1)/4);
d2f=(1/h^2)*(d2y(1)-d3y(1)+(11/12)*d4y(1));
```

Parametrii de intrare:

fun – fişier funcţie ce conţine expresia funcției *f(x)*. Acest fişier are ca variabilă de intrare *X*, iar ca variabilă de ieşire un vector *F*, ale cărui valori reprezintă expresia ecuaţiei evaluate în punctul *X*. Fişierul Fun poate fi apelat astfel:

function *F* = **Fun**(*X*)

x1 –punctul in care se calculează derivata.

Parametrii de ieşire:

d1f– valoarea derivatei de ordinul 1;

d2f– valoarea derivatei de ordinul 2.

B.6. Integrarea numerică

B.6.1. Metoda trapezelor

Caseta B.21. Funcţia **met_trapezelor** pentru determinarea numerică a valorii integralei unei funcţii *f*(*x*)

```
function I = met_trapezelor(f,a,b,n)


h  = (b - a)/n;
I = 0;
for k=1:(n-1),
  x = a + h*k;
  I = I + feval(f,x);
end
I = h*(feval(f,a)+feval(f,b))/2  + h*I;
```

Parametrii de intrare:

f – numele funcţiei care se integrează;

a – valoarea limită inferioară a intervalului de integrare;

b – valoarea limită superioară a intervalului de integrare;

n – numărul de intervale elementare.

Parametrii de ieşire:

I – valoarea integralei calculată cu formula trapezelor.

B.6.2. Metoda Simpson

Caseta B.22. Funcţia **met_simpson** pentru determinarea numerică a valorii integralei unei funcţii f(x)

```
function I = met_Simpson(f,a,b,n)

h  = (b - a)/(2*n);
I1 = 0; I2 = 0;
for k=1:n
  x = a + h*(2*k-1);
  I1 = I1 + feval(f,x);
end
for k=1:(n-1)
  x = a + h*2*k;
  I2 = I2 + feval(f,x);
  end
I = h*(feval(f,a)+feval(f,b)+4*I1+2*I2)/3;
```

Parametrii de intrare şi de ieşire corespunzători fişierului funcţie **met_simpson** sunt:

Parametrii de intrare:

f - numele funcţiei care se evaluează;

a - valoarea limită inferioară a intervalului ;

b - valoarea limită superioară a intervalului;

n – numărul de intervale elementare.

Parametrii de ieşire:

I – valoarea integralei calculată cu formula Simpson.

B.7. Vectori și valori proprii

B.7.1. Metoda puterii inverse

Caseta B.23. Funcţia **met_putere_inv** pentru determinarea valorii proprii minime in valoare absolută şi a vectorului asociat

```
function [ L1,X1,Niter ] = met_putere_inv(A,s,Nmax,eps)
n=size(A,1)
A=A-eye(n)*s
X=rand(n,1)
Xn=sqrt(dot(X,X))
X=X/Xn
for i=1:Nmax
   Xv=X
   X=inv(A)*X
   Xn=sqrt(dot(X,X))
   X=X/Xn
   Xsign=sign(dot(Xv,X))
   X=X*Xsign
   if sqrt(dot(Xv-X,Xv-X))<eps
      L1=s+Xsign/Xn
      X1=X
      Niter=i
      return
   end
end
error('depasire numar maxim de iteratii')
```

Parametrii de intrare:

A - matricea coeficienţilor;

s – deplasarea fată de origine;

eps – abaterea maximă între două iteraţii succesive;

Nmax – numărul maxim de iteraţii.

Parametrii de ieşire:

L1– valoarea proprie minimă;

X1– vectorul propriu asociat valorii proprii *L1*;

Niter – numărul de iteraţii în care este obţinută soluţia.

B.7.2. Metoda puterii directe

Caseta B.24. Funcţia **met_putere_directe** pentru determinarea valorii proprii dominate in valoare absolută şi a vectorului asociat

```
function [Ln,Xn,Niter] = met_putere_directe(A,Nmax,eps)
n=size(A,1)
X=[1;0;0]%rand(n,1)
for i=1:Nmax
   Xv=X
   X=A*X
   Xn=sqrt(dot(X,X))
   X=X/Xn
   Xsign=sign(dot(Xv,X))
   X=X*Xsign
   if sqrt(dot(Xv-X,Xv-X))<eps
      Ln=Xsign*Xn
      Xn=X
      Niter=i
      return
   end
end
error('depasire numar maxim de iteratii')
```

Parametrii de intrare:

A - matricea coeficienţilor;

eps – abaterea maximă între două iteraţii succesive;

Nmax – numărul maxim de iteraţii.

Parametrii de ieşire:

Ln– valoarea proprie dominantă;

Xn– vectorul propriu asociat valorii proprii *Ln*;

Niter – numărul de iteraţii în care este obţinută soluţia problemei.

B.8. Soluționarea numerică a ecuațiilor diferențiale

B.8.1. Metoda Taylor

Caseta B.25. Funcţia **met_Taylor** pentru soluţionarea numerică a ecuaţiilor diferenţiale

```
function [Y,X] = met_Taylor(fun,a,b,y0,n)
h=(b-a)/n
X=zeros(1,n+1)
Y=zeros(1,n+1)
X(1)=a
Y(1)=y0
for j=1:n
   xj=X(j)
   yj=Y(j)
   D=feval('fun',xj,yj)
   Y(j+1)=yj+h*(D(1)+h*(D(2)/2+h*D(3)/6+h*D(4)/24)))
   X(j+1)=a+h*j
end
```

Parametrii de intrare:

fun – fişier funcţie ce conţine expresia ecuaţiei diferenţiale. Acest fişier are ca variabile de intrare *x* şi *y*, iar ca variabilă de ieşire un vector *z*, ale cărui valoare reprezintă valoarea ecuaţiei diferenţiale evaluată în punctele *x*. Fişierul **fun** poate fi apelat astfel:

function *z* = **fun**(*x,y*)

y0 – condiţia iniţială;

n – numărul de paşi;

a, *b* – extremităţile intervalului în care se calculează soluţia.

Parametrii de ieşire:

Y– soluţia aproximativă;

X– punctele din intervalul [a, b] în care se calculează soluţia *Y*.

B.8.2. Metoda Euler

Caseta B.26. Funcţia **met_euler** pentru soluţionarea numerică a ecuaţiilor diferenţiale

```
function [Y,X] = met_euler(fun,a,b,y0,n)
h=(b-a)/n
X=zeros(1,n+1)
Y=zeros(1,n+1)
X(1)=a
Y(1)=y0
for j=1:n
   Y(j+1)=Y(j)+h*feval('fun2',X(j),Y(j))
   X(j+1)=a+h*j
end
```

Parametrii de intrare:

fun – fişier funcţie ce conţine expresia ecuaţiei diferenţiale. Acest fişier are ca variabile de intrare *x* şi *y*, iar ca variabilă de ieşire un vector *z*, ale cărui valoare reprezintă valoarea ecuaţiei diferenţiale evaluată în punctele *x*. Fişierul **fun** poate fi apelat astfel:

function *z* = **fun**(*x,y*)

y0 – condiţia iniţială;

n – numărul de paşi;

a, *b* – extremităţile intervalului în care se calculează soluţia.

Parametrii de ieşire:

Y– soluţia aproximativă;

X– punctele din intervalul [a, b] în care se calculează soluţia *Y*.

B.8.3. Metode de tip Runge-Kutta

Caseta B.27. Funcţia **met_RK** pentru soluţionarea numerică a ecuaţiilor diferenţiale

```
function [Y,X] = met_RK(fun,a,b,y0,n)
h=(b-a)/n
X=zeros(1,n+1)
Y=zeros(1,n+1)
X(1)=a
```

```
Y(1)=y0
for j=1:n
   xj=X(j)
   yj=Y(j)
   k1=h*feval(fun,xj,yj)
   k2=h*feval(fun,xj+h/2,yj+k1/2)
   k3=h*feval(fun,xj+h/2,yj+k2/2)
   k4=h*feval(fun,xj+h,yj+k3)
   Y(j+1)=yj+(k1+2*k2+2*k3+k4)/6
   X(j+1)=a+h*j
end
```

Parametrii de intrare:

fun – fişier funcţie ce conţine expresia ecuaţiei diferenţiale. Acest fişier are ca variabile de intrare *x* şi *y*, iar ca variabilă de ieşire un vector *z*, ale cărui valoare reprezintă valoarea ecuaţiei diferenţiale evaluată în punctele *x*. Fişierul **fun** poate fi apelat astfel:

function *z* = **fun**(*x*,*y*)

y0 – condiţia iniţială;

n – numărul de paşi;

a, *b* – extremităţile intervalului în care se calculează soluţia.

Parametrii de ieşire:

Y– soluţia aproximativă;

X– punctele din intervalul [a, b] în care se calculează soluţia *Y*.