

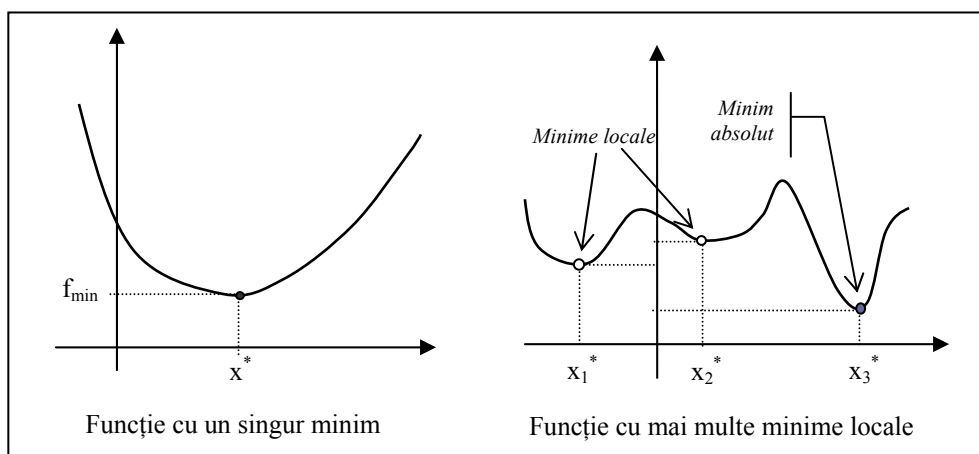
5.8. Calcule asupra funcțiilor

5.8.1 Minimul funcțiilor

Minimul funcțiilor interesează în problemele de extremum. Prin minimizarea unei funcții înțelegem determinarea valorii argumentului, pentru care funcția prezintă valoarea minimă. În cazul funcțiilor *continue și derivabile* minimul lor se determină ca un extremum global care se găsește printre soluțiile sistemului de ecuații:

$$\begin{cases} f'(x) = 0 \\ f''(x) > 0 \end{cases}$$

Operația de minimizare a funcțiilor este foarte utilizată în problemele ingierești vizând optimizarea sistemelor și în controlul automat.



Mediul Matlab pune la dispoziție funcții de minimizare pentru trei situații:

- Funcția de minimizat este de o singură variabilă;
- Funcția de minimizat este de mai multe variabile;
- Funcția de minimizat are zerouri și acestea interesează ca minime relative.

A) Minimul funcțiilor de o singură variabilă

Minimul funcției $F(x)$ se calculează cu funcția `fmin` sau cu funcția `fminbnd` (care o înlocuiește pe prima în versiunile recente ale Matlab-ului), folosind următoarele sintaxe:

```
x=fmin('F',x1,x2)
x=fmin('F',x1,x2,options)
[x,options]=fmin('F',x1,x2,options,p1,p2,...)
```

respectiv:

```
x=fminbnd(FUN,x1,x2)
x=fminbnd(FUN,x1,x2,optiuni)
[x,options]=fminbnd(FUN,x1,x2,options,p1,p2,...)
[x,fval]=fminbnd(...)
[x,fval,exitflag]=fminbnd(...)
[x,fval,exitflag,output]=fminbnd(...)
```

unde:

x – valoarea argumentului pentru care $F(x)$ este minimă;

'F' – o variabilă șir reprezentând numele unui fișier Matlab de tip `function` în care se găsește definită funcția de minimizat, sau o variabilă șir care reprezintă chiar expresia funcției scrisă în sintaxa Matlab;

`FUN` – variabilă generală ce conține funcția definită printr-una din următoarele metode, care vor fi exemplificate ulterior:

- `FUN='expresie funcție';`
- cu directiva `inline`, ce construiește un obiect funcție pornind de la o expresie precizată ca argument;
- cu *handler* (mâner) `@nume_funcție;`
- nume fișier `function`;

$x1, x2$ – sunt limitele între care se caută minimul ($x1 < x^* < x2$);

`options` – vector pentru controlul opțiunilor având 18 componente, dintre care:

- `options(1)/Display` – pentru afișarea valorilor intermediare ale rezolvării (implicit 0 – nu afișează),
- `options(2)/TolX` – impune toleranța de calcul (valoarea implicită este 10^{-4}),
- `options(14)/MaxFunEvals` – controlează numărul maxim de pași pentru căutarea minimului (implicit 500),

- semnificația tuturor opțiunilor se poate obține cu comanda `help foptions` din linia de comandă, (o parte dintre acestea fiind prevăzute pentru dezvoltări ulterioare ale Matlab-ului),
- structura de date `options` poate să apară și în lista parametrilor de ieșire, returnând vectorul celor 18 parametri opționali, sau în cazul funcției `fminbnd` o structură de date ce conține valoarea parametrului `options(8)`-minimul funcției,
- prin tastarea comenzii `opt` este returnat vectorul linie cu valorile celor 18 parametri ai execuției curente,
- structura de date `options` poate fi stabilită cu valori implicite optimizate create de funcția `optimset`.

`p1, p2, ...` - reprezintă parametri adiționali (maxim 10) ce pot fi transmiși cu rol de argumente funcției $F(x, p1, p2, \dots)$;

`fval` -variabilă ce conține valoarea curentă a funcției obiectiv, inclusiv pe cea finală minimă obținută (calculată pentru valoarea minimă $F(x^*)$);

`Exitflag` – variabilă de ieșire care indică care a fost condiția de terminare a execuției funcției, astfel: 1-dacă minimul a fost găsit cu toleranța dată, 0-dacă execuția s-a terminat ca urmare a depășirii numărului maxim de pași;

`Output` – variabilă de ieșire ce returnează o structură de date cu trei componente:

- `iterations` -numărul de pași efectuat de algoritm pentru găsirea minimului și terminarea execuției,
- `funcCount` -numărul de valori ale funcției calculate,
- `algorithm` -algoritmul aplicat de funcție pentru căutarea minimului.

Aplicațiile prezentate în continuare exemplifică modul de utilizare a funcției de aflare a minimului pentru diferite variante de definire a funcției argument. Exemplele sunt relevante inclusiv pentru modurile în care pot fi definite și transmise ca argumente expresiile analitice cu rol de funcție în cadrul apelurilor funcțiilor Matlab.

a) Definirea și transferul expresiei argumentului funcție ca șir de caractere

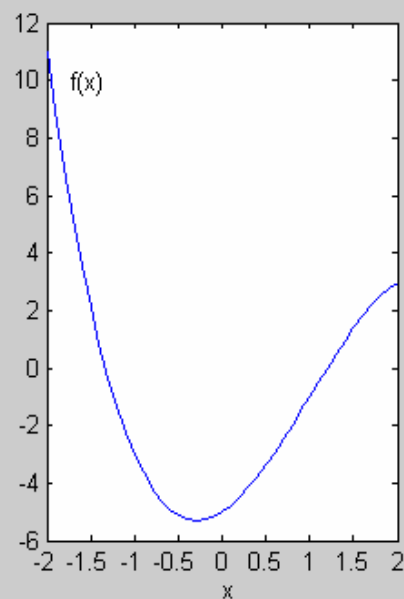
Exemplu. Să se determine minimul funcției polinomiale $f(x) = -x^3 + 3x^2 + 2x - 5$ aflat în intervalul $[-2, 2]$, punând în evidență condiția de terminare a execuției (exitflag) și parametrii de ieșire output.

```
%definire functie
>> fun='-x.^3+3.*x.^2+2.*x-5'

fun =
    -x.^3+3.*x.^2+2.*x-5
```

```
%reprezentare functie
>> p=[-1 3 2 -5]
p =
    -1     3     2    -5

>> x=-2:0.1:2;
>> y=polyval(p,x);
>> plot(x,y)
```



```
%determinarea minimului
>> [xs,fval,exitflag,output]=fminbnd(fun,-2,2)

xs =
    -0.2910

fval =
    -5.3033

exitflag =
     1

output =
    iterations: 10
    funcCount: 12
    algorithm:'golden            section            search,            parabolic
    interpolation'
```

b) Definirea și transferul funcției cu *handler* @nume_func•ie

Exemplu. Să se determine valoarea minimă a funcției sinus în intervalul $[-2, 2]$, punând în evidență condiția de terminare a execuției (exitflag) și parametrii de ieșire output.

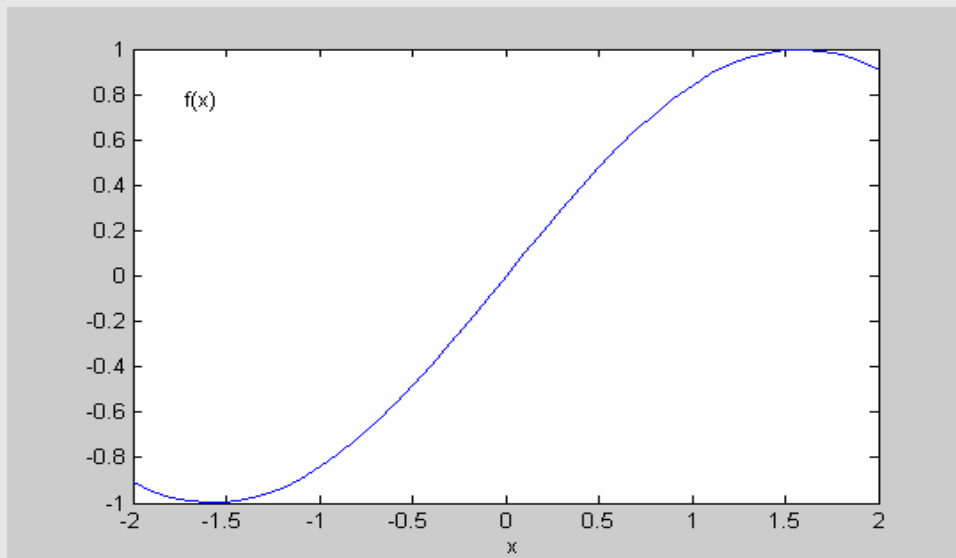
```
>> [xs,fval,exitflag,output]=fminbnd(@sin,-2,2)

xs =
    -1.5708

fval =
    -1.0000

exitflag =
         1

output =
    iterations: 9
    funcCount: 11
    algorithm:'golden section search,parabolic interpolation'
```



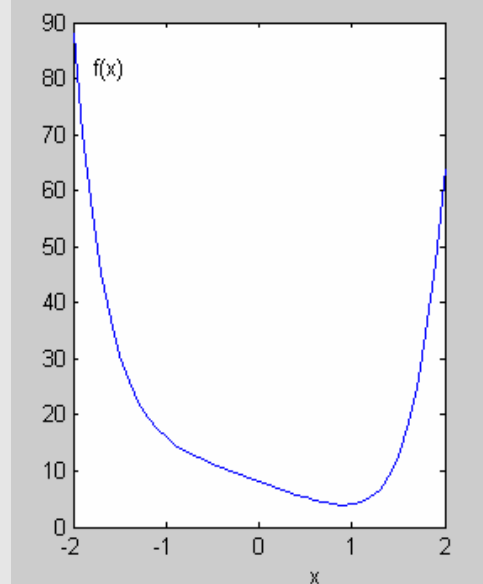
c) Definirea și transferul funcției prin fișier `function`

Exemplu. Să se determine minimul aflat în intervalul $[-2, 2]$ pentru funcția polinomială $f(x) = x^6 + (x-3)^2 - 1$ definită în fișierul `func•ie.m`, punând în evidență condiția de terminare a execuției (`exitflag`) și parametrii de ieșire `output`.

```
%fișier functie.m
function f=functie(x)
    f=x.^6+(x-3).^2-1;
```

```
%Secvența pentru
reprezentare grafică
```

```
>> x=-2:0.1:2;
>> f=functie(x);
>> plot(x,f)
```



```
%determinarea minimului
```

```
>> [xs,fval,exitflag,output]=fminbnd('functie',-2,2)
```

```
xs =
    0.9286
```

```
fval =
    3.9319
```

```
exitflag =
    1
```

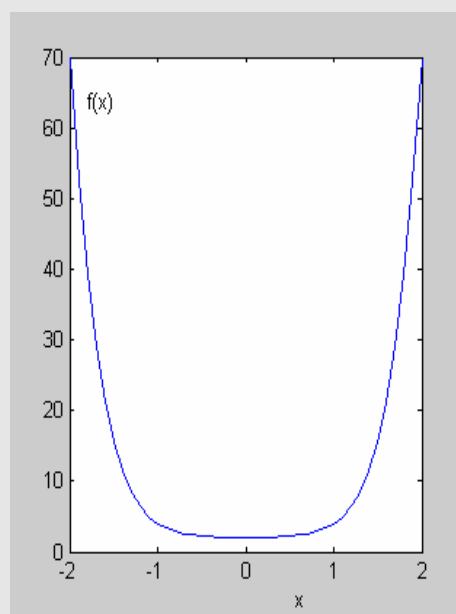
```
output =
    iterations: 12
    funcCount: 14
    algorithm: 'golden section search,parabolic interpolation'
```

d) Exemplu pentru funcții cu opțiuni și parametri

Aplicația de mai jos reia exemplul de la punctul (c) pentru o formă mai generală a funcției polinomiale $f(x) = x^6 + (x - m)^2 - n$ în care intervin parametrii m și n . Se exemplifică utilizarea sintaxei celei mai generale, care permite transferul unor parametrii adiționali funcției argument, precum și a unor valori opționale de intrare.

```
Function f=functie(x,p1,p2)
    f=x.^6+(x-p1).^2-p2;
```

```
>> x=-2:0.1:2;
>> f=functie(x,0,-2);
>> plot(x,f)
```



```
>> [xs,fval,exitflag,output]=fminbnd('functie',
-2,2,optimset('TolX',1e-12,'Display','off'),0,-2)
```

```
xs =
    1.2443e-012
```

```
fval =
     2
```

```
exitflag =
     1
```

```
output =
    iterations: 46
    funcCount: 48
    algorithm: 'golden section search,parabolic interpolation'
```

B) Minimul funcțiilor de mai multe variabile

Minimul unei funcții de forma $F(x_1, x_2, \dots)$ se determină cu funcția `fmins` respectiv cu funcția `fminsearch` (care o înlocuiește pe prima în versiunile recente ale Matlabului), cu următoarele sintaxe:

```
X = fmins('F', x0)
X = fmins('F', x0, options)
[X, optiuni] = fmins('F', x0, options, [], p1, p2, ...)
```

respectiv:

```
X = fminsearch(fun, x0)
X = fminsearch(fun, x0, options)
X = fminsearch(fun, x0, options, p1, p2, ...)
[X, fval] = fminsearch(...)
[X, fval, exitflag] = fminsearch(...)
[X, fval, exitflag, output] = fminsearch(...)
```

unde:

`x` – vector linie/coloană ce conține valorile argumentelor pentru care funcția argument `F` sau `fun` este minimă;

`'F'` – o variabilă șir reprezentând numele unui fișier Matlab de tip `function` în care se găsește definită funcția de minimizat de mai multe variabile, sau o variabilă șir care reprezintă chiar expresia funcției de mai multe variabile scrisă în sintaxa Matlab;

`fun` – se definește prin aceleași metode ca la funcțiile de o variabilă;

`x0` – vector linie/coloană în care se definește o estimare inițială a minimului (foarte importantă în cazul când funcția are mai multe minime locale), algoritmul de minimizare va căuta minimul funcției în zona (subdomeniul) sugerat de estimția inițială.

Toate considerațiile privitoare la semnificația parametrilor de intrare, respectiv a celor returnați de funcțiile `fmins` și `fminsearch` rămân valabile de la cele pentru funcțiile de o singură variabilă (vezi pct. A).

Se menționează că aceste două funcții folosesc metoda directă de căutare a minimului cunoscută sub numele de metoda simplex Nelder-Mead.

Notă. Alte funcții Matlab relative la problema determinării minimului funcțiilor de una sau mai multe variabile sunt: `foptions`, `optimset`, `inline`, caracterul special `@`. Se recomandă consultarea help-ului Matlab.

Exemplu comentat.

Fie funcția de două variabile și parametru a :
 $F(x_1, x_2, a) = 100(x_2 - x_1^2)^2 + (a - x_1)^2 + 2$. Să se determine
 minimul funcției F cu estimăția inițială $x_0 = (-1, 5; 0, 5)$ și $a = 3$.

Se efectuează următoarele operații:

- Se scrie fișierul funcției și se salvează chiar cu numele acesteia, de pildă `test.m`

```
function F=test(x,a)
F=100*(x(2)-x(1)^2)^2+(a-x(1))^2+2;
```

- Cu secvența de instrucțiuni scrise în linie de comandă:

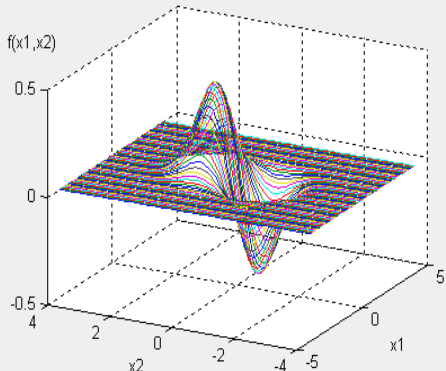
```
X0=[-1.5, 0.5];
a=3;
[xmin,opt]=fmins('test',x0,[0,1e-8],[],a);
xmin
Fmin=test(xmin,a)
Nr_pasi=opt(10)
```

Se obțin succesiv:

```
xmin=[3.000 9.000]
Fmin=2.000
Nr_pasi=317
```

Exemple.

(1) Să se găsească minimul funcției de două variabile
 $f(x_1, x_2, p) = x_2 \cdot e^{(-x_1^2 - x_2^2)}$ pornind de la două estimății
 inițiale: $[0; 0.5]$ respectiv $[-0.25; 0.5]$. Se vor scrie
 instrucțiuni suplimentare pentru reprezentarea grafică a funcției
 de două variabile.

<pre>%fisierul functiei function f=functionie(x) f=x(2).*exp(-x(1).^2-x(2).^2)</pre>	
<pre>%reprezentare grafica >>x1=-4:0.1:4 >>x2=-4:0.1:4 >>[x1i,x2i]=meshgrid(x1,x2) >>f=x2i.*exp(-x1i.^2.-x2i.^2) >>plot3(x1i,x2i,f)</pre>	
<pre>%determinarea minimului pentru prima estimatie >> [X,FVAL,EXITFLAG,OUTPUT]=FMINSEARCH('functie',[0;0.5]) X = 0.0000 -0.7071 FVAL = -0.4289 EXITFLAG = 1 OUTPUT = iterations: 43 funcCount: 82 algorithm: 'Nelder-Mead simplex direct search'</pre>	
<pre>%determinarea minimului pentru a doua estimatie >> [X,FVAL,EXITFLAG,OUTPUT]=FMINSEARCH('functie',[-0.25;0.5]) X = 0.0000 -0.7071 FVAL = -0.4289 EXITFLAG = 1 OUTPUT = iterations: 46 funcCount: 90 algorithm: 'Nelder-Mead simplex direct search'</pre>	

Observație. Indiferent de estimăția inițială rezultatul găsit este același, dar efortul de calcul (numărul de iterații și valorile calculate ale funcției) este ușor diferit în cele două situații.

(2) Exemplu pentru o funcție de două variabile cu un singur parametru: $f(x_1, x_2, p) = x_2 \cdot e^{\left(\frac{-x_1^2 - x_2^2}{p}\right)}$. Se remarcă invocarea funcției `optimset` pentru comunicarea parametrilor de lucru optimi.

```
%fisierul functiei
function f=functie(x,p1)
f=x(2)*exp((-x(1)^2-x(2)^2)/p1);

>>[xmin,fmin,exitflag,output]= fminsearch('functie',[0;0.5], optimset,10)

xmin =
    -0.0000
   -2.2361

fmin =
   -1.3562

exitflag =
         1

output =
    iterations: 57
    funcCount: 105
    algorithm: 'Nelder-Mead simplex direct search'
```

C) Calculul zerourilor funcțiilor de o variabilă reală

În general, zerourile unei funcții dau informații despre locul (locurile) în care graficul funcției taie axa absciselor, iar funcția schimbă semnul. Valorile abscisei în aceste puncte reprezintă rădăcinile ecuației $F(x)=0$ sau în particular, rădăcinile *polinomului caracteristic* asociat funcției respective. În figura 5.10 este ilustrat conceptul de zero pentru o funcție oarecare.

Zerourile unei funcții pot fi interpretate și ca minime relative. De exemplu, dacă într-un punct de zero funcția nu își schimbă semnul, aceasta înseamnă că funcția a atins valoarea sa minimă.

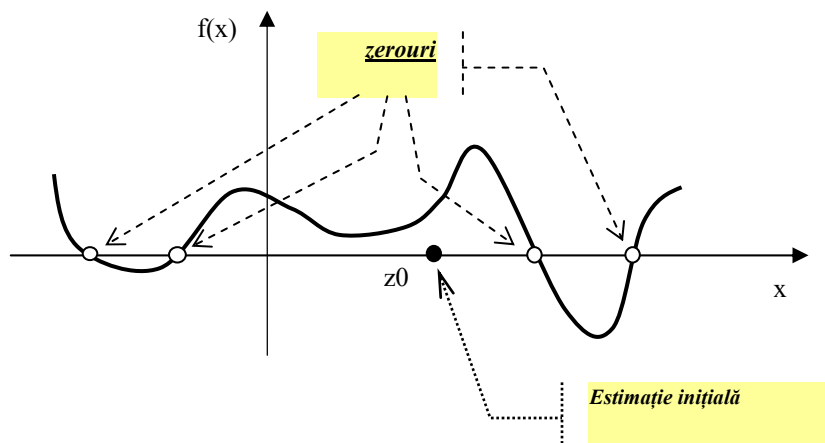


Fig. 5.10. Punerea în evidență a zerourilor și a unei estimății inițiale

Zerourile unei funcții se determină cu funcția Matlab `fzero` folosită cu următoarele sintaxe:

```
z = fzero(fun,z0)
z = fzero(fun,z0,options)
z = fzero(fun,z0,options,p1,p2,...)
[z,fval]= fzero(fun,...)
[z,fval,exitflag] = fzero(...)
[z,fval,exitflag,output] = fzero(...)
```

unde:

`z` – valoarea abscisei pentru care $F(x)$ are zeroul cel mai apropiat de o estimăție inițială dată z_0 ;

`fun` – variabilă generală ce conține funcția obiectiv ale cărei zerouri se caută, definită printr-una din următoarele metode amintite deja:

- `fun='expresie funcție'`,
- cu directiva `inline`,
- cu *handler* (mâner) `@nume_funcție`,
- cu nume fișier `function`,

`z0` – valoare reprezentând estimăția inițială a punctului în care funcția are un zero; alegerea acestuia influențează numărul pașilor (iterațiilor) de căutare a zerourilor;

`options` – vector pentru controlul opțiunilor, această structură de date putând fi stabilită cu valori implicite optimizate create de funcția `optimset`;

`p1, p2, ...` – reprezintă parametri adiționali (maxim 10), ce pot fi transmiși cu rol de argumente funcțiilor de forma $F(x, p1, p2, \dots)$;

`fval` – variabilă ce conține valoarea curentă a funcției obiectiv ale cărei zerouri se caută, inclusiv pe cea finală $f(x)=0$, dacă aceasta are un zero;

`exitflag` – variabilă de ieșire care indică care a fost condiția de terminare a execuției funcției, având valori astfel: o valoare pozitivă - dacă a fost găsit un zero; un număr negativ - dacă nu există nici un interval în care funcția își schimbă semnul sau pe timpul căutării s-a produs o valoare de tip NaN ori inf, fie s-a obținut o valoare complexă (soluție imaginară);

`output` – variabilă de ieșire ce returnează o structură de date cu trei componente:

- `iterations` – numărul de pași efectuat de algoritm pentru găsirea minimului și terminarea execuției,
- `funcCount` – numărul de valori ale funcției calculate,
- `algorithm` – algoritmul aplicat de funcție pentru căutarea minimului.

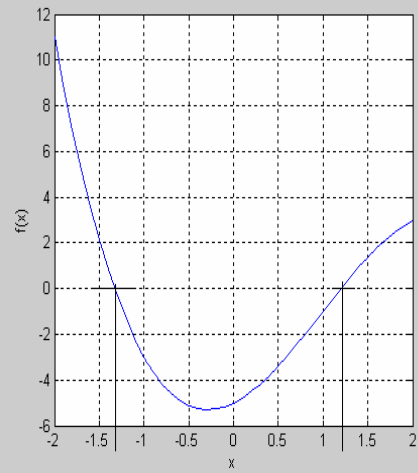
Observație. Pentru funcții polinomiale, zerourile acestora pot fi calculate și cu funcția `roots`, (vezi secțiunea 5.7.4).

Exemplu.

Să se determine succesiv zerourile funcției polinomiale $f(x) = -x^3 + 3x^2 + 2x - 5$ pornind de la două estimări inițiale $x_0 = -2$ respectiv $x_0 = 2$ cu afișarea tuturor parametrilor returnați de funcția `fzero`.

Valorile determinate ale zerourilor se pun în evidență și prin reprezentare pe graficul funcției.

```
%definire functie
>> fun='-x.^3+3.*x.^2+2.*x-5'
fun =
    -x.^3+3.*x.^2+2.*x-5
```



```
>> [Z,FVAL,exitflag,output]= FZERO(fun,-2)
Z =
    -1.3301
FVAL =
    -1.7764e-015
exitflag =
         1
output =
    iterations: 26
    funcCount: 26
    algorithm: 'bisection, interpolation'
```

```
>> [Z,FVAL,exitflag,output]= FZERO(fun,2)
Z =
     1.2016
FVAL =
    -8.8818e-016
exitflag =
         1
output =
    iterations: 25
    funcCount: 25
    algorithm: 'bisection, interpolation'
```

5.8.2 Derivarea numerică a funcțiilor

Derivarea ca și integrarea sunt concepte fundamentale în modelarea matematică a fenomenelor. Dacă funcțiile care descriu o anumită problemă sunt exprimate analitic și în plus sunt derivabile, atunci ele pot fi derivate respectiv integrate după ce eventual au fost descompuse în funcții elementare, pentru care există derivate respective primitive (imEDIATE). Operațiile de derivare respectiv de integrare au efect reciproc, fiind considerate complementare.

În multe situații impuse de practică funcțiile nu admit descrieri analitice și nu pot fi descompuse în funcții elementare fără a fi approximate. În astfel de situații derivarea/integrarea numerică devine o practică generală în rezolvarea problemelor științifice, care oferă soluții aproximative satisfăcătoare.

Din punctul de vedere al calculelor în inginerie, problematica derivării funcțiilor vizează următoarele aspecte practice:

- a) Aproximarea numerică a derivatelor:
 - cu diferențe finite,
 - cu polinomul de interpolare Newton;
- b) Aproximarea numerică a Laplaceanului unei funcții;
- c) Aproximarea numerică a gradientului unei funcții.

A) Derivarea numerică a funcțiilor de o variabilă

Derivata unei funcții semnifică *viteza de variație a funcției* pe un (sub)domeniu de definiție al argumentului acesteia. Această interpretare calitativă a conceptului de derivată este intuitivă. Din punct de vedere cantitativ derivata unei funcții $f(x)$ este o măsură caracteristică punctuală a funcției și se definește la limită pentru oricare valoare x_0 din domeniul său de definiție, care se notează convențional cu $f'(x_0)$ reprezentând derivata de ordinul întâi, astfel:

$$\lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0} = f'(x_0).$$

Semnificația geometrică a derivatei ca tangentă la graficul funcției ajută și mai mult la perceperea intuitivă a conceptului în sensul evaluării unghiului (panta) tangentei, ca măsură a vitezei de variație a funcției. Acest lucru este evidențiat în figura 5.11.

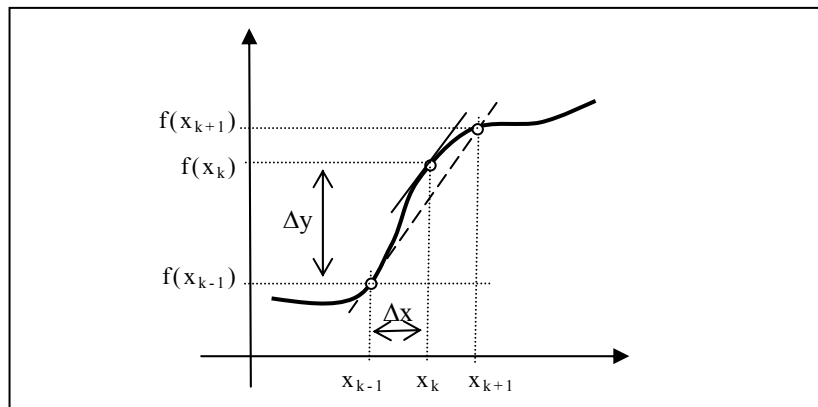


Fig. 5.11. Ilustrarea conceptului de derivată

a) Aproximarea derivatei cu diferențe finite

Operația de derivare numerică presupune discretizarea funcției $f(x)$ rezultând o mulțime ordonată de valori discrete $\{..., f(x_{k-1}), f(x_k), f(x_{k+1}), ...\}$. Aproximarea numerică se produce la trecerea din domeniul continuu în cel discret prin asocierea diferențelor finite variațiilor infinit mici $\Delta x \cong dx$ respectiv $\Delta f \cong df$, astfel:

$$f'(x) = \frac{df}{dx} \cong \frac{\Delta f}{\Delta x}.$$

Prin urmare, o aproximare numerică a derivatei în punctul x_k se poate exprima cu diferențe finite astfel:

- $f'(x_k) \cong \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$ (diferențe regresive),
- $f'(x_k) \cong \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}$ (diferențe progresive),
- $f'(x_k) \cong \frac{f(x_{k+1}) - f(x_{k-1}))}{x_{k+1} - x_{k-1}}$ (diferențe centrale),

Diferențele finite de la numitor Δx semnifică pasul de discretizare (divizare) pe domeniul *variabilei independente* (argumentul funcției), iar

cele de la numărător Δf reprezintă variația finită a funcției, considerată *variabilă dependentă*. Gradul de aproximare la derivarea numerică depinde de mărimea pasului de discretizare. La limită, pentru $\Delta x \rightarrow 0$ se obține aproximația cea mai bună, echivalentă rezultatului analitic (dacă acesta există).

b) Aproximarea derivatei cu polinomul de interpolare Newton

Și în acest caz se presupun cunoscute valori discrete ale funcției $f(x_k)$ pentru mulțimea de puncte *echidistante* $\{x_0, x_1, \dots, x_{k-1}, x_k, x_{k+1}, \dots\}$, ca în cazul unei probleme tipice de interpolare. Această metodă se bazează tot pe noțiunea de diferență finită, care se extinde pentru ordine superioare astfel:

$\Delta x = x_{k+1} - x_k = h$ este pasul de divizare,

$\Delta f_k = f(x_{k+1}) - f(x_k)$ sunt diferențe finite de ordinul 1,

$\Delta^2 f_k = \Delta f_{k+1} - \Delta f_k$ reprezintă diferențele finite de ordinul 2,

$\Delta^3 f_k = \Delta^2 f_{k+1} - \Delta^2 f_k$ reprezintă diferențele finite de ordinul 3, etc.

Pe baza acestor entități calculul numeric al derivatei se face cu precizie mai bună luând în considerare diferențele finite de ordin superior. De exemplu, considerând diferențele finite până la ordinul patru se calculează:

- derivata de ordinul întâi $f'(x_k) = \frac{1}{h} \left(\Delta f_k - \frac{\Delta^2 f_k}{2} + \frac{\Delta^3 f_k}{3} - \frac{\Delta^4 f_k}{4} \right)$,
- derivata de ordinul doi $f''(x_k) = \frac{1}{h^2} \left(\Delta^2 f_k - \Delta^3 f_k + \frac{11}{12} \Delta^4 f_k \right)$.

A găsi derivata unei funcții înseamnă a *diferenția* funcția. Geometric, diferențiala funcției este reprezentată prin variația ordonatei punctului funcției Δy corespunzător variației argumentului Δx , (vezi figura 5.11). Derivata funcției se obține prin raportul a două diferențiale (ce pot fi approximate cu diferențe finite).

Diferențele finite pot fi calculate cu funcția Matlab `diff` cu una din următoarele sintaxe de apel:

```
y = diff(x)
y = diff(x,n)
y = diff(x,n,dim)
```

unde:

x - un vector, o matrice sau un tablou n-dimensional conținând valorile numerice date;

y - variabila de ieșire în care se returnează diferențele finite calculate astfel:

- $[X(2)-X(1) \ X(3)-X(2) \ \dots \ X(n)-X(n-1)]$ dacă X este un vector,
- matricea diferențelor rândurilor $[X(2:n,:) - X(1:n-1,:)]$, dacă X este o matrice,
- diferențele calculate pe direcția primei dimensiuni nenule a tabloului n-dimensional X .

n - este ordinul diferențelor finite care vor fi calculate pe direcția primei dimensiuni nesingulare.

Notă. Dacă $n \geq \text{size}(x, \text{dim})$, adică ordinul diferențelor solicitate este mai mare sau egal cu numărul de elemente pe direcția dimensiunii respective, calculul nu este posibil și se vor calcula diferențele pe următoarea direcție valabilă.

dim - dimensiunea precizată ce definește direcția pentru calculul diferențelor finite: de exemplu, în cazul tablourilor bidimensionale pentru $\text{dim}=1$ (cazul implicit) diferențele sunt calculate între rânduri (deci pe verticală), iar dacă $\text{dim}=2$ acestea se vor calcula între coloane (adică pe orizontală).

Notă. În general, dacă se solicită diferențe finite de ordin mai mare sau egal decât oricare dintre dimensiunile tabloului de date dat, adică $n \geq \text{size}(x, \text{dim})$, atunci funcția `diff` va returna un tablou gol.

Exemple.

(1) Calculul diferențelor finite de ordine succesiv crescătoare pentru un vector cu 7 elemente:

```
>> x=[0 1 2 4 7 8 10]
x =
    0     1     2     4     7     8    10

>> n=length(x)
n =
     7

>> y1=diff(x)
y1 =
     1     1     2     3     1     2

>> y2=diff(x,2)
y2 =
     0     1     1    -2     1

>> y3=diff(x,3)
y3 =
     1     0    -3     3

>> y4=diff(x,4)
y4 =
    -1    -3     6

>> y5=diff(x,5)
y5 =
    -2     9

>> y6=diff(x,6)
y6 =
    11

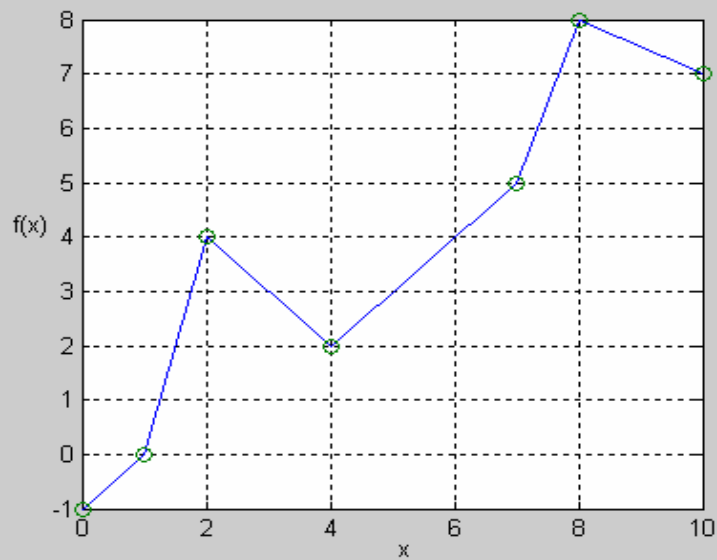
>> y7=diff(x,7)
y7 =
    []
```

(2) Calculul diferențelor finite de ordine succesiv crescătoare pe direcțiile unui tablou de date bidimensional 4×3:

<pre> x1 = 1 2 3 0 3 4 5 6 7 8 9 10 </pre>	
Calculul pe verticală (dim=1)	Calculul pe orizontală (dim=2)
<pre> >> y1=diff(x1,1,1) y1 = -1 1 1 5 3 3 3 3 3 >> y2=diff(x1,2,1) y2 = 6 2 2 -2 0 0 >> y3=diff(x1,3,1) y3 = -8 -2 -2 >> y4=diff(x1,4,1) y4 = Empty matrix: 0-by-3 </pre>	<pre> >> y1=diff(x1,1,2) y1 = 1 1 3 1 1 1 1 1 >> y2=diff(x1,2,2) y2 = 0 -2 0 0 >> y3=diff(x1,3,2) y3 = Empty matrix: 4-by-0 </pre>

(3) Calculul numeric al derivatei cu diferențe finite pentru o funcție definită prin puncte:

```
>> x
>> x=[0 1 2 4 7 8 10]
x =
    0     1     2     4     7     8    10
>> f=[-1 0 4 2 5 8 7]
f =
   -1     0     4     2     5     8     7
```



```
% calculul valorilor derivatei
>> f1=diff(f)./diff(x)
f1 =
    1.0    4.0000 -1.0000    1.0000    3.0000 -0.5000
```

B) Derivarea numerică a funcțiilor de mai multe variabile

Conceptul de *derivată parțială* se aplică la funcțiile de mai multe variabile de forma $f(x, y, z, \dots)$ în raport cu fiecare dintre argumentele funcției x, y, z, \dots considerând că doar unul este variabil iar celelalte sunt constante. Derivatele parțiale se notează și se definesc astfel:

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x, y, z, \dots) - f(x, y, z, \dots)}{\Delta x},$$

$$\frac{\partial f}{\partial y} = \lim_{\Delta y \rightarrow 0} \frac{f(x, y + \Delta y, z, \dots) - f(x, y, z, \dots)}{\Delta y},$$

$$\frac{\partial f}{\partial z} = \lim_{\Delta z \rightarrow 0} \frac{f(x, y, z + \Delta z, \dots) - f(x, y, z, \dots)}{\Delta z},$$

...

În practica inginerescă calculul derivatelor parțiale apare în contextul unor probleme de extremum pentru funcții de mai multe variabile precum și în aproximarea numerică a Laplaceanului unei funcții și la calculul gradientului funcțiilor de câmp.

a) Aproximarea numerică a Laplaceanului unei funcții

Laplaceanul este un tip de operator diferențial care se aplică în general funcțiilor de trei variabile $f(x, y, z)$, fiind exprimat formal astfel:

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

În majoritatea problemelor ingineresti însă, Laplaceanul interesează pentru funcții de două variabile $u(x, y)$, în contextul ecuației lui Laplace $\Delta u = 0$. Cu alte cuvinte, interesează o aproximare numerică a operatorului

diferențial aplicat funcției u , astfel: $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.

Calculul Laplaceanului discret pentru funcții de n variabile se face cu funcția Matlab `del2` care se poate apela cu următoarele sintaxe:

```
L = del2 (U)
L = del2 (U, H)
L = del2 (U, HX, HY)
L = del2 (U, HX, HY, HZ, . . . )
```

unde:

- U este o matrice sau un tablou n -dimensional, ce conține valori discrete ale funcției $u(x, y, \dots)$;

- L este o aproximare discretă a lui $\frac{\Delta^2 u}{4}$ sub forma unei matrice cu aceeași dimensiune ca U , având fiecare element egal cu diferența (cu semn schimbat) dintre un element al lui U și media aritmetică a celor patru elemente vecine acestuia:

$$l_{i,j} = \frac{1}{4}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}) - u_{i,j}.$$

- Argumentele opționale H, HX, HY, HZ ce intră în apelul funcției `del2` sunt mărimi scalare sau vectoriale prin care se poate specifica pasul dintre punctele considerate la calculul numeric al Laplaceanului.

Exemplu.

```
>> U=[10 6 3 7; 1 4 5 8;-10 -4 0 1;0 2 9 -1]

U =
    10     6     3     7
     1     4     5     8
    -10    -4     0     1
     0     2     9    -1

>> L=del2 (U)

L =
   -7.5000   -6.2500   -5.2500   -2.0000
   -2.0000   -2.0000   -1.2500   -0.5000
    5.0000    3.0000    2.7500    0.2500
   17.7500    9.7500    4.5000   -5.2500
```

b) Aproximarea numerică a gradientului

Conceptul de gradient

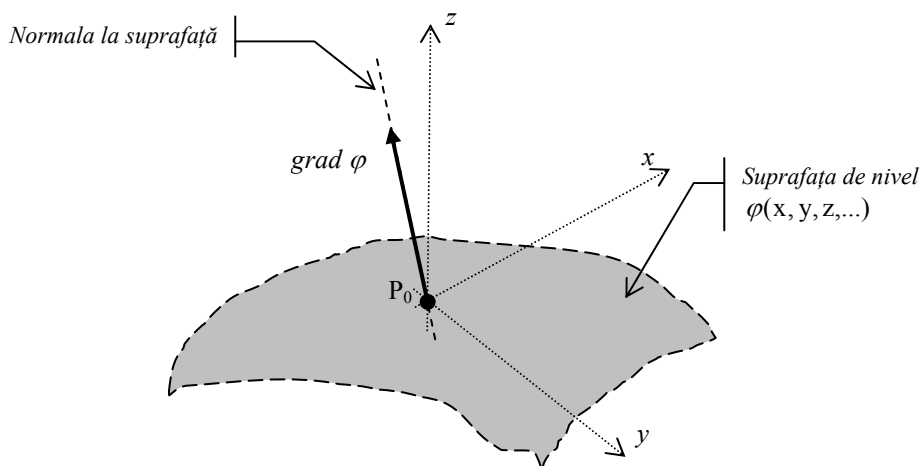
În practica inginerescă apar probleme în care anumite stări și distribuții ale unor mărimi fizice pot fi descrise ca și *câmpuri scalare* reprezentate formal prin *funcții scalare de punct*: $\varphi(x, y, z) = \varphi(P)$, unde $x, y, z \in \mathbb{R}$. Acestea sunt reprezentări de tip suprafață. Astfel de exemple sunt: câmpul temperaturilor, câmpul presiunilor dintr-o încălț, câmpul înălțimilor (cotelor de nivel) unei suprafețe planetare, câmpul sarcinilor electrice distribuite spațial, etc.. Din multitudinea de suprafețe ale unui câmp scalar se disting *suprafețele de nivel* ca fiind locul geometric al punctelor pentru care funcția de câmp are aceeași valoare: $\varphi(x, y, z) = \text{const.}$.

Conceptul de gradient este legat de cel de derivată (parțială). Derivatele parțiale sunt cele care dau mărimea (modulul) gradientului.

Astfel, în contextul descris mai sus, *gradientul* unui câmp scalar într-un punct P_0 la suprafața de nivel este un *vector* cu punctul de aplicație în P_0 , cu direcția dată de normala la suprafață și sensul versorului acesteia, exprimat astfel:

$$\text{grad } \varphi = \frac{\partial \varphi}{\partial x} \cdot \bar{i} + \frac{\partial \varphi}{\partial y} \cdot \bar{j} + \frac{\partial \varphi}{\partial z} \cdot \bar{k}$$

În cazul general, gradientul se poate calcula pentru orice funcție scalară de punct într-un spațiu n-dimensional $\varphi(x, y, z, \dots)$.



Ilustrarea conceptului de gradient

În Matlab, calculul numeric al gradientului se face cu funcția `gradient` apelată cu următoarele particularități sintactice:

`[FX,FY]=gradient(F)` returnează **gradientul numeric 2D** pentru un câmp scalar bidimensional descris numeric prin matricea F , unde FX , respectiv FY sunt valorile derivatelor parțiale $\frac{\partial F}{\partial x}$ și $\frac{\partial F}{\partial y}$ pe cele două direcții ale funcției scalare. Spațiul între punctele câmpului scalar (pasul) pe fiecare direcție se presupune egal cu unu. Gradientul pentru un câmp scalar unidimensional, când F este deci un vector se obține apelând funcția cu sintaxa `DF=gradient(F)`, unde DF conține valorile **gradientului 1D**.

Cu sintaxa `[FX,FY]=gradient(F,H)` se poate specifica valoarea spațiului între punctele câmpului (altă decât cea implicită) pe cele două direcții prin variabila (scalară) de intrare H . Utilizând sintaxa de apelare `[FX,FY]=gradient(F,HX,HY)` pentru gradientul 2D se pot impune spațieri diferite pe cele două direcții prin intermediul variabilelor de intrare HX și HY care pot fi:

- *scalari* specificând spațiul (pasul) dintre coordonatele punctelor,
- *vectori* specificând chiar coordonatele punctelor.

Calculul numeric al **gradientului 3D** se face apelând funcția `gradient` cu una dintre sintaxele următoare:

```
[FX,FY,FZ] = gradient(F)
[FX,FY,FZ] = gradient(F,H)
[FX,FY,FZ] = gradient(F,HX,HY,HZ)
```

În general, **gradientul nD** se aproximează numeric cu apelul de funcție extins la n variabile de ieșire și $n+1$ argumente, astfel:

```
[FX,FY,FZ,...] = gradient(F,...)
```

Exemplu.

Să se calculeze numeric gradientul 2D pentru funcția $f(x,y)=x \cdot e^{(-x^3-y^2)}$ ce definește un câmp scalar bidimensional pe domeniul $[1,2] \times [0,2]$.

Evaluarea gradientului se va face cu pasul de 0,25 pe ambele direcții. Se vor reprezenta următoarele: suprafața definită de

funcție, curbele de nivel și gradientul sub formă de vectori definiți prin componentele lor.

```
% definirea punctelor (grid-ului) domeniului scalar
>> [x,y] = meshgrid(1:.2:2, 0:.2:2);

%calculul valorilor functiei (pe grid-ul dat)
>> f = x.* exp(-x.^3- y.^2)

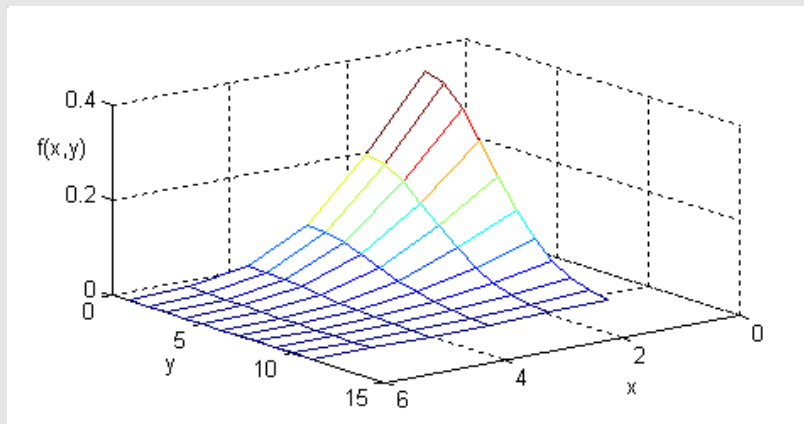
f =
    0.3679    0.2132    0.0900    0.0266    0.0053    0.0007
    0.3535    0.2048    0.0865    0.0256    0.0051    0.0006
    0.3135    0.1816    0.0767    0.0227    0.0045    0.0006
    0.2567    0.1487    0.0628    0.0186    0.0037    0.0005
    0.1940    0.1124    0.0475    0.0140    0.0028    0.0004
    0.1353    0.0784    0.0331    0.0098    0.0019    0.0002
    0.0872    0.0505    0.0213    0.0063    0.0013    0.0002
    0.0518    0.0300    0.0127    0.0038    0.0007    0.0001
    0.0284    0.0165    0.0070    0.0021    0.0004    0.0001
    0.0144    0.0083    0.0035    0.0010    0.0002    0.0000
    0.0067    0.0039    0.0016    0.0005    0.0001    0.0000

% calculul gradientului pe suprafata definita de functie
>> [fx,fy] = gradient(f,.25,.25)

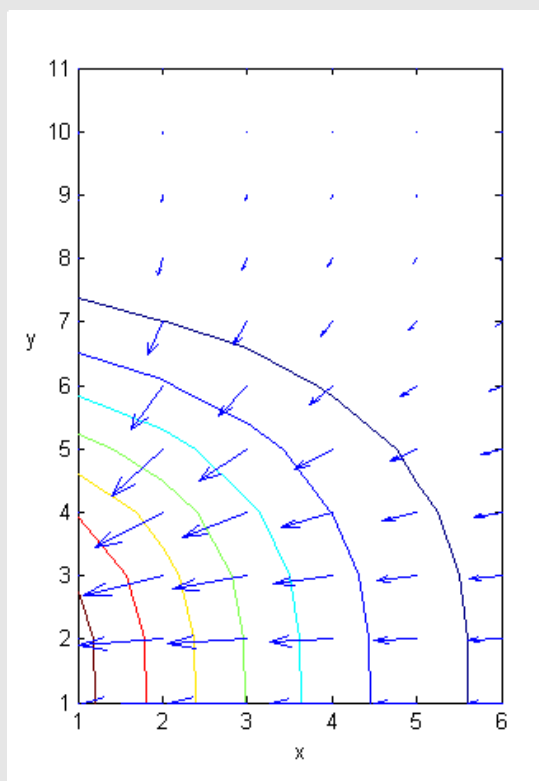
fx =
   -0.6188   -0.5557   -0.3731   -0.1695   -0.0519   -0.0184
   -0.5946   -0.5339   -0.3585   -0.1629   -0.0499   -0.0177
   -0.5273   -0.4735   -0.3179   -0.1445   -0.0442   -0.0157
   -0.4318   -0.3877   -0.2603   -0.1183   -0.0362   -0.0129
   -0.3263   -0.2930   -0.1967   -0.0894   -0.0274   -0.0097
   -0.2277   -0.2044   -0.1373   -0.0624   -0.0191   -0.0068
   -0.1466   -0.1317   -0.0884   -0.0402   -0.0123   -0.0044
   -0.0872   -0.0783   -0.0526   -0.0239   -0.0073   -0.0026
   -0.0478   -0.0430   -0.0288   -0.0131   -0.0040   -0.0014
   -0.0242   -0.0218   -0.0146   -0.0066   -0.0020   -0.0007
   -0.0113   -0.0102   -0.0068   -0.0031   -0.0010   -0.0003

fy =
   -0.0577   -0.0334   -0.0141   -0.0042   -0.0008   -0.0001
   -0.1088   -0.0630   -0.0266   -0.0079   -0.0016   -0.0002
   -0.1936   -0.1122   -0.0474   -0.0140   -0.0028   -0.0004
   -0.2390   -0.1385   -0.0585   -0.0173   -0.0034   -0.0004
   -0.2427   -0.1406   -0.0594   -0.0176   -0.0035   -0.0004
   -0.2136   -0.1238   -0.0523   -0.0155   -0.0031   -0.0004
   -0.1670   -0.0968   -0.0409   -0.0121   -0.0024   -0.0003
   -0.1174   -0.0681   -0.0287   -0.0085   -0.0017   -0.0002
   -0.0748   -0.0434   -0.0183   -0.0054   -0.0011   -0.0001
   -0.0434   -0.0251   -0.0106   -0.0031   -0.0006   -0.0001
   -0.0307   -0.0178   -0.0075   -0.0022   -0.0004   -0.0001

% reprezentarea suprafetei
>> mesh(f)
```



```
% reprezentarea vectorială a gradientului
>> contour(f), hold on, quiver(fx,fy), hold off
```



Observație. Gradientul are valori mai mari în zona abruptă (cu variații mari de cotă) a suprafeței.

5.8.3. Integrarea numerică

Conceptul de integrare a funcțiilor reprezintă complementarul derivării, care împreună se utilizează pe scară largă în știință și inginerie.

Orice funcție poate fi integrată numeric pe intervale finite prin așa numite metode de *cuadratură*, astfel:

- se aproximează funcția dată cu alta care poate fi integrată prin metode analitice (exacte). O bună aproximare a funcției date conduce la o bună estimare a integralei;
- se aproximează funcția dată printr-un set de funcții *liniare* sau *pătratice* pe porțiuni cât mai mici și se însumează suprafețele elementare astfel determinate;

Se disting astfel două metode pentru integrarea numerică a funcțiilor de o variabilă:

- *metoda trapezelor*, când aproximarea se face pe porțiuni cu funcții liniare;
- *metoda Simpson*, una dintre cele mai utilizate metode de cuadratură, când aproximarea se face cu funcții pătratice pe porțiuni;

A) Integrarea funcțiilor de o variabilă

Semnificația fizică a integralei unei funcții de o variabilă are o interpretare geometrică directă prin aria suprafeței plane mărginită de graficul funcției (curba) $f(x)$ și delimitată de axa Ox și dreptele $x=a$ și $y=b$, așa cum se exemplifică în figura 5.12a.

Calculul aproximativ al integralei prin metoda trapezelor este exemplificat intuitiv în figura 5.12b, estimația integralei fiind dată de expresia:

$$S = \frac{b-a}{2n} \left[f(a) + f(b) + 2 \sum_{i=2}^{n-1} f(x_i) \right],$$

în care x_i cu $i=0,1,\dots,n$ sunt nodurile de integrare echidistante cu pasul $h = \frac{b-a}{n}$, iar $f(x_i)$ sunt valorile funcției date în nodurile de integrare.

Matlab pune la dispoziție funcția `trapz` pentru calculul integralei prin metoda trapezelor, ce poate fi apelată cu următoarele sintaxe:

```
S = trapz(Y)
Z = trapz(X,Y)
Z = trapz(X,Y,DIM) sau trapz(Y,DIM),
```

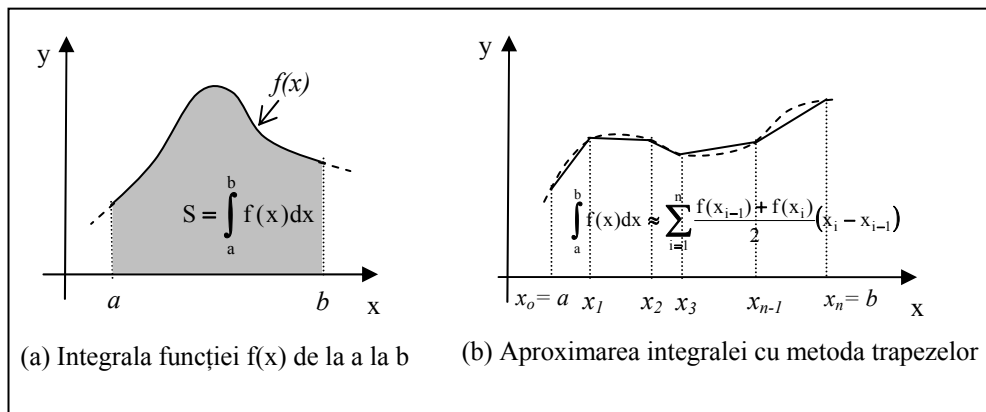
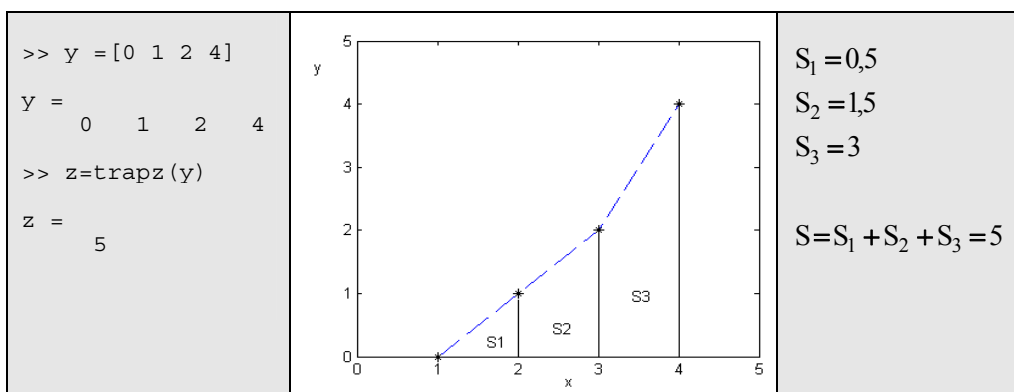


Fig. 5.12. Ilustrarea conceptului de integrală

a) Utilizarea sintaxei `S=trapz(Y)`

Apelarea funcției cu această sintaxă minimală necesită furnizarea variabilei de intrare Y , de regula sub formă de vector ce conține valorile funcției în nodurile de integrare, iar S este variabila de ieșire ce conține valoarea calculată a integralei. Deoarece nodurile de integrare nu sunt furnizate explicit și nici limitele intervalului de integrare nu sunt precizate, în acest caz limitele de integrare sunt luate implicit între 1 și $N = \text{length}(Y)$, echidistante cu pasul egal cu unitatea.

Exemplu



O particularitate de aplicare a funcției `trapz` cu sintaxa minimală este cazul când variabila de intrare `Y` este o matrice. În această situație funcția de returnează un vector linie `Z` ce conține valorile integralei considerând nodurile pe coloanele matricei.

Exemplu.

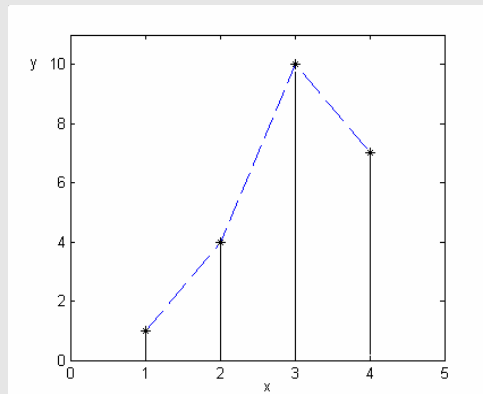
```
>> Y = [1 2; 4 6; 10 12; 7 8]
```

```
Y =
```

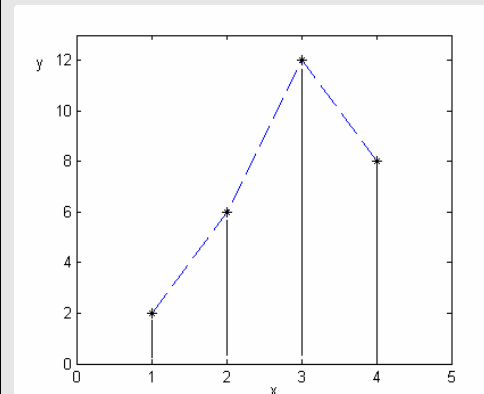
```
    1    2
    4    6
   10   12
    7    8
```

```
>> Z = trapz(Y)
```

```
Z =
   18   23
```



S1=18



S2=23

b) Utilizarea sintaxei `S=trapz(X, Y)`

Apelarea funcției cu această sintaxă permite precizarea explicită a nodurilor de integrare prin variabila de intrare `X`, care poate fi:

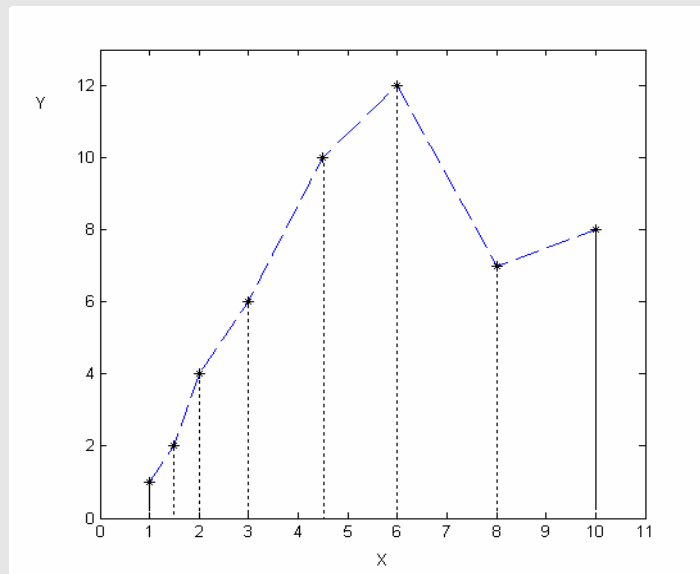
- vector (linie) de aceeași lungime ca `Y`,
- vector coloană, dacă `Y` este un tablou care va avea obligatoriu cel puțin o dimensiune egală cu `length(X)`.

Exemplu.

```
>> Y = [1 2 4 6 10 12 7 8]
Y =
     1     2     4     6    10    12     7     8

>> X = [1 1.5 2 3 4.5 6 8 10]
X =
     1.000     1.500     2.000     3.000     4.500     6.000     8.000    10.000

>> S=trapz(X,Y)
S =
    69.750
```



c) Utilizarea sintaxei `S=trapz (X, Y, DIM)` sau `trapz (Y, DIM)`

În cazul tablourilor multidimensionale `Y`, calculul integralei se poate face precizând dimensiunea `DIM` de-a lungul căreia se calculează integrala, pe nodurile `X` sau implicit pe noduri echidistante, cu pasul 1 pe intervalul

`[1, length(X)]`. În general, pentru ambele situații numărul nodurilor trebuie să fie egal cu `size(Y,DIM)`.

Exemplu.

```
>> Y =[0 1 2 3;8 9 10 11]
Y =
     0     1     2     3
     8     9    10    11

>> size(Y,1)
ans =
     2

>> size(Y,2)
ans =
     4

>> X1=[2 3]
X1 =
     2     3

>> S1=trapz(X1,Y,1)
S1 =
     4     5     6     7

>> X2=[2 3 4 5]
X2 =
     2     3     4     5

>> S2=trapz(X2,Y,2)
S2 =
    4.5000
   28.5000
```


d) Utilizarea funcției cumtrapz

Funcția Matlab `cumtrapz` permite calculul integralei prin metoda trapezelor furnizând la ieșire valoarea cumulată a suprafețelor elementare. Această funcție poate fi utilizată cu una din sintaxele următoare:

```
Z = cumtrapz(Y)
Z = cumtrapz(X,Y)
Z = cumtrapz(X,Y,DIM) sau cumtrapz(Y,DIM)
```

cu mențiunea că rezultatul returnat (variabila *Z*) va fi întotdeauna cel puțin un vector conținând valorile succesive (intermediare) ale integralei calculate până în punctul curent.

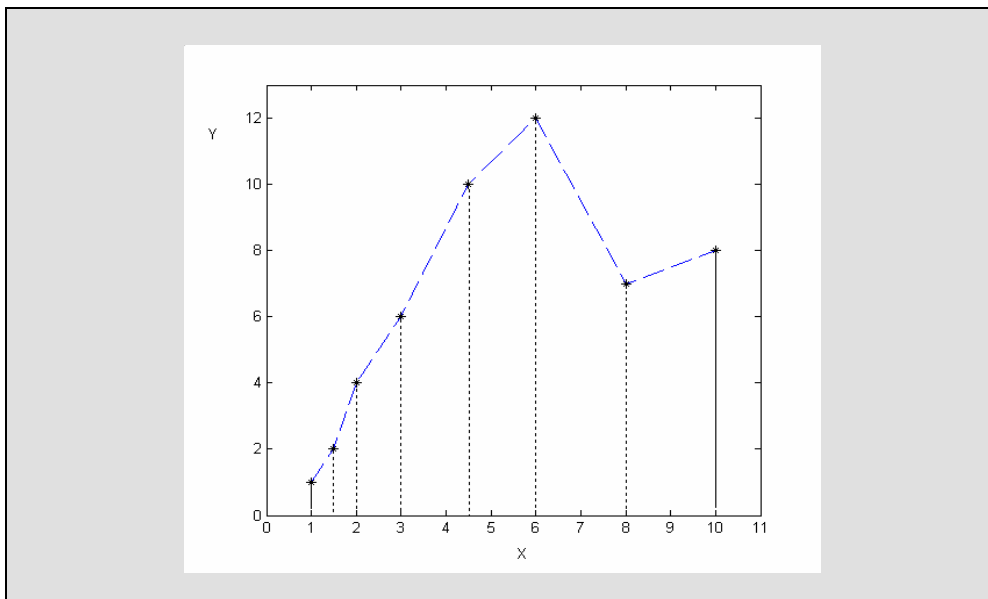
Observație. Funcția `cumtrapz` combină calculul suprafețelor elementare prin metoda trapezelor cu efectuarea sumei cumulate a acestora cu funcția `cumsum`.

Exemplu.

```
>> Y = [1 2 4 6 10 12 7 8]
Y =
    1     2     4     6    10    12     7     8

>> X = [1 1.5 2 3 4.5 6 8 10]
X =
    1.000    1.500    2.000    3.000    4.500    6.000    8.000   10.000

>> Z=cumtrapz(X,Y)
Z =
    0    0.750    2.250    7.250   19.250   35.750   54.750   69.750
```



e) Utilizarea funcțiilor de integrare quad și quadl

Funcția Matlab `quad` calculează integrala unei funcții prin metoda de cuadratură adaptivă a lui Simpson. Această funcție se utilizează cu una dintre sintaxele următoare:

```
Q = quad(FUN, A, B)
Q = quad(FUN, A, B, TOL)
[Q, FCNT] = quad(...)
               quad(FUN, A, B, TOL, TRACE)
               quad(FUN, A, B, TOL, TRACE, P1, P2, ...)
```

Funcția returnează în variabila de ieșire `Q` valoarea aproximată a integralei funcției `FUN` (numită și *integrand*) pe intervalul `[A,B]`, cu eroare de cel mult 10^{-6} , utilizând metoda adaptivă recursivă a lui Simpson. Adăugând în lista variabilelor de ieșire variabila `FCNT` funcția va returna inclusiv numărul pașilor de calcul (evaluărilor) efectuați pentru calculul integralei.

Cu parametrul de intrare opțional `TOL` se poate preciza valoarea absolută a toleranței de calcul, altă decât cea implicită (10^{-6}). Parametrul adițional `TRACE` setat la o valoare diferită de zero permite activarea opțiunii de afișare pe timpul calculului recursiv a unei structuri de date de forma unui vector cu patru elemente: `[FCNT Xi B-Xi Q]`, unde `Xi` este valoarea curentă a nodului de integrare. `P1, P2,...` reprezintă parametri

adiționali ce pot fi transmiși direct ca argumente funcției de integrat $F(x, p1, p2, \dots)$. Folosirea valorilor implicite pentru TOL și TRACE se obține punând matricea goală pentru acestea (simbolul `[]` în lista parametrilor de intrare).

Funcția Matlab `quadl` folosește metoda de cuadratură adaptivă a lui Lobatto fiind considerată mai eficientă (mai precisă) în cazul funcțiilor netede. Pentru aceeași precizie, cu funcția `quadl` rezultatul se calculează cu mai puține iterații (pași). Utilizarea ei se face cu aceleași sintaxe ca și `quad`.

Exemplu.

Să se integreze numeric funcția $f(x) = \frac{1}{x^3 + 2e^x}$ pe intervalul $[0,2]$. Să se reprezinte graficul funcției pe intervalul de integrare.

```
>> F = inline('1./(x.^3+2*exp(x))')
F =
    Inline function:
    F(x) = 1./(x.^3+2*exp(x))

>> [Q,fcnt] = quad(F,0,2,[],0)

Q =
    0.3911

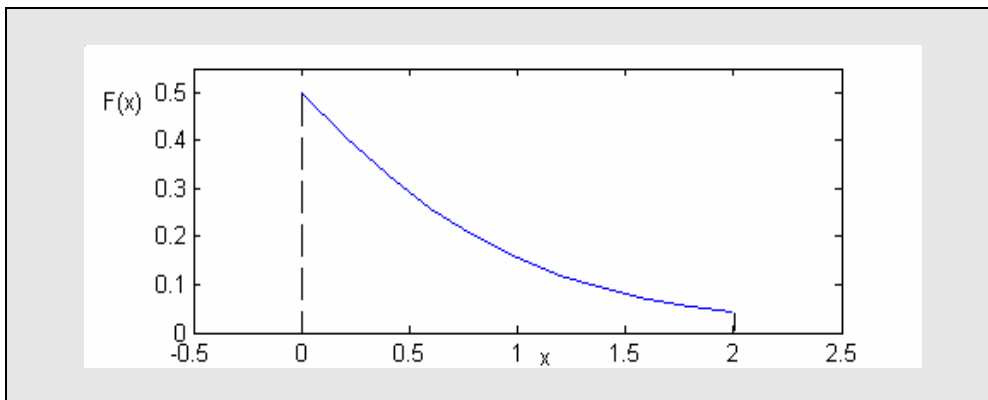
fcnt =
    21

>> [Q,fcnt] = quad(F,0,2,[],1)

     9    0.0000000000    5.43160000e-001    0.2072850003
    11    0.0000000000    2.71580000e-001    0.1186927413
    13    0.2715800000    2.71580000e-001    0.0885917289
    15    0.5431600000    9.13680000e-001    0.1500214164
    17    0.5431600000    4.56840000e-001    0.0966057931
    19    1.0000000000    4.56840000e-001    0.0534157377
    21    1.4568400000    5.43160000e-001    0.0337670459

Q =
    0.3911

fcnt =
    21
```



Observații.

Funcția de integrat FUN poate fi specificată fie ca *obiect inline*, ca în exemplul de mai sus, fie utilizând un *handle* (mâner) pentru aceasta, astfel:

```
Q = quad(@funint, 0, 2)
```

unde `funint.m` este un fișier Matlab, de pildă:

```
function y = myfun(x)
y = 1./(x.^3+2*exp(x))
```

La descrierea analitică a funcțiilor de integrat se recomandă utilizarea operatorilor pentru tablouri `.*`, `./` și `.^` pentru a putea fi evaluate cu argumente vectori.

B) Calculul numeric al integralelor multiple

Conceptul de integrală a unei funcții de o variabilă se extinde corespunzător la funcții de două și trei variabile cu aplicabilitate largă în modelarea problemelor ingineresti.

Astfel, integrala dublă a funcției $f(x,y)$ extinsă la domeniul S se mai numește și *integrală pe suprafață* și reprezintă o valoare definită în modul următor:

$$\iint_S f(x,y) dS = \lim_{\Delta S_i \rightarrow 0} \sum_{i=1}^n f(x_i, y_i) \Delta S_i$$

Integrala triplă a unei funcții $f(x,y,z)$ extinsă la un domeniu tridimensional V se numește *integrală pe volum* și reprezintă o valoare definită în modul următor:

$$\iiint_V f(x,y,z)dV = \lim_{\Delta V_i \rightarrow 0} \sum_{i=1}^n f(x_i, y_i, z_i) \Delta V_i$$

Calculul integralei multiple se reduce la calculul succesiv a unor integrale simple definite. Astfel, în coordonate carteziene variația elementară a domeniului de integrare este $dS = dx \cdot dy$ (aria elementară) respectiv $dV = dx \cdot dy \cdot dz$ (volumul elementar). Prin urmare se poate exprima astfel:

$$\iint_S f(x,y)dS = \iint_S f(x,y)dx dy = \int_a^b dx \int_{y_1(x)}^{y_2(x)} f(x,y)dy,$$

respectiv:

$$\begin{aligned} \iiint_V f(x,y,z)dV &= \iiint_V f(x,y,z)dx dy dz = \\ &= \iint_S dx dy \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z)dz = \\ &= \int_a^b dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z)dz \end{aligned}$$

Evaluarea numerică a integralelor multiple se poate realiza în Matlab cu funcțiile `dblquad` respectiv `triplequad` cu următoarele sintaxe de apelare:

```
dblquad(FUN,XMIN,XMAX,YMIN,YMAX)
dblquad(FUN,XMIN,XMAX,YMIN,YMAX,TOL)
dblquad(FUN,XMIN,XMAX,YMIN,YMAX,TOL,@QUADL)
dblquad(FUN,XMIN,XMAX,YMIN,YMAX,TOL,@MYQUADF)
dblquad(FUN,XMIN,XMAX,YMIN,YMAX,TOL,@QUADL,P1,P2,...)

triplequad(FUN,XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX)
triplequad(FUN,XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX,TOL)
triplequad(FUN,XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX,TOL,@QUADL)
triplequad(FUN,XMIN,XMAX,YMIN,YMAX,TOL,ZMIN,ZMAX,@MYQUADF)
triplequad(FUN,XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX,TOL,@QUADL,P1,P2,...)
```

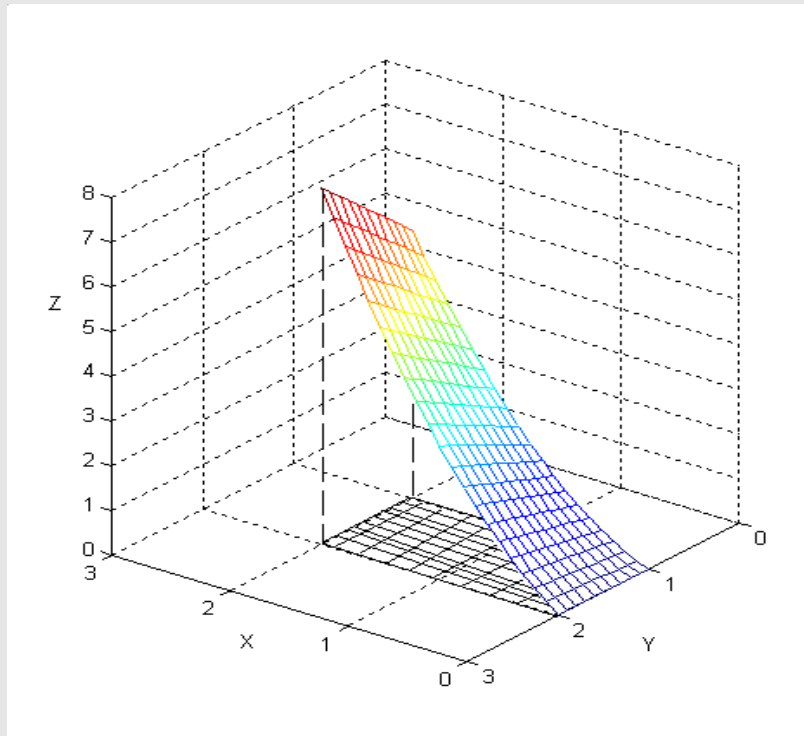
Referitor la integrarea multiplă în continuare se fac unele observații:

- Integrala dublă a funcției $F(X,Y)$ este evaluată numeric pe un domeniu rectangular $XMIN \leq X \leq XMAX$, $YMIN \leq Y \leq YMAX$ cu toleranța implicită de ordinul 10^{-6} sau cu o valoare precizată prin argumentul TOL.
- Metoda de integrare implicit utilizată este metoda adaptiv recursivă a lui Simpson sau opțional, fie metoda de cuadratură adaptivă a lui Lobatto introducând argumentul @QUADL, fie o funcție de cuadratură proprie introdusă prin argumentul @MYQUADF.
- Funcția de integrat poate avea parametri $FUN(X,Y,P1,P2,...)$ care pot fi transferați ca argumente cu ultima dintre sintaxe.
- Precizările de mai sus sunt valabile și în cazul funcției pentru integrarea triplă a funcțiilor de forma $FUN(X,Y,Z)$ respectiv $FUN(X,Y,Z,P1,P2,...)$, cu precizarea că integrarea se efectuează pe domeniul tridimensional rectangular $XMIN \leq X \leq XMAX$, $YMIN \leq Y \leq YMAX$, $ZMIN \leq Z \leq ZMAX$.

Exemplu.

Să se calculeze valoarea integralei duble $Q = \iint_S (yx + x^2) dx dy$ pe domeniul $S = [0,2] \times [1,2]$. Să se reprezinte grafic funcția pe domeniul de integrare.

```
>> Q = dblquad(inline('y.*x+x.^2'), 0, 2, 1, 2)
Q =
    5.6667
```

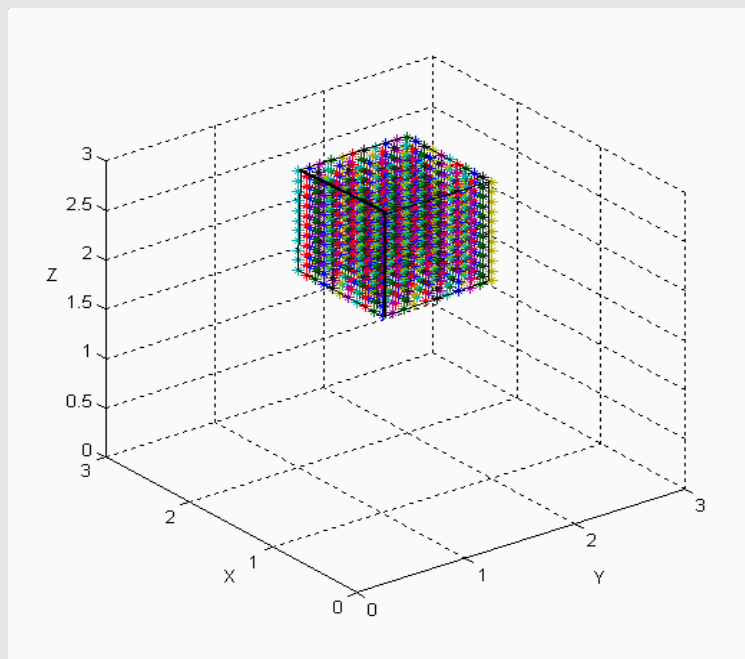


Valoarea integralei duble reprezintă volumul proiectat de suprafața $z = yx + x^2$ pe planul XY.

Exemplu.

Să se calculeze valoarea integralei triple
 $Q = \iiint_V (yx + zy + x^2) dx dy dz$ pe domeniul tridimensional
 $V = [1,2] \times [1,2] \times [2,3]$.

```
>> Q=triplequad(inline('y.*x+z.*y+x.^2'), 1, 2, 1, 2, 1, 2)
Q =
    6.8333
```



Reprezentarea domeniului tridimensional de integrare.

Observație. Valoarea numerică a integralei nu are neapărat o semnificație fizică în acest caz. În funcție de problema concretă de modelat, integrala triplă poate să determine volumul în sens geometric, momente de inerție mecanice, masa unui corp cu densitatea variabilă și în general parametri fizici în sisteme distribuite, acolo unde intervin *funcții scalare de punct*.

C) Particularități la integrarea pe intervale infinite și a funcțiilor cu singularități

Din punctul de vedere al integrării numerice, tratarea situațiilor în care apare problema nedeterminării funcțiilor face apel la metode analitice de prelucrare preliminară a acestora. Astfel, pentru calculul unei integrale definite pe *interval infinit* de tipul:

$$I = \int_a^{\infty} f(x) dx$$

în care funcția se presupune convergentă se poate folosi una dintre metodele următoare:

- 1) Se încearcă o schimbare de variabilă, care ar putea transforma intervalul $[a, \infty)$ într-un interval închis.
- 2) Se procedează la separarea integralei în două integrale însumate:

$$\int_a^b f(x) dx = \int_a^b f(x) dx + \int_b^{2b} f(x) dx ,$$

în care prima integrală se evaluează prin metodele numerice cunoscute, iar a doua se tratează astfel:

- se neglijează pur și simplu, dacă b este ales suficient de mare și în plus $\int_a^b f(x) dx \gg \int_b^{2b} f(x) dx$;
- se aproximează funcția $f(x)$ cu o funcție $g(x) \cong f(x)$, care pentru x suficient de mare permite calculul integralei $g(x)$.

Tratarea *nesingularităților* în cazul integrării numerice se face prin următoarele metode:

- schimbare de variabilă;
- folosirea metodei lui Simpson cu eliminarea singularității prin înlocuirea ei cu $s - \varepsilon$, unde ε este suficient de mic;
- folosirea unei metode de cuadratură cu aproximarea funcției cu polinoame.

Exemplu.

În expresia $\int_0^1 \frac{1}{1-x} dx$ funcția de integrat prezintă o nesingularitate la capătul superior al intervalului de integrare, când $x=1$.

Încercarea de calcul numeric a integralei eșuează astfel:

```
>> Q=quad(inline('1./(1-x)'),0,1)
Warning: Divide by zero.
Q =
    Inf
```

Încercarea de a calcula integrala pe un interval mai mare, care cuprinde nesingularitatea, eșuează de asemenea:

```
>> Q=quad(inline('1./(1-x)'),0,2)
Warning: Divide by zero.
Q =
    NaN
```

Eliminarea singularității prin înlocuirea ei cu valori aproximative succesive, conduce la următoarele rezultate:

```
>> Q=quad(inline('1./(1-x)'),0,0.9999)
Q =
    9.2103
>> Q=quad(inline('1./(1-x)'),0,0.9999999)
Q =
   16.1181
>> Q=quad(inline('1./(1-x)'),0,0.9999999999)
Q =
   23.0259
>> Q=quad(inline('1./(1-x)'),0,0.9999999999999)
Q =
   29.9333
>> Q=quad(inline('1./(1-x)'),0,0.9999999999999999)
Warning: Minimum step size reached; singularity possible.
Q =
   36.6196
```