TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

MATLB: přednáška 5

Optimalizace

Jaroslav Čmejla

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt ESF CZ.1.07/2.2.00/28.0050
Modernizace didaktických metod a inovace výuky technických předmětů.

Základy používání Optimization Toolboxu

- Optimization Toolbox je určen k hledání optimálních a extrémních bodů skalárních reálných funkcí (tzv. objektivní funkce, kritérium nebo kontrastní funkce)
- Numerická optimalizace: můžeme si ji představit jako hledání hmatem poslepu. Každý dotyk, který nám poskytuje informaci k orientaci, znamená vyhodnocení optimalizované funkce.
- Neomezené (unconstrained) nebo omezené (constrained) optimalizace
- Omezení např. na interval, vazbu proměnných (varietu)
- Řešení úlohy lineárního a kvadratického programování

Základy používání Optimization Toolboxu

- Optimization Toolbox je určen k hledání optimálních a extrémních bodů skalárních reálných funkcí (tzv. objektivní funkce, kritérium nebo kontrastní funkce)
- Numerická optimalizace: můžeme si ji představit jako hledání hmatem poslepu. Každý dotyk, který nám poskytuje informaci k orientaci, znamená vyhodnocení optimalizované funkce.
- Neomezené (unconstrained) nebo omezené (constrained) optimalizace
- Omezení např. na interval, vazbu proměnných (varietu)
- Řešení úlohy lineárního a kvadratického programování

Základy používání Optimization Toolboxu

- Optimization Toolbox je určen k hledání optimálních a extrémních bodů skalárních reálných funkcí (tzv. objektivní funkce, kritérium nebo kontrastní funkce)
- Numerická optimalizace: můžeme si ji představit jako hledání hmatem poslepu. Každý dotyk, který nám poskytuje informaci k orientaci, znamená vyhodnocení optimalizované funkce.
- Neomezené (unconstrained) nebo omezené (constrained) optimalizace
- Omezení např. na interval, vazbu proměnných (varietu)
- Řešení úlohy lineárního a kvadratického programování

Základy používání Optimization Toolboxu

- Optimization Toolbox je určen k hledání optimálních a extrémních bodů skalárních reálných funkcí (tzv. objektivní funkce, kritérium nebo kontrastní funkce)
- Numerická optimalizace: můžeme si ji představit jako hledání hmatem poslepu. Každý dotyk, který nám poskytuje informaci k orientaci, znamená vyhodnocení optimalizované funkce.
- Neomezené (unconstrained) nebo omezené (constrained) optimalizace
- Omezení např. na interval, vazbu proměnných (varietu)
- Řešení úlohy lineárního a kvadratického programování

Základy používání Optimization Toolboxu

- Optimization Toolbox je určen k hledání optimálních a extrémních bodů skalárních reálných funkcí (tzv. objektivní funkce, kritérium nebo kontrastní funkce)
- Numerická optimalizace: můžeme si ji představit jako hledání hmatem poslepu. Každý dotyk, který nám poskytuje informaci k orientaci, znamená vyhodnocení optimalizované funkce.
- Neomezené (unconstrained) nebo omezené (constrained) optimalizace
- Omezení např. na interval, vazbu proměnných (varietu)
- Řešení úlohy lineárního a kvadratického programování

Část I

Funkce jedné proměnné

Příklad vlastní optimalizace: Newtonova metoda

- Minimalizujeme funkci

$$f(x) = e^{-2x} + x - 3$$

- Newtonova iterace s inicializací x_0

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \qquad n = 0, 1, \ldots$$

čili v tomto případě

$$x_{n+1} = x_n - \frac{1 - 2e^{-2x_n}}{4e^{-2x_n}}$$

- Implementace:

```
>> x=1; % inicializace
>> for i=1:8
>> x(i+1)=x(i)-(1-2*exp(-2*x(i)))/(4*exp(-2*x(i)));
>> end
```

Příklad vlastní optimalizace: Newtonova metoda

- Minimalizujeme funkci

$$f(x) = e^{-2x} + x - 3$$

- Newtonova iterace s inicializací x_0

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \qquad n = 0, 1, \ldots$$

čili v tomto případě

$$x_{n+1} = x_n - \frac{1 - 2e^{-2x_n}}{4e^{-2x_n}}$$

- Implementace:

```
>> x=1; % inicializace
>> for i=1:8
>> x(i+1)=x(i)-(1-2*exp(-2*x(i)))/(4*exp(-2*x(i)));
>> end
```

Příklad vlastní optimalizace: Newtonova metoda

- Minimalizujeme funkci

$$f(x) = e^{-2x} + x - 3$$

- Newtonova iterace s inicializací x_0

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}, \qquad n = 0, 1, \ldots$$

čili v tomto případě

$$x_{n+1} = x_n - \frac{1 - 2e^{-2x_n}}{4e^{-2x_n}}$$

- Implementace:

```
>> x=1; % inicializace
>> for i=1:8
>> x(i+1)=x(i)-(1-2*exp(-2*x(i)))/(4*exp(-2*x(i)));
>> end
```

Newtonova metoda: výsledek příkladu

- Výsledek

```
>> x'
ans =
    1.000000000000000
   -0.347264024732662
    0.027908464760413
    0.263557443124567
    0.340048003196198
    0.346531191644278
    0.346573588482379
    0.346573590279973
    0.346573590279973
```

- Analytické řešení: $x = \frac{\ln 2}{2}$

```
>> log(2)/2
ans =
    0.346573590279973
```

Newtonova metoda: výsledek příkladu

- Výsledek

```
>> x'
ans =
   1.000000000000000
  -0.347264024732662
   0.027908464760413
   0.263557443124567
   0.340048003196198
   0.346531191644278
   0.346573588482379
   0.346573590279973
   0.346573590279973
```

- Analytické řešení: $x = \frac{\ln 2}{2}$

```
>> log(2)/2
ans =
   0.346573590279973
```

Minimalizace funkce jedné proměnné: `fminbnd`

- Použití příkazu `fminbnd`

```
>> fminbnd(@funkce1,x1,x2,opt)
```

- `@funkce1` vrací odkaz (handle) na funkci `funkce1` (viz `help function_handle`). Příkaz `fminbnd` pomocí tohoto odkazu volá `funkce1`. Proto se také příkazům jako je `fminbnd` říká "funkce funkcí" (function functions).
- `x1` a `x2` určují interval, na kterém se hledá optimální bod.
- `opt` obsahuje parametry optimalizace. Tuto proměnnou lze nastavit pomocí příkazu `optimset`.
- `fminbnd` předpokládá, že `funkce1` je na intervalu spojitá
- Chceme-li najít maximum, stačí u optimalizované funkce otočit znaménko.
- Má-li funkce parametry `a1`, `a2`,...

```
>> fminbnd(@(x)funkce1(x,a1,a2),x1,x2,opt)
```

Minimalizace funkce jedné proměnné: `fminbnd`

- Použití příkazu `fminbnd`

```
>> fminbnd(@funkce1,x1,x2,opt)
```

- `@funkce1` vrací odkaz (handle) na funkci `funkce1` (viz `help function_handle`). Příkaz `fminbnd` pomocí tohoto odkazu volá `funkce1`. Proto se také příkazům jako je `fminbnd` říká "funkce funkcí" (function functions).
- `x1` a `x2` určují interval, na kterém se hledá optimální bod.
- `opt` obsahuje parametry optimalizace. Tuto proměnnou lze nastavit pomocí příkazu `optimset`.
- `fminbnd` předpokládá, že `funkce1` je na intervalu spojitá
- Chceme-li najít maximum, stačí u optimalizované funkce otočit znaménko.
- Má-li funkce parametry `a1`, `a2`,...

```
>> fminbnd(@(x)funkce1(x,a1,a2),x1,x2,opt)
```

Minimalizace funkce jedné proměnné: `fminbnd`

- Použití příkazu `fminbnd`

```
>> fminbnd(@funkce1,x1,x2,opt)
```

- `@funkce1` vrací odkaz (handle) na funkci `funkce1` (viz `help function_handle`). Příkaz `fminbnd` pomocí tohoto odkazu volá `funkce1`. Proto se také příkazům jako je `fminbnd` říká “funkce funkcí” (function functions).
- `x1` a `x2` určují interval, na kterém se hledá optimální bod.
- `opt` obsahuje parametry optimalizace. Tuto proměnnou lze nastavit pomocí příkazu `optimset`.
- `fminbnd` předpokládá, že `funkce1` je na intervalu spojitá
- Chceme-li najít maximum, stačí u optimalizované funkce otočit znaménko.
- Má-li funkce parametry `a1`, `a2`,. . .

```
>> fminbnd(@(x)funkce1(x,a1,a2),x1,x2,opt)
```

Minimalizace funkce jedné proměnné: fminbnd

- Použití příkazu fminbnd

```
>> fminbnd(@funkce1,x1,x2,opt)
```

- @funkce1 vrací odkaz (handle) na funkci funkce1 (viz help function_handle). Příkaz fminbnd pomocí tohoto odkazu volá funkce1. Proto se také příkazům jako je fminbnd říká "funkce funkcí" (function functions).
- x1 a x2 určují interval, na kterém se hledá optimální bod.
- opt obsahuje parametry optimalizace. Tuto proměnnou lze nastavit pomocí příkazu optimset.
- fminbnd předpokládá, že funkce1 je na intervalu spojitá
- Chceme-li najít maximum, stačí u optimalizované funkce otočit znaménko.
- Má-li funkce parametry a1, a2,...

```
>> fminbnd(@(x)funkce1(x,a1,a2),x1,x2,opt)
```

Minimalizace funkce jedné proměnné: `fminbnd`

- Použití příkazu `fminbnd`

```
>> fminbnd(@funkce1,x1,x2,opt)
```

- `@funkce1` vrací odkaz (handle) na funkci `funkce1` (viz `help function_handle`). Příkaz `fminbnd` pomocí tohoto odkazu volá `funkce1`. Proto se také příkazům jako je `fminbnd` říká “funkce funkcí” (function functions).
- `x1` a `x2` určují interval, na kterém se hledá optimální bod.
- `opt` obsahuje parametry optimalizace. Tuto proměnnou lze nastavit pomocí příkazu `optimset`.
- `fminbnd` předpokládá, že `funkce1` je na intervalu spojitá
- Chceme-li najít maximum, stačí u optimalizované funkce otočit znaménko.
- Má-li funkce parametry `a1`, `a2`,. . .

```
>> fminbnd(@(x)funkce1(x,a1,a2),x1,x2,opt)
```

Minimalizace funkce jedné proměnné: `fminbnd`

- Použití příkazu `fminbnd`

```
>> fminbnd(@funkce1,x1,x2,opt)
```

- `@funkce1` vrací odkaz (handle) na funkci `funkce1` (viz `help function_handle`). Příkaz `fminbnd` pomocí tohoto odkazu volá `funkce1`. Proto se také příkazům jako je `fminbnd` říká "funkce funkcí" (function functions).
- `x1` a `x2` určují interval, na kterém se hledá optimální bod.
- `opt` obsahuje parametry optimalizace. Tuto proměnnou lze nastavit pomocí příkazu `optimset`.
- `fminbnd` předpokládá, že `funkce1` je na intervalu spojitá
- Chceme-li najít maximum, stačí u optimalizované funkce otočit znaménko.
- Má-li funkce parametry `a1`, `a2`,...

```
>> fminbnd(@(x)funkce1(x,a1,a2),x1,x2,opt)
```

Minimalizace funkce jedné proměnné: `fminbnd`

- Použití příkazu `fminbnd`

```
>> fminbnd(@funkce1,x1,x2,opt)
```

- `@funkce1` vrací odkaz (handle) na funkci `funkce1` (viz `help function_handle`). Příkaz `fminbnd` pomocí tohoto odkazu volá `funkce1`. Proto se také příkazům jako je `fminbnd` říká "funkce funkcí" (function functions).
- `x1` a `x2` určují interval, na kterém se hledá optimální bod.
- `opt` obsahuje parametry optimalizace. Tuto proměnnou lze nastavit pomocí příkazu `optimset`.
- `fminbnd` předpokládá, že `funkce1` je na intervalu spojitá
- Chceme-li najít maximum, stačí u optimalizované funkce otočit znaménko.
- Má-li funkce parametry `a1`, `a2`,...

```
>> fminbnd(@(x)funkce1(x,a1,a2),x1,x2,opt)
```

Příklad: hledání minima funkce $\sin(x)$

- Vytvoříme funkci, kterou chceme optimalizovat.

```
function y=mojefunkce(x)
% vstup i výstup musí být skalár
y=sin(x);
```

- Hledáme minimum na intervalu $[0, \pi/2]$ pomocí příkazu `fminbnd`

```
>> fminbnd(@mojefunkce,0,pi/2)
ans =

     6.4177e-005
```

- Chceme-li např. nastavit vyšší přesnost

```
>> op=optimset('tolx',1e-8);
>> fminbnd(@mojefunkce,0,10,op)
ans =

  3.5313e-009
```

Příklad: hledání minima funkce $\sin(x)$

- Vytvoříme funkci, kterou chceme optimalizovat.

```
function y=mojefunkce(x)
% vstup i výstup musí být skalár
y=sin(x);
```

- Hledáme minimum na intervalu $[0, \pi/2]$ pomocí příkazu `fminbnd`

```
>> fminbnd(@mojefunkce,0,pi/2)
ans =

     6.4177e-005
```

- Chceme-li např. nastavit vyšší přesnost

```
>> op=optimset('tolx',1e-8);
>> fminbnd(@mojefunkce,0,10,op)
ans =

   3.5313e-009
```

Příklad: hledání minima funkce $\sin(x)$

- Vytvoříme funkci, kterou chceme optimalizovat.

```
function y=mojefunkce(x)
% vstup i výstup musí být skalár
y=sin(x);
```

- Hledáme minimum na intervalu $[0, \pi/2]$ pomocí příkazu `fminbnd`

```
>> fminbnd(@mojefunkce,0,pi/2)
ans =

     6.4177e-005
```

- Chceme-li např. nastavit vyšší přesnost

```
>> op=optimset('tolx',1e-8);
>> fminbnd(@mojefunkce,0,10,op)
ans =

  3.5313e-009
```

Příklad: hledání minima funkce $\sin(x)$

- Výsledek pochopitelně závisí na intervalu

```
>> fminbnd(@funkcef,pi,2*pi,op)

ans =

    4.7124
```

Příklad: $f(x) = e^{-2x} + x - 3$

- Rychlá implementace funkce pomocí anonymní funkce (dříve `inline`):

```
>> mojefunkce= @(x) exp(-2*x)+x-3
mojefunkce =

    function_handle with value:

    @(x)exp(-2*x)+x-3
```

- Použití `fminbnd`

```
>> fminbnd(mojefunkce,0,2)

ans =

   0.346580442393996
```

Příklad: $f(x) = e^{-2x} + x - 3$

- Rychlá implementace funkce pomocí anonymní funkce (dříve `inline`):

```
>> mojefunkce= @(x) exp(-2*x)+x-3
mojefunkce =

    function_handle with value:

    @(x)exp(-2*x)+x-3
```

- Použití `fminbnd`

```
>> fminbnd(mojefunkce,0,2)

ans =

   0.346580442393996
```

Příklad: polynom s extrémy v bodech 1, 2 a 5

- Hledáme polynom, jehož derivace bude v bodech 1, 2 a 5 nulová (bude tam extrém). Použijeme Symbolic Toolbox

```
>> syms x
>> int((x-1)*(x-2)*(x-5))

ans =

x^4/4 - (8*x^3)/3 + (17*x^2)/2 - 10*x
```

- Definujeme funkci

```
function y=mojefunkce(x)
y=x.^4/4 - (8*x.^3)/3 + (17*x.^2)/2 - 10*x;
```

Příklad: polynom s extrémy v bodech 1, 2 a 5

- Hledáme polynom, jehož derivace bude v bodech 1, 2 a 5 nulová (bude tam extrém). Použijeme Symbolic Toolbox

```
>> syms x
>> int((x-1)*(x-2)*(x-5))

ans =

x^4/4 - (8*x^3)/3 + (17*x^2)/2 - 10*x
```

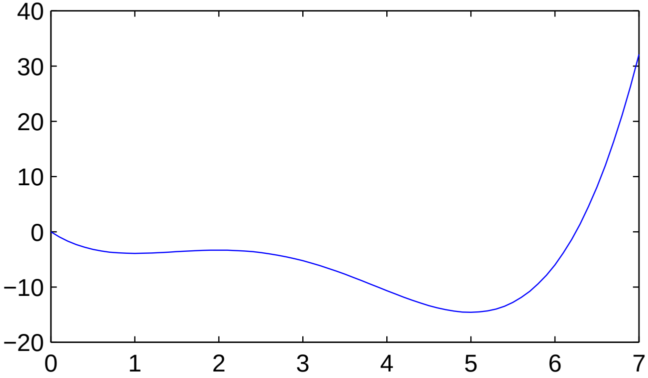
- Definujeme funkci

```
function y=mojefunkce(x)
y=x.^4/4 - (8*x.^3)/3 + (17*x.^2)/2 - 10*x;
```

Příklad: polynom s extrémy v bodech 1, 2 a 5

- Vykreslíme průběh

```
plot(0:0.1:7,mojefunkce(0:0.1:7))
```



Příklad: polynom s extrémy v bodech 1, 2 a 5

- Optimalizace je závislá na inicializaci (odvozená z intervalu)

```
>> op=optimset('tolx',1e-2);
>> fminbnd(@funkcef,0,2,op)
ans =
    0.9998
>> fminbnd(@funkcef,0,3,op)
ans =
    1.0011
>> fminbnd(@funkcef,1,10,op)
ans =
    4.9992
>> fminbnd(@funkcef,-50,5,op)
ans =
    1.0003
```

Hledání nulového bodu $f(x) = 0$

- Příkaz `fzero`

```
>> x=fzero(@funkce4,x0)
```

- `fzero` hledá bod, kde funkce mění znaménko.
- Funkce musí být spojitá
- Pokud se funkce nuly pouze dotýká (např. $f(x) = x^2$), `fzero` nelze použít.

Hledání nulového bodu $f(x) = 0$

- Příkaz `fzero`

```
>> x=fzero(@funkce4,x0)
```

- `fzero` hledá bod, kde funkce mění znaménko.
- Funkce musí být spojitá
- Pokud se funkce nuly pouze dotýká (např. $f(x) = x^2$), `fzero` nelze použít.

Hledání nulového bodu $f(x) = 0$

- Příkaz `fzero`

```
>> x=fzero(@funkce4,x0)
```

- `fzero` hledá bod, kde funkce mění znaménko.
- Funkce musí být spojitá
- Pokud se funkce nuly pouze dotýká (např. $f(x) = x^2$), `fzero` nelze použít.

Hledání nulového bodu $f(x) = 0$

- Příkaz `fzero`

```
>> x=fzero(@funkce4,x0)
```

- `fzero` hledá bod, kde funkce mění znaménko.
- Funkce musí být spojitá
- Pokud se funkce nuly pouze dotýká (např. $f(x) = x^2$), `fzero` nelze použít.

Část II

Funkce více proměnných

Minimalizace funkce více proměnných

- Použití příkazu `fminsearch`

```
>> fminsearch(@funkce2,x0,opt)
```

- `funkce2` má jednu proměnnou (ostatní jsou parametry), která je vektor, jehož složky jsou jednotlivé proměnné.
- `x0` je inicializace
- `fminsearch` provádí neomezenou optimalizaci (neomezený interval)
- `fminsearch` nepoužívá derivaci
- Příklad funkce $f(x, y) = \cos(x)\sin(y)$

```
function y=funkce2(x)
y=cos(x(1))*sin(x(2));
```

hledáme minimum s inicializací v bodě $[0, 0]$

```
fminsearch(@funkce2,[0,0])
ans =

    0.0000   -1.5708
```

Minimalizace funkce více proměnných

- Použití příkazu `fminsearch`

```
>> fminsearch(@funkce2,x0,opt)
```

- `funkce2` má jednu proměnnou (ostatní jsou parametry), která je vektor, jehož složky jsou jednotlivé proměnné.
- `x0` je inicializace
- `fminsearch` provádí neomezenou optimalizaci (neomezený interval)
- `fminsearch` nepoužívá derivaci
- Příklad funkce $f(x, y) = \cos(x)\sin(y)$

```
function y=funkce2(x)
y=cos(x(1))*sin(x(2));
```

hledáme minimum s inicializací v bodě $[0, 0]$

```
fminsearch(@funkce2,[0,0])
ans =

      0.0000   -1.5708
```

Minimalizace funkce více proměnných

- Použití příkazu `fminsearch`

```
>> fminsearch(@funkce2,x0,opt)
```

- `funkce2` má jednu proměnnou (ostatní jsou parametry), která je vektor, jehož složky jsou jednotlivé proměnné.
- `x0` je inicializace
- `fminsearch` provádí neomezenou optimalizaci (neomezený interval)
- `fminsearch` nepoužívá derivaci
- Příklad funkce $f(x, y) = \cos(x)\sin(y)$

```
function y=funkce2(x)
y=cos(x(1))*sin(x(2));
```

hledáme minimum s inicializací v bodě $[0, 0]$

```
fminsearch(@funkce2,[0,0])
ans =

     0.0000    -1.5708
```

Minimalizace funkce více proměnných

- Použití příkazu `fminsearch`

```
>> fminsearch(@funkce2,x0,opt)
```

- `funkce2` má jednu proměnnou (ostatní jsou parametry), která je vektor, jehož složky jsou jednotlivé proměnné.
- `x0` je inicializace
- `fminsearch` provádí neomezenou optimalizaci (neomezený interval)
- `fminsearch` nepoužívá derivaci
- Příklad funkce $f(x, y) = \cos(x)\sin(y)$

```
function y=funkce2(x)
y=cos(x(1))*sin(x(2));
```

hledáme minimum s inicializací v bodě $[0, 0]$

```
fminsearch(@funkce2,[0,0])
ans =

    0.0000   -1.5708
```

Minimalizace funkce více proměnných

- Použití příkazu `fminsearch`

```
>> fminsearch(@funkce2,x0,opt)
```

- `funkce2` má jednu proměnnou (ostatní jsou parametry), která je vektor, jehož složky jsou jednotlivé proměnné.
- `x0` je inicializace
- `fminsearch` provádí neomezenou optimalizaci (neomezený interval)
- `fminsearch` nepoužívá derivaci
- Příklad funkce $f(x,y) = \cos(x)\sin(y)$

```
function y=funkce2(x)
y=cos(x(1))*sin(x(2));
```

hledáme minimum s inicializací v bodě $[0,0]$

```
fminsearch(@funkce2,[0,0])
ans =

     0.0000    -1.5708
```

Minimalizace funkce více proměnných

- Použití příkazu `fminsearch`

```
>> fminsearch(@funkce2,x0,opt)
```

- `funkce2` má jednu proměnnou (ostatní jsou parametry), která je vektor, jehož složky jsou jednotlivé proměnné.
- `x0` je inicializace
- `fminsearch` provádí neomezenou optimalizaci (neomezený interval)
- `fminsearch` nepoužívá derivaci
- Příklad funkce $f(x, y) = \cos(x)\sin(y)$

```
function y=funkce2(x)
y=cos(x(1))*sin(x(2));
```

hledáme minimum s inicializací v bodě $[0, 0]$

```
fminsearch(@funkce2,[0,0])
ans =

    0.0000   -1.5708
```

Minimalizace funkce více proměnných

- Použití příkazu `fminunc` - umí využívat gradient funkce.
- Gradient předáváme v druhém výstupním parametru funkce.
- Příklad

```
function [y g]=funkce2(x)
y=cos(x(1))*sin(x(2));

% parc. derivace podle x(1)
g(1)=-sin(x(1))*sin(x(2));

% parc. derivace podle x(2)
g(2)=cos(x(1))*cos(x(2));
```

- Typické použití

```
>> op=optimset('GradObj','on');
>> fminunc(@funkce2,[0,0],op)
Local minimum found.
```

Minimalizace funkce více proměnných

- Použití příkazu `fminunc` - umí využívat gradient funkce.
- Gradient předáváme v druhém výstupním parametru funkce.
- Příklad

```
function [y g]=funkce2(x)
y=cos(x(1))*sin(x(2));

% parc. derivace podle x(1)
g(1)=-sin(x(1))*sin(x(2));

% parc. derivace podle x(2)
g(2)=cos(x(1))*cos(x(2));
```

- Typické použití

```
>> op=optimset('GradObj','on');
>> fminunc(@funkce2,[0,0],op)
Local minimum found.
```

Minimalizace funkce více proměnných

- Použití příkazu `fminunc` - umí využívat gradient funkce.
- Gradient předáváme v druhém výstupním parametru funkce.
- Příklad

```
function [y g]=funkce2(x)
y=cos(x(1))*sin(x(2));

% parc. derivace podle x(1)
g(1)=-sin(x(1))*sin(x(2));

% parc. derivace podle x(2)
g(2)=cos(x(1))*cos(x(2));
```

- Typické použití

```
>> op=optimset('GradObj','on');
>> fminunc(@funkce2,[0,0],op)
Local minimum found.
```

Minimalizace funkce více proměnných

- Použití příkazu `fminunc` - umí využívat gradient funkce.
- Gradient předáváme v druhém výstupním parametru funkce.
- Příklad

```
function [y g]=funkce2(x)
y=cos(x(1))*sin(x(2));

% parc. derivace podle x(1)
g(1)=-sin(x(1))*sin(x(2));

% parc. derivace podle x(2)
g(2)=cos(x(1))*cos(x(2));
```

- Typické použití

```
>> op=optimset('GradObj','on');
>> fminunc(@funkce2,[0,0],op)
Local minimum found.
```

Minimalizace funkce více proměnných s omezením

- Omezení může být lineární nebo nelineární vazbou proměnných (přímka, rovina, kruh, …) tzv. varietou a definičním oborem (např. interval)
- Příkaz `fmincon`

```
>> fmincon(@fun,x0,A,b,Aeq,beq,lb,ub,@nonlcon,op)
```

  řeší úlohu

$$\min_x fun(x)$$

  za podmínek

$$Ax \leq b$$
$$A_{eq}x = b_{eq}$$
$$lb \leq x \leq ub$$
$$C(x) \leq 0$$
$$C_{eq}(x) = 0$$

Minimalizace funkce více proměnných s omezením

- Omezení může být lineární nebo nelineární vazbou proměnných (přímka, rovina, kruh, . . .) tzv. varietou a definičním oborem (např. interval)
- Příkaz `fmincon`

```
>> fmincon(@fun,x0,A,b,Aeq,beq,lb,ub,@nonlcon,op)
```

řeší úlohu

$$\min_x fun(x)$$

za podmínek

$$Ax \leq b$$
$$A_{eq}x = b_{eq}$$
$$lb \leq x \leq ub$$
$$C(x) \leq 0$$
$$C_{eq}(x) = 0$$

Minimalizace funkce více proměnných s omezením

- Nechceme-li některé omezení, zadáváme [ ].
- Příklad

$$\min_{x,y,z} -\sin(x)\sin(y)\sin(z)$$

za podmínky

$$\underbrace{0 \leq x + 2y + 2z \leq 72}_{\text{dvě lineární nerovnosti}}$$

- Podmínky přepíšeme do tvaru $Ax \leq b$

$$x + 2y + 2z \leq 72$$
$$-x - 2y - 2z \leq 0$$

$$A = \begin{bmatrix} 1 & 2 & 2 \\ -1 & -2 & -2 \end{bmatrix} \qquad b = \begin{bmatrix} 72 \\ 0 \end{bmatrix}$$

Minimalizace funkce více proměnných s omezením

- Nechceme-li některé omezení, zadáváme [ ].
- Příklad

$$\min_{x,y,z} -\sin(x)\sin(y)\sin(z)$$

za podmínky

$$\underbrace{0 \leq x + 2y + 2z \leq 72}_{\text{dvě lineární nerovnosti}}$$

- Podmínky přepíšeme do tvaru $Ax \leq b$

$$x + 2y + 2z \leq 72$$
$$-x - 2y - 2z \leq 0$$

$$A = \begin{bmatrix} 1 & 2 & 2 \\ -1 & -2 & -2 \end{bmatrix} \qquad b = \begin{bmatrix} 72 \\ 0 \end{bmatrix}$$

Minimalizace funkce více proměnných s omezením

- Nechceme-li některé omezení, zadáváme [ ].
- Příklad

$$\min_{x,y,z} -\sin(x)\sin(y)\sin(z)$$

za podmínky

$$\underbrace{0 \leq x + 2y + 2z \leq 72}_{\text{dvě lineární nerovnosti}}$$

- Podmínky přepíšeme do tvaru $Ax \leq b$

$$x + 2y + 2z \leq 72$$
$$-x - 2y - 2z \leq 0$$

$$A = \begin{bmatrix} 1 & 2 & 2 \\ -1 & -2 & -2 \end{bmatrix} \qquad b = \begin{bmatrix} 72 \\ 0 \end{bmatrix}$$

Minimalizace funkce více proměnných s omezením

- Funkce

```
function y=funkce3(x)
y=-sin(x(1))*sin(x(2))*sin(x(3));
% nebo
y=-prod(sin(x));
```

- Použití fmincon

```
>> A=[1 2 2;-1 -2 -2];
>> b=[72;0];
>> fmincon(@funkce3,[1;1;1],A,b)
ans =

    1.5708
    1.5708
    1.5708
```

Minimalizace funkce více proměnných s omezením

- Funkce

```
function y=funkce3(x)
y=-sin(x(1))*sin(x(2))*sin(x(3));
% nebo
y=-prod(sin(x));
```

- Použítí `fmincon`

```
>> A=[1 2 2;-1 -2 -2];
>> b=[72;0];
>> fmincon(@funkce3,[1;1;1],A,b)
ans =

    1.5708
    1.5708
    1.5708
```

Část III

Lineární programování

Řešení úlohy lineárního programování

- Hledáme minimum lineární funkce (více proměnných)

$$\min_x f_1 x_1 + f_2 x_2 + \cdots + f_n x_n = \min_x f^T x$$

kde

$$f = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

za podmínek

$$Ax \leq b$$
$$A_{eq}x = b_{eq}$$
$$lb \leq x \leq ub$$

```
>> linprog(f,A,b,Aeq,beq,lb,ub,x0,opt)
```

Příklad úlohy lineárního programování: maximalizace zisku v obchodu s potravinami

potravina	ks	cena [Kč]	min. ks	max. ks	zisk z 1ks [Kč]
houska	x_1	2	100	200	0.2
chléb	x_2	10	20	50	1
rohlík	x_3	5	20	50	0.5

Peníze na nákup zboží: 1000 Kč

Úloha:

$$\min -(0.2x_1 + x_2 + 0.5x_3)$$

$$2x_1 + 10x_2 + 5x_3 \leq 1000$$

$$100 \leq x_1 \leq 200$$

$$20 \leq x_2 \leq 50$$

$$20 \leq x_3 \leq 50$$

Příklad úlohy lineárního programování: maximalizace zisku v obchodu s potravinami

```
>> A=[2 10 5];
>> b=1000;
>> lb=[100 20 20];
>> ub=[200 50 50];
>> f=-[0.2 1 0.5];
>> linprog(f,A,b,[],[],lb,ub)
Optimization terminated.

ans =

  176.9449
   46.8141
   35.5939
```

Část IV

Kvadratické programování

Úloha kvadratického programování

Hledáme minimum kvadratické funkce (více proměnných)

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{f}^T\mathbf{x}$$

kde

$$\mathbf{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

a **H** je symetrická za podmínek

$$Ax \leq b$$
$$A_{eq}x = b_{eq}$$
$$lb \leq x \leq ub$$

```
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,opt)
```

Příklad úlohy kvadratického programování

- Minimalizujeme funkci

$$f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2$$

vzhledem k podmínce

$$\begin{aligned} x_1 + x_2 &\le 2 \\ -x_1 + 2x_2 &\le 2 \\ 2x_1 + x_2 &\le 3 \\ 0 \le x_1, 0 &\le x_2 \end{aligned}$$

- Zápis pomocí matic a vektorů

$$\mathbf{H} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}, \mathbf{f} = \begin{pmatrix} -2 \\ -6 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 1 & 1 \\ -1 & 2 \\ 2 & 1 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix}$$

Příklad úlohy kvadratického programování

- Minimalizujeme funkci

$$f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1 x_2 - 2x_1 - 6x_2$$

vzhledem k podmínce

$$\begin{aligned} x_1 + x_2 &\leq 2 \\ -x_1 + 2x_2 &\leq 2 \\ 2x_1 + x_2 &\leq 3 \\ 0 \leq x_1, 0 &\leq x_2 \end{aligned}$$

- Zápis pomocí matic a vektorů

$$\mathbf{H} = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}, \mathbf{f} = \begin{pmatrix} -2 \\ -6 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 1 & 1 \\ -1 & 2 \\ 2 & 1 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 2 \\ 2 \\ 3 \end{pmatrix}$$

Příklad úlohy kvadratického programování

```
>> H = [1 -1; -1 2];
>> f = [-2; -6];
>> A = [1 1; -1 2; 2 1];
>> b = [2; 2; 3];
>> lb = zeros(2,1);
>> x = quadprog(H,f,A,b,[],[],lb);
Warning: Large-scale algorithm does not currently solve this pro
formulation,
using medium-scale algorithm instead.
> In quadprog at 291
Optimization terminated.
>> x
x =
    0.6667
    1.3333
```

Část V

Paralelní výpočty pomocí cyklu `parfor`

Cyklus `parfor`

- Příklad použití:

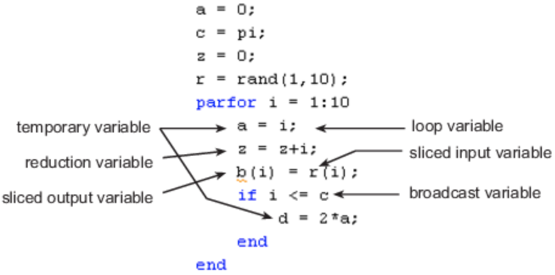
```
parfor i = 1:10000
% tělo cyklu
end
```

- Při první běhu je na standardním výpočetním klastru aktivován "Parallel Pool" s přednastaveným počtem "Workerů."
- Tělo cyklu je prováděno paralelně na Workerech, přičemž pořadí cyklů není předem určené a může být ovlivněno mnoha faktory.
- Neplatí, že kterýkoliv cyklus `for` lze nahradit cyklem `parfor`!

Cyklus `parfor`

- V cyklech `parfor` jsou proměnné klasifikovány na 5 tříd: Loop, Sliced, Broadcast, Reduction, Temporary.
- Pokud nelze některou z proměnných klasifikovat, je to chyba.
- Matlab se klasifikací proměnných snaží zoptimalizovat paralelizaci a předejít chybám programátora.

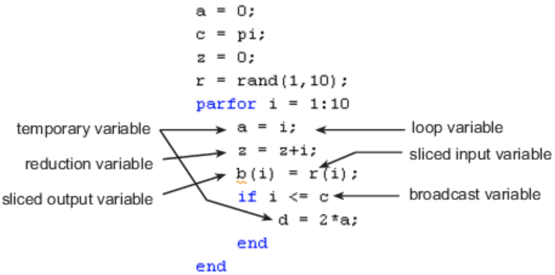
Klasifikace proměnných v cyklech `parfor`



Loop Indexující proměnná cyklu. Přiřazení nové hodnoty této proměnné uvnitř cyklu není možné a nelze v ní indexovat.

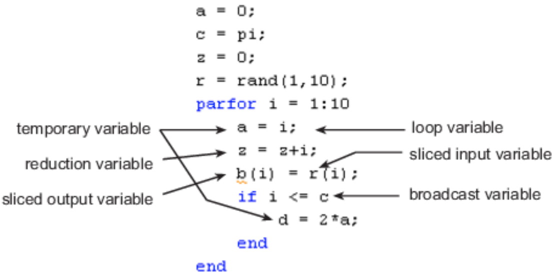
Temporary Proměnná, která je přiřazena (nebo vytvořena) uvnitř cyklu a současně není třídy Reduction. Proměnná je vždy na začátku cyklu smazána.

Klasifikace proměnných v cyklech `parfor`



Reduction Akumuluje hodnoty závislé na všech iteracích avšak nezávisle na pořadí iterací. Matlab rozezná redukční proměnnou pouze pokud se vyskytuje ve výrazu určitého tabulkového typu (viz help).

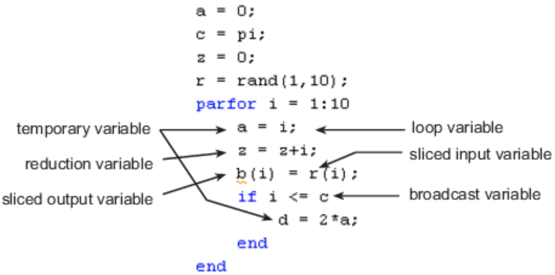
Klasifikace proměnných v cyklech `parfor`



Příklad proměnné typu Reduction:

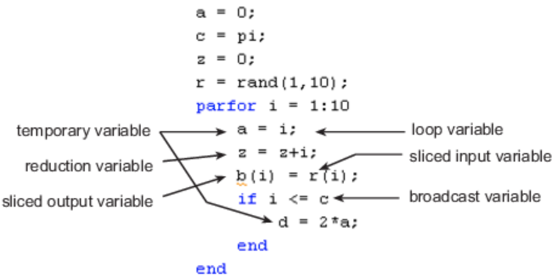
```
X = ...;
parfor i = 1:n
    X = X + d(i);
end
```

Klasifikace proměnných v cyklech `parfor`



Broadcast Proměnná definovaná před cyklem, jejíž hodnota je použitá uvnitř cyklu ale nikdy není v cyklu přiřazena. Hodnota této proměnné je na začatku cyklu poslána všem Workerům.

Klasifikace proměnných v cyklech `parfor`



Sliced Pole, jehož pouze některé části jsou použity v jednotlivých iteracích. Může být “rozděleno” a pouze použité části jsou rozesílány mezi klientem a Workerem. Pole smí být indexované vždy jen stejným výrazem (zjednodušeně řečeno). Rozměry pole se nesmí změnit v průběhu cyklu a proto nejsou tolerována přiřazení obsahující [] nebo ’ ’.

Tento materiál vznikl v rámci projektu ESF CZ.1.07/2.2.00/28.0050
Modernizace didaktických metod a inovace výuky technických předmětů,
který je spolufinancován Evropským sociálním fondem a státním rozpočtem ČR.