Programy pro řešení úlohy lineárního programování

23. dubna 2012

Přehled

- Mathematica
- Sage
- AMPL
- GNU Linear Programming Kit (GLPK)

Mathematica

Mathematika je program pro numerické a symbolické počítání.

- maticové počítání
- kreslení 2D i 3D funkcí
- řešení úloh lineárního programování(i jiných optimalizačních úloh), diofantických rovnic
- integrace
- ...

Mathematica

Mathematika je program pro numerické a symbolické počítání.

- maticové počítání
- kreslení 2D i 3D funkcí
- řešení úloh lineárního programování(i jiných optimalizačních úloh), diofantických rovnic
- integrace
- ...

```
Minimize[{x+2y,
    −5x+y=7 && x+y>=26 && x>=3 && y>=4},
    {x, y}]
```

Na prvním řádku je cílová funkce, na druhém jsou podmínky a na třetím jsou proměnné, které se mají minimalizovat.

```
LinearProgramming[{1, 2},
    {{-5, 1}, {1, 1}},
    {{7, 0}, {26, 1}},
    {{3, Infinity}, {4, Infinity}}]
```

Na prvním řádku je vektor cílové funkce, na druhém matice je “A” a na čtvrtém je dolní a horní omezení na každou proměnnou.
Na třetím řádku je pro každou podmínku dvojice čísel, kde první je pravá strana podmínky a druhé udává, zda se jedná o rovnost (0), alespoň (1) nebo nejvýše (-1).

Alternativy

Instalační CD programu Mathematica je v knihovně zdarma pro studijní účely. Podrobnosti a dokumentace na adrese `http://www.wolfram.com/products/mathematica/index.html`

Alternativy

Instalační CD programu Mathematica je v knihovně zdarma pro studijní účely. Podrobnosti a dokumentace na adrese `http://www.wolfram.com/products/mathematica/index.html`

Další podobné programy: Maple, Matlab, Octave, ...

Alternativy

Instalační CD programu Mathematica je v knihovně zdarma pro studijní účely. Podrobnosti a dokumentace na adrese `http://www.wolfram.com/products/mathematica/index.html`

Další podobné programy: Maple, Matlab, Octave, ...

Opensource varianta k programu Mathematica je například Sage, jehož syntaxe je založena na jazyku Python.

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`
- Pojmenování vektoru proměnných (`p` může pracovat zaráz s více vektory proměnných): `x=p.new_variable()`

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`
- Pojmenování vektoru proměnných (`p` může pracovat zaráz s více vektory proměnných): `x=p.new_variable()`
- Zadání cílové funkce:
  `p.set_objective(x[1] + x[2])`

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`
- Pojmenování vektoru proměnných (`p` může pracovat zaráz s více vektory proměnných): `x=p.new_variable()`
- Zadání cílové funkce:
  `p.set_objective(x[1] + x[2])`
- Přidání omezující podmínky:
  `p.add_constraint(-3*x[1] + 2*x[2], max=10)`

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`
- Pojmenování vektoru proměnných (`p` může pracovat zaráz s více vektory proměnných): `x=p.new_variable()`
- Zadání cílové funkce:
  `p.set_objective(x[1] + x[2])`
- Přidání omezující podmínky:
  `p.add_constraint(-3*x[1] + 2*x[2], max=10)`
- Zadání rozsahu proměnné:
  `p.set_min(x[1],None)#implicitne 0`

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`
- Pojmenování vektoru proměnných (`p` může pracovat zaráz s více vektory proměnných): `x=p.new_variable()`
- Zadání cílové funkce:
  `p.set_objective(x[1] + x[2])`
- Přidání omezující podmínky:
  `p.add_constraint(-3*x[1] + 2*x[2], max=10)`
- Zadání rozsahu proměnné:
  `p.set_min(x[1],None)#implicitne 0`
  `p.set_max(x[2],3)#implicitne None...+oo`

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`
- Pojmenování vektoru proměnných (`p` může pracovat zaráz s více vektory proměnných): `x=p.new_variable()`
- Zadání cílové funkce:
  `p.set_objective(x[1] + x[2])`
- Přidání omezující podmínky:
  `p.add_constraint(-3*x[1] + 2*x[2], max=10)`
- Zadání rozsahu proměnné:
  `p.set_min(x[1],None)#implicitne 0`
  `p.set_max(x[2],3)#implicitne None...+oo`
- Vyřešení úlohy: `optimum=p.solve(solver="GLPK")`
  `#implicitne solver="Coin"`

Sage – třída MixedIntegerLinearProgram

- Vytvoření instance:
  `p=MixedIntegerLinearProgram(maximization=True)`
- Pojmenování vektoru proměnných (`p` může pracovat zaráz s více vektory proměnných): `x=p.new_variable()`
- Zadání cílové funkce:
  `p.set_objective(x[1] + x[2])`
- Přidání omezující podmínky:
  `p.add_constraint(-3*x[1] + 2*x[2], max=10)`
- Zadání rozsahu proměnné:
  `p.set_min(x[1],None)#implicitne 0`
  `p.set_max(x[2],3)#implicitne None...+oo`
- Vyřešení úlohy: `optimum=p.solve(solver="GLPK")`
  `#implicitne solver="Coin"`
- Získání hodnoty `x1=p.get_values(x[1])`

Sage – jednoduchý příklad

Lineární program:

$$\text{Maximalizuj: } 2x_1 + x_2 \tag{1}$$

$$\text{Za podmínek: } 3x_1 + 4x_2 \leq 2.5 \tag{2}$$

$$0.5 \leq 1.5x_1 + 0.5x_2 \leq 4 \tag{3}$$

$$x_1, x_2 \geq 0 \tag{4}$$

Sage kód:

```
p=MixedIntegerLinearProgram(maximization=True)
x=p.new_variable()
p.set_objective(2*x[1]+x[2])
p.add_constraint(3*x[1]+4*x[2],max=2.5)
p.add_constraint(1.5*x[1]+0.5*x[2],max=4,min=0.5)
print str(p.solve())+":"+str(p.get_values(x))
```

Vypíše:

```
1.66666666667:{1: 0.83333333333333326, 2: 0.0}
```

Odkazy

Web Sage (+webový klient):

`http://www.sagemath.org/`

`http://www.sagenb.org/`

Dokumentace (průvodce k (celočíselnému) LP + reference k MixedIntegerLinearProgram):

`http://www.sagemath.org/doc/thematic_tutorials/linear_programming.html`

`http://www.sagemath.org/doc/reference/sage/numerical/mip.html`

AMPL

AMPL je modelovací jazyk pro matematické programování.
Možnosti AMPL sahají až za lineární programování.

Příklad

Továrna produkuje dva druhy barev - modrou a zlatou. Je možné vyrobit 40 litrů modré barvy za hodinu nebo 30 litrů zlaté barvy. Modrá barva se prodává za 10 Kč/litr, zlatá 15 Kč/litr. Není možné prodat více než 1000 litrů modré barvy a 860 litrů zlaté barvy za týden. Pracovní týden má 40 hodin.

Příklad

Továrna produkuje dva druhy barev - modrou a zlatou. Je možné vyrobit 40 litrů modré barvy za hodinu nebo 30 litrů zlaté barvy. Modrá barva se prodává za 10 Kč/litr, zlatá 15 Kč/litr. Není možné prodat více než 1000 litrů modré barvy a 860 litrů zlaté barvy za týden. Pracovní týden má 40 hodin.

$$\max 10m + 15z$$
$$\frac{1}{40}m + \frac{1}{30}z \leq 40$$
$$0 \leq m \leq 1000$$
$$0 \leq z \leq 860$$

AMPL program

Vstup:

```
var m;
var z;

maximize profit: 10*m + 15*z;
subject to time: (1/40)*m + (1/30)*z <= 40;
subject to m_limit: 0 <= m <= 1000;
subject to z_limit: 0 <= z <= 860;
```

AMPL program

Vstup:

```
var m;
var z;

maximize profit: 10*m + 15*z;
subject to time: (1/40)*m + (1/30)*z <= 40;
subject to m_limit: 0 <= m <= 1000;
subject to z_limit: 0 <= z <= 860;
```

Výstup:

```
MINOS 5.51: optimal solution found.
2 iterations, objective 17433.33333
:       _varname          _var       :=
1    ''paint['blue']''    453.333
2    ''paint['gold']''    860
```

shrnutí

- Klíčové slovo **var** uvozuje proměnné
- Vše musí být pojmenované(poměnné, cílová funkce, pomínky), jména musejí být jedinečná
- Vše končí středníkem
- AMPL je case-sensitive
- Lze mít program i data v jednom souboru(jako výše), ale je také možno tyto části oddělit(program *.mod, data *.dat). Webové řešiče chtějí oddělené soubory.

Složitější problém

n počet barev

t pracovních hodin

p_i zisk z prodeje jednoho litru i-té barvy

r_i rychlost výroby i-té barvy(v litrech za hodinu)

m_i maximální množství vyrobené(prodané) i-té barvy

Odpovídající program

```
param n;
param t;
param p{i in 1..n};
param r{i in 1..n};
param m{i in 1..n};

var paint{i in 1..n}; #n barev

maximize profit: sum{i in 1..n}
            p[i]*paint[i];
subject to time: sum{i in 1..n}
            (1/r[i])*paint[i] <= t;
subject to capacity{i in 1..n}:
            0 <= paint[i] <= m[i];
```

Data programu

```
paran n:= 2;
param t:= 40;

param p:= 1 10 2 15;
param r:= 1 40 2 50;
param m:= 1 1000 2 860;
```

Data programu

```
paran n:= 2;
param t:= 40;

param p:= 1 10 2 15;
param r:= 1 40 2 50;
param m:= 1 1000 2 860;

# parametry lze zadávat chytřeji
param: p r m:=
1 10 40 1000
2 15 30 860;
```

Další možnosti

Množina (set P) - namísto i in 1..n lze psát i in P

Další možnosti

Množina (set P) - namísto i in 1..n lze psát i in P

```
set P;
param t;
param p{i in P};
param r{i in P};
param m{i in P};
var paint{i in P};

maximize profit: sum{i in P}
          p[i]*paint[i];
subject to time: sum{i in P}
          (1/r[i])*paint[i] <= t;
subject to capacity{i in P}:
          0 <= paint[i] <= m[i];
```

Další možnosti

Množina (set P) - namísto i in 1..n lze psát i in P

```
set P;
param t;
param p{i in P};
param r{i in P};
param m{i in P};
var paint{i in P};

maximize profit: sum{i in P}
          p[i]*paint[i];
subject to time: sum{i in P}
          (1/r[i])*paint[i] <= t;
subject to capacity{i in P}:
          0 <= paint[i] <= m[i];

set P:= blue gold;
param t:= 40;
```

Omezení na proměnné:

- nerovnostmi
- celočíselné - var x integer
- binární - var y binary

AMPL prakticky

Zkuste si AMPL na webu:
`http://www.ampl.com/TRYAMPL/startup.html`
nebo lokálně (GNU řešení):

```
glpsol --mps
```

Další informace o AMPL:
`http://www.ampl.com/REFS/amplmod.pdf`

API pro jazyk C

GNU Linear Programming Kit (GLPK) je knihovna napsaná v jazyce ANSI C, která umí řešit úlohy lineárního a celočíselného lineárního programování.

API pro jazyk C

GNU Linear Programming Kit (GLPK) je knihovna napsaná v jazyce ANSI C, která umí řešit úlohy lineárního a celočíselného lineárního programování.
K řešení úlohy lineárního programování používá simplexovou metodu a metody vnitřního bodu a pro celočíselne LP používá branch-and-bound a Gomory cutting-plane.

API pro jazyk C

GNU Linear Programming Kit (GLPK) je knihovna napsaná v jazyce ANSI C, která umí řešit úlohy lineárního a celočíselného lineárního programování.
K řešení úlohy lineárního programování používá simplexovou metodu a metody vnitřního bodu a pro celočíselne LP používá branch-and-bound a Gomory cutting-plane.

Příklad

API pro jazyk C

GNU Linear Programming Kit (GLPK) je knihovna napsaná v jazyce ANSI C, která umí řešit úlohy lineárního a celočíselného lineárního programování.
K řešení úlohy lineárního programování používá simplexovou metodu a metody vnitřního bodu a pro celočíselne LP používá branch-and-bound a Gomory cutting-plane.

Příklad

Nejprve je nutné inkludovat soubor glpk.h.

```
#include <stdio.h>
#include <stdlib.h>
#include <glpk.h>
```

Vytvoření úlohy

Vytvoříme hlavní objekt pro naši úlohu a přiřadíme ji jméno.

```
int main(void)
{ glp_prob *lp;
lp = glp_create_prob();
glp_set_prob_name(lp, "sample");
```

Vytvoření úlohy

Vytvoříme hlavní objekt pro naši úlohu a přiřadíme ji jméno.

```
int main(void)
{ glp_prob *lp;
lp = glp_create_prob();
glp_set_prob_name(lp, "sample");
```

V naší úloze budeme maximalizovat.

```
glp_set_obj_dir(lp, GLP_MAX);
```

Popis podmínek

Budeme mít tři podmínky.

```
glp_add_rows(lp, 3);
```

Popis podmínek

Budeme mít tři podmínky.

```
glp_add_rows(lp, 3);
```

První podmínce přiřadíme symbolický název (pomocnou proměnnou) “p”.

```
glp_set_row_name(lp, 1, "p");
```

Popis podmínek

Budeme mít tři podmínky.

```
glp_add_rows(lp, 3);
```

První podmínce přiřadíme symbolický název (pomocnou proměnnou) “p”.

```
glp_set_row_name(lp, 1, "p");
```

První podmínka bude $x_1 + x_2 + x_3 \leq 100$, a proto nastavíme nerovnost a pravou stranu.

```
glp_set_row_bnds(lp, 1, GLP_UP, 0.0, 100.0);
```

Popis podmínek

Budeme mít tři podmínky.

```
glp_add_rows(lp, 3);
```

První podmínce přiřadíme symbolický název (pomocnou proměnnou) “p”.

```
glp_set_row_name(lp, 1, "p");
```

První podmínka bude $x_1 + x_2 + x_3 \leq 100$, a proto nastavíme nerovnost a pravou stranu.

```
glp_set_row_bnds(lp, 1, GLP_UP, 0.0, 100.0);
```

Analogicky nastavíme druhou a třetí podmínku.

```
glp_set_row_name(lp, 2, "q");
glp_set_row_bnds(lp, 2, GLP_UP, 0.0, 600.0);
glp_set_row_name(lp, 3, "r");
glp_set_row_bnds(lp, 3, GLP_UP, 0.0, 300.0);
```

Popis proměnných — sloupců

Budeme mít tři proměnné (sloupce).

```
glp_add_cols(lp, 3);
```

Popis proměnných — sloupců

Budeme mít tři proměnné (sloupce).

```
glp_add_cols(lp, 3);
```

První proměnnou pojmenujeme x_1.

```
glp_set_col_name(lp, 1, "x1");
```

Popis proměnných — sloupců

Budeme mít tři proměnné (sloupce).

```
glp_add_cols(lp, 3);
```

První proměnnou pojmenujeme x_1.

```
glp_set_col_name(lp, 1, "x1");
```

Proměnná x_1 je omezena jen zdola a to nulou (je nezáporná).

```
glp_set_col_bnds(lp, 1, GLP_LO, 0.0, 0.0);
```

Popis proměnných — sloupců

Budeme mít tři proměnné (sloupce).

```
glp_add_cols(lp, 3);
```

První proměnnou pojmenujeme x_1.

```
glp_set_col_name(lp, 1, "x1");
```

Proměnná x_1 je omezena jen zdola a to nulou (je nezáporná).

```
glp_set_col_bnds(lp, 1, GLP_LO, 0.0, 0.0);
```

Analogicky nastavíme druhou a třetí proměnnou.

```
glp_set_col_name(lp, 2, "x2");
glp_set_col_bnds(lp, 2, GLP_LO, 0.0, 0.0);
glp_set_col_name(lp, 3, "x3");
glp_set_col_bnds(lp, 3, GLP_LO, 0.0, 0.0);
```

Cílová funkce

Cílová funkce je $10x_1 + 6x_2 + 4x_3$.

```
glp_set_obj_coef(lp, 1, 10.0);
glp_set_obj_coef(lp, 2, 6.0);
glp_set_obj_coef(lp, 3, 4.0);
```

Matice “A”

GLPK používá řídké matice, a proto pro každý nenulový prvek $A_{i,j}$ musíme do pole řádkových indexů ia uložit i, do pole sloupcových indexů ar uložit j a do pole hodnot ar uložit $A_{i,j}$.

Matice “A”

GLPK používá řídké matice, a proto pro každý nenulový prvek $A_{i,j}$ musíme do pole řádkových indexů ia uložit i, do pole sloupcových indexů ar uložit j a do pole hodnot ar uložit $A_{i,j}$.

$x_1 + x_2 + x_3 \leq 100$

$10x_1 + 4x_2 + 5x_3 \leq 600$

$2x_1 + 2x_2 + 6x_3 \leq 300$

Matice “A”

GLPK používá řídké matice, a proto pro každý nenulový prvek $A_{i,j}$ musíme do pole řádkových indexů ia uložit i, do pole sloupcových indexů ar uložit j a do pole hodnot ar uložit $A_{i,j}$.

$x_1 + x_2 + x_3 \leq 100$
$10x_1 + 4x_2 + 5x_3 \leq 600$
$2x_1 + 2x_2 + 6x_3 \leq 300$

```
int ia[10], ja[10]; double ar[10];
ia[1] = 1, ja[1] = 1, ar[1] = 1.0;
ia[2] = 1, ja[2] = 2, ar[2] = 1.0;
ia[3] = 1, ja[3] = 3, ar[3] = 1.0;
ia[4] = 2, ja[4] = 1, ar[4] = 10.0;
ia[5] = 3, ja[5] = 1, ar[5] = 2.0;
ia[6] = 2, ja[6] = 2, ar[6] = 4.0;
ia[7] = 3, ja[7] = 2, ar[7] = 2.0;
ia[8] = 2, ja[8] = 3, ar[8] = 5.0;
ia[9] = 3, ja[9] = 3, ar[9] = 6.0;
```

Načtení matice “A”

Hodnoty matice *A* jsou uloženy v polích ia, ja, ar, které jsou indexovány od 1 a máme 9 nenulových prvků matice *A*.

```
glp_load_matrix(lp, 9, ia, ja, ar);
```

Simplexová metoda

Spustíme simplexovou metodu.

```
glp_simplex(lp, NULL);
```

Simplexová metoda

Spustíme simplexovou metodu.

```
glp_simplex(lp, NULL);
```

Druhým argumentem můžou být kontrolní parametry:

- množství informací vypisovaných na terminál,
- typ simplexové metody,
- pivotovací pravidlo,
- maximální čas pro výpočet,
- ošetřování numerické nestability Gaussovské eliminace
- a další.

Simplexová metoda

Spustíme simplexovou metodu.

```
glp_simplex(lp, NULL);
```

Druhým argumentem můžou být kontrolní parametry:

- množství informací vypisovaných na terminál,
- typ simplexové metody,
- pivotovací pravidlo,
- maximální čas pro výpočet,
- ošetřování numerické nestability Gaussovské eliminace
- a další.

Návratová hodnota určuje, jak výpočet skončil.

Získání výsledků

Optimální hodnota cílové funkce.

```
double z = glp_get_obj_val(lp);
```

Získání výsledků

Optimální hodnota cílové funkce.

```
double z = glp_get_obj_val(lp);
```

Optimální vrchol.

```
double x1 = glp_get_col_prim(lp, 1),
   x2 = glp_get_col_prim(lp, 2),
   x3 = glp_get_col_prim(lp, 3);
```

Získání výsledků

Optimální hodnota cílové funkce.

```
double z = glp_get_obj_val(lp);
```

Optimální vrchol.

```
double x1 = glp_get_col_prim(lp, 1),
    x2 = glp_get_col_prim(lp, 2),
    x3 = glp_get_col_prim(lp, 3);
```

Výpis výsledků.

```
printf("\nz = %g; x1 = %g; x2 = %g; x3 = %g\n",
                    z, x1, x2, x3);
```

Konec

Zrušíme objekt úlohy LP

```
glp_delete_prob(lp);
```

Konec

Zrušíme objekt úlohy LP

```
glp_delete_prob(lp);
```

a zkončíme.

```
return 0;
}
```

Kompilace

Náš soubor `example.c` zkompilujeme příkazem

```
gcc -c example.c
```

Kompilace

Náš soubor `example.c` zkompilujeme příkazem

```
gcc -c example.c
```

a při linkování musíme přilinkovat matematickou a GLPK knihovnu.

```
gcc example.o -lglpk -lm -o example
```

Výstup z programu

```
* 0: obj = 0.000000000e+00 infeas = 0.000e+00 (0)
* 2: obj = 7.333333333e+02 infeas = 0.000e+00 (0)
OPTIMAL SOLUTION FOUND

z = 733.333; x1 = 33.3333; x2 = 66.6667; x3 = 0
```

Výstup z programu

```
*  0: obj =  0.000000000e+00 infeas =  0.000e+00 (0)
*  2: obj =  7.333333333e+02 infeas =  0.000e+00 (0)
OPTIMAL SOLUTION FOUND

z = 733.333; x1 = 33.3333; x2 = 66.6667; x3 = 0
```

Průběžné výstupy funkce `glpk_simplex` mají tvar

```
nnn: obj = xxx infeas = yyy (ddd)
```

- nnn Pořadové číslo kroku simplexové metody.
- xxx Aktualní hodnota cílové funkce.
- yyy Numerická nepřesnost.
- ddd Aktuální počet pevných bázických proměnných.

Získání GLPK

Hlavní stránka GLPK

`http://www.gnu.org/software/glpk/`

Získání GLPK

Hlavní stránka GLPK
`http://www.gnu.org/software/glpk/`

GLPK je ve všech běžných Linuxových distribucích a verze pro windows je na
`http://gnuwin32.sourceforge.net/packages/glpk.htm`.

Získání GLPK

Hlavní stránka GLPK
`http://www.gnu.org/software/glpk/`

GLPK je ve všech běžných Linuxových distribucích a verze pro windows je na
`http://gnuwin32.sourceforge.net/packages/glpk.htm`.

Licence: GNU General Public License