Kapitel 2

Softwarewerkzeuge für die Lineare Programmierung

Inhalt

Python

scipy.optimize.linprog

scipy.optimize.linprog(*c*, *A_ub=None*, *b_ub=None*, *A_eq=None*, *b_eq=None*, *bounds=None*, *method='highs'*, *callback=None*, *options=None*, *x0=None*, *integrality=None*) [source]

Linear programming: minimize a linear objective function subject to linear equality and inequality constraints.

Linear programming solves problems of the following form:

$$\begin{aligned} \min_x \ & c^T x \\ \text{such that } & A_{ub} x \leq b_{ub}, \\ & A_{eq} x = b_{eq}, \\ & l \leq x \leq u, \end{aligned}$$

where x is a vector of decision variables; c, b_{ub}, b_{eq}, l, and u are vectors; and A_{ub} and A_{eq} are matrices.

Hinweise zu den Parametern

- für Maximiereung $-\mathbf{c}$ angeben
- analog für $\geq$-Nebenbedingungen
- Variablenbeschränkungen l und u in Parameter `bounds`
- Standard: alle Variablen sind vorzeichenbeschränkt
- Empfehlung für `method`: `highs-ds`
  recht performanter Simplex-Algorithmus, siehe `https://highs.dev/`

Ergebnis von `linprog`

Dictionary mit u. a. diesen Attributen:

- `success`
  boolean, Wurde ein Optimum gefunden?
- `status`
  `status` = 0 bedeutet Erfolg
- `fun`
  Zielfunktionswert
- `x`
  Komponenten der optimalen Lösung **x**

Beispiele: Eisverkäufer (`eis.py`), Bodenbelag(`boden.py`), siehe Homepage

GNU Linear Programming Kit

In erster Linie eine C-Bibliothek für die

- lineare Programmierung (LP) und
- gemischt-ganzzahlige Programmierung (MIP).

Weitere Eigenschaften:

- open source, GPL
- Homepage: `https://www.gnu.org/software/glpk/`

GLPK Funktionalitäten

- Primaler und dualer Simplex-Algorithmus (LP)
- Innere-Punkte-Algorithmus (LP)
- Branch-and-Bound-Algorithmus (MIP, kombinatorische Optimierung)
- Schnittebenenverfahren (MIP, kombinatorische Optimierung)
- reichhaltige Einstellmöglichkeiten (z. B. Pivot-, Branching- und Backtrack-Strategien)
- C API
- Bindings für andere Sprachen, z. B. Java
- Modellierungssprache
- Stand-alone LP- bzw. MIP-Solver

Problemformen

GLPK kann LPs der folgenden Form verarbeiten:

Maximiere oder minimiere

$$z = c_0 + c_1 x_1 + \cdots + c_n x_n$$

unter linearen Nebenbedingungen

$$\begin{array}{ccccccc} a_{11}x_1 & + & \cdots & + & a_{1n}x_n & \theta & b_1 \\ \vdots & & & & \vdots & \vdots & \vdots \\ a_{m1}x_1 & + & \cdots & + & a_{mn}x_n & \theta & b_m \end{array}$$

mit $\theta \in \{\leq, =, \geq\}$ und Variablenbeschränkungen

$$\begin{array}{ccccc} l_1 & \leq & x_1 & \leq & u_1 \\ l_2 & \leq & x_2 & \leq & u_2 \\ \vdots & & \vdots & & \vdots \\ l_n & \leq & x_n & \leq & u_n \end{array}$$

Hierbei ist $l_i = -\infty$ und $u_i = \infty$ möglich.

Für MIPs oder kombinatorische Probleme können die Varibalen x_i zusätzlich auf

- $x_i \in \mathbb{Z}$ oder
- $x_i \in \{0, 1\}$ eingeschränkt werden.

Eingabeformate

- Am einfachsten ist es, ein LP in einem der möglichen textuellen Eingabeformate zu formulieren und
- dieses mit dem Stand-alone-Solver zu lösen.

Mögliche textuelle Eingabeformate:

- MPS-Format (IBM, 60er Jahre)
- CPLEX-Format (CPLEX Optimization Inc., 80er Jahre)
- GLPK-Format
- GNU MathProg Modellierungssprache

Wir nutzen das CPLEX-Format!

CPLEX-Format

- Plain Text
- zeilenorientiert
- formatfrei, d.h. Zwischenraum ist nicht relevant
- Syntaxelemente (Tokens):
  - Schlüsselwörter
  - Identifier
  - numerische Konstanten und Operatorsymbole
  - Trennzeichen

CPLEX-Format: Aufbau einer LP-Datei

1. Zielfunktion
2. Nebenbedingungen
3. Variablenbeschränkungen
4. Zuordnung der Variablen zu den Wertebereichen $\mathbb{R}, \mathbb{Z}$ und $\{0, 1\}$
5. Schlüsselwort `end`

CPLEX-Format: Beschreibung der Zielfunktion

$$\left\{ \begin{array}{l} \texttt{maximize} \\ \texttt{minimize} \end{array} \right\} [\textit{identifier} :] \textit{expression}$$

- *identifier* ist ein optionaler Bezeichner für den Zielfunktionswert.
- Bezeichner sind nach den übliche syntaktischen Regeln aufgebaut.
- Standardbezeichner für Zielfunktionswert: `obj`
- *expression*:

$$s\ c\ x\ s\ c\ x \ldots s\ c\ x$$

  - $s =$ Vorzeichen
  - $c =$ numerische Konstante als Koeffizient (otional)
  - $x =$ Bezeichner für eine Variable

- Das erste Vorzeichen (Standard: +) und die Koeffizienten (Standard: `1.0`) sind optional.
- Koeffizienten werden in den für Programmiersprachen üblichen Notationen geschrieben.
- Variablen müssen nicht deklariert werden.

Beispiel:

```
Minimize Z : - x1 + 2 x2 - 3.5 x3 + 4.997e3x4 + x5 + x6 +
    x7 - .01x8
```

CPLEX-Format: Nebenbedingungen

`subject to`
nebenbedingung$_1$
...
nebenbedingung$_m$

Jede *nebenbedingung*$_i$ in der Form:

$$[r :] \mathit{expression} \left\{ \begin{array}{c} \texttt{<=} \\ \texttt{>=} \\ \texttt{=} \end{array} \right\} b$$

- r ist optionaler Bezeichner für die Nebenbedingung; Standard: `r.`iii
- b ist numerische Konstante mit optionalem Vorzeichen

Beispiel für Produktionsproblem (siehe Beispiel 4.6):

```
maschine1: 40 a + 24 b <= 480
maschine2: 24 a + 48 b <= 480
maschine3:       60 b <= 480
```

CPLEX-Format: Variablenbeschränkungen

```
bounds
bound_1
...
bound_p
```

- Dieser Abschnitt ist optional.
- Standard: Für alle definierten Variablen x_i gilt $x_i \geq 0$.
- Auch wenn Abschnitt vorhanden, muss nicht für jede Variable eine Beschränkung $bound_j$ definiert werden.
- Standard: Für alle Variablen x_i, für die keine Beschränkung definiert ist, gilt $x_i \geq 0$.
- Jede Definition einer Beschränkung $bound_j$ muss auf einer neuen Zeile erfolgen.

Die Beschränkungen $bound_j$ können wie folgt aufgebaut sein:

x >= l	untere Schranke l für Variable x
l <= x	untere Schranke l für Variable x
x <= u	obere Schranke u für Variable x
l <= x <= u	untere Schranke l und obere Schranke u für Variable x
x = t	fixer Wert t für Variable x
x `free`	Variable x ist unbeschränkt

- Schlüsselwörter `inf` bzw. `infinity` stehen für ∞.
- `-inf` entspricht somit $-\infty$.

CPLEX-Format: Wertebereiche von Variablen

- optional; Standard: für alle Variablen x_i : $x_i \in \mathbb{R}$
- notwendig für ganzzahlige oder kombinatorische Probleme
- Alle ganzzahligen bzw. kombinatorischen Variablen werden jeweils in einem eigenen Abschnitt aufgelistet.
- In jeder Zeile nur eine Variable!

ganzzahlige Variablen ($x_i \in \mathbb{Z}$):

```
Integer
```

x_1

...

x_q

binäre Variablen ($x_i \in \{0, 1\}$):

```
Binary
```

x_1

...

x_q

CPLEX-Format: Einfaches LP-Beispiel

Beispiel 2.1

Produktionsplanung von Beispiel 4.6:

```
Maximize
    10 a + 40 b
subject to
    maschine1: 40 a + 24 b <= 480
    maschine2: 24 a + 48 b <= 480
    maschine3:        60 b <= 480
end
```

CPLEX-Format: IP-Beispiel

Beispiel 2.2

Schnittproblem, siehe Folie 37 ff.:

```
Minimize
   x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
subject to
   3 x1 + x2              + 2 x5 +   x6        + x8       >= 10
          x2        + x4                                  >= 45
               2 x3                              + x8 + x9 >= 21
                         x4 +   x5 + 2 x6 + 3 x7      + x9 >= 42
integer
   x1
   x2
   x3
   ...
end
```

Stand-alone Solver

Kommando:

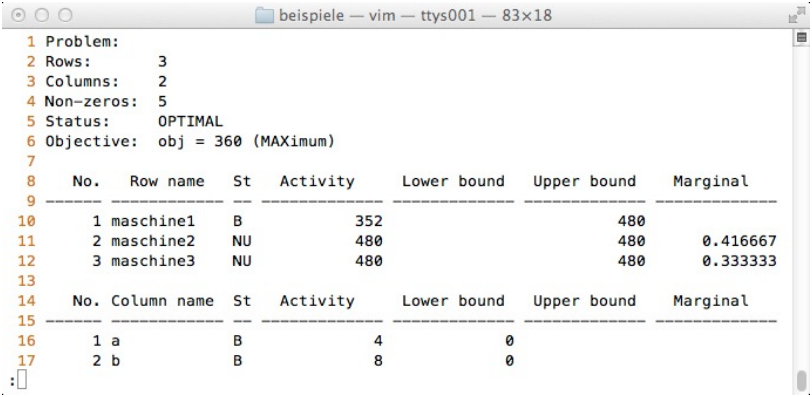
`glpsol` [*options* ...] *filename*

- Löst LP aus *filename*
- Für LP/MIP im CPLEX-Format: Option `--lp`
- Ausgabe der optimalen Lösung in Datei: Option `-o` *filename*
- In Ausgabedatei: optimale Lösung

Aufruf:

```
glpsol --lp production.lp -o production.sol
```

Inhalt von `production.sol`:



Installation (Debian, Ubuntu, Mint)

```
sudo apt-get install glpk-doc glpk-utils libglpk-dev
```

- `glpk-doc`: Dokumentation, Manual(e)
- `glpk-utils`: Stand-alone-Solver, Beispiele
- `libglpk-dev`: C-Header-Datei, statische (`.a`) und dyamische (`.so`) Bibliothek

Gurobi

- Homepage: `www.gurobi.com`
- Probleme:
  - lineare Programmierung (LP)
  - gemischt-ganzzahlige Programmierung (MIP)
  - quadratische Programmierung (QP): quadratische Zielfunktion und/oder Nebenbedingungen
- kommerziell effizientester LP/MIP-Solver.
- Im Vergleich zu GLPK insbesondere bei MIP deutlichst leistungsfähiger.

Konkurrenten: CPLEX (IBM), FICO Xpress (FICO)

Weitergehende Funktionalitäten

- Native Unterstützung für C, C++, Java, Python, Microsoft.NET, Matlab, R
- Unterstützung spezieller Arten von Nebenbedingungen
- Branch-and-Bound-Framework mit Lazy-Constraints
- Quadratische Programmierung
- parallele Implementierung der Verfahren
- Compute Cluster und Remote Services
- kostenlos nutzbar im akademischen Bereich

Installation

1. Registrieren Sie sich bei `www.gurobi.com`.
2. Gurobi-Software herunterladen:
   - `Downloads` → `Gurobi Software`
   - unterstützte Betriebssyteme: Linux (64-bit), macOS (64-bit), Windows (32- und 64-bit), AIX
3. Software installieren:
   - Linux: `.tar.gz` auspacken
   - evtl. Pfade und Environment-Variablen anpassen
4. Lizenz erzeugen:
   `Downloads` → `Lizenzen` → `Akademische Lizenz beantragen`
5. Lizenzschlüssel installieren:
   - Kommando `grbgetkey`
   - genauere Informationen mithilfe der Lizenzübersicht
   - Voraussetzung: Sie sind im Netzwerk der Hochschule Bonn-Rhein-Sieg.
   - Nachdem der Lizenzschlüssel installiert ist, müssen Sie für die Benutzung von Gurobi nicht mehr im Hochschulnetz sein.

Stand-alone Solver

Eingabeformat: CPLEX

Aufruf:

```
gurobi_cl ResultFile=production.sol production.lp
```

Inhalt von `production.sol`:

```
# Objective value = 360
a 4
b 8
```

Programmierbeispiele

C:

- Produktionsproblem: `production_c.c`
- Verschnittproblem: `verschnitt_c.c`

Java:

- Produktionsproblem: `Production.java`
- Verschnittproblem: `Verschnitt.java`

Erzeugung der Executables, Class-Dateien sowie Java-Ausführung:

- `Makefile`

Zusammenfassung

- GLPK: Open Source LP/MIP-Solver
- Gurobi: kommerzieller LP/MIP-Solver
- jeweils C-Bibliothek, Anbindung für weitere Sprachen
- einfaches Eingabeformat: CPLEX
- Programmierbeispiele für Gurobi