SOLVING LINEAR PROGRAMS WITH GLPK

Tutorial with Examples

GLPK

- Why GLPK?
  - Open source, available free of cost to everyone.
  - API for programmers + Input languages that are easy to use.

- Availability: http://www.gnu.org/software/glpk/

Downloading + Installing GLPK

www.gnu.org/software/glpk/

Introduction | Downloading | Documentation | Mailing Lists/Newsgroups | Request an Enhancement | Report a Bug | Maintainer

Introduction to GLPK

The GLPK (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

GLPK supports the *GNU MathProg modeling language*, which is a subset of the AMPL language.

The GLPK package includes the following main components:

- primal and dual simplex methods
- primal-dual interior-point method
- branch-and-cut method
- translator for GNU MathProg
- application program interface (API)
- stand-alone LP/MIP solver

Downloading GLPK

The GLPK distribution tarball can be found on http://ftp.gnu.org/gnu/glpk/ [via http] and ftp://ftp.gnu.org/gnu/glpk/ [via FTP]. It can also be found on one of our FTP mirrors; please use a mirror if possible.

To make sure that the GLPK distribution tarball you have downloaded is intact you need to download the corresponding .sig file and run a command like this:

```
gpg --verify glpk-4.32.tar.gz.sig
```

If that command fails because you do not have the required public key, run the following command to import it:

```
gpg --keyserver keys.gnupg.net --recv-keys 5981E818
```

and then re-run the previous command.

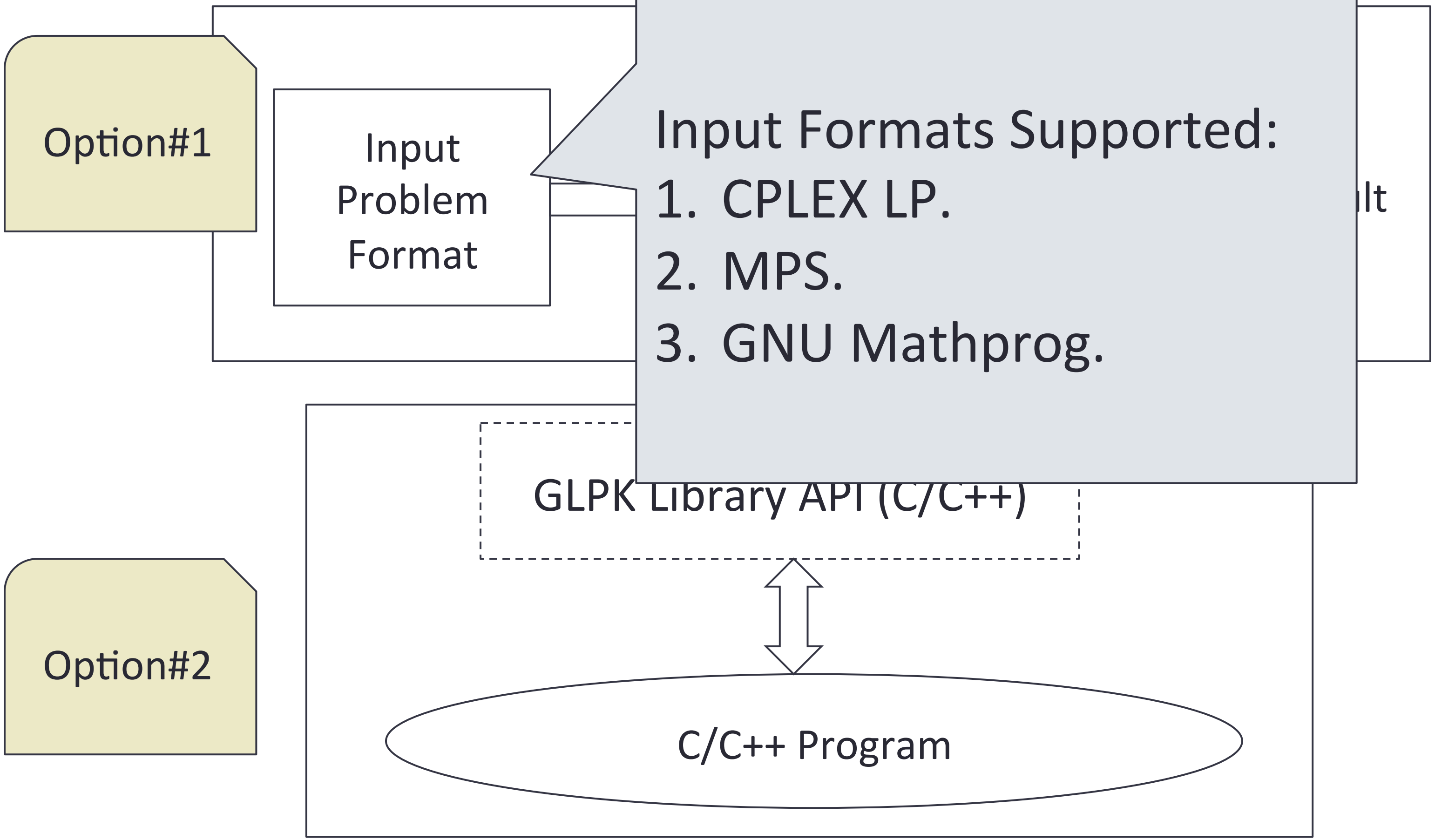
Documentation

The GLPK documentation consists of the Reference Manual and the description of the GNU MathProg modeling language. Both these documents are included in the distribution (in LaTeX, DVI, and PostScript formats).

Mailing Lists/Newsgroups

GLPK has two mailing lists: *help-glpk@gnu.org* and *bug-glpk@gnu.org*.

Solving LPs through GLPK



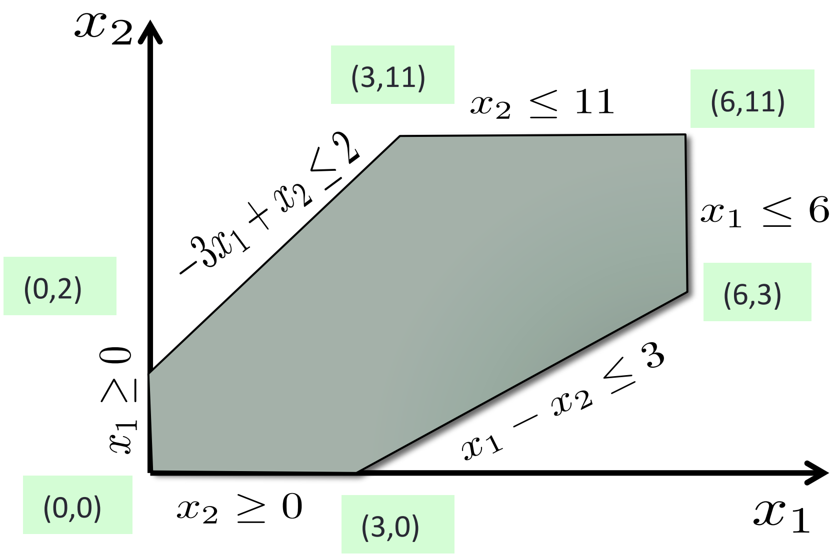
Input Formats Supported:

1. CPLEX LP.
2. MPS.
3. GNU Mathprog.

Example #1

$$\begin{array}{llllll}
\text{max.} & x_1 & +2x_2 & & \\
\text{s.t.} & -3x_1 & +x_2 & \leq & 2 \\
 & & +x_2 & \leq & 11 \\
 & x_1 & -x_2 & \leq & 3 \\
 & x_1 & & \leq & 6 \\
 & x_1, & x_2 & \geq & 0
\end{array}$$

Not drawn to scale



Solution: $x_1 = 6$, $x_2 = 11$Opt. Objective Value: 28

Specifying Problem: Mathprog Format

$$
\begin{array}{rrrrr}
\text{max.} & x_1 & +2x_2 & & \\
\text{s.t.} & -3x_1 & +x_2 & \leq & 2 \\
 & & +x_2 & \leq & 11 \\
 & x_1 & -x_2 & \leq & 3 \\
 & x_1 & & \leq & 6 \\
 & x_1, & x_2 & \geq & 0
\end{array}
$$

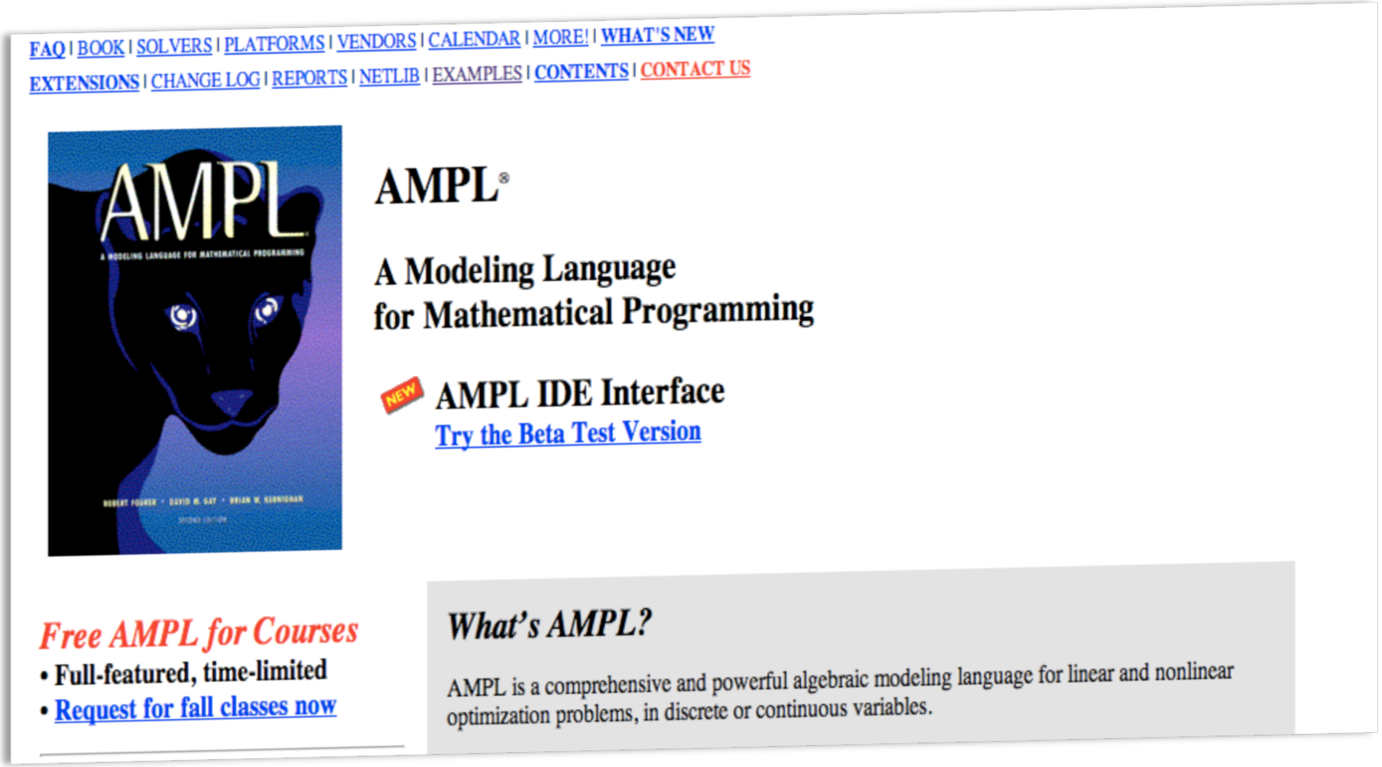
```
var x1 >= 0;
var x2 >= 0;
maximize obj: x1 + 2 * x2;
c1: -3 * x1 + x2 <= 2;
c2: x2 <= 11;
c3: x1 - x2 <= 3;
c4: x1 <= 6;
solve;
display x1;
display x2;
end;
```

Running GLPSOL

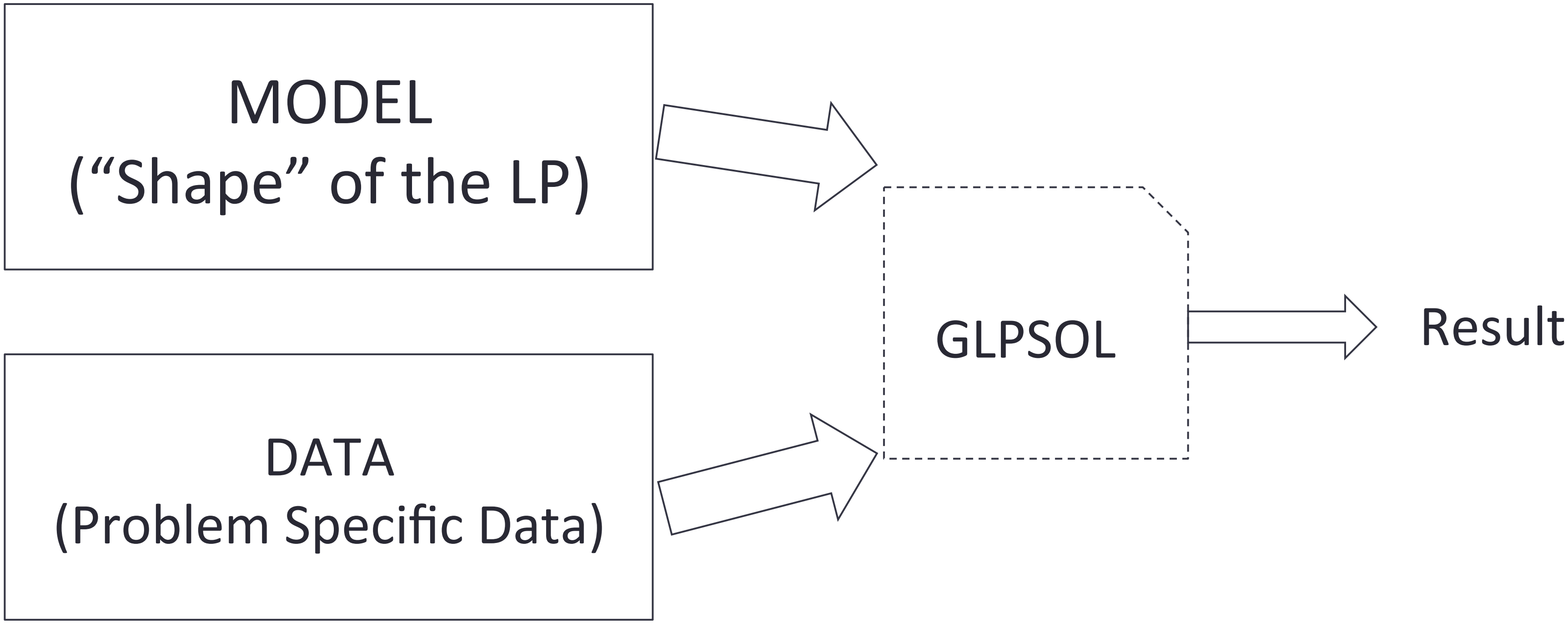
```
bash-3.2$ glpsol --math ex1.ampl
GLPSOL: GLPK LP/MIP Solver, v4.48
OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Display statement at line 12
x1.val = 6
Display statement at line 13
x2.val = 11
Model has been successfully processed
```

Using Mathprog Language

- Very close to AMPL



Mathprog input format



Example

$$
\begin{array}{rcccc}
\text{max.} & x_1 & +2x_2 & & \\
\text{s.t.} & -3x_1 & +x_2 & \leq & 2 \\
 & & +x_2 & \leq & 11 \\
 & x_1 & -x_2 & \leq & 3 \\
 & x_1 & & \leq & 6 \\
 & x_1, & x_2 & \geq & 0
\end{array}
$$

"Shape of the LP"

$$
\begin{array}{llll}
\max & \sum_{j=1}^{n} \boxed{c_i} x_i & & \\
\text{s.t.} & \sum_{j=1}^{n} \boxed{a_{i,j}} x_j \leq \boxed{b_i} & & [i \in \{1, \ldots, \boxed{m}\}] \\
 & x_j \geq 0 & & [j \in \{1, \ldots, \boxed{n}\}]
\end{array}
$$

Mathprog Format

```
# number of constraints
param m;
# number of decision variables
param n;

#problem parameters
param c { i in 1..n};
param A { i in 1..m, j in 1..n};
param b { i in 1..m};

#declare variables
var x { i in 1..n} >= 0;
```

"Shape of the LP"

$$
\begin{array}{llll}
\max & \sum_{j=1}^{n} c_i x_i & & \\
\text{s.t.} & \sum_{j=1}^{n} a_{i,j} x_j \leq & b_i & [i \in \{1, \ldots, m\}] \\
 & x_j \geq & 0 & [j \in \{1, \ldots, n\}]
\end{array}
$$

```
#objective
maximize obj: sum{ i in 1..n} c[i] * x[i];
s.t.
e{j in 1..m}: sum{i in 1..n} A[j,i] * x[i] <= b[j];

solve;
display x;
end;
```

Running GLPSOL

```
$ glpsol --model standardForm.model --data ex1.data
GLPSOL: GLPK LP/MIP Solver, v4.48
OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (130844 bytes)
Display statement at line 20
x[1].val = 6
x[2].val = 11
Model has been successfully processed
```

SOLVING THE DIET PROBLEM IN GLPK

Modeling the Diet Problem

- Problem Data

Name	Type	Meaning
NFoods	SCALAR (INT)	Number of Food Items
NNutrients	SCALAR (INT)	Number of Nutrients
costs	NFoods x 1	Cost per unit of each food
caloricData	Nfoods x NNutrients	Nutrients per unit of each food.
upperBnd	NNutrients x 1	Upper bound on nutrients reqd.
lowerBnd	NNutrients x 1	Lower bound on nutrients reqd.

```
param NFoods;
param NNutrients;
param caloricDat {i in 1..NFoods, j in 1..NNutrients};
param lb {i in 1..NNutrients};
param ub {i in 1..Nnutrients};
param costs { i in 1..NFoods};

var x { i in 1..NFoods} >= 0;

minimize obj: sum{i in 1..NFoods} costs[i] * x[i];

bnds {k in 1..NNutrients}:
   lb[k]<= sum{i in 1..NFoods} x[i]*caloricDat[i,k] <=
ub[k];

solve;
display x;
```

```
param NFoods := 64;
param NNutrients := 11;

param caloricMatrix :
        1       2      3      4       5      6      7      8       9      10     11
:=
  1   73.8      0    0.8    68.2   13.6    8.5      8   5867.4  160.2   159    2.3
  2   23.7      0    0.1    19.2    5.6    1.6    0.6   15471     5.1   14.9   0.3
  3    6.4      0    0.1    34.8    1.5    0.7    0.3    53.6     2.8   16     0.2
  4   72.2      0    0.6     2.5   17.1      2    2.5   106.6     5.2    3.3   0.3
  5    2.6      0      0     1.8    0.4    0.3    0.2    66       0.8    3.8   0.1
  6   20        0    0.1     1.5    4.8    1.3    0.7   467.7    66.1    6.7   0.3
  7  171.5      0    0.2    15.2   39.9    3.2    3.7     0      15.6   22.7   4.3
  8   88.2      0    5.5     8.1    2.2    1.4    9.4    98.6     0.1  121.8   6.2
  9  277.4  129.9   10.8   125.6      0      0   42.2    77.4       0   21.9   1.8
 10  358.2      0   12.3  1237.1   58.3   11.6    8.2  3055.2    27.9   80.2   2.3
```

```
$ glpsol --model diet.model --data diet.dat
GLPSOL: GLPK LP/MIP Solver, v4.48
OPTIMAL SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.4 Mb (377283 bytes)
Display statement at line 19
x[1].val = 0
x[2].val = 0.235817810889784
x[3].val = 0
x[4].val = 0
x[5].val = 0
x[6].val = 0
x[7].val = 3.5449447765207l
x[8].val = 0
x[9].val = 0
x[10].val = 0
. . .
```

More Advanced Model

```
set Foods;
set Nutrients;

param calDat { i in Foods, j in Nutrients};
param bounds { i in Nutrients, j in 1..2};
param costs { i in Foods};

var x { i in Foods} >= 0;

minimize obj: sum{i in Foods} costs[i] * x[i];
s.t. bndConstr {k in Nutrients}:
     bounds[k,1] <= sum{i in Foods} x[i] * calDat[i,k] <= bounds[k,2];
solve;
display x;
printf '----- \n';
printf {i in Foods: x[i] >= 0.001}: 'Optimizer: %f units of food %s \n', x[i], i;
printf ' The bill for the food will be \$ %f \n', obj;
printf 'Bon appetit! \n -----\n';
```

Result

```
bash-3.2$ glpsol --model dietSet.model --data dietSet.data
GLPSOL: GLPK LP/MIP Solver, v4.48
-----
Optimizer says to eat 0.235818 units of food Carrots_Raw
Optimizer says to eat 3.544945 units of food Potatoes_Baked
Optimizer says to eat 2.167849 units of food Skim_Milk
Optimizer says to eat 3.600776 units of food Peanut_Butter
Optimizer says to eat 4.823229 units of food Popcorn_Air-
Popped
 The bill for the food will be $ 0.956008
Bon appetit!
 -----
Model has been successfully processed
```