Using GPLK to solve Mixed Integer Programming (MIP) Linear Optimization Models

Mr Ashlin Darius Govindasamy
University of South Africa

October 4, 2022

Abstract

This paper is a guide to using the Octave programming language and module (GPLK) to solve linear optimization models.

Contents

Chapter 1

GPLK Parameters and Flags

1.0.1 Introduction

Octave can solve Linear Programming problems using the glpk function. That is, Octave can solve

```
min C'*x
```

subject to the linear constraints

$$A * x = b \quad \text{where } x \geq 0 \tag{1.1}$$

the **glpk** function also supports variations of this problem.

1.0.2 Syntax

```
: [xopt, fmin, errnum, extra] = glpk (c, A, b, lb, ub, ctype, vartype, sense, param)
```

Given 3 Parameters, **gplk** solves the following standard form linear programming problem:

$$\begin{aligned} &\min c^T x \\ &\text{subject to } Ax = b \\ &x \geq 0 \end{aligned} \tag{1.2}$$

but it can also solve the following variations of this problem:

```
[min|max] c'*x
```

subject to the linear constraints

```
A*x [ "=" | "<=" | ">=" ] b
x >= LB
x <= UB
```

1.0.3 Parameters and Flags

Input arguments:

c

A column array containing the objective function coefficients.

A

A matrix containing the constraints coefficients.

b

A column array containing the right-hand side value for each constraint in the constraint matrix.

lb

An array containing the lower bound on each of the variables. If *lb* is not supplied, the default lower bound for the variables is zero.

ub

An array containing the upper bound on each of the variables. If *ub* is not supplied, the default upper bound is assumed to be infinite.

ctype

An array of characters containing the sense of each constraint in the constraint matrix. Each element of the array may be one of the following values

`"F"`

A free (unbounded) constraint (the constraint is ignored).

`"U"`

An inequality constraint with an upper bound (`A(i,:)*x <= b(i)`).

`"S"`

An equality constraint (`A(i,:)*x = b(i)`).

`"L"`

An inequality with a lower bound (`A(i,:)*x >= b(i)`).

`"D"`

An inequality constraint with both upper and lower bounds (`A(i,:)*x >= -b(i)`) *and* (`A(i,:)*x <= b(i)`).

vartype

A column array containing the types of the variables.

`"C"`

A continuous variable.

`"I"`

An integer variable.

sense

If *sense* is 1, the problem is a minimization. If *sense* is -1, the problem is a maximization. The default value is 1.

param

A structure containing the following parameters used to define the behavior of solver. Missing elements in the structure take on default values, so you only need to set the elements that you wish to change from the default.

Integer parameters:

`msglev (default: 1)`

Level of messages output by solver routines:

0 (*GLP_MSG_OFF*)

No output.

1 (*GLP_MSG_ERR*)

Error and warning messages only.

2 (*GLP_MSG_ON*)

Normal output.

3 (*GLP_MSG_ALL*)

Full output (includes informational messages).

`scale (default: 16)`

Scaling option. The values can be combined with the bitwise OR operator and may be the following:

1 (*GLP_SF_GM*)

Geometric mean scaling.

16 (*GLP_SF_EQ*)

Equilibration scaling.

32 (*GLP_SF_2N*)

Round scale factors to power of two.

64 (*GLP_SF_SKIP*)

Skip if problem is well scaled.

Alternatively, a value of 128 (*GLP_SF_AUTO*) may be also specified, in which case the routine chooses the scaling options automatically.

dual (default: 1)

Simplex method option:

1 (*GLP_PRIMAL*)

Use two-phase primal simplex.

2 (*GLP_DUALP*)

Use two-phase dual simplex, and if it fails, switch to the primal simplex.

3 (*GLP_DUAL*)

Use two-phase dual simplex.

price (default: 34)

Pricing option (for both primal and dual simplex):

17 (*GLP_PT_STD*)

Textbook pricing.

34 (*GLP_PT_PSE*)

Steepest edge pricing.

itlim (default: intmax)

Simplex iterations limit. It is decreased by one each time when one simplex iteration has been performed, and reaching zero value signals the solver to stop the search.

outfrq (default: 200)

Output frequency, in iterations. This parameter specifies how frequently the solver sends information about the solution to the standard output.

branch (default: 4)

Branching technique option (for MIP only):

1 (*GLP_BR_FFV*)

First fractional variable.

2 (*GLP_BR_LFV*)

Last fractional variable.

3 (*GLP_BR_MFV*)

Most fractional variable.

4 (*GLP_BR_DTH*)

Heuristic by Driebeck and Tomlin.

5 (*GLP_BR_PCH*)

Hybrid pseudocost heuristic.

btrack (default: 4)

Backtracking technique option (for MIP only):

1 (*GLP_BT_DFS*)

Depth first search.

2 (*GLP_BT_BFS*)

Breadth first search.

3 (*GLP_BT_BLB*)

Best local bound.

4 (*GLP_BT_BPH*)

Best projection heuristic.

presol (default: 1)

If this flag is set, the simplex solver uses the built-in LP presolver. Otherwise the LP presolver is not used.

lpsolver (default: 1)

Select which solver to use. If the problem is a MIP problem this flag will be ignored.

1

Revised simplex method.

2

Interior point method.

rtest (default: 34)

Ratio test technique:

17 (*GLP_RT_STD*)

Standard ("textbook").

34 (*GLP_RT_HAR*)

Harris' two-pass ratio test.

tmlim (default: intmax)

Searching time limit, in milliseconds.

outdly (default: 0)

Output delay, in seconds. This parameter specifies how long the solver should delay sending information about the solution to the standard output.

save (default: 0)

If this parameter is nonzero, save a copy of the problem in CPLEX LP format to the file *"outpb.lp"*. There is currently no way to change the name of the output file.

Real parameters:

tolbnd (default: 1e-7)

Relative tolerance used to check if the current basic solution is primal feasible. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose.

toldj (default: 1e-7)

Absolute tolerance used to check if the current basic solution is dual feasible. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose.

tolpiv (default: 1e-10)

Relative tolerance used to choose eligible pivotal elements of the simplex table. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose.

objll (default: -DBL_MAX)

Lower limit of the objective function. If the objective function reaches this limit and continues decreasing, the solver stops the search. This parameter is used in the dual simplex method only.

objul (default: +DBL_MAX)

Upper limit of the objective function. If the objective function reaches this limit and continues increasing, the solver stops the search. This parameter is used in the dual simplex only.

tolint (default: 1e-5)

Relative tolerance used to check if the current basic solution is integer feasible. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose.

tolobj (default: 1e-7)

Relative tolerance used to check if the value of the objective function is not better than in the best known integer feasible solution. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose.

Output values:

xopt

The optimizer (the value of the decision variables at the optimum).

fopt

The optimum value of the objective function.

errnum

Error code.

0

No error.

1 (*GLP_EBADB*)

Invalid basis.

2 (*GLP_ESING*)

Singular matrix.

3 (*GLP_ECOND*)

Ill-conditioned matrix.

4 (*GLP_EBOUND*)

Invalid bounds.

5 (*GLP_EFAIL*)

Solver failed.

6 (*GLP_EOBJLL*)

Objective function lower limit reached.

7 (*GLP_EOBJUL*)

Objective function upper limit reached.

8 (*GLP_EITLIM*)

Iterations limit exhausted.

9 (*GLP_ETMLIM*)

Time limit exhausted.

10 (*GLP_ENOPFS*)

No primal feasible solution.

11 (*GLP_ENODFS*)

No dual feasible solution.

12 (*GLP_EROOT*)

Root LP optimum not provided.

13 (*GLP_ESTOP*)

Search terminated by application.

14 (*GLP_EMIPGAP*)

Relative MIP gap tolerance reached.

15 (*GLP_ENOFEAS*)

No primal/dual feasible solution.

16 (*GLP_ENOCVG*)

No convergence.

17 (*GLP_EINSTAB*)

Numerical instability.

18 (*GLP_EDATA*)

Invalid data.

19 (*GLP_ERANGE*)

Result out of range.

extra

A data structure containing the following fields:

`lambda`

Dual variables.

`redcosts`

Reduced Costs.

`time`

Time (in seconds) used for solving LP/MIP problem.

`status`

Status of the optimization.

1 (*GLP_UNDEF*)

Solution status is undefined.

2 (*GLP_FEAS*)

Solution is feasible.

3 (*GLP_INFEAS*)

Solution is infeasible.

4 (*GLP_NOFEAS*)

Problem has no feasible solution.

5 (*GLP_OPT*)

Solution is optimal.

6 (*GLP_UNBND*)

Problem has no unbounded solution.

Chapter 2

Implementation and Examples

2.0.1 Implementation and Examples

2.0.1.1 Example 1: Minimizing Optimization Multiple Integer Programming Problem

Consider the following optimization problem:

$$\begin{aligned} L &= 2x_1 + 3x_2 + 4x_3 + 5x_4 \\ \text{subject to } & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 100 \\ & 6x_1 + 3x_2 + 7x_3 + 3x_4 \geq 50 \\ & x_1 + x_2 + x_3 + x_4 \leq 10 \\ & x_1 + x_2 + x_3 + x_4 \geq 5 \\ & x_1 \geq 0 \end{aligned} \tag{2.1}$$

The Upper Bound Symbol is $\leq$ and the Lower Bound Symbol is $\geq$

We can solve this problem using the **glpk** function in Octave. The **glpk** function takes 9 parameters. The first 5 parameters are the objective function, the linear constraints, the lower bound and the upper bound. The last 4 parameters are the constraint type, the variable type, the sense and the parameters.

$$\text{glpk}(c, A, b, lb, ub, ctype, vartype, sense, param) \tag{2.2}$$

Lets find our solution using Octave:

$$C = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \tag{2.3}$$

C is the objective function. The objective function is the function we want to minimize or maximize. In this case we want to minimize the objective function.

$$A = \begin{bmatrix} 2 & 3 & 4 & 5 \\ 6 & 3 & 7 & 3 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

A is the matrix of the linear constraints. The linear constraints are the constraints that the variables must satisfy. In this case we have 5 linear constraints.

$$b = \begin{bmatrix} 100 \\ 50 \\ 10 \\ 5 \\ 0 \end{bmatrix} \tag{2.5}$$

b is the matrix of the linear constraints. The linear constraints are the constraints that the variables must satisfy. In this case we have 5 linear constraints.

$$lb = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.6}$$

lb is the lower bound matrix. The lower bound matrix is the matrix of the lower bounds of the variables. In this case we have 4 variables and we have no lower bounds.

$$ub = \begin{bmatrix} \infty \\ \infty \\ \infty \\ \infty \end{bmatrix} \tag{2.7}$$

ub is the upper bound matrix. The upper bound matrix is the matrix of the upper bounds of the variables. In this case we have 4 variables and we have no upper bounds.

$$ctype = \begin{bmatrix} 'U' \\ 'L' \\ 'U' \\ 'L' \\ 'L' \end{bmatrix} \tag{2.8}$$

ctype is the constraint type matrix. The constraint type matrix is the matrix of the constraint types. In this case we have 5 constraints and we have 2 upper bounds, 2 lower bounds and 1 equality constraint.

$$vartype = \begin{bmatrix} 'C' \\ 'C' \\ 'C' \\ 'C' \end{bmatrix} \tag{2.9}$$

vartype is the variable type matrix. The variable type matrix is the matrix of the variable types. In this case we have 4 variables and we have 4 continuous variables.

$$sense = -1 \tag{2.10}$$

sense is the sense of the optimization problem. The sense of the optimization problem is the sense of the objective function. In this case we want to minimize the objective function so we set the sense to -1.

We set our parameters to

```
param.msglev = 1;
```

Msglev is the message level. The message level is the level of the messages that are displayed. In this case we set the message level to 1 so that we can see the messages.

Now we can implement our octave code

```
c = [2; 3; 4; 5];
A = [2 3 4 5; 6 3 7 3; 1 1 1 1; 1 1 1 1; 1 0 0 0];
b = [100; 50; 10; 5; 0];
lb = [0; 0; 0; 0];
ub = [inf; inf; inf; inf];
ctype = ['U'; 'L'; 'U'; 'L'; 'L'];
vartype = ['C'; 'C'; 'C'; 'C'];
sense = 1;
param.msglev = 1;
[x, fmin, status, extra] = glpk (c, A, b, lb, ub, ctype, vartype, sense, param)
```

When we run this code we get the following output:

```
   x =

   8.3333
        0
        0
        0

fmin = 16.667
status = 0
extra =

  scalar structure containing the fields:

    lambda =

            0
       0.3333
            0
            0
            0

    redcosts =

            0
       2.0000
       1.6667
       4.0000

    time = 0
    status = 5
```

In conclusion we solved the question for the most optimized minimal variables we should use

$$\begin{aligned} x_1 &= 8.3333 \\ x_2 &= 0 \\ x_3 &= 0 \\ x_4 &= 0 \end{aligned} \tag{2.11}$$

To get our objective function to be minimized to 16.667

2.0.1.2 Example 2: Real World Example

ADGSTUDIOS manufactures two types of wooden toys: peace-keepers and trains. A peace-keeper sells for R27 and uses R10 worth of raw materials. Each peace-keeper that is manufactured increases ADGSTUDIOS's variable labour and overhead costs by R14. A train sells for R21 and uses R9 worth of raw materials. Each train built increases ADGSTUDIOS's variable labour and overhead costs by R10. The manufacture of wooden peace-keeper and trains requires two types of skilled labor: carpentry and finishing. A peace-keeper requires 2 hours of finishing labour and 1 hour of carpentry labour. A train requires 1 hour of finishing labour and 1 hour of carpentry labour. Each week, ADGSTUDIOS can obtain all the needed raw material but only 100 finishing hours and 80 carpentry hours. Demand for trains is unlimited, but at most 40 peace-keepers are bought each week. ADGSTUDIOS wants to maximize weekly profit (revenues-costs).

$$\begin{aligned} &\text{maximize } (27-10-14)x_1 + (21-9-10)x_2 \\ &\text{subject to } 10x_1 + 9x_2 \leq 100 \\ &\qquad 2x_1 + x_2 \leq 80 \\ &\qquad x_1 \leq 40 \\ &\qquad x_1, x_2 \geq 0 \end{aligned} \tag{2.12}$$

Our objective function is

$$C = 3x_1 + 2x_2 \tag{2.13}$$

Octave code

```
c=[3 2]; % maximise z = 3 x_1 + 2 x_2
% subject to:
A=[2,1;1,1;1,0;1,0;0,1]; % coefficients
b=[100;80;40;0;0] % bounds

lb=[0;0]
ub=[Inf;Inf];
ctype = ["U";"U";"U";"L";"L"]; % indicates upper bound or lower bound
vtype = ["C";"C"]; % continuous variables

sense=-1; % maximises
[xopt,zmx]=glpk(c,A,b,lb,ub,ctype,vtype,sense)
```

Output

```
xopt =
    20
    60

zmx = 180
```

Our optimal solution is

$$\begin{aligned} x_1 &= 20 \\ x_2 &= 60 \end{aligned} \tag{2.14}$$

So maximum revenue for ADGSTUDIOS is R180 each week. The report also shows that ADGSTUDIOS can maximize weekly profit by producing 20 peace-keepers and 60 trains each week.