# Assignment 1 - Linear Programming

## Contents

## The problem

A construction materials company is looking for a way to maximize profit per transportation of their goods. The company has a train available with 4 wagons.

When stocking the wagons they can choose among 3 types of cargo, each with its own specifications. How much of each cargo type should be loaded on which wagon in order to maximize profit?

*Linear programming, maximization problem*

### More data

| TRAIN WAGON $j$ | WEIGHT CAPACITY (TONNE) $w_j$ | VOLUME CAPACITY ($m^2$) $s_j$ |
|---|---|---|
| (wag) 1 | 10 | 5000 |
| (wag) 2 | 8 | 4000 |
| (wag) 3 | 12 | 8000 |
| (wag) 4 | 6 | 2500 |

| CARGO TYPE $i$ | AVAILABLE (TONNE) $a_i$ | VOLUME ($m^2/t$) $v_i$ | PROFIT (PER TONNE) $p_i$ |
|---|---|---|---|
| (cg) 1 | 20 | 500 | 3500 |
| (cg) 2 | 10 | 300 | 2500 |
| (cg) 3 | 18 | 400 | 2000 |

### The decision variables

Define the decision variables for the problem described above.

- $x_1$: how much of cg1 loaded on which wagon
- $x_2$: how much of cg2 loaded on which wagon
- $x_3$: how much of cg3 loaded on which wagon

### The objective function

Define the objective function for the problem described above.

The profit coefficients of these variables are 3500, 2500, and 2000, respectively. Therefore, the objective function is defined multiplying each variable by its corresponding coefficient.

- $f_0(x) = 3500x_1 + 2500x_2 + 2000x_3$

### The constraints

Define the constraints for the problem described above.

- Decision variables can't exceed the weight capacity of the train
  $x_1 + x_2 + x_3 \leq 36$
  ai <= 36
- Volume of the cargo can't exceed Volume Capacity of the wagons
  $500cg_1 + 300cg_2 + 400cg_3 \leq 19500$
  sj <= 19500
- Every decision variable can't exceed the available quantity
  $x_1 \leq 20$
  $x_2 \leq 10$
  $x_3 \leq 18$
- All variables positive
  $x_1, x_2, x_3 \geq 0$

## Building the model

Build and solve the model with a suitable solver. You might want to use the `lpSolveAPI` library.

max $f_0(x)$

s.t. $a_i \leq 36$

$s_j \leq 19500$

$x_1 \leq 20$

$x_2 \leq 10$

$x_3 \leq 18$

$x_1, x_2, x_3 \geq 0$

*Install the package*

```
#LP_SOLVE
if(require(lpSolveAPI)==FALSE) install.packages("lpSolveAPI")
```

```
## Loading required package: lpSolveAPI
```

*Build the Model*

The `make.lp(x,y)` function initializes and defines the structure to be solved by `lpSolve` through the library `lpSolveAPI`. This syntax creates a model with a matrix of coefficients that is `x` rows by `y` columns wide; the number of rows is defined by the number of initial constraints while the number of columns is defined by the **total number of variables** in this system.

```
model = make.lp(0,3)
name.lp(lprec = model, "Cargo - Wagons") # name the model
```

*Define the Objective Function*

The objective function is declared using the functions `set.objfn()` and `lpcontrol()`. In both functions, the first argument is the name of a linear optimization model

- `lp.control()` uses the argument `sense`, that can be set to "max" or "min" to reflect that we want to *maximize* or *minimize* the objective function, respectively.
- `set.objfn()` declares the coefficients of each of the variables in the objective function to be optimized, and takes a vector as an argument into `obj`.

```
lp.control(model, sense="max")
```

```
## $anti.degen
## [1] "fixedvars" "stalling"
##
## $basis.crash
## [1] "none"
##
## $bb.depthlimit
## [1] -50
##
## $bb.floorfirst
## [1] "automatic"
##
## $bb.rule
## [1] "pseudononint" "greedy"       "dynamic"      "rcostfixing"
##
## $break.at.first
## [1] FALSE
##
## $break.at.value
## [1] 1e+30
##
```

```
## $epsilon
##       epsb       epsd      epsel     epsint epsperturb   epspivot
##      1e-10      1e-09      1e-12      1e-07      1e-05      2e-07
##
## $improve
## [1] "dualfeas" "thetagap"
##
## $infinite
## [1] 1e+30
##
## $maxpivot
## [1] 250
##
## $mip.gap
## absolute relative
##    1e-11    1e-11
##
## $negrange
## [1] -1e+06
##
## $obj.in.basis
## [1] TRUE
##
## $pivoting
## [1] "devex"    "adaptive"
##
## $presolve
## [1] "none"
##
## $scalelimit
## [1] 5
##
## $scaling
## [1] "geometric"   "equilibrate" "integers"
##
## $sense
## [1] "maximize"
##
## $simplextype
## [1] "dual"   "primal"
##
## $timeout
## [1] 0
##
## $verbose
## [1] "neutral"
```

```
set.objfn(model, obj = c(3500,2500,2000))
```

*Define the Constraints*

Use the `add.constraint()` function to add the constraints.

- `xt`: The **non-zero coefficients** of the constraining equation. **All coefficients that are not explicitly declared are assigned to zero.**

- `type`: The **equality or inequality operator** to be used as comparator, as a character string (in double quotes.)
- `rhs`: The value of **the right-hand side of the inequality**, which *must be a constant*. Any variables that exist in the constraint must be moved over to the left side of this equation, if they exist.
- `indices`: The **indices of the values established by `xt`**, which need not be consecutive.

```
add.constraint(model,
               xt=c(500,300,400),
               type="<=", rhs=19500,
               indices=1:3)
add.constraint(model,
               xt=c(1,1,1),
               type="<=", rhs=36,
               indices=1:3)
add.constraint(model,
               xt=c(1,0,0),
               type="<=", rhs=20,
               indices=1:3)
add.constraint(model,
               xt=c(0,1,0),
               type="<=", rhs=10,
               indices=1:3)
add.constraint(model,
               xt=c(0,0,1),
               type="<=", rhs=18,
               indices=1:3)
```

*Define the Boundaries*

The upper and lower bounds are set using the `set.bounds()` function, in the arguments `lower` and `upper`, respectively.

```
set.bounds(model, lower = c(0,0,0))
```

*Solve and Inspect Results*

```
model
```

```
## Model name: Cargo - Wagons
##             C1    C2    C3
## Maximize  3500  2500  2000
## R1         500   300   400  <=  19500
## R2           1     1     1  <=     36
## R3           1     0     0  <=     20
## R4           0     1     0  <=     10
## R5           0     0     1  <=     18
## Kind       Std   Std   Std
## Type      Real  Real  Real
## Upper      Inf   Inf   Inf
## Lower        0     0     0
```

```
solve(model)
```

```
## [1] 0
```

To view the results, we can extract the output of the optimized objective function and the values of each of the variables under optimized conditions using the `get.objective()` and `get.variables()` functions, respectively.

```
get.variables(model)
```

```
## [1] 20 10  6
```

```
get.objective(model)
```

```
## [1] 107000
```

```
get.constraints(model)
```

```
## [1] 15400    36    20    10     6
```

## Sensitivity analysis

Perform the sensitivity analysis for the model solved.

```
get.primal.solution(model) # getting the primal solution
```

```
## [1] 107000  15400     36     20     10      6     20     10      6
```

The primal model is the model we have built and solved. This solution contains information in the form [**obj__value, constraints__values, decision__variables**].

```
get.basis(lprec = model, nonbasic = F) # get the optimal basis
```

```
## [1] -1 -8 -6 -7 -5
```

Dual variables (shadow prices):

```
get.dual.solution(model)
```

```
## [1]    1    0 2000 1500  500    0    0    0    0
```

The **shadow prices** are: [0, 2000, 1500, 500, 0]

*Check shadow prices*

If we reduce by one unity the value of b2 the optimal value of the objective function should reduce by 2000.

```r
set.rhs(model,c(19500,35,20,10,18))
solve(model)
```

```
## [1] 0
```

```r
get.objective(model)
```

```
## [1] 105000
```

```r
# pretty-printing the sensitivity analysis

if(require(dplyr)==FALSE)install.packages("dplyr")
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
if(require(tidyr)==FALSE)install.packages("tidyr")
```

```
## Loading required package: tidyr
```

```r
printSensitivityRHS <- function(model){
  options(scipen = 999)
  arg.rhs <- get.sensitivity.rhs(model)
  numRows <- length(arg.rhs$duals)
  symb <- c()
  for (i in c(1:numRows)) symb[i] <- paste("B", i, sep = "")

  rhs <- data.frame(rhs = symb, arg.rhs)

  rhs <- rhs %>%
  mutate(dualsfrom=replace(dualsfrom, dualsfrom < -1.0e4, "-inf")) %>%
  mutate(dualstill=replace(dualstill, dualstill > 1.0e4, "inf")) %>%
  unite(col = "Sensitivity",
        dualsfrom,
        rhs,
        dualstill ,
        sep = " <= ", remove = FALSE) %>%
  select(c("rhs","Sensitivity"))
  colnames(rhs)[1] <- c('Rhs')
  print(rhs)
}
printSensitivityObj <- function(model){
  options(scipen=999)
  arg.obj = get.sensitivity.obj(model)

  numRows <- length(arg.obj$objfrom)
  symb <- c()
  for (i in c(1:numRows)) symb[i] <- paste("C", i, sep = "" )

  obj <- data.frame(Objs = symb, arg.obj)

  obj<-
    obj %>%
    mutate(objfrom=replace(objfrom, objfrom < -1.0e4, "-inf")) %>%
    mutate(objtill=replace(objtill, objtill > 1.0e4, "inf")) %>%
    unite(col = "Sensitivity",
          objfrom, Objs, objtill ,
          sep = " <= ", remove = FALSE) %>%
    select(c("Objs","Sensitivity"))
    print(obj)
}
```

```
printSensitivityObj(model)
```

```
##   Objs       Sensitivity
## 1   C1 2000 <= C1 <= inf
## 2   C2 2000 <= C2 <= inf
## 3   C3    0 <= C3 <= 2500
```

```
printSensitivityRHS(model)
```

```
##   Rhs       Sensitivity
## 1  B1 -inf <= B1 <= inf
## 2  B2 30 <= B2 <= 46.25
## 3  B3      7 <= B3 <= 25
## 4  B4      0 <= B4 <= 15
## 5  B5 -inf <= B5 <= inf
## 6  B6 -inf <= B6 <= inf
## 7  B7 -inf <= B7 <= inf
## 8  B8 -inf <= B8 <= inf
```

We see the allowed range for each coefficient (objective function and rhs).

## Questions about LP

1. Can an LP model have more than one optimal solution. Is it possible for an LP model to have exactly two optimal solutions? Why or why not?

   ***risposta:***

No, it's impossible for a Linear Programming model to have exactly two optimal solutions. A LP model may have zero, one, or an infinite number of optimal solutions. A LP problem has more than 1 optimal solution when the curve intersects the feasible region along an edge and in that all the points of that edge will give the optimal solutions.

2. Are the following objective functions for an LP model equivalent? That is, if they are both used, one at a time, to solve a problem with exactly the same constraints, will the optimal values for $x_1$, $x_2$ and $x_3$ be the same in both cases? Why or why not?

$$\max 2x_1 + 3x_2 - x_3$$

$$\min -2x_1 - 3x_2 + x_3$$

***risposta:***

Yes, an LP model is equivalent for both objective functions.

This is because the constraints decide the feasible region and as long as the they remain the same so does the feasible region, and the corner point (x1, x2, x3) that maximizes or minimizes the objective function.

3. Which of the following constraints are not linear or cannot be included as a constraint in a linear programming problem?

a. $2x_1 + x_2 - 3x_3 \geq 50$

b. $2x_1 + \sqrt{x_2} \geq 60$

c. $4x_1 - \frac{1}{2}x_2 = 75$

d. $\frac{3x_1+2x_2x_1-3x_3}{x_1+x_2+x_3} \leq 0.9$

e. $3x_1^2 + 7x_2 \leq 45$

***risposta:***

Linear constraint: It is a mathematical expression in which the decision variable are added or subtracted and the result of the expression is greater than or equal, less than or equal , or equal to the right hand side value. If all the terms of a constraint are of the first order, the constraint is said to be linear.

So b, d, e are not linear.