Stochastic Control and Optimization

Introduction to Optimization

ASSEMBLING AND TESTING COMPUTERS

- The PC Tech company assembles and then tests two models of computers, Basic and XP.
- For the coming month, the company wants to decide how many of each model to assembly and then test.
- No computers are in inventory from the previous month, and because these models are going to be changed after this month, the company doesn't want to hold any inventory after this month

ASSEMBLING AND TESTING COMPUTERS

- It believes the most it can sell this month are 600 Basics and 1200 XPs. Each Basic sells for $300 and each XP sells for $450.
- The cost of component parts for a Basic is $150; for an XP it is $225.
- Labor is required for assembly and testing. There are at most 10,000 assembly hours and 3000 testing hours available.
- Each labor hour for assembling costs $11 and each labor hour for testing costs $15.
- Each Basic requires five hours for assembling and one hour for testing, and each XP requires six hours for assembling and two hours for testing.

ASSEMBLING AND TESTING COMPUTERS

- PC Tech wants to know how many of each model it should produce (assemble and test) to maximize its net profit, but it cannot use more labor hours than are available, and it does not want to produce more than it can sell.

- Find the best mix of computer models that stays within the company's labor availability and maximum sales constraints.

Problem Formulation

- Choose

$$x_1, x_2$$ (units of basic, XP to be produced)

- to maximize

$$80x_1 + 129x_2$$ (total net profit)

- subject to

$$5x_1 + 6x_2 \leq 10000$$ (assembly hour constraint)

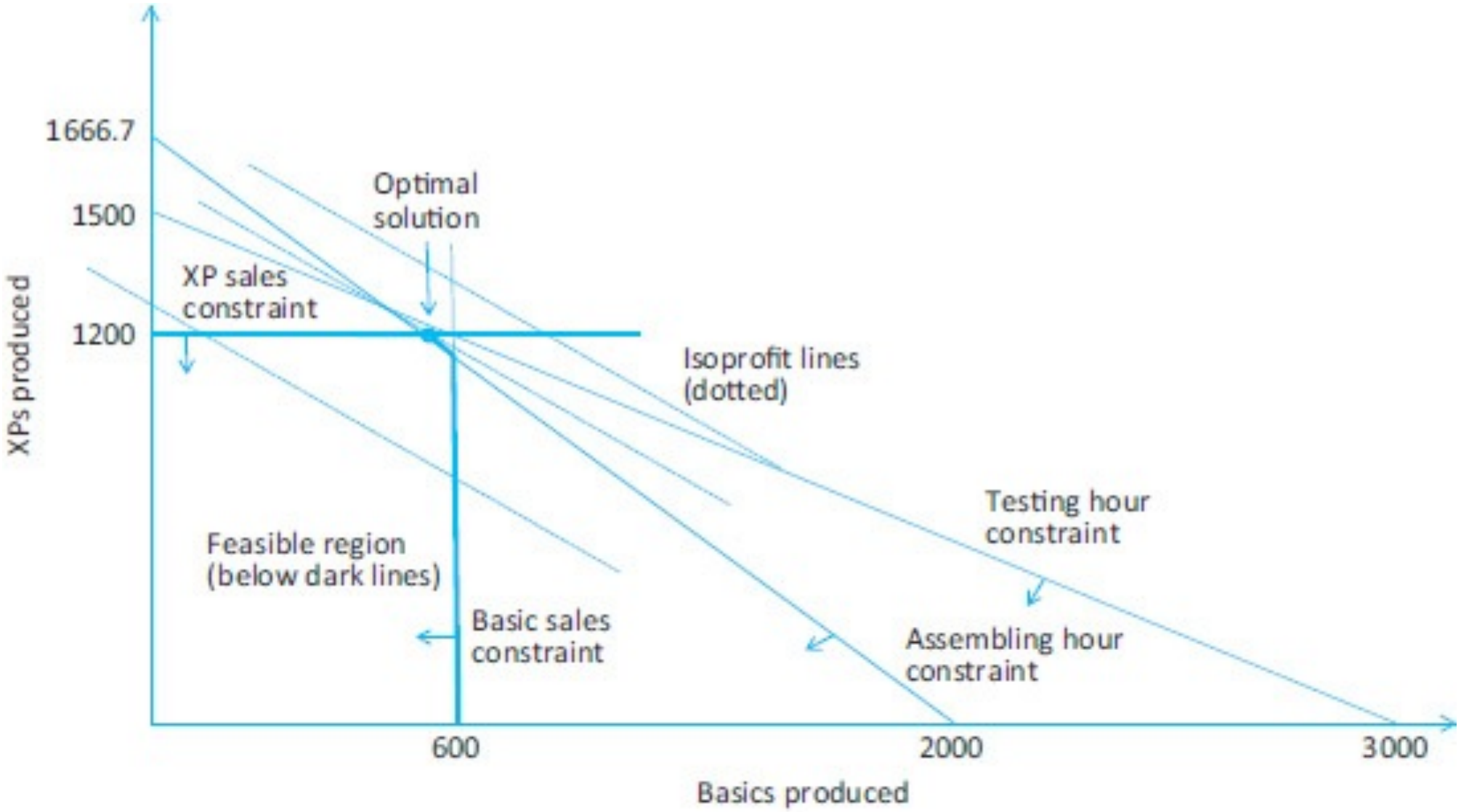
$$x_1 + 2x_2 \leq 3000$$ (testing hours constraint)

$$x_1 \leq 600$$ (demand constraint for Basic)

$$x_2 \leq 1200$$ (demand constraint for XP)

$$x_1, x_2 \geq 0$$ (only nonnegative amounts can be produced)

A Graphical Solution



A Graphical Solution

- The constraints together dictate the feasible region. That is, the set of x1,x2 that satisfy all the constraints
- To see which feasible point maximizes the objective, it is useful to draw a sequence of lines where, for each, the objective is a constant.
- A typical line is of the form 80 x1 + 129 x2 = c, where c is a constant. Any such line has slope –80/129 = –0.620

Solving using R

- Graphical solutions are intuitive, but can only handle toy problems with two decision variables.
- Before we use R, we will have to first write our problem in this format:

  minimize: $c' * x$

  subject to: $A*x <= b, x>=0$

- What are c, A and b?

Solving using R

- A **Linear programming** problem written in the form

  minimize: c' * x

  subject to: A*x <= b, x >= 0

  can be solved in R simply by calling the function **lp** (in package lpSolve)

- You will have to install the package lpSolve using the command install.packages("lpSolve") and then load the library using the command library("lpSolve") before you can use solveLP.

- For eg. first define c, A and b and then call the lp function

```
> c<-c(80,129)
> A<-matrix(c(5,1,1,0,6,2,0,1),4,2)
> dir<-c("<=","<=","<=","<=")
> b<-c(10000,3000,600,1200)
> s<-lp("max",c,A,dir,b)
> s$solution
[1]  560 1200
> s$objval
[1] 199600
```

"dir" is a list of inequality directions. A vector of "<=" in our case.

This will return the best possible production mix subject to the constraints.

- Remember: you can pull help on any R function by either using "?<function name>". Try reading through "? lp" and see how you can get the maximum profit as well.

lp function

```
lp (direction = "min", objective.in, const.mat, const.dir, const.rhs,
        transpose.constraints = TRUE, int.vec, presolve=0, compute.sens=0,
        binary.vec, all.int=FALSE, all.bin=FALSE, scale = 196, dense.const,
        num.bin.solns=1, use.rw=FALSE)
```

Arguments

`direction`	Character string giving direction of optimization: "min" (default) or "max."
`objective.in`	Numeric vector of coefficients of objective function
`const.mat`	Matrix of numeric constraint coefficients, one row per constraint, one column per variable (unless transpose.constraints = FALSE; see below).
`const.dir`	Vector of character strings giving the direction of the constraint: each value should be one of "<," "<=," "=," "==," ">," or ">=". (In each pair the two values are identical.)
`const.rhs`	Vector of numeric values for the right-hand sides of the constraints.
`transpose.constraints`	By default each constraint occupies a row of const.mat, and that matrix needs to be transposed before being passed to the optimizing code. For very large constraint matrices it may be wiser to construct the constraints in a matrix column-by-column. In that case set transpose.constraints to FALSE.
`int.vec`	Numeric vector giving the indices of variables that are required to be integer. The length of this vector will therefore be the number of integer variables.
`presolve`	Numeric: presolve? Default 0 (no); any non-zero value means "yes." Currently ignored.
`compute.sens`	Numeric: compute sensitivity? Default 0 (no); any non-zero value means "yes."
`binary.vec`	Numeric vector like int.vec giving the indices of variables that are required to be binary.
`all.int`	Logical: should all variables be integer? Default: FALSE.
`all.bin`	Logical: should all variables be binary? Default: FALSE.
`scale`	Integer: value for lpSolve scaling. Details can be found in the lpSolve documentation. Set to 0 for no scaling. Default: 196
`dense.const`	Three column dense constraint array. This is ignored if const.mat is supplied. Otherwise the columns are constraint number, column number, and value; there should be one row for each non-zero entry in the constraint matrix.
`num.bin.solns`	Integer: if all.bin=TRUE, the user can request up to num.bin.solns optimal solutions to be returned.
`use.rw`	Logical: if TRUE and num.bin.solns > 1, write the lp out to a file and read it back in for each solution after the first. This is just to defeat a bug somewhere. Although the default is FALSE, we recommend you set this to TRUE if you need num.bin.solns > 1, until the bug is found.

- We just introduced optimization, one of the most powerful and flexible methods of quantitative analysis.
- The specific type of optimization we discussed is linear programming (LP).
- LP is used in many organizations, often on a daily basis, to solve a variety of problems:
    - Risk management
    - Labor scheduling
    - Inventory management
    - Selection of advertising media
    - Bond trading, etc.

- All optimization models have several common elements:
  - Decision variables, the variable whose values the decision maker is allowed to choose. The values of these variables determine such outputs as total cost, revenue, and profit.
  - An objective function (objective, for short) to be optimized – minimized or maximized.
  - Constraints that must be satisfied. They are usually physical, logical, or economic restrictions, depending on the nature of the problem.
    - Non-negativity constraint is very common. It states that changing cells must have nonnegative (zero or positive) values. Non-negativity constraints are usually included for physical reasons. For example, it is impossible to produce a negative number of automobiles.

- There are two basic steps in solving an optimization problem:
  - Model development step
  - Optimization step

- Model development is where most of your efforts go in.
- To optimize means that you must systematically choose the values of the decision variables that make the objective as large (for maximization) or small (for minimization) as possible and cause all of the constraints to be satisfied.

- Any set of values of the decision variables that satisfies all of the constraints is called a feasible solution.
- The set of all feasible solutions is called the feasible region.
- An infeasible solution is a solution that violates at least one constraint.
- The desired feasible solution is the one that provides the best value – minimum for a minimization problem, maximum for a maximization problem – for the objective. This solution is called the optimal solution.
- Much of the published research has been about the optimization step.
    - One algorithm for searching through the feasible region is called the **simplex method**.

Binding and Non-binding constraints

- Of all the inequality constraints, some are satisfied exactly and others are not.
- An inequality constraint is binding if the solution makes it an equality. Otherwise, it is non-binding.
- The positive difference between the two sides of the constraint is called the slack.

- A trading company is looking for a way to maximize profit per transportation of their goods. The company has a train available with 3 wagons. When stocking the wagons they can choose between 4 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?
- The following constraints have to be taken in consideration;
  - Weight capacity per train wagon
  - Volume capacity per train wagon
  - Limited availability per cargo type

TRAIN WAGON	WEIGHT CAPACITY (TONNE)	SPACE CAPACITY (M^2)
w1	10	5000
w2	8	4000
w3	12	8000

CARGO TYPE	AVAILABLE (TONNE)	VOLUME (M^2)	PROFIT (PER TONNE)
c1	18	400	2000
c2	10	300	2500
c3	5	200	5000
c4	20	500	3500

```
c<-rep(c(2000,2500,5000,3500),3)

A=matrix(0,10,12)
A[1,1:4]<-1
A[2,5:8]<-1
A[3,9:12]<-1
A[4,1:4]<-c(400,300,200,500)
A[5,5:8]<-c(400,300,200,500)
A[6,9:12]<-c(400,300,200,500)
A[7:10,1:4]<-diag(4)
A[7:10,5:8]<-diag(4)
A[7:10,9:12]<-diag(4)

dir<-rep("<=",10)

b<-c(10,8,12,5000,4000,8000,18,10,5,20)

s=lp("max",c,A,dir,b,compute.sens = 1)
```

```
> c
 [1] 2000 2500 5000 3500 2000 2500 5000 3500 2000 2500 5000 3500

> A
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
 [1,]    1    1    1    1    0    0    0    0    0     0     0     0
 [2,]    0    0    0    0    1    1    1    1    0     0     0     0
 [3,]    0    0    0    0    0    0    0    0    1     1     1     1
 [4,]  400  300  200  500    0    0    0    0    0     0     0     0
 [5,]    0    0    0    0  400  300  200  500    0     0     0     0
 [6,]    0    0    0    0    0    0    0    0  400   300   200   500
 [7,]    1    0    0    0    1    0    0    0    1     0     0     0
 [8,]    0    1    0    0    0    1    0    0    0     1     0     0
 [9,]    0    0    1    0    0    0    1    0    0     0     1     0
[10,]    0    0    0    1    0    0    0    1    0     0     0     1

> b
 [1]   10    8   12 5000 4000 8000   18   10    5   20
```

```
> s$solution
 [1]  0  5  5  0  0  0  0  8  0  0  0 12
> s$objval
[1] 107500
```

Solution	c1	c2	c3	c4
w1	0	5	5	0
w2	0	0	0	8
w3	0	0	0	12