Table of Contents

1 Elements of Linear Programming

Linear Programming is one of the simpler class optimization models. Specifically it is a form of constrained optimization, where we attempt to find the best point of a function while respecting various constraints. In the real world, these constraints are usually related to limited resources that we are trying to optimize around.

There are four elements of a constrained optimization problem:

- Decision variables
  - The unknown values you're trying to determine
  - Can be thought of as different quantities of a product to produce, amount of money to spend on different options, etc.
- The objective function
  - The mathematical expression that combines the decision variables to express our goal
  - We are trying to either minimize (e.g. an expense) or maximize (e.g. profit) this function
- Constraints
  - The mathematical expression representing limitations, requirements, or restrictions imposed on our solution
- Variable bounds

Building our optimization model requires us to identify these components and create mathematical expressions out of our objective function and constraints, which are linear functions of the decision variables. All variables are continuous, if a variable is required to take an integer value then this requires the use of *integer programming.* No you can't just use LP and round up to an integer, your results may not be optimal, and you will feel bad.

2 Example: Product Machining

A company produces two types of products, Product A and Product B, using two machines, Machine X and Machine Y.

Each unit of Product A requires 3 hours on Machine X and 2 hours on Machine Y, while each unit of Product B requires 1 hour on Machine X and 4 hours on Machine Y.

The company has 120 hours available on Machine X and 100 on Machine Y per week.

Additionally, the company must satisfy the following constraints:

- The company cannot produce more than 50 units of Product A per week.
- The profit per unit of Product A is $50, and the profit per unit of Product B is $40. The company wishes to determine how many units of each product to produce weekly to maximize total profit.

3 LP Model Formulation

3.1 Decision Variables

First we need to identify the decision variables. These are the variables that we can control, but don't know the optimum values for. In the problem above, we can formulate those variables as follows:

- x_1 = the number of Product A produced in a week
- x_2 = the number of Product B produced in a week

3.2 Objective Function

Next we need to determine our *objective function*. In this example, we are asked to maximize profit. The convention for representing the value of our function is *Z*, so using the provided profit per unit we can express our function as:

Maximize $Z = 50x_1 + 40x_2$

3.3 Constraints

Modeling constraints can be challenging. After determining our objective function, we need to scan the prompt and mathematically account for all other constraints on our problem.

The company cannot produce more than 50 units of Product A per week:

$x_1 \leq 50$

Product A requires 3 hours on Machine X, while Product B requires 1 hour. There are only 120 hours available on Machine X per week:

$3x_1 + x_2 \leq 120$

Product A requires 2 hours on Machine Y, while Product B requires 4 hours. There are only 100 hours available on Machine Y per week:

$2x_1 + 4x_2 \leq 100$

You can't produce negative products. If you can, this math can't help you. Seek a physicist and/or spiritual advisor.

$x_1, x_2 \geq 0$

4 Graphically Solving LP

A basic way to solve and demonstrate a linear programming problem is to graph it. This is only possible if your problem has two decision variables, otherwise we can't represent the problem as a linear graph.

4.1 Determine the Feasible Region

First we will find our feasible region by plotting the line for each of our constraints. Treat all inequalities as equals when drawing your lines because the inequalities will be represented by shading in a specific direction.

Plotting these lines requires a lot of advanced middle-school algebra that you probably don't remember. When two variables are present, solve the equation twice while plugging in a zero for each variable. This will give you two points to create the required line.

Plot Constraint 1:

Plot a line at $x_1 = 50$, feasible region is to the left of this line.

Plot Constraint 2:

$3x_1 + x_2 \leq 120$

First point: $3x_1 + 0 = 120$ -> $x_1 = 40, x_2 = 0$, point (40,0)

Second point: $0 + x_2 = 120$ -> $x_1 = 0, x_2 = 120$, point (0,120)

Plot Constraint 3

This time skipping algebra, just like I did in middle school. This will probably come back to haunt me.

$2x_1 + 4x_2 \leq 100$

First point: $x_1 = 50, x_2 = 0$, point (50,0)

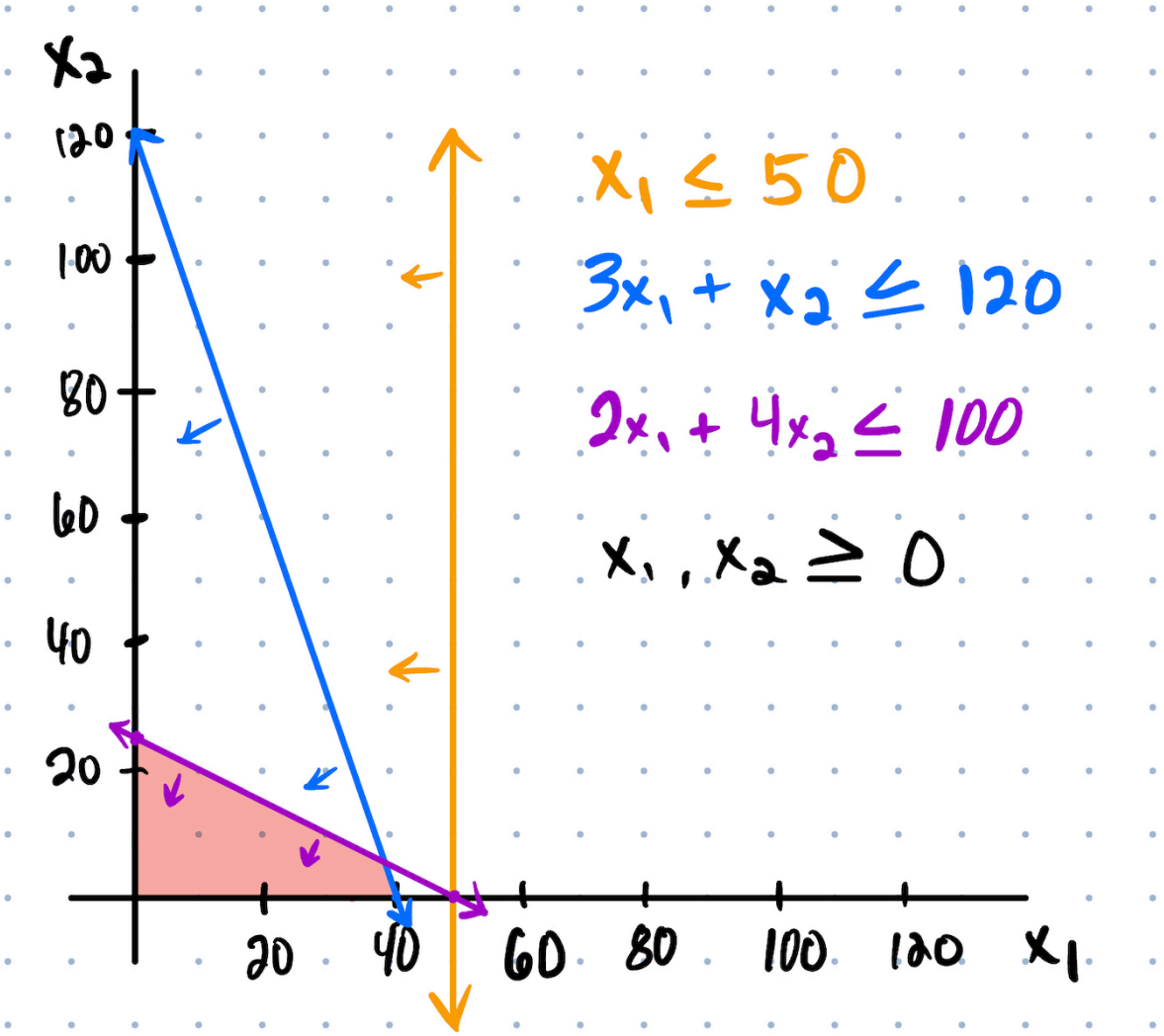
Second point: $x_1 = 0, x_2 = 25$, point (0,25)

Plot Constraint 4

$x_1, x_2 \geq 0$

The feasible region will be in the first quadrant, representing all positive values.

4.1.1 Graph Feasible Region

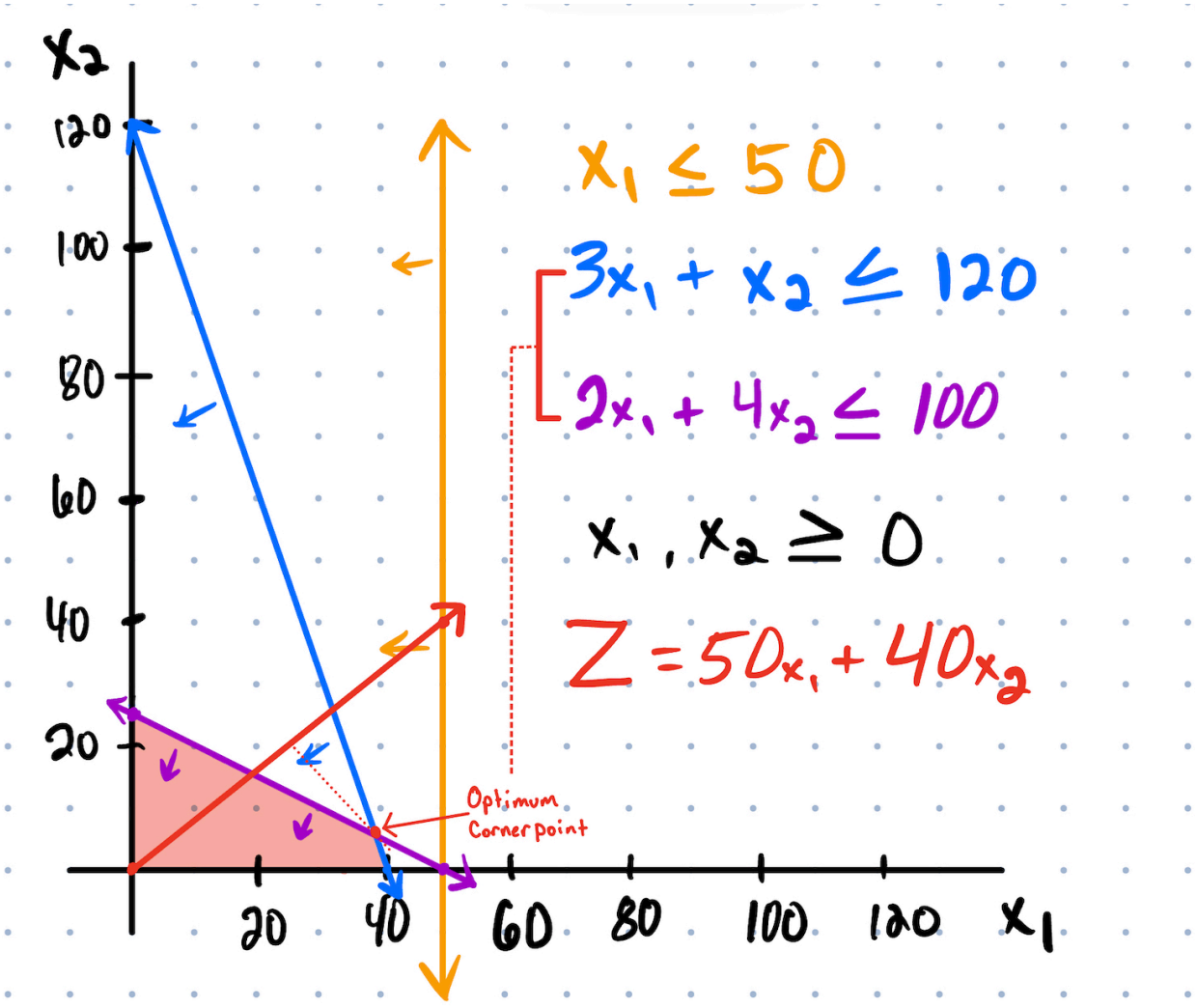

x_2
120
100
80
60
40
20
20
40
60
80
100
120
x_1
$x_1 \leq 50$
$3x_1 + x_2 \leq 120$
$2x_1 + 4x_2 \leq 100$
$x_1, x_2 \geq 0$

4.2 Plot the Objective Function and find Cornerpoints

Our objective function represents our goal that we want to optimize for. We want to find the most optimal point within our feasible area that maximizes (or sometimes minimizes) this function.

In our feasible region, the external edges intersect at cornerpoints. Unless the slope of the objective function is exactly the same as one of the edges, the most optimal solution will always be one of the cornerpoints. If we plot our objective function, we can see which cornerpoint maximizes the function.

Plot our objective function $Z = 50x_1 + 40x_2$ by finding the line between (0,0) and (50,40).



4.3 Solve for the Optimum Values

The cornerpoint we found is at the intersection of two of the constraints we defined above. We can solve for that intersection to get our optimum values for each product.

Solve the intersection of the two constraints

$3x_1 + x_2 \leq 120$

$2x_1 + 4x_2 \leq 100$

Solve the second equation for x_1 in terms of x_2

$x_1 = 50 - 2x_2$

Plug this x_1 in to the first equation

$3(50 - 2x_2) + x_2 = 120 \Rightarrow 150 - 6x_2 + x_2 = 120 \Rightarrow 150 - 5x_2 = 120$

$-5x_2 = -30$

First optimal value: $x_2 = 6$

Solve for second optimal value $3x_1 + (6) = 120$

Second optimal value: $x_1 = 38$

4.4 Solve the Objective Function

Now we can plug our two optimum values into the objective function and find our optimal solution:

$Z = 50x_1 + 40x_2$

$Z = 50(38) + 40(6) = \$2140$

The company should produce 38 units of Product A and 6 units of Product B to achieve a maximum profit of $2,140

5 The Simplex Method

The simplex method is a more tedious and difficult way for humans to find linear programming solutions because it intended for computers. Lucky for us, we get to experience this painful process the old-fashioned way.

This method involves the construction of a simplex tableau, which is a fancy table that contains the same types of objective functions and constraints as above. In fact, we will demonstrate this method with the same example.

5.1 Simplex Preparation

In order to align our function and constraints into a table, we need to do a few preparatory steps.

First we need to move all of our variables in our objective function to the left hand side and set it equal to zero. Taking our objective function above:

$$Z = 50x_1 + 40x_2$$

We will rewrite it as:

$$Z - 50x_1 - 40x_2 = 0$$

Then we take our constraints, change all inequalities to equals, and introduce slack variables that help us align our table. Now, this doesn't make sense yet, but it might soon!

$x_1 \leq 50$ -> $x_1 + s_1 = 50$

$3x_1 + x_2 \leq 120$ -> $3x_1 + x_2 + s2 = 120$

$2x_1 + 4x_2 \leq 100$ -> $2x_1 + 4x_2 + s3 = 100$

5.2 Simplex Tableau

5.2.1 Setup and Finding Pivot

Now we take our new strange equations and fill them into the simplex tableau, aligning the value for each variable in the appropriate column. Note the final column RHS which represents the right-hand side of the equation.

Variable	x1	x2	s1	s2	s3	Z	RHS
s1	1	0	1	0	0	0	50
s2	3	1	0	1	0	0	120
s3	2	4	0	0	1	0	100
Z	-50	-40	0	0	0	0	0

Next we need to determine our "pivot," which will be the item in this table that focuses our row reduction. Finding this pivot is a multi-stage process:

- In the Z row, find the lowest number which will become the pivot column
  - For us this is -50, so x1 becomes our pivot column
- Divide all non-Z RHS values by their respective values in the pivot column
  - $\frac{50}{1}, \frac{120}{3}, \frac{100}{2}$
- Take the lowest value of these as your pivot. If there's a tie, just pick one.
  - For us the s2 row is the smallest ($\frac{120}{3} = 40$)
  - The pivot is 3
- We want to reduce this pivot to 1, performing whatever operation is necessary across the entire row
  - In our case this entails dividing the entire row by 3
- Rewrite the tableau with this new operation, leaving other rows as-is

$\frac{1}{3}R_2 => R_2$

Variable	x1	x2	s1	s2	s3	Z	RHS
s1	1	0	1	0	0	0	50
x1	1	1/3	0	1/3	0	0	40
s3	2	4	0	0	1	0	100
Z	-50	-40	0	0	0	0	0

5.2.2 Row reductions

Our next goal is to reduce the values above and below our same pivot to zero. For each of those values, determine what we can multiply our pivot by in order to add it to the value and equal zero. We will do this for each row.

For example, the value above the pivot is 1. To reduce it, we multiply our pivot by -1 and add it to the value in the row above. We'll actually do this across the entire row, multiple values in our pivot row by -1 and adding that to the entire row above.

This same process is repeated for the row below the pivot for whatever multiple is needed to reduce the value in that column to zero, so we have 3 tableau transformations to make:

1. $-1R_2 + R_1 => R_1$
2. $-2R_2 + R_3 => R_3$
3. $50R_2 + R_4 => R_4$

Variable	x1	x2	s1	s2	s3	Z	RHS
s1	0	-1/3	1	-1/3	0	0	10
x1	1	1/3	0	1/3	0	0	40
s3	0	3 1/3	0	-2/3	1	0	20
Z	0	-23 1/3	0	16 2/3	0	0	2000

If we have any negative numbers in the Z row, we need to repeat this process with a new pivot column if necessary.

- Our lowest value is -23 1/3 (that can't be right) so x2 is our pivot column
- Find the smallest positive value after dividing RHS by the pivot column
  - -30, 13 1/3, 6
- The pivot number is 3 1/3 at x2, s3
  - This implies we multiply the entire row by 3/10s. I have little confidence that this is correct, but we press on

$\frac{3}{10}R_2 => R_2$

Variable	x1	x2	s1	s2	s3	Z	RHS
s1	0	-1/3	1	-1/3	0	0	10
x1	1	1/3	0	1/3	0	0	40
s3	0	1	0	-1/5	3/10	0	6
Z	0	-23 1/3	0	16 2/3	0	0	2000

Row reductions:

1. $\frac{1}{3}R_3 + R_1 => R_1$
2. $-\frac{1}{3}R_3 + R_2 => R_2$
3. $23\frac{1}{3}R_3 + R_4 => R_4$

Variable	x1	x2	s1	s2	s3	Z	RHS
s1	1/3	0	1	-6/15	1/10	0	12
x1	1	0	0	6/15	-1/10	0	38

x2	0	1	0	-1/5	3/10	0	6
Z	0	0	0	12	7	0	2140

Once we have no negatives in the Z row, we are finished.

5.3 Simplex Interpretation

By some miracle, we achieved the same x_1 and x_2 values as our graphical solution as well as the same maximum Z value of $2,140.

We also have a slack value of 12, which represents the unused units of our constraint on producing no more than 50 units of Product A in one week. Our optimum value for Product A is 38, but we have the capacity to produce 12 more.

6 Linear Programming with Python

6.1 Using the PuLP Library

To solve our same LP optimization problem in Python, we will be using an open-source library called PuLP. First we ensure PuLP is installed in our Python environment and begin by importing the module.

In [5]:
```
from pulp import *
```

LpProblem is the PuLP class that creates the actual problem with the parameters of a problem name and the *sense* of the problem, which is either LpMaximize or LpMinimize. We will use the former as this is a maximization problem.

In [10]:
```
prob = LpProblem("ProductDecision", LpMaximize)
x1 = LpVariable('x_1', lowBound = 0)
x2 = LpVariable('x_2', lowBound = 0)
```

Now we add our objective function and constraints to the `prob` LP problem we created. The objective function is added first, then we'll add our three constraints. If we print our `prob` at this point, we can see a summary of our problem so far.

In [11]:
```
# Objective function
prob += 50*x1 + 40*x2, "Obj"

# Constraints
prob += x1 <= 50
prob += 3*x1 + 1*x2 <= 120
prob += 2*x1 + 4*x2 <= 100
print(prob)
```

```
ProductDecision:
MAXIMIZE
50*x_1 + 40*x_2 + 0
SUBJECT TO
_C1: x_1 <= 50

_C2: 3 x_1 + x_2 <= 120

_C3: 2 x_1 + 4 x_2 <= 100

VARIABLES
x_1 Continuous
x_2 Continuous
```

Using `.solve()`, PuLP will actually execute the LP solution. Before looking at our answer, we can review LpStatus to see if we've achieved an optimal solution.

In [12]:
```
prob.solve()
print("status: " + LpStatus[prob.status])
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/jdmini/datapad/lib/python3.9/site-packages/pu
lp/solverdir/cbc/osx/64/cbc /var/folders/1n/nvr79nb55tz9j4lsbrw6hsf
80000gn/T/f93097c6566d4abcbe20be70fc0b47d6-pulp.mps max timeMode el
apsed branch printingOptions all solution /var/folders/1n/nvr79nb55
tz9j4lsbrw6hsf80000gn/T/f93097c6566d4abcbe20be70fc0b47d6-pulp.sol (
default strategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 8 COLUMNS
At line 16 RHS
At line 20 BOUNDS
At line 21 ENDATA
Problem MODEL has 3 rows, 2 columns and 5 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 3 (-1) rows, 2 (0) columns and 4 (-1) elements
```

Seeing we have an optimal solution, we can write a small loop to print each of the optimal variables and objective value. The optimal values for Product A and Product B (x_1 and x_2) match the values found with the graphing and simplex methods above. We also see the same maximum objective value of $2140.

In [13]:
```
for variable in prob.variables():
     print("{}* = {}".format(variable.name, variable.varValue))

print(value(prob.objective))
```

```
x_1* = 38.0
x_2* = 6.0
2140.0
```

6.2 Different LP Outcomes

Using `LpStatus` above, we found that this example had an optimal solution. However, there are four potential outcomes than a solution can have:

- One optimal solution
- Multiple optimal solutions
  - On a graph, this could be represented by an edge of a feasible region being exactly perpendicular to the objective function line
- An unbounded solution: the optimal value can reach into infinity
  - If all constraints do not encapsulate a "region", then the objective value is theoretically infinite
  - Usually this means we've missed a constraint while designing our model
- Solution is infeasible: there is no feasible region
  - If the constraints are such that no feasible region is created, then the solution is infeasible

7 Sensitivity Analysis

7.1 Theory

Up until now we have focused on finding the optimal solution, which is a useful first step. However, we can look more closely at our decision variables to see how they contribute to that solution and what happens if any are changed. There are a few terms that are important to this type of analysis:

Binding constraints: These are constraints that would directly affect the optimal solution if changed as they limit the feasible region. Binding constraints have zero slack.

Slack: The slack of a constraint is the difference between the right-hand side (RHS) and left-hand side (LHS) of the constraint equation at the optimal solution. This tells us the amount of unused capacity for a given constraint. If the slack is greater than zero, the constraint is non-binding and is not limiting the optimal solution.

Shadow price: For a binding constraint, the shadow price represents how much the objective value will change when the RHS of the constraint changes.

- This is essentially the marginal value of the constraint, as it affects the value of the objective function per unit of increase (in a binding constraint in a maximization problem)
- The shadow price is zero for non-binding constraints, we can consider these resources to be in surplus (or "free goods" if you're an economist, but we don't care about them)

Reduced cost: It's possible for the optimum solution for variables to be zero. As an example, it may not be as profitable to produce any of Product A and focus entirely on Product B. Reduced cost helps us determine what would need to change about that variable in order for it to make the cut of our optimal solution.

- Specifically, it tells us how much the coefficient of that variable in the objective function needs to increase for maximization problems, or decrease for minimization problems, in order to be part of the optimal solution.
- If in our example objective function $Z = 50x_1 + 40x_2$ we found that the optimal solution was to have zero units of x_2 (Product B), a reduced cost of 10 would mean that the coefficient of x_2 would need to increase from 40 to 50 for it to be greater than zero in our optimal solution.

7.2 Sensitivity Analysis in PuLP

Continuing our example above, we can report the slack and shadow price still contained in the `prob` solution:

In [15]:
```
for name, c in prob.constraints.items():
    print("\n", name, ":", c, ", Slack=", c.slack, ", Shadow Price=",
```

```
 _C1 : x_1 <= 50 , Slack= 12.0 , Shadow Price= -0.0

 _C2 : 3*x_1 + x_2 <= 120 , Slack= -0.0 , Shadow Price= 12.0

 _C3 : 2*x_1 + 4*x_2 <= 100 , Slack= -0.0 , Shadow Price= 7.0
```

We can also report any reduced costs:

In [16]:
```
for v in prob.variables():
    print ("\n", v.name, "=", v.varValue, ", Reduced Cost=", v.dj)
```

```
 x_1 = 38.0 , Reduced Cost= 0.0

 x_2 = 6.0 , Reduced Cost= 0.0
```

7.3 Slack and Shadow Price Interpretation

These results show us that the constraints on weekly production hours for both Machine X and Machine Y are binding while the constraint on making less than 50 units of Product A per week is not.

The slack for the constraint on Product A is 12 is consistent with our findings using the simplex method and shows that we have a surplus of 12 units for this constraint.

The shadow price of the Machine X constraint reveals that an extra $12 of weekly profit could be achieved if an extra hour of usage could be made available. An extra hour per week on Machine Y would generate an additional $7 of profit per week.

8 Linear Programming Minimization

So far we've only worked with a maximization problem, in our example this involved trying to maximize profit. However, when we are called upon to minimize concerns like cost or risk we need to achieve a minimization solution.

8.1 Minimization and Reduced Cost

Performing minimization in Python is almost an identical process as the maximization problem above, with the small difference of using the `LpMinimize` sense when the problem is defined. The major differences will be in interpreting the output once we find our solution.

In [16]:
```
from pulp import *
minprob = LpProblem("MinimizeDecision", LpMinimize)
x1 = LpVariable('x_1', lowBound = 0)
x2 = LpVariable('x_2', lowBound = 0)
x3 = LpVariable('x_3', lowBound = 0)
```

As before, we add our objective function and constraints to the LpProblem we created. Pay attention to the inequality signs, in minimization problems they are usually greater-than instead of lesser-than relationships.

Printing the problem shows our standard LP definition.

In [17]:
```
# Objective function
minprob += 3*x1 + 4*x2 + 5*x3, "Obj"

# Constraints
minprob += 2*x1 + 1*x3 >= 8
minprob += 1*x2 + 1*x3 >= 6
minprob += 6*x1 + 8*x2 >= 48
print(minprob)
```

```
MinimizeDecision:
MINIMIZE
3*x_1 + 4*x_2 + 5*x_3 + 0
SUBJECT TO
_C1: 2 x_1 + x_3 >= 8

_C2: x_2 + x_3 >= 6

_C3: 6 x_1 + 8 x_2 >= 48

VARIABLES
x_1 Continuous
x_2 Continuous
x_3 Continuous
```

Solving the problem and printing the status shows that we do have an optimal solution.

In [18]:
```
minprob.solve()
print("status: " + LpStatus[minprob.status])
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/jdmini/datapad/lib/python3.9/site-packages/pulp
/solverdir/cbc/osx/64/cbc /var/folders/1n/nvr79nb55tz9j4lsbrw6hsf8000
0gn/T/9495236d618e4513bcfe23fed7246b71-pulp.mps timeMode elapsed bran
ch printingOptions all solution /var/folders/1n/nvr79nb55tz9j4lsbrw6h
sf80000gn/T/9495236d618e4513bcfe23fed7246b71-pulp.sol (default strate
gy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 8 COLUMNS
At line 18 RHS
At line 22 BOUNDS
At line 23 ENDATA
Problem MODEL has 3 rows, 3 columns and 6 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 3 (0) rows, 3 (0) columns and 6 (0) elements
0  Obj 0 Primal inf 16 (3)
3  Obj 34.909091
Optimal - objective value 34.909091
Optimal objective 34.90909091 - 3 iterations time 0.002
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.0
1

status: Optimal
```

Print the optimal variables with a simple loop, then print the optimal objective value. In this case, we have an optimal minimal value of 34.9.

In [19]:
```
for variable in minprob.variables():
    print("{}* = {}".format(variable.name, variable.varValue))

print(value(minprob.objective))
```

```
x_1* = 2.9090909
x_2* = 3.8181818
x_3* = 2.1818182
34.909090899999995
```

8.2 Reduced Cost Interpretation

Here we see that each of our optimal variables has a reduced cost of zero. This means that each of our variables are already at their optimum level, and increasing any of them would not result in a decrease of our objective value.

In [20]:
```
for v in minprob.variables():
    print ("\n", v.name, "=", v.varValue, ", Reduced Cost=", v.dj)
```

```
 x_1 = 2.9090909 , Reduced Cost= 0.0

 x_2 = 3.8181818 , Reduced Cost= 0.0

 x_3 = 2.1818182 , Reduced Cost= 0.0
```

9 LP Investment Portfolio Example

A trust officer at ABC Bank has a collection of bonds and needs to decide how to invest $500,000 in them in order to maximize the annual return.

Bond	Annual Return	Maturity	Risk	Tax-Free
A	9.5%	Long	High	Yes
B	8.0%	Short	Low	Yes
C	9.0%	Long	Low	No
D	9.0%	Long	High	Yes
E	9.0%	Short	High	No

The officer wants to invest at least 50% of the money in short-term issues and no more than 50% in high-risk issues. Also, at least 25% of the funds should go into tax-free investments.

In this problem, we can easily identify our decision variables as each individual bond since we are trying to determine how much to invest in each. Since we are trying to maximize our return, this is a maximization problem.

In [34]:
```
#from pulp import *

bondprob = LpProblem("InvestmentDecision",LpMaximize)
x1 = LpVariable('A', lowBound = 0)
x2 = LpVariable('B', lowBound = 0)
x3 = LpVariable('C', lowBound = 0)
x4 = LpVariable('D', lowBound = 0)
x5 = LpVariable('E', lowBound = 0)
```

The objective function here is the annual return for each bond expressed as a decimal. The scenario expresses our constraints as percentages. Knowing our investment is $500,000, we can convert those percentages to real values.

By referencing our table, we can determine which of our bond variables to include in each constraint. Remember that we need one final constraint that sums all of our decision variables to our total investment.

In [35]:
```
# Objective function
bondprob += .095*x1 + .08*x2 + .09*x3 + .09*x4 + .09*x5, "Obj"

# Constraints
bondprob += x2 + x5 >= 250000, "Short-term"
bondprob += x1 + x4 + x5 <= 250000, "High-risk"
bondprob += x1 + x2 + x4 >= 125000, "Tax-free"
bondprob += x1 + x2 + x3 + x4 + x5 == 500000, "Investment"
print(bondprob)
```

```
InvestmentDecision:
MAXIMIZE
0.095*A + 0.08*B + 0.09*C + 0.09*D + 0.09*E + 0.0
SUBJECT TO
Short_term: B + E >= 250000

High_risk: A + D + E <= 250000

Tax_free: A + B + D >= 125000

Investment: A + B + C + D + E = 500000

VARIABLES
A Continuous
B Continuous
C Continuous
D Continuous
E Continuous
```

Solve and print the status to show that we have an optimal solution, with spoilers for our optimal value.

In [36]:
```
bondprob.solve()
print("status: " + LpStatus[bondprob.status])
```

```
Welcome to the CBC MILP Solver
Version: 2.10.3
Build Date: Dec 15 2019

command line - /Users/jdmini/datapad/lib/python3.9/site-packages/pulp
/solverdir/cbc/osx/64/cbc /var/folders/1n/nvr79nb55tz9j4lsbrw6hsf8000
0gn/T/d18228c1e4b24648a04d6c5ae9ec8865-pulp.mps max timeMode elapsed
branch printingOptions all solution /var/folders/1n/nvr79nb55tz9j4lsb
rw6hsf80000gn/T/d18228c1e4b24648a04d6c5ae9ec8865-pulp.sol (default st
rategy 1)
At line 2 NAME          MODEL
At line 3 ROWS
At line 9 COLUMNS
At line 28 RHS
At line 33 BOUNDS
At line 34 ENDATA
Problem MODEL has 4 rows, 5 columns and 13 elements
Coin0008I MODEL read with 0 errors
Option for timeMode changed from cpu to elapsed
Presolve 4 (0) rows, 4 (-1) columns and 10 (-3) elements
0  Obj -0 Primal inf 875000 (3) Dual inf 0.354996 (4)
4  Obj 44687.5
Optimal - objective value 44687.5
After Postsolve, objective 44687.5, infeasibilities - dual 0 (0), pri
mal 0 (0)
Optimal objective 44687.5 - 4 iterations time 0.002, Presolve 0.00
Option for printingOptions changed from normal to all
Total time (CPU seconds):       0.00   (Wallclock seconds):       0.0
1

status: Optimal
```

Now we can print the optimal amount of investment for each bond, which would give us a maximum return of $44,687.50

In [37]:
```
for variable in bondprob.variables():
    print("{}* = {}".format(variable.name, variable.varValue))

print(value(bondprob.objective))
```

```
A* = 62500.0
B* = 62500.0
C* = 187500.0
D* = 0.0
E* = 187500.0
44687.5
```

9.1 Slack and Shadow Price Interpretation

If we take a look at the slack and shadow prices of our constraints, we can see a few interesting things. First, all of our constraints are binding as evidenced by having zero slack.

The shadow prices reveal more information about our constraints. First we have two constraints with negative shadow prices: the short-term and tax-free constraints. This indicates that increasing the required amount of investment in either would actually result in slightly lower returns.

The high risk constraint shadow price is slightly positive, which tells us that allowing more investment in these bonds would slightly increase our maximum return.

Finally, we see that the investment constraint has a shadow price of 0.09. This means a one-dollar increase in overall investment would give us a 9-cent increase in our optimal value.

It makes sense that increasing our overall investment would give us a greater return, but our model design doesn't actually reflect the real-world marginal value of increasing the overall investment. Our constraints, which were given to us as percentages, were converted to values relative to the $500,000 investment. This is a good example of why thoughtful model design is important, because assumptions baked into the model can limit the utility of its outputs.

In [38]:
```
for name, c in bondprob.constraints.items():
    print("\n", name, ":", c, ", Slack=", c.slack, ", Shadow Price=",
```

```
 Short_term : B + E >= 250000 , Slack= -0.0 , Shadow Price= -0.0075

 High_risk : A + D + E <= 250000 , Slack= -0.0 , Shadow Price= 0.0075

 Tax_free : A + B + D >= 125000 , Slack= -0.0 , Shadow Price= -0.0025

 Investment : A + B + C + D + E = 500000 , Slack= -0.0 , Shadow Price
= 0.09
```

9.2 Reduced Cost Interpretation

Our optimal solution gave us one variable with a value of zero, which was bond D. This means it has a nonzero reduced cost, which in this case is -0.005. This indicates that if the annual return of bond D increased by half a percent, it would be part of optimal solution and improve the objective value.

In [40]:
```
for v in bondprob.variables():
    print ("\n", v.name, "=", v.varValue, ", Reduced Cost=", v.dj)
```

```
A = 62500.0 , Reduced Cost= 0.0

B = 62500.0 , Reduced Cost= 0.0

C = 187500.0 , Reduced Cost= 0.0

D = 0.0 , Reduced Cost= -0.005

E = 187500.0 , Reduced Cost= 0.0
```