## Final project

Created on: 2021-12-03

Name: Manish Aggarwal

## Project


# Linear Programming using Python

Project Report:

Mathematical Modeling of Chemical Engineering Processes

Topic: Linear Programming using Python

Course Advisor: Prof. John Kitchin

Date: Dec 3, 2021

-Manish Aggarwal

# Introduction


Research was conducted at the Star Bakery home industry, Sukarame, Bandar Lampung, Indonesia. This business produces three types of products, namely **flavored bread, mattress bread, and bargain bread**. **Flavored bread** has six flavors, namely chocolate flavor, chocolate cheese flavor, pineapple flavor, blueberry flavor, cheese milk flavor, and strawberry flavor. The price in each package of bread is **Rp.2,500**. **Bread mattress** has three flavors in one bread because it has a larger size. The taste in the bread is the mixing of chocolate flavor, chocolate cheese flavor, pineapple flavor, blueberry flavor, cheese milk flavor, and strawberry flavor. The price in each package of mattress bread is **Rp.6,000**. **Bargain Bread** does not have a flavor variant. In bread packaging there are 12 pieces of white bread. The price of each package of bargain bread is **Rp.5,000**.

**Objective**:

The Objective of the problem is to optimize(maximize) the profit of the bakery by varying the number(packs) of **flavored bread, mattress bread, and bargain bread** produced in the bakery.

The Bakery currently produces 3640 packs of flavored bread, 1300 packs of mattress bread and 520 packs of bargain bread. This information will be used as initial guess for solving the optimization problem.

Note:

1. Rupiah (Rp) is the official currency of Indonesia.
2. Bandar Lampung is the capital of the Indonesian province of Lampung.

This is a problem from literature which shows the use of Linear programming for optimizing the profit of a bakery.

**Reference: Bambang Sri Anggoro et al 2019 Profit Optimization Using Simplex Methods on Home Industry Bintang Bakery in Sukarame Bandar Lampung**

There are 18 constraints in the problem which have to be satisfied for finding the optimal solution.

**Table 2** Production Availability in one period (one month)

| Num. | Production Factor | Availability | Unit |
|---|---|---|---|
| 1. | Raw material | | |
| | Flour | 400 | Kg |
| | Sugar | 250 | Kg |
| | Developer | 90 | Kg |
| | Softener | 40 | Kg |
| | Yellow Butter | 90 | Kg |
| | Salt | 10 | Kg |
| | Milk powder | 60 | Kg |
| | Liquid milk | 60 | Kg |
| | Butter BOS | 90 | Kg |
| | Egg | 70 | Kg |
| | Feeling | 200 | Kg |
| | White butter | 90 | Kg |
| | Calcium | 20 | Kg |
| 2. | Production machine | | |
| | Mixer | 20 | Hour |
| | Divider machine | 7 | Hour |
| | Oven | 105 | Hour |
| 3. | Labor Hours | 208 | Hour |

*Source: Bintang Bakery home industry, 2018*

1. Determine the decision variables in solving linear program problems, namely the types of bread produced in the Bakery star home industry:
   $x_1$ = Bintang Bakery flavor (3640 packs)
   $x_2$ = Bintang Bakery mattress (1300 packs)
   $x_3$ = Bintang Bakery bargain (520 packs)

2. Determine the constraints in the problem:

| | |
|---|---|
| Flour | $= 28x_1 + 100x_2 + 250x_3 \leq 400.000$ |
| Sugar | $=7x_1 + 25x_2 + 62x_3 \leq 250.000$ |
| Developer | $=1x_1 + 9x_2 + 4x_3 \leq 90000$ |
| Softener | $=1x_1 + 6x_2 + 2x_3 \leq 40000$ |
| Yellow butter | $= 5x_1 + 20x_2 + 50x_3 \leq 90000$ |
| Salt | $= 1x_1 + 3x_2 \leq 10000$ |
| Milk powder | $= 1x_1 + 3x_2 + 2x_3 \leq 60000$ |
| Liquid milk | $= 5x_1 + 20x_2 \leq 60000$ |
| BOS butter | $=5x_1 + 20x_2 \leq 90000$ |
| Egg | $=4x_1 + 15x_2 + 25x_3 \leq 70000$ |
| Feeling | $= 14x_1 + 20x_2 \leq 200000$ |
| White butter | $= 5x_3 \leq 90000$ |
| Calcium | $= 2x_3 \leq 20000$ |
| Production machine | $= 32x_1 + 132x_2 + 336x_3 \leq 475200$ |
| Labor | $= 65x_1 + 209x_2 + 450x_3 \leq 748800$ |
| $x_1$ | $\geq 3640$ |
| $x_2$ | $\geq 1300$ |
| $x_3$ | $\geq 520$ |

3. Determine the objective function of the problem.

$$Z = 2500x_1 + 6000x_2 + 5000x_3$$

Here $x_1, x_2, x_3$ are the product variables of Bintang Bakery which are varied in order to get the maximum value of the objective function.

Firstly, we try using minimize function which is generally used for solving optimization problems in python.

In [ ]:

```
import numpy as np
from scipy.optimize import minimize

# x1 = no. of packs of flavor bread
# x2 = no. of packs of mattress bread
# x3 = no. of packs of bargain bread

def c1(X):
    'Flour constraint'
    x1, x2, x3 = X
    return -(28*x1 +100* x2 +250* x3 -400000)

def c2(X):
    'Sugar constraint'
    x1, x2, x3 = X
    return -(7*x1 +25* x2 +62* x3 -250000)

def c3(X):
    'Developer constraint'
    x1, x2, x3 = X
    return -(1*x1 +9* x2 +4* x3 -90000)

def c4(X):
    'x positivity constraint'
    return X[0]-3640

def c5(X):
    'y positivity constraint'
    return X[1] - 1300

def profit(X):
    'Profit function'
    x1, x2, x3 = X
    return -(2500 * x1 + 6000 * x2 + 5000* x3)
```

```python
def c6(X):
    'Softener constraint'
    x1, x2, x3 = X
    return -(1*x1 +6* x2 +2* x3 -40000)

def c7(X):
    'Yellow butter constraint'
    x1, x2, x3 = X
    return -(5*x1 +20* x2 +50* x3 -90000)

def c8(X):
    'Salt constraint'
    x1, x2, x3 = X
    return -(1*x1 +3* x2 -10000)

def c9(X):
    'Milk Powder constraint'
    x1, x2, x3 = X
    return -(1*x1 +3* x2 +2*x3 -60000)

def c10(X):
    'Liquid Milk constraint'
    x1, x2, x3 = X
    return -(5*x1 +20* x2 -60000)

def c11(X):
    'BOS butter constraint'
    x1, x2, x3 = X
    return -(5*x1 +20* x2 -90000)

def c12(X):
    'Egg constraint'
    x1, x2, x3 = X
    return -(4*x1 +15* x2 +25*x3 -70000)

def c13(X):
    'Feeling constraint'
    x1, x2, x3 = X
    return -(14*x1 +20* x2 -200000)

def c14(X):
    'White butter constraint'
    x1, x2, x3 = X
    return -(5*x3-90000)

def c15(X):
    'Calcium constraint'
    x1, x2, x3 = X
    return -(2*x1-20000)

def c16(X):
    'Production Machine constraint'
    x1, x2, x3 = X
    return -(32*x1 +132*x2+336*x3-47520)

def c17(X):
    'Labor constraint'
    x1, x2, x3 = X
    return -(65*x1 +209*x2+450*x3-748800)

def c18(X):
    'z constraint'
    x1, x2, x3 = X
    return (x3-520)



sol = minimize(profit, [3640, 1300, 520], constraints=[{'type': 'ineq', 'fun': f} \
for f in [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18]])
print(sol)
```

```
     fun: -19500000.000014067
     jac: array([-2500., -6000., -5000.])
 message: 'Positive directional derivative for linesearch'
    nfev: 50
     nit: 8
    njev: 4
  status: 8
 success: False
       x: array([3640., 1300.,  520.])
```

In [ ]:

```
# Using different Initial guess

sol = minimize(profit, [2000, 500, 520], constraints=[{'type': 'ineq', 'fun': f}
for f in [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18]])
print(sol)
```

```
     fun: -10600000.000001011
     jac: array([-2500., -6000., -5000.])
 message: 'Positive directional derivative for linesearch'
    nfev: 20
     nit: 6
    njev: 2
  status: 8
 success: False
       x: array([2000.,  500.,  520.])
```

The values of the problem are stuck at initial guesses in the example above, which doesnot solve the problem.

**Different algorithms in minimize function are used to find optimal values of the objective.**

**1. Algorithm: Nelder Mead**

In [ ]:

```
import warnings
warnings.filterwarnings("ignore")

sol = minimize(profit, [3735, 1300, 520], method='Nelder-Mead', options={'xatol': 1, 'disp': True},
constraints=[{'type': 'ineq', 'fun': f} for f in [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12,
                                                  c13, c14, c15, c16, c17, c18]])
print(f'success={sol.success}')
print(f'No. of Iterations = {sol.nit}')
```

```
Warning: Maximum number of function evaluations has been exceeded.
success=False
No. of Iterations = 299
```

**2. Algorithm: Trust-constr**

In [ ]:

```
import warnings
warnings.filterwarnings("ignore")

sol = minimize(profit, [3640, 1300, 520], method='trust-constr',
constraints=[{'type': 'ineq', 'fun': f} for f in [c1, c2, c3, c4,
c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18]])

print(sol.message)
print(f'success={sol.success}')
print(f'No. of Iterations = {sol.nit}')
```

```
The maximum number of function evaluations is exceeded.
success=False
No. of Iterations = 1001
```

The output solution is very long and only the success message of the algorithm is shown.

**3. Algorithm: COBYLA**

In [ ]:

```
sol = minimize(profit, [3640, 1300, 520], method='COBYLA',constraints=[{'type': 'ineq', 'fun': f}
for f in [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18]])

print(f'The values of x1={sol.x[0]}, x2={sol.x[1]}, x3={sol.x[2]}')
print(sol.message)
print(f'success={sol.success}')

# Note COBYLA method does not give the number of iterations
```

```
The values of x1=3553.056408515182, x2=938.2326852358319, x3=-403.4077106325075
Did not converge to a solution satisfying the constraints. See `maxcv` for magnitude of violation.
success=False
```

**The Optimization function does not give a optimum solution of the problem with different algorithms in minimize function.**

# Methods


## Simplex Method

Simplex was invented in 1946–1947 by George B. Dantzig as a means to solve linear optimization problems.

**Introduction to Simplex method**

The Simplex method is an approach to solving linear programming using slack variables as a means to finding the optimal solution of an optimization problem. A linear program is a method of achieving the best outcome given a maximum or minimum equation with linear constraints.

For example, imagine that you've a bakery; you make 3 different products cakes, pastries, and cookies. Each item requires x, y, and z labor hours worth of sweetening, baking, and finishing. And you have a fixed number of labor hours available for each task (sweetening, baking, and finishing.) If cakes, pastries, and cookies bring in a, b, and c dollars — which combination of bakery products should you produce each month to maximize profit subject to labor hour constraints?

Those numbers (and the maximum profit) are precisely what this algorithm will return as output.

So how does it work? The algorithm works by taking inequalities, introducing slack variables, and converting these inequalities into equations:

$1(x_1) + 1(x_2) <= 20$ is equivalent to writing $1(x_1) + 1(x_2) + (Slack\ variable) = 20$

$-2(x_1) + 1(x_2) <= -10$

is equivalent to writing

$-2(x_1) + 1(x_2) + (Slack\ variable) = -10$

### What are Slack Variables?

In an optimization problem, a slack variable is a variable that is added to an inequality constraint to transform it into an equality. Introducing a slack variable replaces an inequality constraint with an equality constraint and a non-negativity constraint on the slack variable.

Slack variables are used in particular in linear programming. As with the other variables in the augmented constraints, the slack variable cannot take on negative values, as the simplex algorithm requires them to be positive or zero.

If a slack variable associated with a constraint is zero at a particular candidate solution, the constraint is binding there, as the constraint restricts the possible changes from that point.

If a slack variable is positive at a particular candidate solution, the constraint is non-binding there, as the constraint does not restrict the possible changes from that point. If a slack variable is negative at some point, the point is infeasible (not allowed), as it does not satisfy the constraint.

Reference : https://en.wikipedia.org/wiki/Slack_variable (https://en.wikipedia.org/wiki/Slack_variable)

**Smaller problems (2 variables and 2-4 constraints) are solved by using simplex method in linear programming using linprog function in python.**

minimize function is also used to solve these problems.

## Linprog function for Linear Optimization.

**Next we use the Linprog function for Linear Optimization.**

**First Question**

**Objective:**

Here the objective is to maximise $5x_1$ + $10x_2$, hence this is our objective function.

The objective function is written as -5, -10 in the program for maximisation of $x_1$ and $x_2$.

The inequalities are written in the form A$x_1$+B$x_2$ <= C

hence the inequalities are written as

-2$x_1$ + 1$x_2$<= -10 --->(1)

and

1$x_1$ + 1$x_2$<= 20 ---->(2)

Here there are no bounds or limitation on $x_1$ or $x_2$

In [ ]:

```
from scipy.optimize import linprog
# Objective function
obj = [-5, -10]
#      ─┬   ─┬
#       │    └┤ Coefficient for x2
#       └─────┤ Coefficient for x1

# Left Hand side of Constraints
lhs_ineq = [[ -2,1],
            [ 1,1],
            ]

# Right Hand side of Constraints
rhs_ineq = [-10,
            20,
             ]

#Simplex Algorithm
opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq,
              method="simplex")

print(opt)
print(f'\n The Maximum objective value is {-1*opt.fun: 1.2f}')
```

```
     con: array([], dtype=float64)
     fun: -150.0
 message: 'Optimization terminated successfully.'
     nit: 2
   slack: array([0., 0.])
  status: 0
 success: True
       x: array([10., 10.])

 The Maximum objective value is  150.00
```

There are no slack variables generated by the linprog function and we have maximized the value of the profit function as 150 and

$x_1$ =10

$x_2$ = 10

and looking at the 2 equations, we have

-2$x_1$ + 1$x_2$<= -10 --->(1)

and

1$x_1$ + 1$x_2$<= 10 ---->(2)

$-2 * 10 + 1 * 10 = -10$ from eq 1

and

$1 * 10 + 1 * 10 = 20$ from eq 2

Both the equations are satisfied.

**Using minimize function**

In [ ]:

```
import numpy as np
from scipy.optimize import minimize

def c11(X):
    # Constraint 1
    x, y = X
    return -(-2*x +1* y + 10)

def c21(X):
    # Constraint 2
    x, y = X
    return -(1*x +1* y -20)

def profit1(X):
    'Profit function (Objective)'
    x, y = X
    return -(5 * x + 10 * y)



sol = minimize(profit1, [1,1], constraints=[{'type': 'ineq', 'fun': f} for f in [c11, c21]])

print(sol)
print(f'\n The Maximum objective value is {-1*sol.fun: 1.2f}')
```

```
     fun: -150.0000000000483
     jac: array([ -5., -10.])
 message: 'Optimization terminated successfully.'
    nfev: 12
     nit: 3
    njev: 3
  status: 0
 success: True
       x: array([10., 10.])

 The Maximum objective value is  150.00
```

In [ ]:

```
sol1 = minimize(profit1, [1,1], method='COBYLA', constraints=[{'type': 'ineq', 'fun': f} for f in [c11, c21]])
sol1
```

Out[ ]:

```
     fun: -150.0
   maxcv: 0.0
 message: 'Optimization terminated successfully.'
    nfev: 27
  status: 1
 success: True
       x: array([10., 10.])
```

Here we see that the minimize function also gives a correct value of the function and the optimization is terminated successfully.

Minimize function is able to give correct values of the Objective function for a small problem, but doesnot give a solution for the main problem, hence we will use specialized function in python.

Note: simplex function uses 2 iterations and minimize function uses 3 iterations to solve the first question.

**Second question**

The objective in the second question is to maximise the value of

$1x_1 + 2x_2$

In [ ]:

```
from scipy.optimize import linprog
obj = [-1, -2]
#      ┬  ┬
#      │  └┤ Coefficient for x2
#      └──┤ Coefficient for x1

lhs_ineq = [[ 2,  1],
            [-4,  5],
            [ 1, -2]]

rhs_ineq = [20,
            10,
             2]

lhs_eq = [[-1, 5]]
rhs_eq = [15]

bnd = [(0, float("inf")),
       (0, float("inf"))]

opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq,
              A_eq=lhs_eq, b_eq=rhs_eq, bounds=bnd,
              method="simplex")
print(opt)
print(f'\n The Maximum objective value is {-1*opt.fun: 1.2f}')
```

```
     con: array([3.55271368e-15])
     fun: -16.818181818181817
 message: 'Optimization terminated successfully.'
     nit: 5
   slack: array([ 0.        , 18.18181818,  3.36363636])
  status: 0
 success: True
       x: array([7.72727273, 4.54545455])

 The Maximum objective value is  16.82
```

Slack variables are generated by the simplex algorithm and the slack values are generated by the solver.

slack: array([ 0. , 18.18181818, 3.36363636])

There are 3 inequalities and 1 equality constraint in the problem and hence we have slack variables for the 3 inequalities.

Here the objective function is to maximise $1x_1$ + $2x_2$

and looking at the 3 equations, we have

$2x_1$ + $1x_2$<= 20 --->(1)

and

-$4x_1$ + $5x_2$<= 10 ---->(2)

and

$1x_1$ - $2x_2$<= 2 ---->(3)

and equality constraint as

-$1x_1$ + $5x_2$= 15 ---->(4)

There are slack variables generated by the linprog function and we have maximized the value of the profit function as 16.818 and

$x_1$=7.72727273,

$x_2$=4.54545455

From Equation 1, we have

$2 * 7.72727273 + 4.54545455 + S_1 = 20$ from $eq1$

$S_1$ is the slack variable, which is 0.

and

From Equation 2, we have

$-4$*7.72727273 + 5*$4.54545455$+ $S_2 = 10$

where $S_2$ is the slack variable. Solving for $S_2$, we get $S_2$ = 18.1818

From Equation 3, we have

$1x_1$ - $2x_2$<= 2 ---->(3)

can be rewritten as $1x_1$ - $2x_2$+ $S_3$= 2

$1 * 7.72727273 - 2 * 4.54545455 + S_3 = 2$

we get

$S_3 = 3.36363636$

and the 4th equality equation

-$1x_1$ + $5x_2$= 15 ---->(4)

$-1 * 7.72727273 + 5 * 4.54545455 = 15$

is satisfied.

The Maximum objective value obtained is 16.82

**To do List**

**To do List using linprog**

1. Change the Objective Function if it is maximisation function. Change the equations in the form A$x_1$+B$x_2$ <= C form for the Linprog function.
2. The value received from the Linprog function in fun: is the optimised value: maximum or minimum. (opt.fun command is used to get the optimum value of the objective function).

In [ ]:

```
-1*opt.fun
```

Out[ ]:

16.818181818181817

1. The success of the solver can be checked by using the command opt.success

In [ ]:

```
opt.success
```

Out[ ]:

```
True
```

1. opt.slack gives the slack variables.

In [ ]:

```
opt.slack
```

Out[ ]:

```
array([ 0.        , 18.18181818,  3.36363636])
```

1. opt.message tells whether the Optimization is completed successfully.

In [ ]:

```
opt.message
```

Out[ ]:

```
'Optimization terminated successfully.'
```

1. opt.x gives the values of the variables.

In [ ]:

```
opt.x
```

Out[ ]:

```
array([7.72727273, 4.54545455])
```

Note:

A maximization problem is optimized when the slack variables are “squeezed out,” maximizing the true variables’ effect on the objective function.

Conversely, a minimization problem is optimized when the slack variables are “stuffed,” minimizing the true variables’ effect on the objective function. By the way, the objective function is the relation between true variables. In most business applications, it’s cost or profit.

Profit = $5(x_1)$ + $10(x_2)$ for the First question and

Profit = $1(x_1)$ + $2(x_2)$ for the second question.

# Results


### Profit Optimization Using Simplex Methods

**Profit Optimization using Simplex Methods on Home Industry Bintang Bakery in Sukarame Bandar Lampung**

The results of the production optimization model processing show that the production carried out by the Bakery star home industry in factual conditions are not optimal. This can be seen from the condition of the total production that is different from the optimal conditions.

**Table 3** Optimal Bintang Bakery Production

| Num. | Bread type | Variable | Production Level | |
|---|---|---|---|---|
| | | | Factual | Optimal |
| 1 | Taste bread | $x_1$ | 3640 | 3740 |
| 2 | Bread mattress | $x_2$ | 1300 | 1300 |
| 3 | White bread | $x_3$ | 520 | 520 |

*Source: Data Processed, 2018.*

Based on table 3 the results of processing the production optimization model on the Bakery star home industry in factual conditions have not shown optimal results. This is shown by the results produced in factual conditions different from the results obtained in optimal conditions. In factual conditions, there were 3640 flavored Bakery stars, 1300 star Bakery mattresses, and 520 tasteless Bakery stars. 3740 flavored Bakery stars, 1300 Bakery stars, and 520 Bakery stars were obtained for optimal conditions.

**Table 4** Earnings of Each Product in Factual Conditions and Optimal Conditions

| No | Bread type | Variable | Production Level | |
|---|---|---|---|---|
| | | | Factual | Optimal |
| 1 | Taste bread | $x_1$ | 9.100.000 | 9.350.000 |
| 2 | Bread mattress | $x_2$ | 7.800.000 | 7.800.000 |
| 3 | White bread | $x_3$ | 2.600.000 | 2.600.000 |
| | Total | | 19.500.000 | 19.750.000 |

*Source: Data Processed, 2018.*

Profit optimization calculations using the simplex method, get optimal results if the home industry produces 3740 packs of flavored Bakery stars, 1300 star Bakery mattresses and 520 packaged Bakery stars, Rp. 19,750,000 obtained. If the Bakery star home industry produces each type of bread according to optimal results, the benefits that will be obtained will get optimal results. The optimal results obtained are Rp. 19,750,000 while the optimal results obtained from factual conditions are Rp. 19,500,000. The benefits obtained from factual conditions to optimal conditions are increased by Rp. 250,000

**Note:**

1. flavor bread is referred as Taste bread in the solution above.
2. bargain bread is referred as White bread in the solution above.

In [ ]:

```
from scipy.optimize import linprog
obj = [-2500, -6000, -5000]
#      ─┬       ─┬      └┤ Coefficient for x3
#       │        └┤ Coefficient for x2
#       └─────┤ Coefficient for x1

# x1 = no. of packs of flavor bread (referred as Taste bread in the solution)
# x2 = no. of packs of mattress bread
# x3 = no. of packs of bargain bread(referred as White bread in the solution)

lhs_ineq = [[ 28,100,250],
            [ 7,25,62],
            [ 1,9,4],
            [1,6,2],
            [5,20,50],
            [1,3,0],
            [1,3,2],
            [5,20,0],
            [5,20,0],
            [4,15,25],
            [14,20,0],
            [0,0,5],
            [0,0,2],
            [32,132, 336],
            [65, 209,450]]


rhs_ineq = [400000,
            250000,
             90000,
             40000,
             90000,
             10000,
             60000,
             60000,
            90000,
            70000,
            200000,
            90000,
            20000,
            475200,
            748800]


bnd = [(3640, float("inf")),  # Bounds of x1
      (1300, float("inf")),  # Bounds of x2
      (520, float("inf"))]  # Bounds of x3

opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq, bounds=bnd,
              method="simplex")
```

In [ ]:

```
print(f'The values of x1={opt.x[0]}, x2={opt.x[1]}, x3={opt.x[2]}')
print(opt.message)
print(f'success={opt.success}')
print(f'\n The Maximum profit value is {-1*opt.fun: 1.2f} Rp')
```

```
The values of x1=3740.0, x2=1300.0, x3=520.0
Optimization terminated successfully.
success=True

 The Maximum profit value is  19750000.00 Rp
```

In [ ]:

```
# Full output

print(opt)
```

```
     con: array([], dtype=float64)
     fun: -19750000.0
 message: 'Optimization terminated successfully.'
     nit: 18
   slack: array([ 35280., 159080.,  72480.,  27420.,  19300.,   2360.,  51320.,
        15300.,  45300.,  22540., 121640.,  87400.,  18960.,   9200.,
            0.])
  status: 0
 success: True
       x: array([3740., 1300.,  520.])
```

**Revised Simplex algorithm in linprog**

In [ ]:

```
opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq, bounds=bnd,
              method="revised simplex")
print(f'The values of x1={opt.x[0]}, x2={opt.x[1]}, x3={opt.x[2]}')
print(opt.message)
print(f'success={opt.success}')
print(f'No. of Iterations = {opt.nit}')
print(f'\n The Maximum profit  is {-1*opt.fun: 1.2f} Rp')
```

```
The values of x1=3740.0, x2=1300.0, x3=520.0
Optimization terminated successfully.
success=True
No. of Iterations = 2

 The Maximum profit  is  19750000.00 Rp
```

Note:

The no. of iterations in Revised simplex method is 2 compared to 18 iterations in the simplex method.

## Differences between Simplex and Revised Simplex Method

**Simplex :-**

1. Simplex method starts with a non-optimal but feasible solution.
2. It maintains the feasibility during successive iterations.
3. It follows the shortest route to reach the optimal solution from the starting point.
4. It is less efficient and accurate as compared to revised simplex.

**Revised Simplex:-**

1. Revised simplex is an improvement over simplex method.
2. It is computationally more efficient and accurate.
3. It clearly comprehends in case of large Linear Programming problems.
4. Instead of maintaining a tableau unlike Simplex method, it maintains a representation of a basis of the matrix representing the constraints.
5. It is more efficient and accurate as compared to simplex.

The simplex and revised simplex algorithm (using linprog) successfully gives an optimal solution of the problem in literature.

## Finding tightest constraint

**Finding constraint having the smallest slack variable.**

From the salt constraint we have $1x_1 + 3x_2 < 10000.$

If $x_2$ is taken as 1300, we have $x_1$ <= 6100. We will get more such equations when solve for all equations.

We have from Labor constraint $65 * x_1 + 209 * x_2 + 450 * x_3$ <=748800

If $x_2$ is taken as 1300 and $x_3$ is taken as 520 we have $65 * x_1 + 271,700 + 234,000$ <=748800

$65 * x_1$<243,100

$x_1$<= 3740

which gives the optimal value of $x_1$

The constraint equation with the slack variable equal to zero (smallest) is the tightest constraint.

In [ ]:

```
import numpy as np
from scipy.optimize import minimize

# x1 = no. of packs of flavor bread
# x2 = no. of packs of mattress bread
# x3 = no. of packs of bargain bread

def c4(X):
    'x positivity constraint'
    return X[0]-3640

def c5(X):
    'y positivity constraint'
    return X[1] - 1300

def profit(X):
    'Profit function'
    x1, x2, x3 = X
    return -(2500 * x1 + 6000 * x2 + 5000* x3)

def c8(X):
    'Salt constraint'
    x1, x2, x3 = X
    return -(1*x1 +3* x2 -10000)

def c14(X):
    'White butter constraint'
    x1, x2, x3 = X
    return -(5*x3-90000)

def c15(X):
    'Calcium constraint'
    x1, x2, x3 = X
    return -(2*x1-20000)

def c17(X):
    'Labor constraint'
    x1, x2, x3 = X
    return -(65*x1 +209*x2+450*x3-748800)

def c18(X):
    'z constraint'
    x1, x2, x3 = X
    return (x3-520)



sol = minimize(profit, [3640, 1300, 520], constraints=[{'type': 'ineq', 'fun': f} \
for f in [c4, c5, c8, c14, c15, c17, c18]])

print(f'The values of x1={sol.x[0]}, x2={sol.x[1]}, x3={sol.x[2]}')
print(sol.message)
print(f'success={sol.success}')
print(f'\n The Maximum Profit is {-1*sol.fun: 1.2f} Rp')
```

```
The values of x1=3740.000009363215, x2=1300.000022524273, x3=520.0000187720634
Positive directional derivative for linesearch
success=False

 The Maximum Profit is  19750000.25 Rp
```

In [ ]:

```
# Algorithm SLSQP

sol = minimize(profit, [3540, 1200, 220], method='SLSQP',constraints=[{'type': 'ineq', 'fun': f} \
for f in [c4, c5, c8, c14, c15, c17, c18]])

print(f'The values of x1={sol.x[0]}, x2={sol.x[1]}, x3={sol.x[2]}')
print(sol.message)
print(f'Algorithm SLSQP success={sol.success}')
print(f'\nThe Maximum Profit is {-1*sol.fun: 1.2f} Rp')
```

```
The values of x1=3740.000060117627, x2=1300.0001540876276, x3=520.0001227478584
Positive directional derivative for linesearch
Algorithm SLSQP success=False

The Maximum Profit is  19750001.69 Rp
```

With fewer constraints minimize function gives almost the correct answer to the problem in literature. But the solver still has the success message as False as the exact optimum value is not found.

Let us try the problem with another algorithm COBYLA

Algorithm COBYLA with 7 constraints

In [ ]:

```
# Algorithm COBYLA with 7 constraints

sol = minimize(profit, [3540, 1200, 220], method='COBYLA',constraints=[{'type': 'ineq', 'fun': f} \
for f in [c4, c5, c8, c14, c15, c17, c18]])

print(f'The values of x1={sol.x[0]}, x2={sol.x[1]}, x3={sol.x[2]}')
print(sol.message)
print(f'success={sol.success}')
print(f'The Maximum Profit is {-1*sol.fun: 1.2f} Rp')
```

```
The values of x1=3740.000000000001, x2=1300.0, x3=520.0
Optimization terminated successfully.
success=True
The Maximum Profit is  19750000.00 Rp
```

Algorithm COBYLA with 4 constraints

In [ ]:

```
# Algorithm COBYLA with 4 constraints

import numpy as np
from scipy.optimize import minimize

def c4(X):
    'x positivity constraint'
    return X[0]-3640

def c5(X):
    'y positivity constraint'
    return X[1] - 1300

def profit(X):
    'Profit function'
    x1, x2, x3 = X
    return -(2500 * x1 + 6000 * x2 + 5000* x3)

def c17(X):
    'Labor constraint'
    x1, x2, x3 = X
    return -(65*x1 +209*x2+450*x3-748800)

def c18(X):
    'z constraint'
    x1, x2, x3 = X
    return (x3-520)




sol = minimize(profit, [3640, 1300, 520], method='COBYLA',constraints=[{'type': 'ineq', 'fun': f} \
for f in [c4, c5, c17, c18]])

print(f'The values of x1={sol.x[0]:1.2f}, x2={sol.x[1]}, x3={sol.x[2]}')
print(sol.message)
print(f'success={sol.success}')
print(f'The Maximum objective Profit is {-1*sol.fun: 1.2f} Rp')
```

```
The values of x1=3740.00, x2=1300.0, x3=520.0
Optimization terminated successfully.
success=True
The Maximum objective Profit is  19750000.00 Rp
```

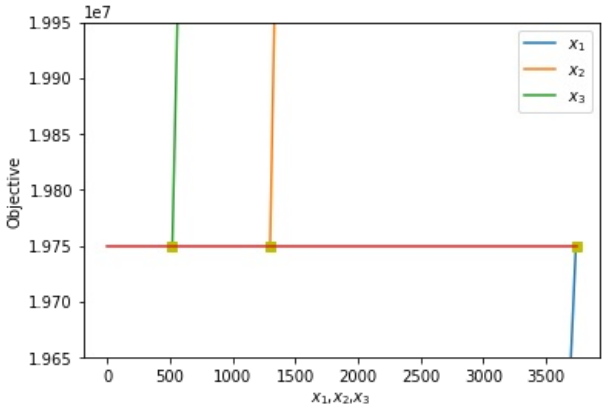
Note:

With the help of **COBYLA** algorithm, minimize function finds the optimized value of the profit function with 7 and 4 constraints.

# Graphs

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

#x=x1, y=x2, z=x3
x= np.linspace(3640, 3740)  # from Labor constraint x1<=3740
y = 1300
z = 520
c = 2500*x + 6000*y + 5000*z

y1= np.linspace(1300, 1400)
x1= 3740
z1=520
c1 = 2500*x1 + 6000*y1 + 5000*z1

z2= np.linspace(520, 600)
x2= 3740
y2=1300
c2 = 2500*x2 + 6000*y2 + 5000*z2

c3=-opt.fun

plt.plot(x, c, label='$x_1$')
plt.plot(y1, c1, label='$x_2$')
plt.plot(z2, c2, label='$x_3$')

plt.xlabel('$x_1$,$x_2$,$x_3$')
plt.ylabel('Objective')
plt.plot(opt.x[0],-opt.fun, 'ys')
plt.plot(opt.x[1],-opt.fun, 'ys')
plt.plot(opt.x[2],-opt.fun, 'ys')
point1 = [opt.x[0], -opt.fun]
point2 = [opt.x[1], -opt.fun]
point3=  [opt.x[2], -opt.fun]
x_values = [point1[0], point2[0], point3[0], 0]
y_values = [point1[1], point2[1], point3[1], -opt.fun]
plt.ylim(19650000.0, 19950000.0)
plt.plot(x_values, y_values)
plt.legend()
plt.show();

#The graph shows the optimal value of x1, x2 and x3 and
# the max objective (profit) value
```



**Optimal Values of $x_1, x_2, x_3$**

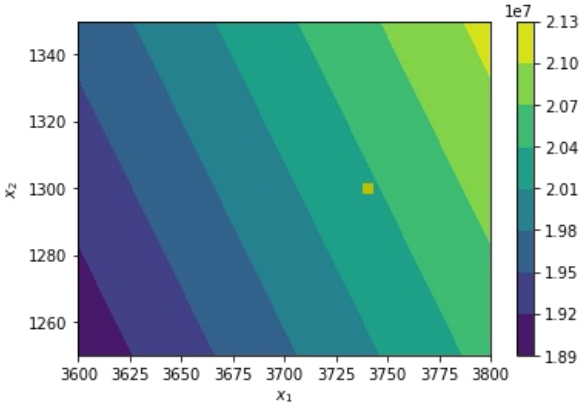
**The Graph below shows the Optimal value of $x_1, x_2$**

**in $x_2$ vs $x_1$ plot**

The objective (Profit) value range is shown in the side graph attached with the graph.

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

#x=x1, y=x2, z=x3
x, y= np.meshgrid(np.linspace(3600, 3800), np.linspace(1250, 1350))
z = np.linspace(500, 700)
c = 2500*x + 6000*y + 5000*z

plt.contourf(x, y, c)
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.plot(opt.x[0], opt.x[1], 'ys')
plt.colorbar();
```

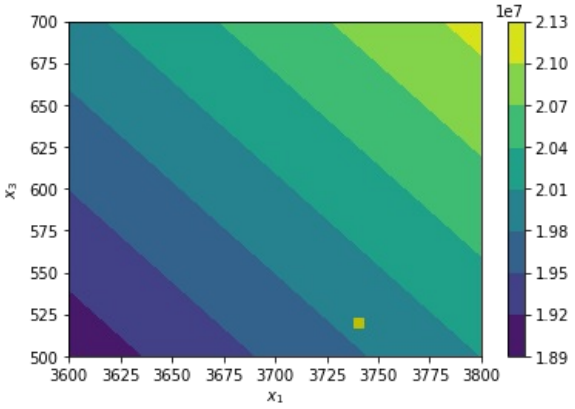



**The Graph below shows the Optimal value of $x_1, x_3$ in**

**$x_3$ vs $x_1$ plot.**

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

x, z= np.meshgrid(np.linspace(3600, 3800), np.linspace(500, 700))
y = np.linspace(1250, 1350)
c = 2500*x + 6000*y + 5000*z

plt.contourf(x, z, c)
plt.xlabel('$x_1$')
plt.ylabel('$x_3$')
plt.plot(opt.x[0], opt.x[2], 'ys')
plt.colorbar();
```



# Conclusions


1. The use of Linear programming in solving a problem in literature is shown in this project.
2. **minimize function** from the scipy.optimize library is generally used for solving the optimization problems. The problem which is used in the project has 18 linear constraints for solving the profit optimization function at the star Bakery home industry, Sukarame, Bandar Lampung. The minimize function is not able to give the optimum value of the optimization problem and is stuck at the initial guess.
3. **linprog function** which is also from scipy.optimize library is specifically used to solve the problems in linear programming. The linprog function successfully reaches to the optimum value of the objective fucntion. The linprog function uses many algorithms among which one is the simplex method. Simplex method uses slack variables to solve problems in Linear programming.
4. This project shows the **limitation** of using the minimize function in python. linprog function which is specifically designed for use in linear programming has to be used for the problem, eventhough minimize function also has Simplex algorithm method (Nelder-Mead) to solve problems which is tested in the project.
5. **With fewer constraints** (7 instead of 18) minimize function gives an answer very near to the optimum value of $x_1$, $x_2$, $x_3$ as the problem in literature. But for that we have to find the equations which have the tightest constraints and that is not always possible (or easy) to find.
6. Using **COBYLA** algorithm, minimize function finds the optimized value of the profit function with 7 constraints and 4 constraints.
7. Two different algorithms **(Simplex and Revised simplex)** are tested with linprog function. Revised simplex is an improvement over the simplex method. Revised simplex uses less iterations compared to simplex algorithm to solve the problem.

# Acknowledgement

The author thanks Professor John Kitchin for his invaluable supervision, guidance and helpful comments for improvement of the project.

In [ ]:

```
!pip install git+git://github.com/jkitchin/pycse &> /dev/null
from pycse.colab import pdf
%pdf
```