Python Frameworks

Solving linear programming problems

General Statements

- Linear programming is an important branch of Operations Research. This mathematical technique consists of a set of methods used to obtain the best solution to linear optimization problems subject to constraints, such as practical contexts when the optimal distribution of limited resources must be found.
- There is a wide variety of problems which can be represented by a linear programming model. Some examples are the problem of assigning resources to tasks, production planning, transportation of goods, product-mix problems, etc.
- In linear programming, a mathematical model is used to describe the problem. The adjective linear means that all functions used to define the model are linear.

The linear model

A linear model deals with optimizing (maximizing or minimizing) a linear function with several variables, given certain linear constraint inequalities.

$$\text{opt}\, z = \mathbf{c}^T \mathbf{x} \qquad (1.1)$$

subject to

$$\mathbf{A}\mathbf{x} \leq > \mathbf{b} \qquad (1.2)$$

$$\mathbf{x} \geq 0 \qquad (1.3)$$

The elements that appear in the model are the following:

- **x**, is the vector of *decision variables*, and contains *n* variables.
- $\mathbf{c}^T$, is the vector of *cost coefficients*, and contains *n* constants.
- **b**, is the *right-hand-side vector*, with *m* constants.

The linear model

Matrix **A**, with *m* rows and *n* columns is called the *constraint matrix*. Each coefficient a_{ij} in **A** represents the amount of resource *i, i = 1, …, m*, needed to perform a unit of activity *j, j = 1, … , n*, and are called the technological coefficients.

Vectors $\mathbf{c}^T$ and **b** and matrix **A** are known parameters in the linear model; vector x contains the variables whose values have to be determined in order to find the optimal way to assign resources to activities.

Notations

The linear model can be stated using different notations:

1. We can write the linear model in the following way:

$$
\begin{aligned}
\text{opt } z &= c_1x_1 + c_2x_2 + \cdots + c_nx_n \\
\text{subject to} & \\
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\lessgtr b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\lessgtr b_2 \\
\vdots \quad\quad \vdots \quad\quad \ddots \quad\quad \vdots \quad \vdots & \\
a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\lessgtr b_m \\
x_1, x_2, \ldots, x_n &\geq 0
\end{aligned}
$$

Notations

2. Matrix notation:

$$\text{opt } z = (c_1, \ldots, c_n) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

subject to

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{matrix} \leq \\ = \\ \geq \end{matrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

$$(x_1, x_2, \cdots, x_n)^T \geq (0, 0, \cdots, 0)^T$$

Notations

3. Denoting by $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n$ the n columns of matrix **A**, the linear model can be represented as follows:

$$\text{opt } z = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

subject to

$$\mathbf{a}_1 x_1 + \mathbf{a}_2 x_2 + \cdots + \mathbf{a}_n x_n \overset{\leq}{\geq} \mathbf{b}$$

$$x_j \geq 0\ ,\ j = 1, \ldots, n$$

Linear programming modeling

- The first stage in the analysis and solution of a linear programming problem is to formulate the problem by writing a model that represents it.
- The process of transcribing the verbal description of a problem into a mathematical form that allows the application of linear programming techniques is usually called modeling, and it is a particularly difficult aspect.
- However, it is important, because the solution obtained for the problem will depend on the model that has been formulated.
- Care must be taken to ensure that the model represents correctly the problem being analyzed. That is why it is worth focusing on the development of the necessary skills to formulate the appropriate models.

Example

Problem statement: The factory produces two types of paints: the first one for exterior and the second one for interior. Two ingredients are used for the production of paints: A and B. The maximum possible daily stocks of these ingredients are 6 and 8 tons, respectively. The following table shows how many tons of ingredients should be spend for manufacturing 1 ton of paint of each type.

Ingredients	Ingredients consumption rate, t of ingr. / t of paint		Stocks, t of ingr./day
	1st paint	2d paint	
A	1	2	6
B	2	1	8
Price, thousand UAH	3	2	

Market research has shown that the daily demand for paint of type 2 never exceeds the demand for paint of type 1 by more than 1 ton.

In addition, the demand for type 2 paint never exceeds 2 tons per day.

The prices of one ton of paints are equal: 3 thousand UAH for paint of the 1st type; 2 thousand UAH for paint of the 2nd type.

It is necessary to build a mathematical model that allows you to determine how much paint of each type should be produced to maximize revenue from sales.

Model

1. Decision variables

x_j : the amount of paint of each type that must be produced , j = 1, 2.

2. Objective function: To maximize the benefit

$$F(x) = 3x_1 + 2x_2 \rightarrow max$$

3. Constraints:

$$\begin{cases} x_1 + 2x_2 \leq 6 \\ 2x_1 + x_2 \leq 8 \\ x_2 - x_1 \leq 1 \\ x_2 \leq 2 \\ x_1, x_2 \geq 0 \end{cases}$$

Graphical solution

In general, and even though not all linear problems can be solved graphically, they all can be geometrically interpreted. It is worth to study the graphical solution of linear problems, because it enables to observe graphically important concepts in linear programming, such as the improvement of a solution, types of solutions, extreme points, etc.

- The set of solutions or feasible region of a linear inequality system can be graphically illustrated by representing the equation associated with each inequality and determining the half-space that contains the points that satisfy the inequality.
- By the non-negativity constraints, the points can only fall in the first quadrant. By proceeding this way, we will obtain the polygon of solutions.
- The objective function is a family of parallel straight lines, one for each value of *z*.
- The line representing the objective function is moved in the optimization direction as much as possible, until the optimal point is reached. If there a bounded optimal solution to the problem exists, then the optimal value for the objective function will be found in an extreme point of the polygon.

Graphical solution

- The objective is to choose x_1 and x_2 such that they verify the constraints and maximize the objective value $z = 3x_1 + 2x_2$.
- We can represent graphically the set of points that satisfy the linear inequalities. Each constraint in the model is a half-space in the plane.
- For example, in order to represent the set of points satisfying $x_1 + 2x_2 \leq 6$, we draw the straight line $x_1 + 2x_2 = 6$. This straight line divides the plane in two half-spaces. The points satisfying the constraint are contained in one of the two half-spaces.
- We can test whether one point, the origin for instance, satisfies the constraint to decide which one of the two half-spaces contains all the points satisfying the constraint. In the graphical representation, we illustrate it by using small arrows.
- After representing graphically all the constraints of the problem, including the non-negativity constraints, we obtain the set of solutions of the problem, which is shown by the shaded region in the following graphical representation:

Graphical solution

In the constraints of the problem, replace the signs of inequalities with signs of exact equality and construct the corresponding lines.

1) $x_1 + 2x_2 \le 6 \rightarrow x_1 + 2x_2 = 6$

$$\begin{Bmatrix} x_1 = 0 \\ x_2 = 3 \end{Bmatrix} \begin{Bmatrix} x_1 = 6 \\ x_2 = 0 \end{Bmatrix}$$

2) $2x_1 + x_2 \le 8 \rightarrow 2x_1 + x_2 = 8$

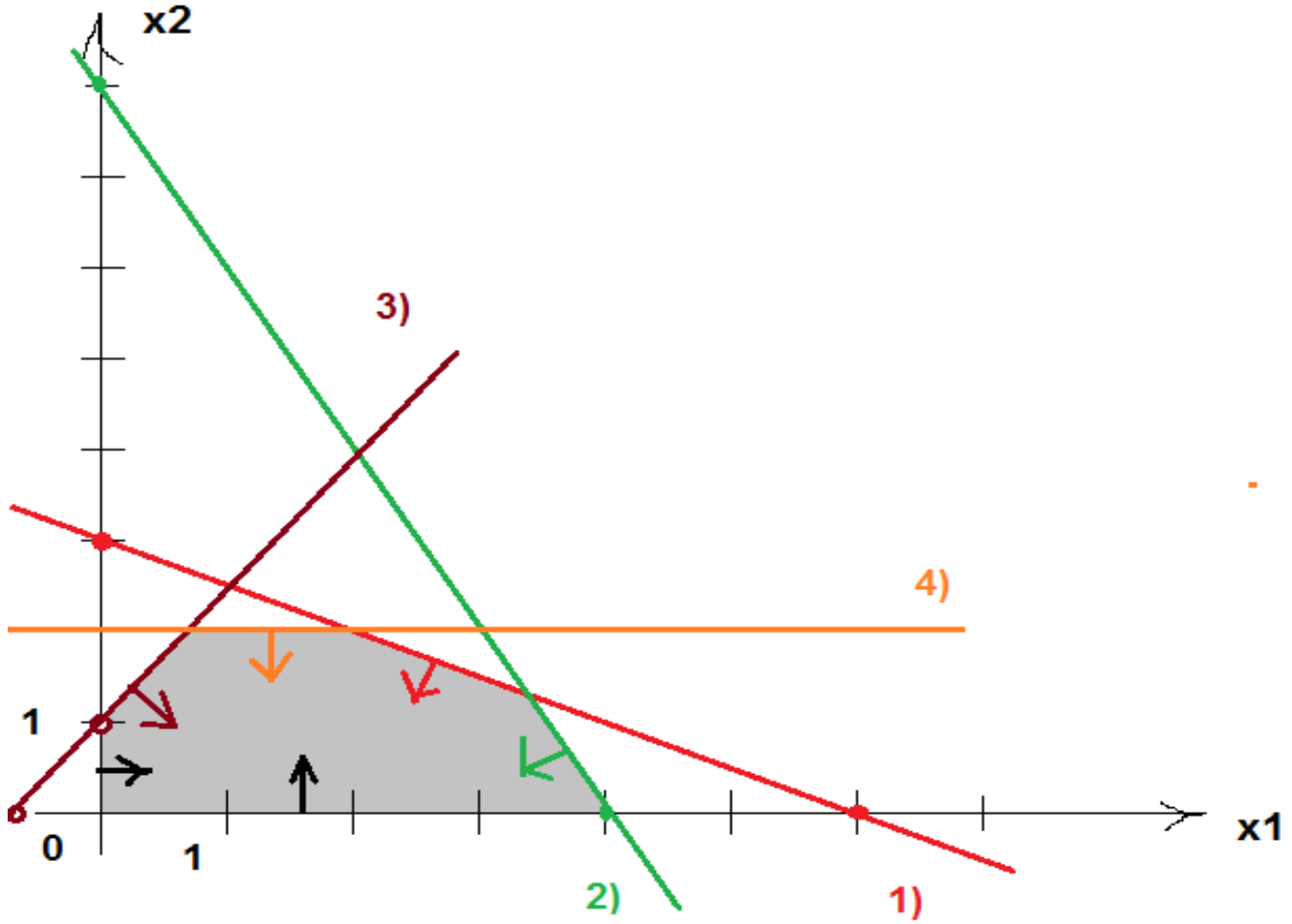
$$\begin{Bmatrix} x_1 = 0 \\ x_2 = 8 \end{Bmatrix} \begin{Bmatrix} x_1 = 4 \\ x_2 = 0 \end{Bmatrix}$$

3) $x_2 - x_1 \le 1 \rightarrow x_2 - x_1 = 1$

$$\begin{Bmatrix} x_1 = 0 \\ x_2 = 1 \end{Bmatrix} \begin{Bmatrix} x_1 = -1 \\ x_2 = 0 \end{Bmatrix}$$

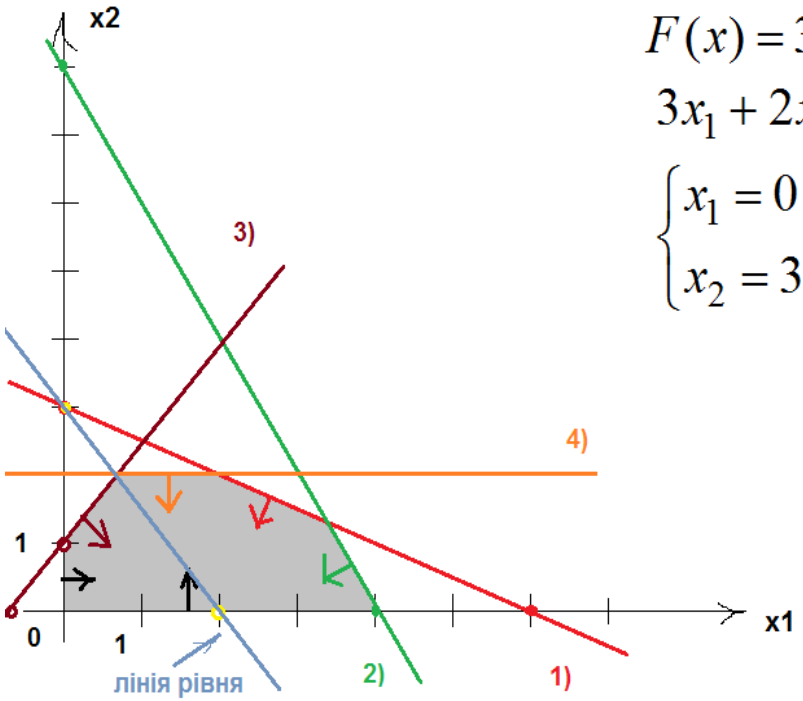
4) $x_2 \le 2$

Lines and feasible region



Drawing the level curve

- The feasible region is a polygon and a convex set. The extreme-points in the convex set can be determined by solving linear equations systems.
- We now search for the optimal solution, which will be the point in the feasible region with the largest value of *z*.



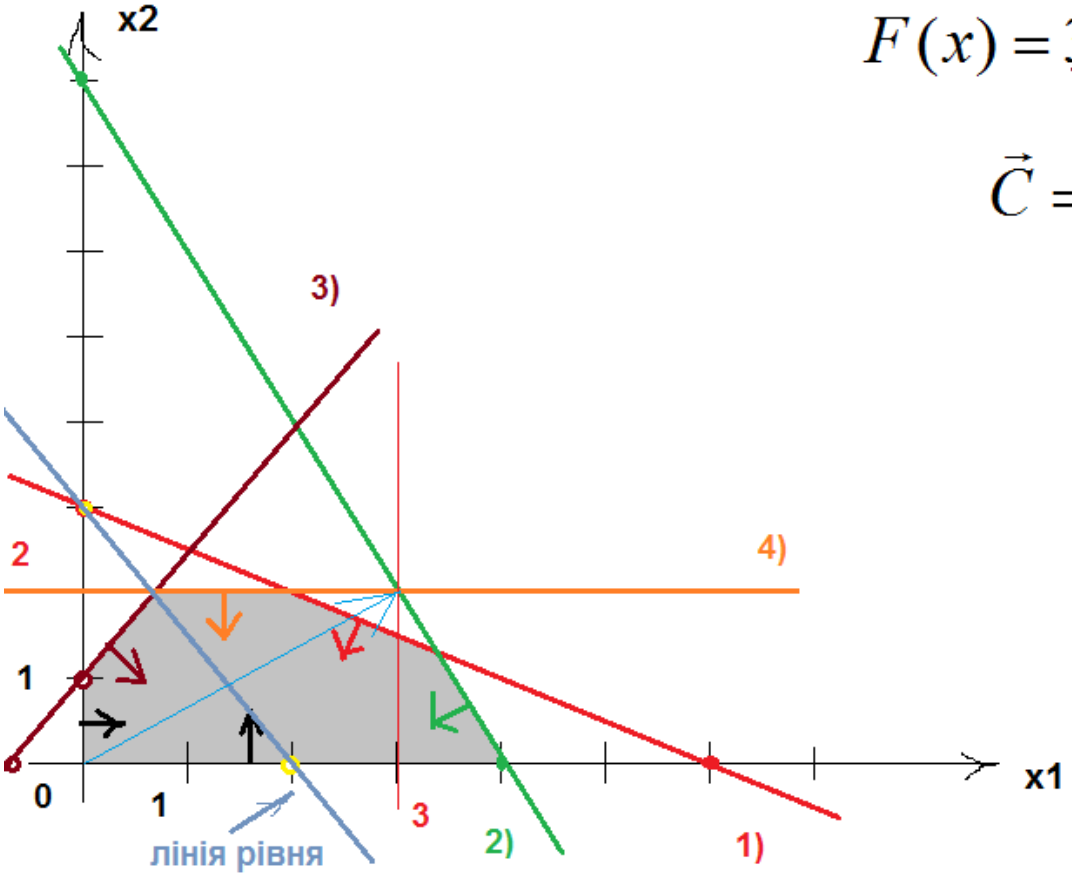
$$F(x) = 3x_1 + 2x_2$$

$$3x_1 + 2x_2 = 6$$

$$\begin{cases} x_1 = 0 \\ x_2 = 3 \end{cases} \begin{cases} x_1 = 2 \\ x_2 = 0 \end{cases}$$

To draw the objective function for a particular value of z, we choose any point in the feasible region and compute its *z* value. We can find all other objective function lines by moving parallel to the line we have drawn.
Thus, we move the line that represents the objective function in the direction that increases *z*.

Gradient vector

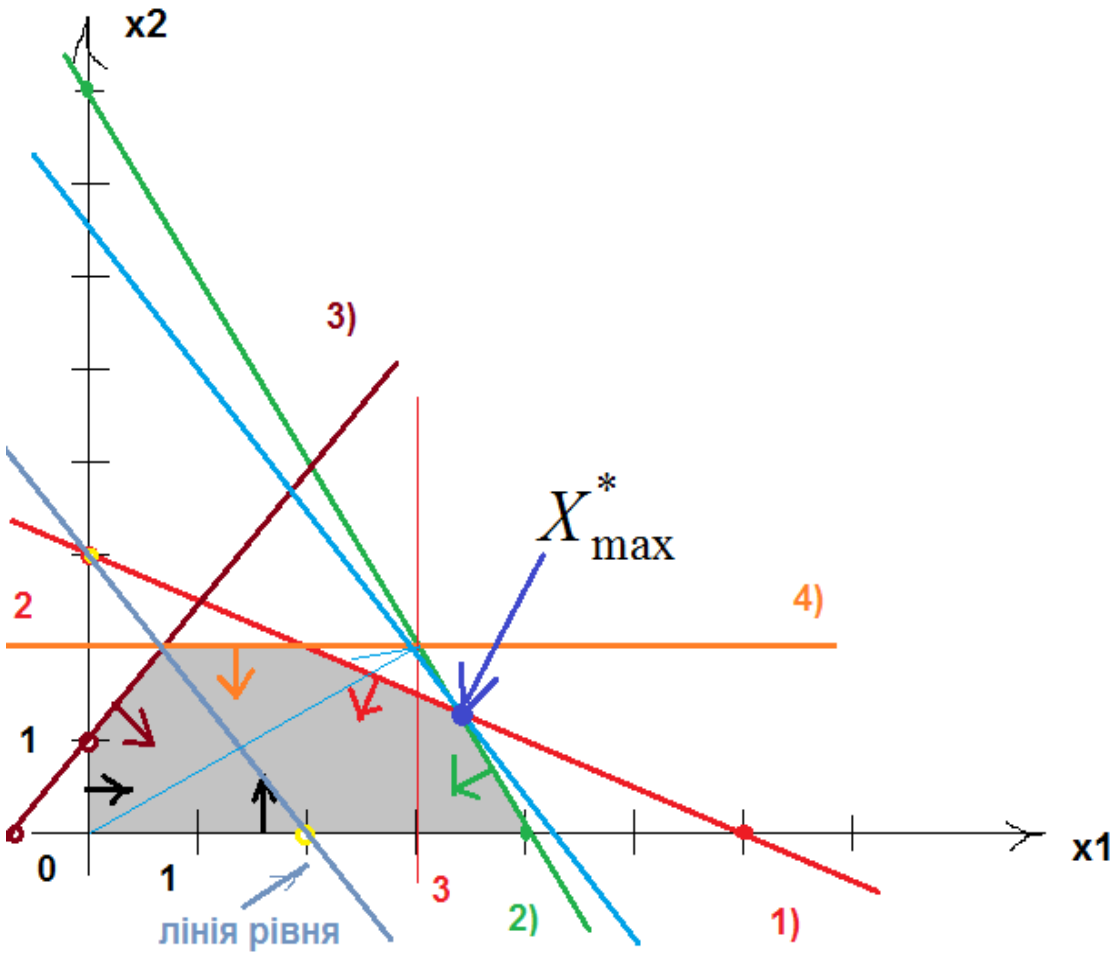


$$F(x) = 3x_1 + 2x_2$$

$$\vec{C} = (3, 2)$$

Алгоритм розв’язання задачі граф. методом

Note that we move the line as long as it intersects with the feasible region. Once the border of the feasible region is reached, the optimal solution is found. The graphical representation shows that the optimal solution is the point that the optimal objective value is z = 12.667.



Finding the optimal point

- Determine the coordinates of the point max (min) of the target function
- To calculate the coordinates of the optimal point X *, solve the system of equations of lines at the intersection of which is X *

$$\begin{cases} x_1 + 2x_2 = 6 \\ 2x_1 + x_2 = 8 \end{cases}$$

$$x_1 = 6 - 2x_2$$

$$2(6 - 2x_2) + x_2 = 8$$

$$12 - 4x_2 + x_2 = 8$$

$$-3x_2 = -4$$

$$x_2 = \frac{4}{3} = 1\frac{1}{3}$$

$$x_1 = 6 - 2 \cdot \frac{4}{3} = \frac{18}{3} - \frac{8}{3} = \frac{10}{3} = 3\frac{1}{3}$$

$$X^* = \left(3\frac{1}{3}, 1\frac{1}{3}\right)$$

$$F(x) = 3 \cdot 3\frac{1}{3} + 2 \cdot 1\frac{1}{3} =$$

$$= \frac{30}{3} + \frac{8}{3} = \frac{38}{3} = 12\frac{2}{3}$$

Solving LP problems using scipy.optimize

```
from scipy.optimize import linprog
import time
start = time.time()
c = [-3,-2] #Функция цели
A = [[1,2],[2,1],[-1,1],[0,1]]  #'A'
b = [6,8,1,2] #'b'
x0_bounds = (0, None)
x1_bounds = (0, None)
res = linprog(c, A_ub=A, b_ub=b, bounds=[x0_bounds, x1_bounds])
print(res)
stop = time.time()
print ("Time :")
print(stop - start)
```

Solving LP problems using scipy.optimize

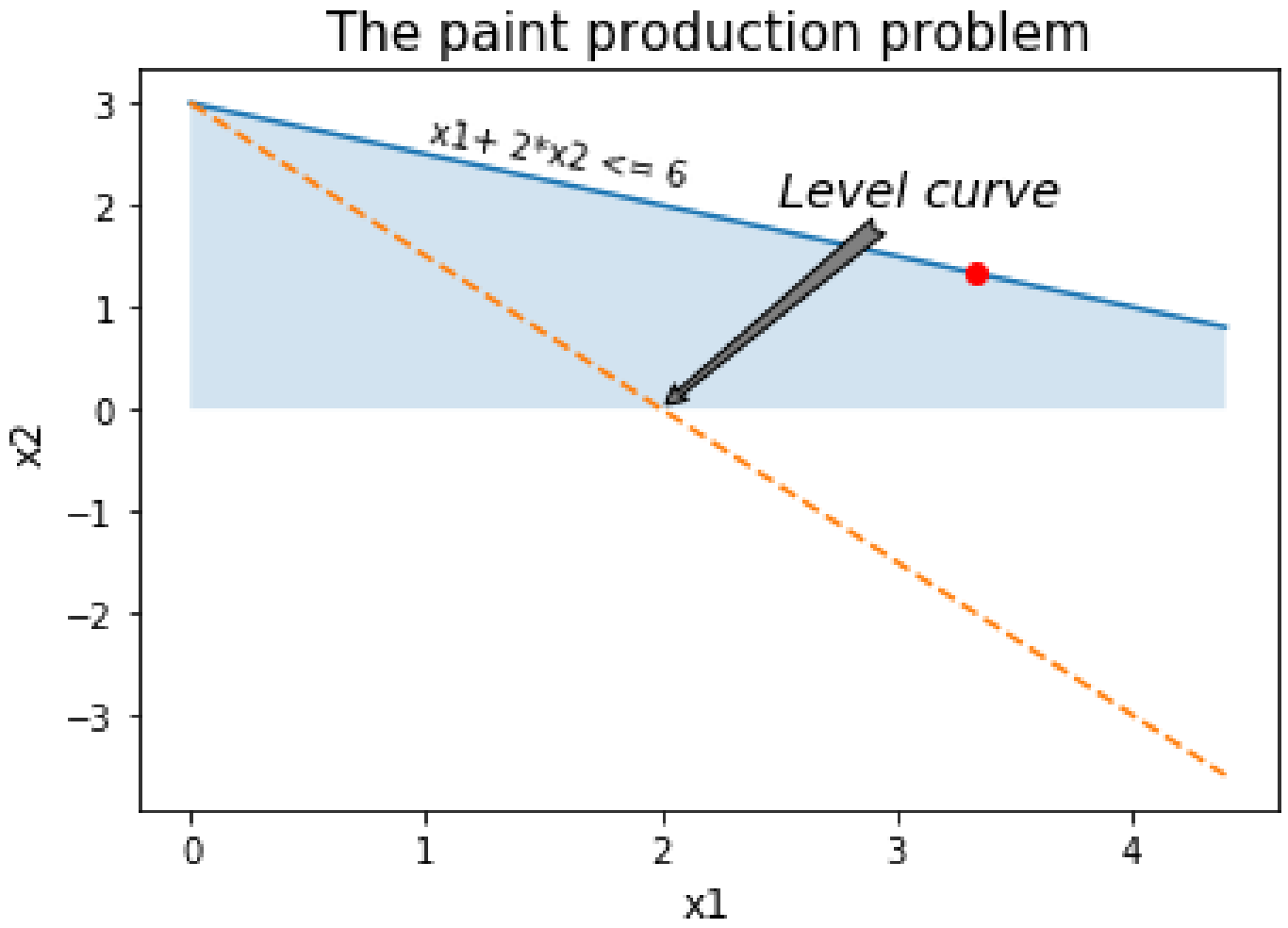
```
     con: array([], dtype=float64)
     fun: -12.666666666663677
 message: 'Optimization terminated successfully.'
     nit: 5
   slack: array([9.94582194e-12, 1.99360528e-11, 3.00000000e+00,
6.66666667e-01])
  status: 0
 success: True
       x: array([3.33333333, 1.33333333])
Time :
0.018000841140747 07
```

For graphical representation

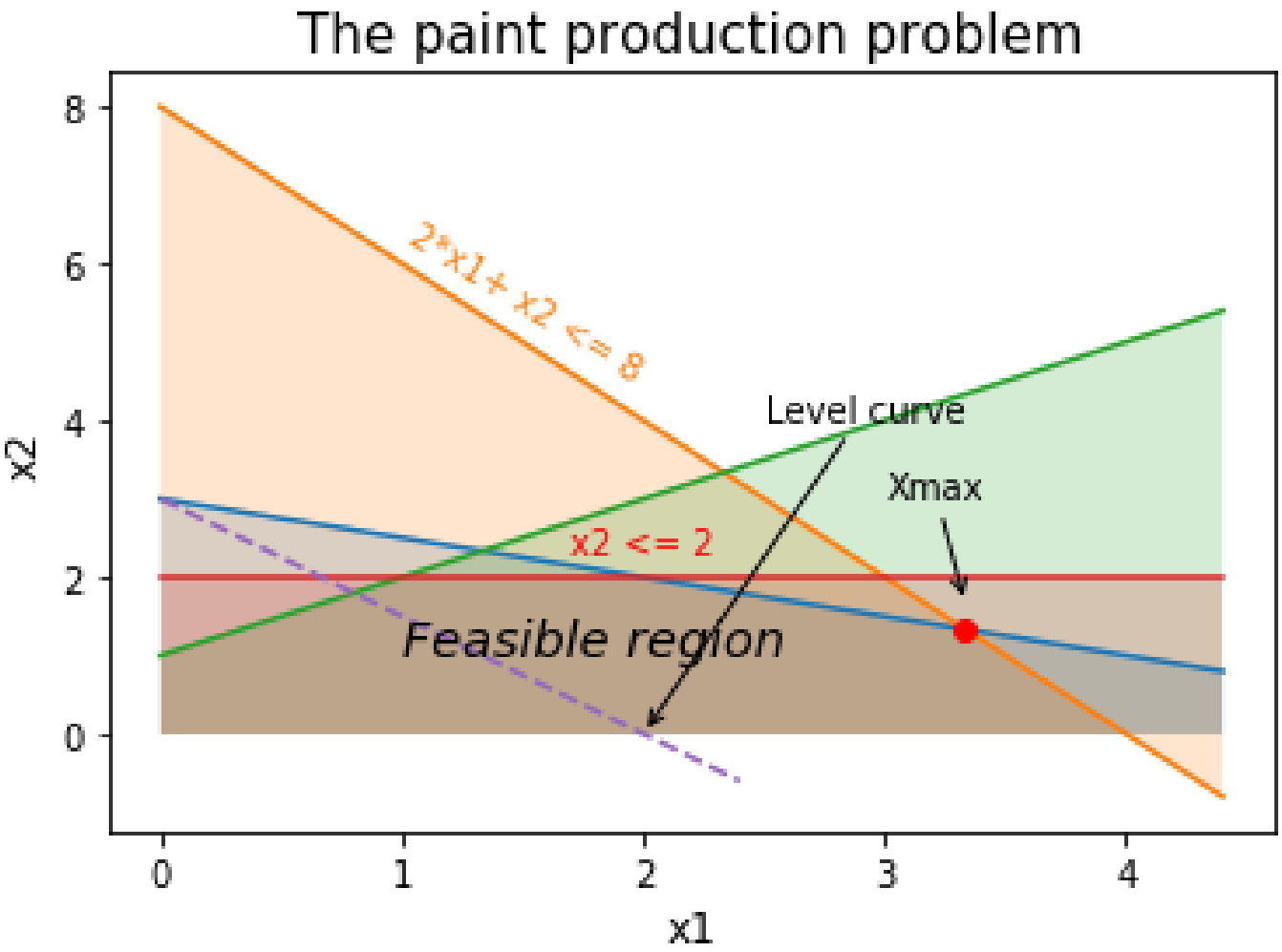
```
def f(a1,a2,b,x):
    return (b-a1*x)/a2 #define the lines
```

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0,4.5,0.1)
y = f(b=b[0],a1=A[0][0],x=x,a2=A[0][1])

plt.plot(x,y)
plt.fill_between(x,y, alpha=0.2)
plt.text(1, 2.2, 'x1+ 2*x2 <= 6', rotation=-10 % 360)
plt.plot(x,(c[0]*c[1]-np.abs(c[0])*x)/np.abs(c[1]),'--')

plt.annotate('Level curve', xy=(2, 0),
             xytext=(2.5, 2), size='x-large', style="italic",
             arrowprops=dict(facecolor='gray',arrowstyle="fancy"))

plt.plot(res.x[0], res.x[1], 'ro')

plt.title('The paint production problem', fontsize=15)
plt.xlabel('x1', fontsize=12)
plt.ylabel('x2', fontsize=12, color='black')
plt.savefig("paint.png", dpi=300)
plt.show;
```

For graphical representation



An approximate view of a graphical solution for laboratory work



Solving LP problem using PuLP

```
from pulp import *
import time
start = time.time()
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
problem = pulp.LpProblem('0',LpMaximize)
problem += 3*x1 +2*x2, "Objective function"
problem += x1+ 2*x2 <= 6, "1-st constrain"
problem += 2*x1+ x2 <= 8, "2-d constrain "
problem += x2 - x1 <= 1, "3-d constrain "
problem += x2 <=2, "4th constrain "
problem.solve()
print ("Result:")
for variable in problem.variables():
    print (variable.name, "=", variable.varValue)
print ("Income:")
print (value(problem.objective))
stop = time.time()
print ("Time :")
print(stop - start)
```

```
Result:
x1 = 3.3333333
x2 = 1.3333333
Income:
12.666666500000002
Time :
0.15200871896362305
```

Thank you for your attention!