Linear programming

Linear Programming

Linear programming (LP, also called linear optimization) is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships. Linear programming is a special case of mathematical programming (also known as mathematical optimization).

More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. Its feasible region is a convex polytope, which is a set defined as the intersection of finitely many half spaces, each of which is defined by a linear inequality. Its objective function is a real-valued affine (linear) function defined on this polyhedron. A linear programming algorithm finds a point in the polytope where this function has the smallest (or largest) value if such a point exists.

Linear programs are problems that can be expressed in *canonical form* as

\begin{aligned}& {\text{maximize}}&&\mathbf{c}^{T}\mathbf{x} \\ &{\text{subject to}}&&A\mathbf{x}\leq\mathbf{b} \\ & &&\mathbf{x}\geq\mathbf{0}. \end{aligned}

where $x\in\mathbb{R}^n$ is the unknown *decision variable*, while $c\in\mathbb{R}^n$, $A\in\mathbb{R}^{m\times n}$ and $b\in\mathbb{R}^{m}$ are given parameters of the linear program problem.

Linear programming can be applied to various fields of study. It is widely used in mathematics, and to a lesser extent in business, economics, and for some engineering problems. Industries that use linear programming models include transportation, energy, telecommunications, and manufacturing. It has proven useful in modeling diverse types of problems in planning, routing, scheduling, assignment, and design.

Algorithms for LP

It can be seen that, in case a solution of a given LP problem exists, it is most likely a vertex of the feasibility set. In general, we can apply different algorithms to find this solution:

- Simplex algorithm
  - Bland's rule — rule to avoid cycling in the simplex method
  - Klee–Minty cube — perturbed (hyper)cube; simplex method has exponential complexity on such a domain
  - Criss-cross algorithm — similar to the simplex algorithm

  - Big M method — variation of simplex algorithm for problems with both "less than" and "greater than" constraints
- Interior point method
  - Ellipsoid method
  - Karmarkar's algorithm
  - Mehrotra predictor–corrector method
- Column generation
- k-approximation of k-hitting set — algorithm for specific LP problems (to find a weighted hitting set)

Example of LP problem

Fiat is projecting the new "Topolino", in two different versions:

- Hybrid
- Full Electric

Fiat's Business Analysts forecast a demand of 80 "Topolino Hybrid" and 20 "Topolino Full Electric" cars per day during the first year. Taking into account the production cost and the capacity of each industrial plant:

Site ID	Site	Hourly Capacity	Hourly Cost
0	Torino	7 Hybrid, 1 Full Electric	9000
1	Buenos Aires	4 Hybrid, 2 Full Electric	6000

determine the optimal decision to achieve the production requirements with the minimum cost.

It is worth noticing that the above problem can be written as follows:

\begin{aligned}& {\text{minimize}}&& 9000 x_0 + 6000 x_1 \\ &{\text{subject to}}&& 80 \leq 7 x_0 + 4 x_1 \\ & && 20 \leq x_0 + 2 x_1 \\ & && 0 \leq x_0 \leq 24 \\ & && 0 \leq x_1 \leq 24 \\ \end{aligned}

where the decision variables are:

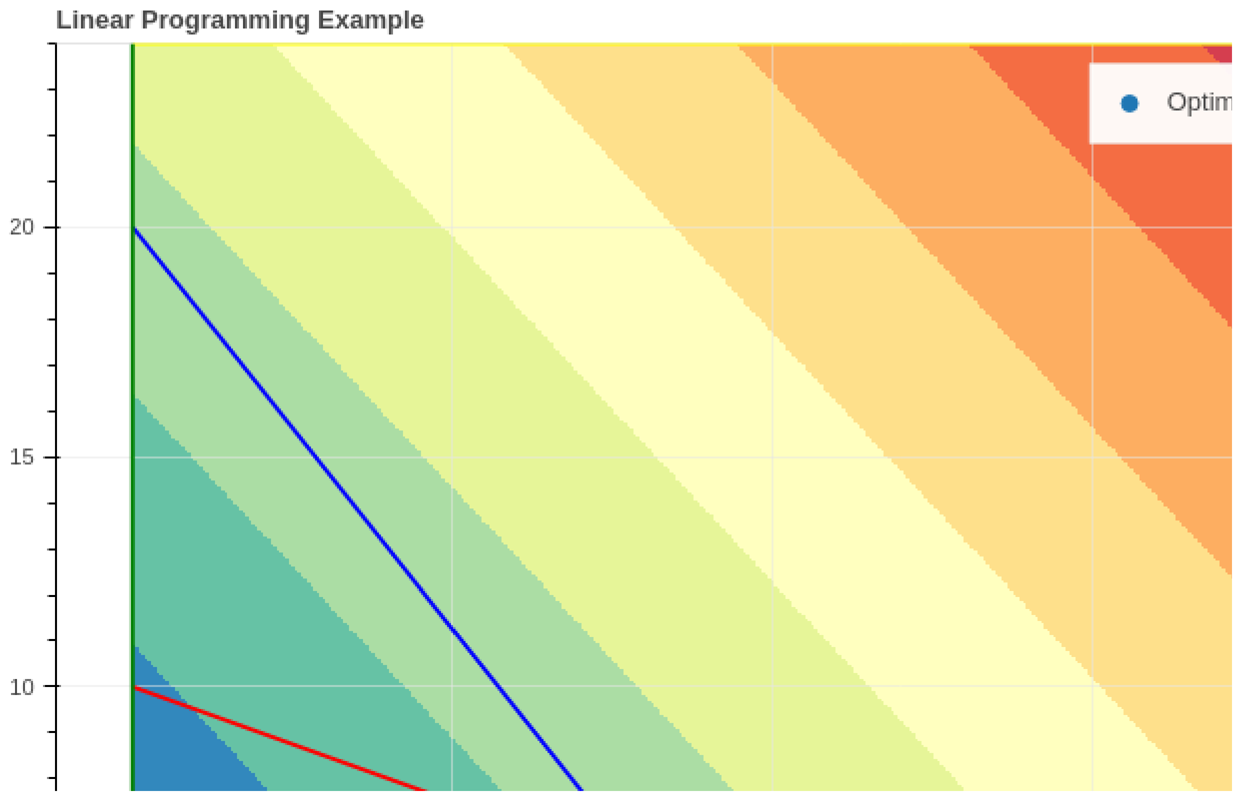
- $x_0$: number of working hours per day in Torino
- $x_1$: number of working hours per day in Buenos Aires

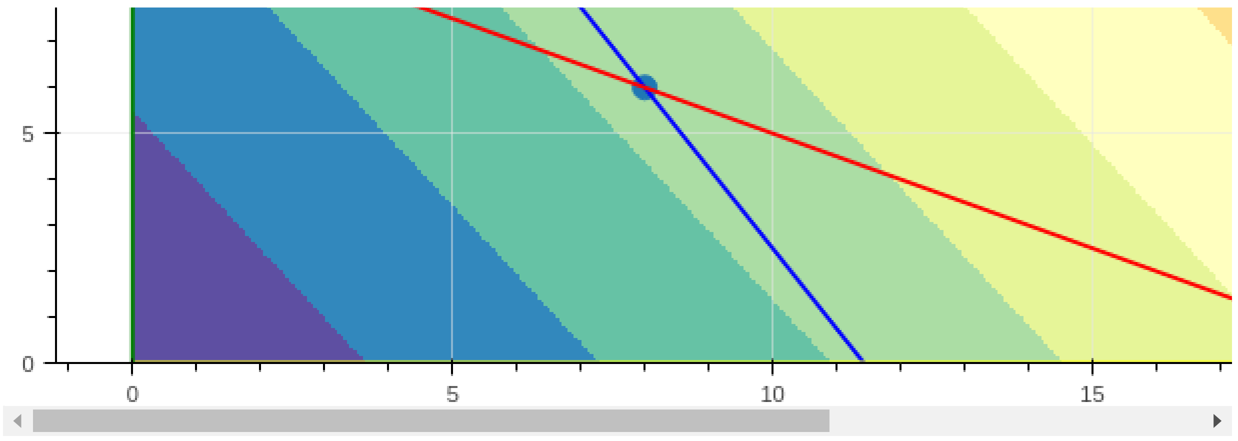
In [1]:
```
# Start Bokeh server
from ipywidgets import interact
import numpy as np

from bokeh.io import push_notebook, show, output_notebook
from bokeh.plotting import figure
output_notebook()
```

BokehJS 2.3.3 successfully loaded.

In [2]:
```
from scipy.optimize import linprog

# Solve the linear programming problem
obj = [9000, 6000]
lhs_ineq = [
    [-7, -4],
    [-1, -2]
]
rhs_ineq = [
    -80,
    -20
]
bnd = [
    [0, 24],
    [0, 24]
]

opt = linprog(
    c=obj,
    A_ub=lhs_ineq,
    b_ub=rhs_ineq,
    bounds=bnd,
    method="revised simplex"
)
print(opt)
# Define elements to plot:
# - Feasible set
# - Objective function
# - Optimal solution
x_0 = np.linspace(0, 24, 2000)
x_1 = np.linspace(0, 24, 2000)

ineq_0 = 20 - 1.75 * x_0
ineq_1 = 10 - 0.5 * x_0


N = 500
X_0 = np.linspace(0, 24, N)
X_1 = np.linspace(0, 24, N)
XX_0, XX_1 = np.meshgrid(X_0, X_1)
f = 9000 * XX_0 + 6000 * XX_1

# Plot the elements
p = figure(
    title="Linear Programming Example",
    plot_width=900,
    y_range=(0,24),
    tooltips=[("x_0", "$x"), ("x_1", "$y"), ("cost", "@image")]
)

p.circle(opt.x[0], opt.x[1], legend_label="""
Optimal solution: ({:.0f}, {:.0f}) -
Optimal cost: {:.0f}
""".format(opt.x[0], opt.x[1], opt.fun), size=12)

p.line(x_0, ineq_0, line_color="blue", line_width=2)
p.line(x_0, ineq_1, line_color="red", line_width=2)
p.line(x_0, 0, line_color="yellow", line_width=2)
p.line(x_0, 24, line_color="yellow", line_width=2)
p.line(0, x_1, line_color="green", line_width=2)
p.line(24, x_1, line_color="green", line_width=2)

p.image(image=[f], x=0, y=0, dw=24, dh=24, palette="Spectral11",
level="image")
p.grid.grid_line_width = 0.5

show(p, notebook_handle=True)
```

```
     con: array([], dtype=float64)
     fun: 108000.0
 message: 'Optimization terminated successfully.'
     nit: 2
   slack: array([0., 0.])
  status: 0
 success: True
       x: array([8., 6.])
```





Out[2]:

```
<Bokeh Notebook handle for In[2]>
```

In [ ]: