In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog
```

1. Massage the Jeklo Ruše toy LP example:

$$\max 3x_1 + 2x_2$$
$$\text{for } 2x_1 + 2x_2 \leq 480$$
$$3x_1 + x_2 \leq 600$$
$$x_1, x_2 \geq 0$$

- Rephrase the problem in the form of (1). Rewrite as a minimization problem and use equalities.

$$\min -3x_1 - 2x_2$$
$$\text{for } 2x_1 + 2x_2 + s_1 = 480$$
$$3x_1 + x_2 + s_2 = 600$$
$$x_1, x_2, s_1, s_2 \geq 0$$

- Write also the corresponding dual problem (2).

Here we should note that s_1 and s_2 are not the same variables as in primal, but we use the same notation so it fits the article notation for artificial problems later on.

$$\max -480y_1 - 600y_2$$
$$\text{for } 2y_1 + 3y_2 - s_1 = 3$$
$$2y_1 + y_2 - s_2 = 2$$
$$s_1, s_2 \geq 0$$

- Add a pair of variables (x_{m+1}, x_{m+2}, Section 4) as in (6). Use scaling and try to figure which M is sufficient.

In the following we use W = 240, since it obtains nice solutions for $d = [2, 2.5]$ and $\rho = [-2, -1.5]$. Also we know the optimal solution is $x_1 = 180, x_2 = 60$. And W has to satisfy $x_i^* \leq W$ for all i. \ Here x_i = x_i/W.

$$\min -3x_1 - 2x_2 + Mx_{m+2}$$
$$\text{for } 2x_1 + 2x_2 - 2x_{m+2} + s_1 = 2$$
$$3x_1 + x_2 - 1.5x_{m+2} + s_2 = 2.5$$
$$x_1 + x_2 + x_{m+1} + x_{m+2} = m + 2$$
$$x_1, x_2, x_{m+1}, x_{m+2}, s_1, s_2 \geq 0$$

- Write also the corresponding dual problem (7).

$$\max -2y_1 - 2.5y_2 + (m+2)y_{n+1}$$
$$\text{for } 2y_1 + 3y_2 + y_{n+1} - s_1 = 3$$
$$2y_1 + y_2 + y_{n+1} - s_2 = 2$$
$$-2y_1 - 1.5y_2 + y_{n+1} + s_{m+2} = M$$
$$y_{n+1} + s_{m+1} = 0$$
$$s_1, s_2, s_{m+1}, s_{m+2} \geq 0$$

- Find a strictly feasible initial solution for (6) and (7). Again, you will have to be somewhat creative with the scaling of variables.

In [ ]:

```
m, n = 2, 2
M = 3
c = np.array([-3, -2, 0, M, 0, 0])
A = np.array([[2,2,0,-2, 1, 0],
              [3,1,0,-1.5,0,1],
              [1,1,1,1,0,0]])
b = np.array([2, 2.5, m+2])
bounds = [(0, None) for i in range(6)]
linprog(c, A_eq=A, b_eq=b, bounds=bounds, method='highs-ipm').x
```

Out[ ]:

```
array([0.75, 0.25, 3.  , 0.  , 0.  , 0.  ])
```

We obtained the optimal solution for our primal Jeklo Ruše problem. $x_1' = \frac{3}{4}$ and $x_2' = \frac{1}{4}$. Since we used W = 240 for our scaling parameter. Our optimal solution for the original problem are: $x_1 = \frac{3}{4} * 240 = 180$ and $x_2 = \frac{1}{4} * 240 = 60$

In [ ]:

```
m, n = 2, 2
M = 3
c = -np.array([-2, -2.5, m+2, 0, 0, 0, 0])
A = np.array([[2, 3, 1, -1, 0, 0, 0],
              [2, 1, 1, 0 , -1, 0, 0],
              [-2, -1.5, 1, 0, 0, 0, 1],
              [0, 0, 1, 0, 0, 1, 0]])
b = np.array([3, 2, M, 0])
bounds = [(None,None) for i in range(3)] # y are unbounded
bounds += [(0, None) for i in range(4)] # s are bounded
linprog(c, A_eq=A, b_eq=b, bounds=bounds, method='highs-ipm').x
```

Out[ ]:

```
array([ 0.75,  0.5 , -0.  ,  0.  ,  0.  ,  0.  ,  5.25])
```

We obtained the optimal solution for our dual Jeklo Ruše problem. $y_1 = \frac{3}{4}$ and $y_2 = \frac{1}{2}$

In [ ]:

```
m, n = 2, 2
W = 240
M = 3
A = np.array([[2, 2], [3, 1]])
b = np.array([[480], [600]])
c = np.array([[-3], [-2]])
d = b / W
e = np.ones((n, 1))
rho = d - A @ e
mu = 6 * np.sqrt(M**2 + sum(c**2))

x = np.ones((m, 1))
x_m1 = 1
x_m2 = 1
y = np.zeros((n, 1))
y_n1 = mu
s = -c + e * mu
s_m1 = -mu
s_m2 = M - mu

print("Check feasibility for primal problem:")
print(all(np.isclose(A @ x + rho * x_m2, d)))
print(all(np.isclose(e.T @ x + x_m1 + x_m2, m + 2)))

print("Check feasibility for dual problem:")
print(all(np.isclose(A.T @ y + e * y_n1 - s, c)))
```

```
print(all(np.isclose(rho.T @ y + y_n1 + s_m2, M)))
print(all(np.isclose(y_n1 + s_m1, 0)))
```

```
Check feasibility for primal problem:
True
True
Check feasibility for dual problem:
True
True
True
```

In [ ]:

```
# check for the article lower bound solutions

m, n = 2,2
c = np.array([3,2])
U = 600
W = (n*U)**n
L = 1/W**2 * 1 / (2*m*((n+1)*U)**(n+1))
M = (4*m*U)/L
M
```

Out[ ]:

```
2.3219011584e+26
```

In [ ]:

```
mu = 6 * np.sqrt(M**2 + sum(c**2))
mu
```

Out[ ]:

```
1.39314069504e+27
```

Jeklo Ruše example find vertices

Rewriting the Jeklo Ruše example with slack variables we obtain

$$\begin{gathered}\max 3x_1 + 2x_2 \\ \text{for } 2x_1 + 2x_2 + x_3 = 480 \\ 3x_1 + x_2 + x_4 = 600 \\ x_1, x_2, x_3, x_4 \geq 0\end{gathered}$$

Every vertex in a feasible solution set can be obtained by choosing a pair of variables in $\{x_1, x_2, x_3, x_4\}$, setting these to zero, and computing the values of the remaining pair from the above constraints.

However, not every pair of set-to-zero variables gives a feasible vertex. Naively we could still do the following.

2. For every pair of variables from $\{x_1, x_2, x_3, x_4\}$, set these to zero, compute the values in the remaining pair, test whether you get a feasible solution, and if so, compute the cost function. Finally, comment on your results.

a) Set x_1 and x_2 to 0:

$$\begin{gathered}x_3 = 480 \\ x_4 = 600 \\ COST = 0\end{gathered}$$

We get a solution in vertex (0,0) of the original problem, which is feasible.

b) Set x_1 and x_3 to 0:

$$2x_2 = 480,\ x_2 = 240$$
$$x_2 + x_4 = 600,\ x_4 = 360$$
$$COST = 480$$

We get a solution in vertex (0,240) of the original problem, which is feasible.

c) Set x_1 and x_4 to 0:

$$2x_2 + x_3 = 480$$
$$x_2 = 600$$
$$COST = NOT\ FEASIBLE$$

We cannot satisfy the constraint such that all x_i are positive, so the solution is not feasible.

d) Set x_2 and x_3 to 0:

$$2x_1 = 480,\ x_1 = 240$$
$$3x_1 + x_4 = 600$$
$$COST = NOT\ FEASIBLE$$

We cannot satisfy the constraint such that all x_i are positive, so the solution is not feasible.

e) Set x_2 and x_4 to 0:

$$2x_1 + x_3 = 480,\ x_3 = 80$$
$$3x_1 = 600,\ x_1 = 200$$
$$COST = 600$$

We get a solution in vertex (200, 0) of the original problem, which is feasible.

f) Set x_3 and x_4 to 0:

$$2x_1 + 2x_2 = 480$$
$$3x_1 + x_2 = 600$$
$$x_1 = 180,\ x_2 = 60$$
$$COST = 660$$

We get a solution in vertex (180, 60) of the original problem, which is feasible and optimal solution, since it has the highest cost.

In [ ]:

```
x = np.arange(0, 240, 0.5)

plt.style.use('seaborn-whitegrid')

y1 = 240 - x
y2 = 600 - 3*x
y3 = np.minimum(y1, y2)

plt.ylim(0, 600)

plt.plot(x, y1, '-k', label='constraints')
plt.plot(x, y2, '-k')
plt.vlines(0, ymin=0, ymax=600, linestyles='dashed', color='black')
plt.hlines(0, xmin=0, xmax=250, linestyles='dashed', color='black')

plt.fill_between(x, y3, color='grey', alpha=0.5, label = 'feasible region')

x = [0, 0, 0, 240, 200, 180]
y = [0, 240, 600, 0, 0, 60]
```

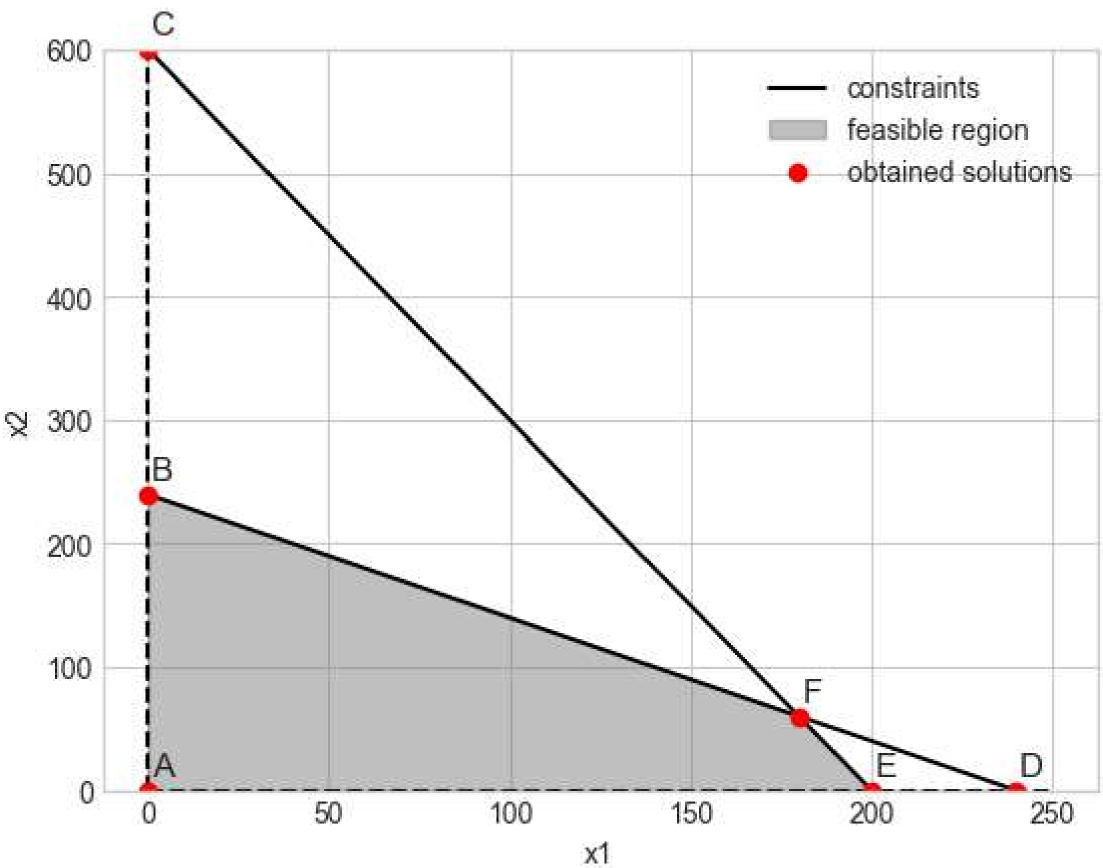
```python
labels = ['A', 'B', 'C', 'D', 'E', 'F']

plt.plot(x, y , 'o', color='red', label = 'obtained solutions')
for i in range(len(x)):
    plt.text(x[i] + 4, y[i] + 20, labels[i], fontsize=12, ha='center', va='center')

plt.xlabel('x1')
plt.ylabel('x2')

plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x18af0002be0>



We can see that when we set every pair of variables from $\{x_1, x_2, x_3, x_4\}$ to zero we obtain edges for our problem. We observe that we obtain unfeasible solutions for C and D, since they lie outside of feasible region.