Table of Contents

1 Import of standard libraries

In [1]:

```
# Import required library
import matplotlib.pyplot as plt
import scipy.optimize as opt
```

executed in 572ms, finished 10:39:27 2020-05-28

2 The Linear Programming Problem

A decision has to be made on the usage of farm land for wheat and barley.

Variables:

- x_0: area of wheat (ha)
- x_1: area of barley (ha)

Condition based on farm size:

$$x_0 + x_1 \leq f$$

In [2]:
```python
farm_size=40 # ha
```
executed in 3ms, finished 10:39:27 2020-05-28

Production costs:

- Wheat: c_w (€/ha)
- Barley: c_b (€/ha)

Condition based on available reserves:

$$c_w x_0 + c_b x_1 \leq R$$

In [3]:
```python
cost_wheat=1360 # €/ha     1350
cost_barley=1150 # €/ha    1150
reserves=50000 # €
```
executed in 3ms, finished 10:39:27 2020-05-28

Optimize margin (without labour cost)

In [4]:
```python
price_wheat=170 # €/t
yield_wheat=10 # t/ha
price_wheat_straw=243 # €/ha
income_wheat=price_wheat*yield_wheat+price_wheat_straw # €/ha

price_barley=160 # €/t
yield_barley=9 # t/ha
price_barley_straw=300 # €/ha
income_barley=price_barley*yield_barley+price_barley_straw # €/ha
```
executed in 6ms, finished 10:39:27 2020-05-28

In [5]:
```
# Input parameters and bounds
A = [[1,1], [cost_wheat, cost_barley]]
b = [farm_size,reserves]
c = [-income_wheat, -income_barley]
limits=((0, farm_size),(0, farm_size))
```
executed in 5ms, finished 10:39:27 2020-05-28

In [6]:
```
# Run the model
res = opt.linprog(c, A, b, bounds=limits,method='simplex')
```
executed in 11ms, finished 10:39:27 2020-05-28

In [7]:
```
# Get results
res
```
executed in 6ms, finished 10:39:27 2020-05-28

Out[7]:
```
     con: array([], dtype=float64)
     fun: -73466.66666666667
 message: 'Optimization terminated successfully.'
     nit: 5
   slack: array([0., 0.])
  status: 0
 success: True
       x: array([19.04761905, 20.95238095])
```

In [8]:
```
# Access each variable x1 and x2
x0 = res.x[0]
x1 = res.x[1]
(x0, x1)
```
executed in 6ms, finished 10:39:28 2020-05-28

Out[8]:
```
(19.047619047619055, 20.952380952380945)
```

In [9]:
```
opt_val = int(res.fun)
opt_val
```
executed in 5ms, finished 10:39:28 2020-05-28

Out[9]:
```
-73466
```

3 Graphical Illustration

3.1 Utilities

$a_{i,0}x + a_{i,1}y \leq b_i$ implies $y \leq \frac{b_i - a_{i,0}x}{a_{i,1}}$

In [10]:
```
def line(A,b,i,x):
    return (b[i]-A[i][0]*x)/A[i][1]
```
executed in 4ms, finished 10:39:28 2020-05-28

$c_0x + c_1y = g$ implies $y = \frac{g - c_0x}{c_1}$

In [11]:
```
def grad(c,g,x):
    return (g-c[0]*x)/c[1]
```
executed in 4ms, finished 10:39:28 2020-05-28

In [12]:
```
def MIN(a, b):
    return b if a==None else a if b==None else min(a,b)

def MAX(a, b):
    return b if a==None else a if b==None else max(a,b)

def plot_linopt(A, b, c, bounds, res,
                borders=None, dx=5, dy=5,
                title=None, labels=None,
                solution=None, legend=None, output=False):

    ax=plt.axes()
    ax.grid(True)

    if borders==None:
        borders=[(res.x[0]-dx, res.x[0]+dx),
                 (res.x[1]-dx, res.x[1]+dx)]
```

```python
                 (res.x[1]-dy, res.x[1]+dy)]

    # set drawing region (xmin, xmax) (ymin, ymax)
    xmin = borders[0][0]
    xmax = borders[0][1]
    ymin = borders[1][0]
    ymax = borders[1][1]

    ax.set_xlim((xmin,xmax))
    ax.set_ylim((ymin,ymax))
    if labels!=None:
        plt.xlabel(labels[0])
        plt.ylabel(labels[1])

    if legend==None:
        legend=[]
        for i in range(0, len(A)):
            legend+=['Constraint '+str(i)]

    if solution==None:
        solution='Solution'



    # compute visual bounds (drawing limits if there is no bound)
    xleft = MAX(bounds[0][0], borders[0][0])
    xright = MIN(bounds[0][1], borders[0][1])
    ybottom = MAX(bounds[1][0], borders[1][0])
    ytop = MIN(bounds[1][1], borders[1][1])

    # plot constraints
    x=[xmin,xmax]
    lines=[]
    for i in range(0, len(A)):
        y = [line(A,b,i,xmin), line(A,b,i,xmax)]
        l=plt.plot(x,y,label=legend[i])
        plt.fill_between(x, y, ymin if A[i][1]>0 else ymax, alpha=0.3)
        lines=lines+[l[0]]

    # plot bounding box
```

```
    # plot bounding box
    rangex=[xleft, xright, xright, xleft, xleft]
    rangey=[ybottom, ybottom, ytop, ytop, ybottom]
    l=plt.plot(rangex, rangey,label='Bounds')
    plt.fill_between(rangex, rangey, alpha=0.3)
    lines+=[l[0]]

    # plot optimal cost function
    x=[xmin,xmax]
    lopt=plt.plot(x, [grad(c,res.fun,xmin),grad(c,res.fun,xmax)],
                  color='red', label=solution)

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

    if legend!=None:
        plt.legend(handles=lines+[lopt[0]])

    if title!=None:
        plt.suptitle(title)

    if output:
        print(solution, '=', res.fun)
        for i in range(0, len(c)):
            print(labels[i],'=',res.x[i])
```

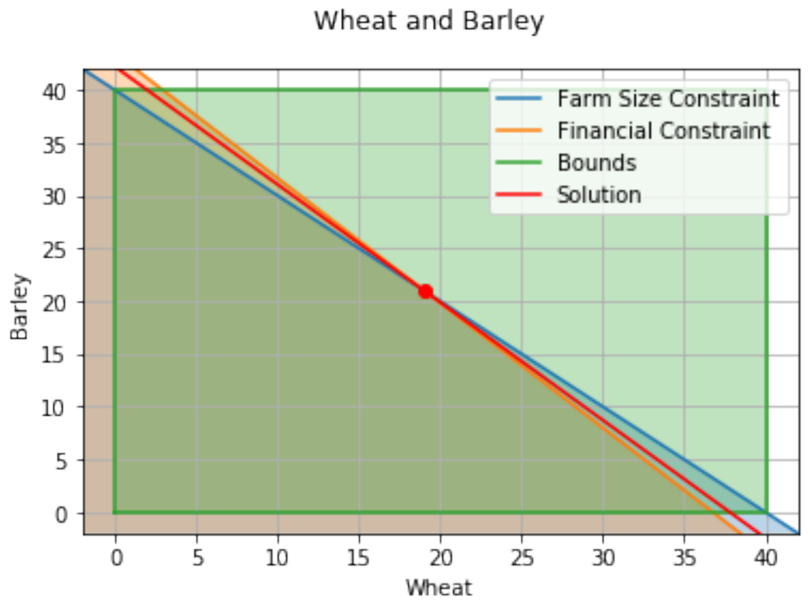
executed in 32ms, finished 10:39:28 2020-05-28

3.2 Global Visualistion

In [13]:

```
# Drawing
# fig=plt.figure(figsize=(4,4), dpi=300)
plot_linopt(A, b, c, limits, res,
            borders=[(-2, 42), (-2,42)],
            title='Wheat and Barley',
            labels=['Wheat', 'Barley'],
            legend=['Farm Size Constraint',
                    'Financial Constraint'])
```

executed in 373ms, finished 10:39:29 2020-05-28



3.3 Visualisation around Optimal Solution

In [14]:

```python
# Drawing
# fig=plt.figure(figsize=(4,4), dpi=300)
plot_linopt(A, b, c, limits, res,
            borders=[(18, 22), (18,22)],
            title='Wheat and Barley',
            labels=['Wheat', 'Barley'],
            legend=['Farm Size Constraint',
                    'Financial Constraint'])
```

executed in 315ms, finished 10:39:29 2020-05-28