# OPT_2

May 21, 2023

## 1 Minimization problem, application to a uniformly-loaded simple beam

In this example, we will look at a minimization problem linked to the deflection of a uniformly-loaded simple beam. The beam is made of a cylinder of length $L$ with an inner diameter $d$ and thickness $t$.

The deflection of this beam is expressed as:

$$u = \frac{5qL^4}{284EI}$$

where $q$ is the load applied to the beam, E its Young's modulus and I is the area moment of inertia.

In this minimization problem, we want to minimize the mass of the beam while ensuring that the deflection $u$ is lower than 250 mm.

The variables we want to optimize are therefore the diameter $d$ and the thickness $t$ of the beam. The thickness of the beam should be restricted to $2 < t < 10$ and the diameter should be restricted to $30 < d < 60$

[ ]:
```
# We import here the python modules we need for this example
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
```

We define three python functions to compute:

1. the are moment of inertia $I$, as function of the diameter $d$ and thickness $t$ of the cylinder (function I(t,d))
2. the mid-point deflection $u$ as function of the applied load $q$, the length of the beam $L$, its Young's modulus $E$ and the area moment of inertia $I$ (function mid_point(q,L,E,I)).
3. the mass of the beam given its diameter $d$, thickness $t$ and density $\rho$ (function mass(rho,L,d,t))

[ ]:
```
def I(t,d):
    return np.pi*((d+2.0*t)**4.0-d**4.0)/64
```

[ ]:
```
def mid_point(x,data):
    [q,L,E,rho] = data
```

```
    t = x[0]
    d = x[1]
    return (5*q*L**4.0)/(384.0*E*I(t,d))
```

[ ]:
```
def mass(rho,L,t,d):
    V_tot = L*rho*np.pi*(d/2.0+t)**2.0
    V_in  = L*rho*np.pi*(d/2.0)**2.0
    return (V_tot-V_in)*1000.0
```

We now give values to the parameters of our problem.

[ ]:
```
q = 100.0 #N/mm
L = 1000.0 #mm
E = 70000.0 #MPa
rho = 2.7e-9
```

In this exercise you need to: 1. define the cost function of the problem 2. define the bounds for the optimization of the thickness $t$ and diameter $d$ of the beam 3. define the inequality constraint equation for the deflection of the beam $u$ 4. use the SLSQP algorithm to determine the optimal combination of thickness and diameter for the beam. Here the initial values can be chosen in the middle of the allowable range for the thickness and diameter of the beam. 5. evaluate the local/global minima by carrying out two more optimization with different starting points. 6. visualize the cost function of the problem using a filled contour plot 7. add the starting points as well as the optimal thicknesses and diameters obtained in questions 4 and 5. What can you conclude on this problem?

**Question 1**

In this optimization problem, the cost function we wish to minimize is the mass of the beam. We therefore built a function which takes in the variables $x$ as well as the additional parameters of the problem.

[ ]:
```
def cost_function(x,data):
    [q,L,E,rho] = data
    t = x[0]
    d = x[1]
    m = mass(rho,L,t,d)
    return m
```

**Question 2**

the bounds are defined as a list of tuples starting with the thickness of the beam $t$ and its diameter $d$

[ ]:
```
bounds = [] # Define bounds
bounds.append((2.0,10.0))  #t
bounds.append((30.0,60.0)) #d
```

**Question 3**

Next we define the inequality constraint relevant to this problem which is that deflection of the

beam $u$ must be lower than 250 mm. The additional variables which are needed are given as extra arguments to the constraint equation.

[ ]:
```
constraints = []
constraints.append({'type': 'ineq', 'fun': lambda x: 250.0-mid_point(x,data)})␣
 ↪# u <= 250.0 mm
```

### Question 4

To prepare the minimization we setup some additional variables. As initial values for the optimization we choose the middle of the allowable range for the thickness $t$ and the diameter $d$, therefore $x_0$ is defined as $[t = 6 \text{ mm}, d = 45 \text{ mm}]$.

[ ]:
```
# We define here some variables we need for the minimization problem
options = {'disp': True, 'maxiter':1000} # display the results
data = [q,L,E,rho]
arguments = (data, ) # This list of parameters is needed by our different␣
 ↪functions
```

[ ]:
```
x0 = [6.0,45.0]
res = optimize.minimize(cost_function, x0,␣
 ↪method='SLSQP',bounds=bounds,options=options,args=arguments,constraints=constraints)
x_opt = res.x
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.7728618131007461
            Iterations: 9
            Function evaluations: 28
            Gradient evaluations: 9
```

The resulting thickness and diameter as well as mass are given in the next cell. We can see that the constraint equation is fulfill despite some numerical errors.

[ ]:
```
print('The optimum thickness of the beam is {}'.format(x_opt[0]))
print('The optimum diameter of the beam is {}'.format(x_opt[1]))
print('The resulting deflection is {} mm'.format(mid_point(x_opt,data)))
print('The resulting mass of the beam is {} kg'.
 ↪format(mass(rho,L,x_opt[0],x_opt[1])))
```

```
The optimum thickness of the beam is 2.0
The optimum diameter of the beam is 43.557325141462314
The resulting deflection is 250.0000000000195 mm
The resulting mass of the beam is 0.7728618131007461 kg
```

### Question 5

We carry out two more optimization starting from the lowest and the highest allowable bounds to check for potential local minima.

[ ]:
```
x0s = [[2.0,30.0],[10.0,60]]
x_opt_2 = []
```

```
#
for x0 in x0s:
    res = optimize.minimize(cost_function, x0,␣
 ↪method='SLSQP',bounds=bounds,options=options,args=arguments,constraints=constraints)
    x_opt_2.append(res.x)
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.7728618131004773
            Iterations: 11
            Function evaluations: 34
            Gradient evaluations: 11
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.772861813100767
            Iterations: 12
            Function evaluations: 37
            Gradient evaluations: 12
```

We can then print the obtained thicknesses and diameters using a simple for loop. What we can observe is that independently of the starting point for the optimization we end up with the same optimal values. This is a good sign that the problem does not possess several local minimas but a single global minimum.

[ ]:
```
for x in x_opt_2:
    print('Thickness t = {}, diameter d 0 {}'.format(*x))
```

```
Thickness t = 2.000000000000002, diameter d 0 43.55732514144642
Thickness t = 2.0, diameter d 0 43.55732514146354
```

### Question 6 & 7

The cost function is plotted in the next cell using a filled contour plot. We add the starting points (in red) for the optimization as well as the optimum values (in black). We can see that the cost function is rather smooth and does not exhibit local minimas.

[ ]:
```
t = np.linspace( 2.0,10.0,100)
d = np.linspace(30.0,60.0,100)

[T,D] = np.meshgrid(t,d)

plt.contourf(T,D,cost_function([T,D],data))
plt.colorbar()

plt.scatter([2.0,6.0,10.0],[30.0,45.0,60.0],c='r',s=100,marker='^')

plt.
 ↪scatter([x_opt[0],x_opt_2[0][0],x_opt_2[1][0]],[x_opt[1],x_opt_2[0][1],x_opt_2[1][1]],c='k'

plt.xlim([1.0,11.0])
plt.ylim([25.0,65.0])
```

```
plt.xlabel('Thickness t (mm)')
plt.ylabel('Diameter d (mm)')
```

[ ]: Text(0, 0.5, 'Diameter d (mm)')