In [ ]:
```
import numpy as np
import roboticstoolbox as rtb
from spatialmath import *
from math import pi
import matplotlib.pyplot as plt
from matplotlib import cm
np.set_printoptions(linewidth=100, formatter={'float': lambda x: f"{x:8.4g}" if abs

%matplotlib notebook
```

The solution of the inverse kinematics problem, similarly to the forward kinematics, must be started with the creation of a model of the manipulator.

In [ ]:
```
L1 = rtb.DHLink(d=1.0, alpha=pi/2, theta=0.0, a=0.5)
L2 = rtb.DHLink(theta=0.0, a=0.7)
robot = rtb.DHRobot([L1, L2])
```

In the next step, it is necessary to determine the position and orientation of the manipulator tip for which the problem is to be solved. This position and orientation should be presented in the form of a homogeneous matrix. One of the simplest ways is to define a translation and rotation matrix. The translation matrix is created using the **SE3** command as in the example:

```
trans = SE3(0.1, 0.2, 0.3)
```
The subsequent arguments correspond to the x, y and z coordinates of the given point.

In [ ]:
```
trans = SE3(0.5, 0.0, 1.7)
trans
```

This creates a homogeneous matrix, the rotation part of which is an identity matrix. To create the appropriate rotation matrix, use the command **SE3.OA**.

```
y = [0,0,1]
z = [1,0,0]
rot = SE3.OA(y, z)
```
Parameters y and z refers to:

- y - vector parallel to the y axis of the tool
- z - vector parallel to the z axis of the tool

In [ ]:
```
y = [0,0,1]
z = [1,0,0]
rot = SE3.OA(y, z)
rot
```

Note that vectors y and z cannot be zero or parallel. However, it is not necessary to normalize the vectors or ensure their perpendicularity. In the case of a pair of non-perpendicular vectors, vector z will be kept in the resulting matrix and vector y will be fitted to it. This operation results from the fact that the vector z determines the so-called direction

of approach, or simply the position of the main axis of the tool, the y vector is responsible for the rotation of the tool around this axis.

In [ ]:
```
y = [1,0,0.5]
z = [1,0,0]
rot = SE3.OA(y, z)
rot
```

By multiplying the translation and rotation matrices obtained in that way, one can easily create the desired homogeneous transformation matrix.

In [ ]:
```
T = trans * rot
T
```

The solution to the inverse kinematics problem can be obtained by calling the **ikine_LM** method on the robot object and passing a homogeneous matrix as an argument.

```
sol = robot.ikine_LM(T)
```
The returned object *sol* contains solution of the problem: vector of joint coordinates and information about whether the given position was reached.

In [ ]:
```
sol = robot.ikine_LM(T)
sol
```

In the example above, you can see that the calculated joint coordinates vector is [1.458, -0.5544]. However, the parameter success=False means that the given position has not been reached (i.e., there is no solution for inverse kinematics problem).

In [ ]:
```
print(sol.success)
print(sol.q)
```

In this case, the problem with the solution is due to the very simple structure of the manipulator, which gives very limited mobility. In such cases, it may not be possible or necessary to maintain all position/orientation constraints. The tool anticipates such situations and makes it possible to specify in the **ikine_LM** method a mask responsible for which elements of the given position must be exactly reached. The mask should be a six-element array of 0s and 1s, in which successive elements mean the need to accurately reproduce, respectively: x, y, and z positions and rotations around the x, y and z axes.

```
mask = np.array([0, 1, 1, 0, 0, 0])   # it is required to exactly reach y and z coordinates
sol = robot.ikine_LM(T, mask=mask)
```
You have to remember that the number of 1s in the mask cannot be larger than the number of degrees of freedom of the manipulator.

In [ ]:
```
mask = np.array([0, 1, 1, 0, 0, 0])
sol = robot.ikine_LM(T, mask=mask)
sol
```

In [ ]:
```
print(sol.success)
print(sol.q)
```

Often, specifying only the given position may not be sufficient to find a solution, even though the position is reachable. If it is possible, it is also worth providing the initial value of the joint variables to the **ikine_LM** method.

```
mask = np.array([0, 1, 1, 0, 0, 0])
q0 = np.array([0.0, 1.0])
sol = robot.ikine_LM(T, q0=q0, mask=mask)
```

In [ ]:
```
mask = np.array([0, 1, 1, 0, 0, 0])
q0 = np.array([0.0, 1.0])
sol = robot.ikine_LM(T, q0=q0, mask=mask)
print(sol.success)
print(sol.q)
```