Tutorial 1: Loading a Robot in CASPR

This tutorial introduces the basics in order to load a robot into the CASPR environment using a script.

```
clear;
```

Loading a model configuration

First, instantiate the ModelConfig object with the name of the robot. In this example, the "Example spatial" robot will be used.

```
model_config = ModelConfig('Example spatial');
```

The list of available robots can be found within the `CASPR_GUI` or `model_config.robotNamesList`

```
robot_names = model_config.robotNamesList
```

```
robot_names = 1×36 cell
'Example point mass XZ'    'Example planar XY'    'Example spatial'    'Example ···
```

Loading a robot model

The `ModelConfig` object is not the robot (kinematics and dynamics) model itself, but must be created. At this point, the "cable set" that define the properties and arrangement of the cables must be specified based on the cable set's ID within the XML cable configuration file. The list of cable sets is stored in the variable `model_config.cableSetNamesList`

```
cable_set_names = model_config.cableSetNamesList
```

```
cable_set_names = 1×3 cell
'basic_7_cables'    'alternative_7_cables'    'basic_8_cables'
```

Furthermore, the name of the default cable set as specified in the XML is stored within

```
default_cable_set_name = model_config.defaultCableSetId
```

```
default_cable_set_name = 
'basic_7_cables'
```

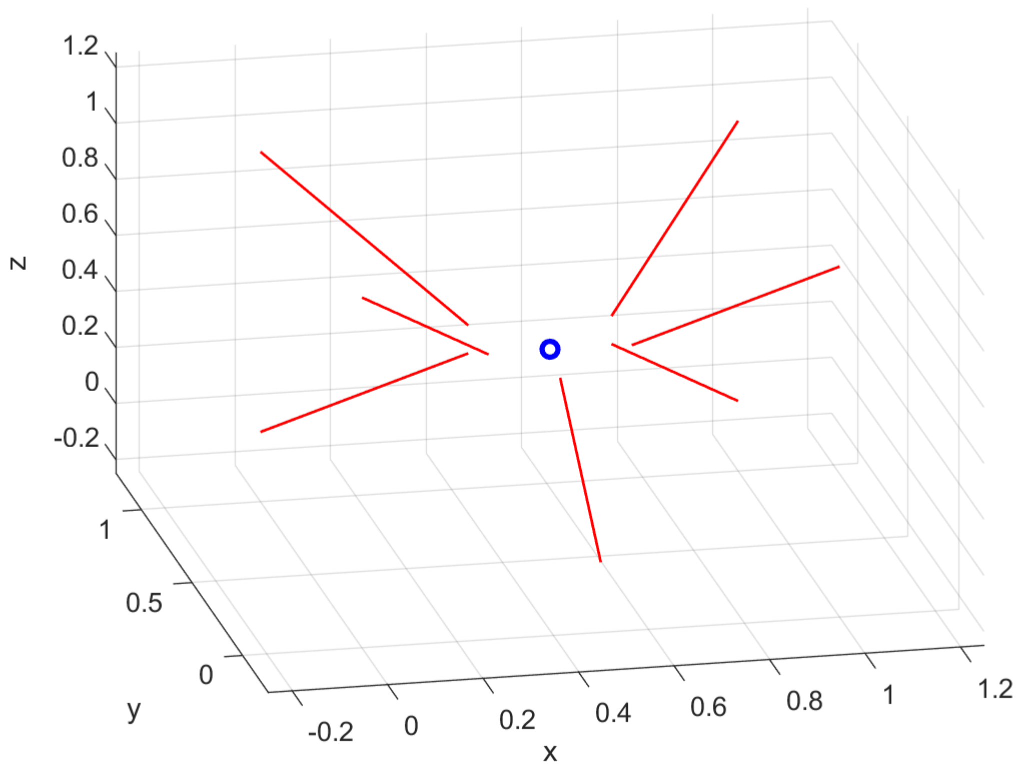
With the cable set, the CDR model can be easily created.

```
cdpr_model = model_config.getModel('basic_7_cables');
```

Visualise the robot

After a robot model is created, it can be easily visualised using the function `MotionSimulatorBase.PlotFrame(...)`

```
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



The `model_config.displayRange` is a vector in the form `[x_min x_max y_min y_max z_min z_max]` and can be specified within the XML cable configuration file

```
disp_range = model_config.displayRange
```

```
disp_range = 1×6
   -0.2500    1.2500   -0.2500    1.2500   -0.2500    1.2500
```

The `model_config.viewAngle` is in the form `[az el]` referring to the azimuth and elevation angles (standard in MATLAB) and can be specified within the XML cable configuration file

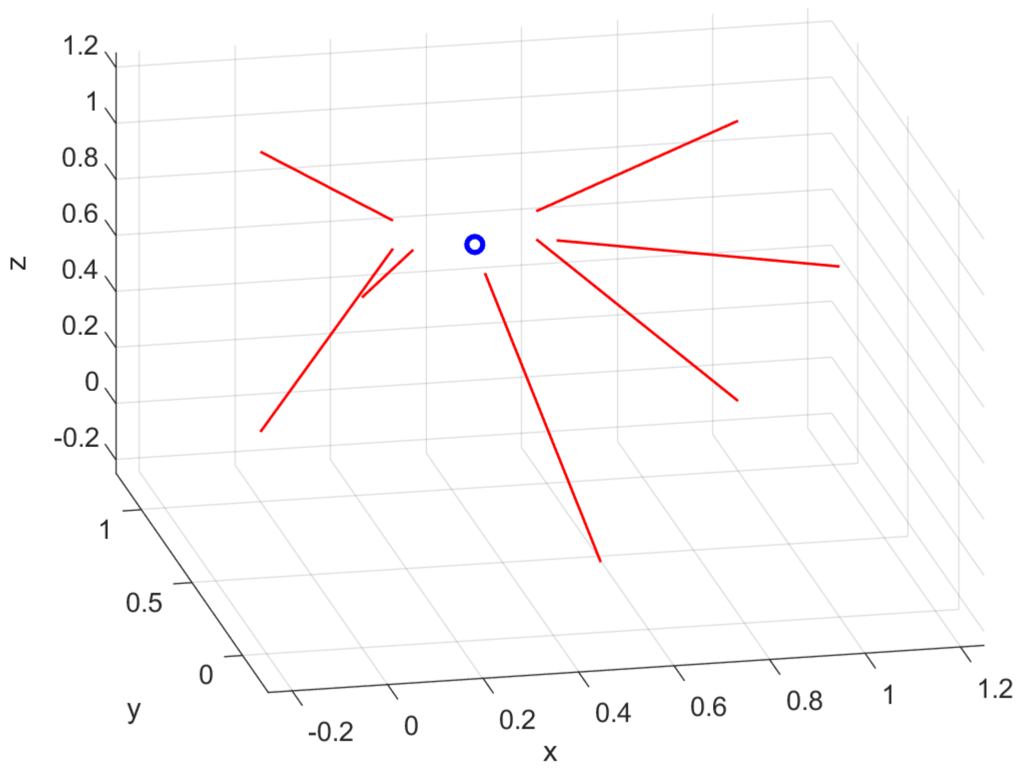
```
view_angle = model_config.viewAngle
```

```
view_angle = 1×2
   -12    28
```

Update the robot pose

The model of the cable robot can be easily updated by providing the joint space position, velocity, acceleration and external wrench

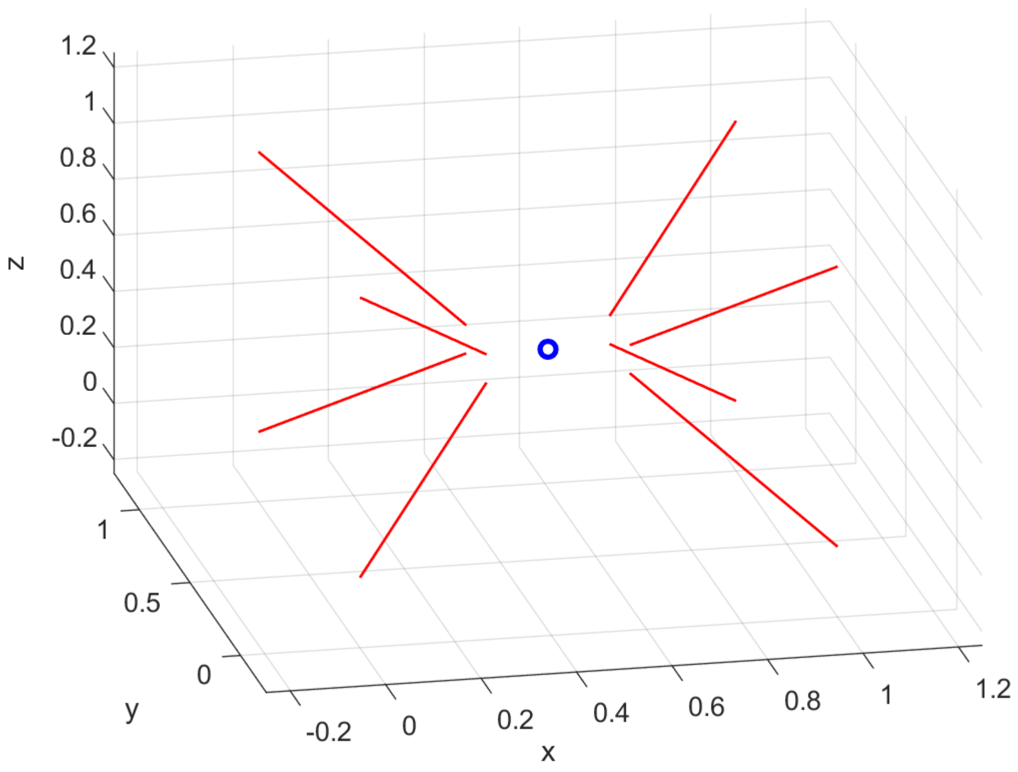
```
q = [0.3; 0.3; 1.0; 0; 0; 0];
q_dot = [0; 0; 0; 0; 0; 0];
q_ddot = [0; 0; 0; 0; 0; 0];
w_ext = [0; 0; 0; 0; 0; 0];
cdpr_model.update(q, q_dot, q_ddot, w_ext); % Update robot model
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



Loading a different cable configuration

This can be easily achieved by loading a different cable set

```
cdpr_model = model_config.getModel('basic_8_cables');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```


1.2
1
0.8
0.6
0.4
0.2
0
-0.2
z
1
0.5
0
y
-0.2
0
0.2
0.4
0.6
0.8
1
1.2
x

Tutorial 1: Loading a Robot in CASPR

This tutorial introduces the basics in order to load a robot into the CASPR environment using a script.

```
clear;
```

Loading a model configuration

First, instantiate the ModelConfig object with the name of the robot. In this example, the "Example spatial" robot will be used.

```
model_config = ModelConfig('Example spatial');
```

The list of available robots can be found within the `CASPR_GUI` or `model_config.robotNamesList`

Loading a robot model

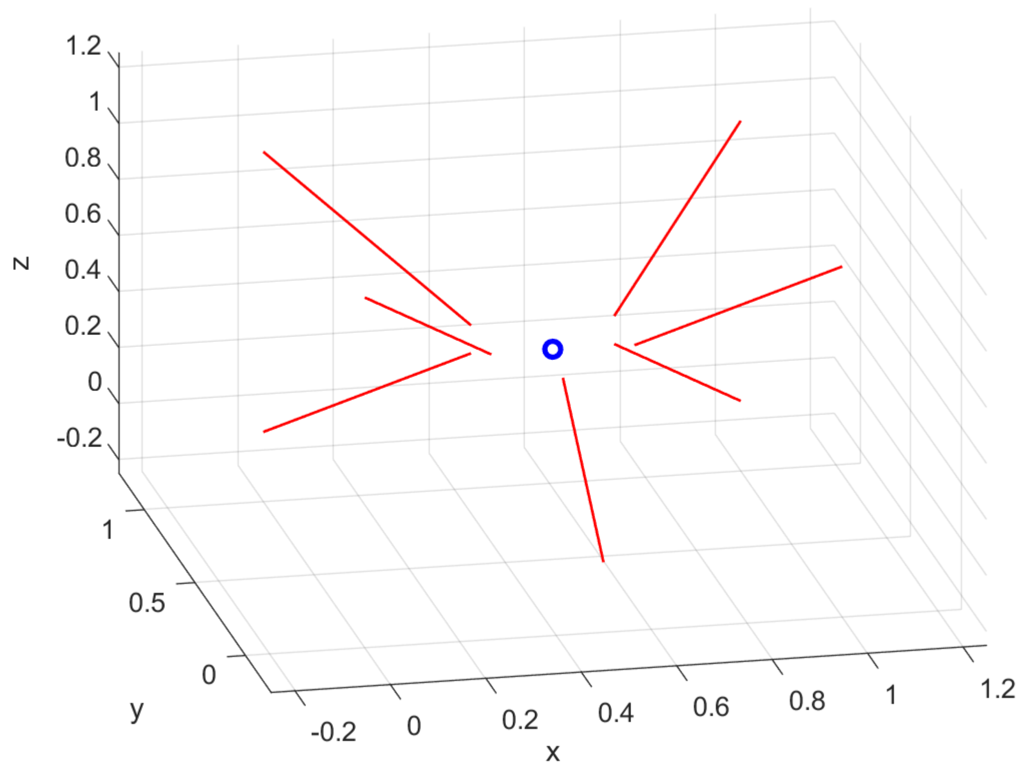
The `ModelConfig` object is not the robot (kinematics and dynamics) model itself, but must be created. At this point, the "cable set" that define the properties and arrangement of the cables must be specified based on the cable set's ID within the XML cable configuration file.

```
cdpr_model = model_config.getModel('basic_7_cables');
```

Visualise the robot

After a robot model is created, it can be easily visualised using the function `MotionSimulatorBase.PlotFrame(...)`

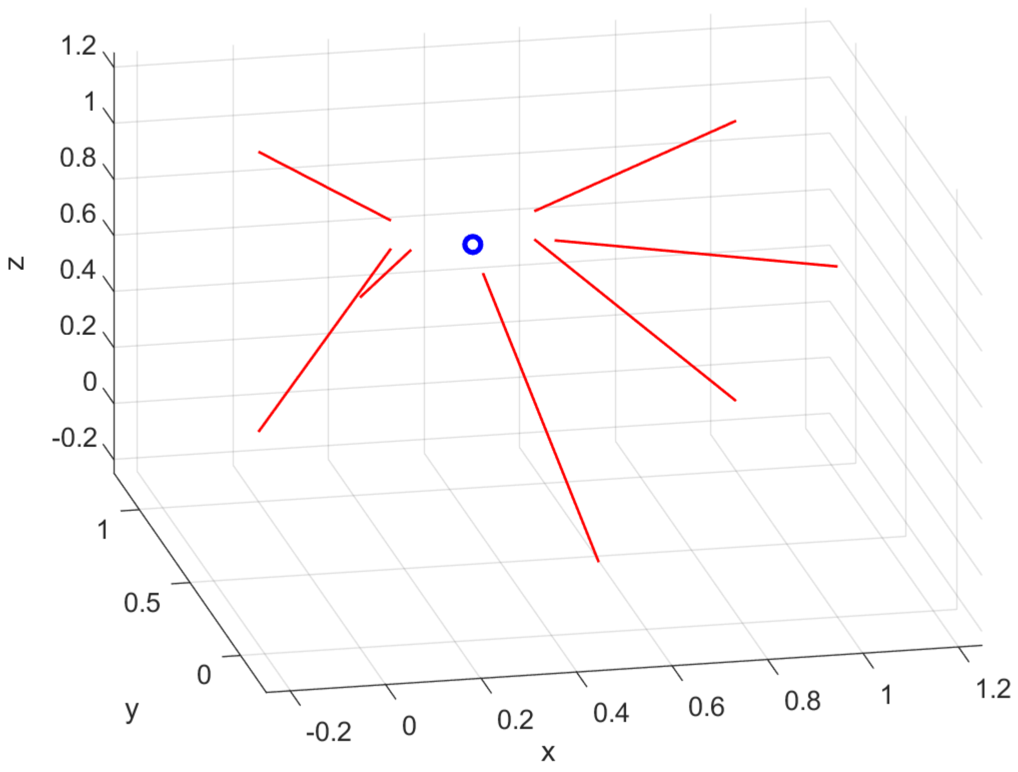
```
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



Update the robot pose

The model of the cable robot can be easily updated by providing the joint space position, velocity, acceleration and external wrench

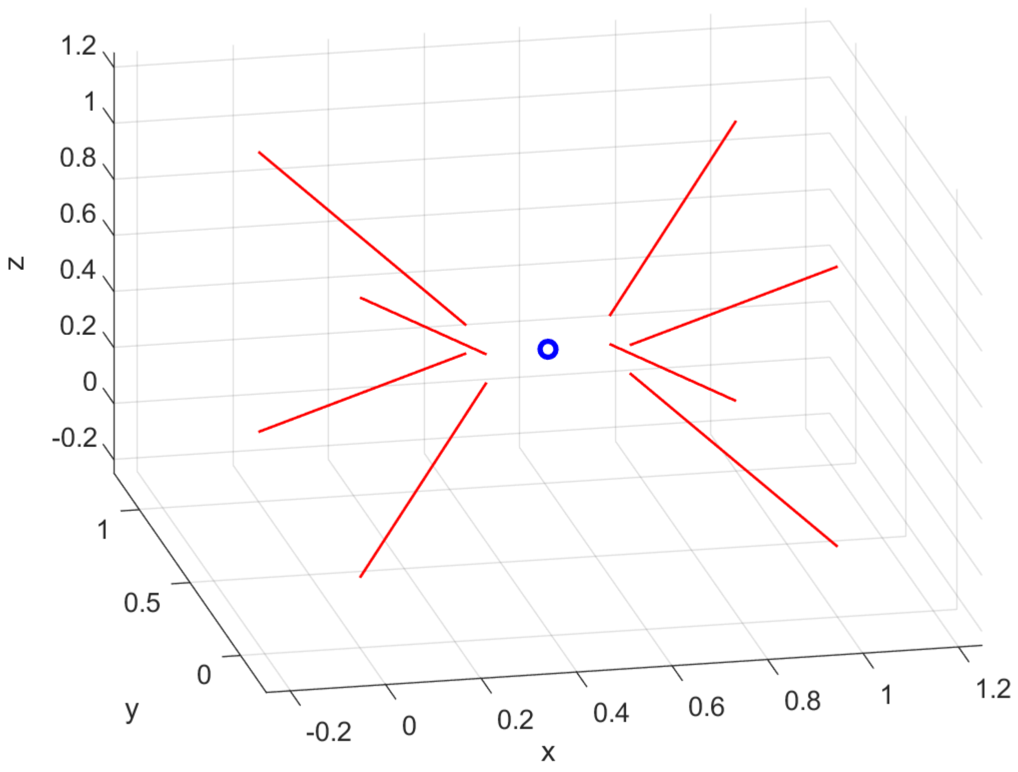
```
q = [0.3; 0.3; 1.0; 0; 0; 0];
q_dot = [0; 0; 0; 0; 0; 0];
q_ddot = [0; 0; 0; 0; 0; 0];
w_ext = [0; 0; 0; 0; 0; 0];
cdpr_model.update(q, q_dot, q_ddot, w_ext); % Update robot model
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



Loading a different cable configuration

This can be easily achieved by loading a different cable set

```
cdpr_model = model_config.getModel('basic_8_cables');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



1.2
1
0.8
0.6
0.4
0.2
0
-0.2
z
1
0.5
0
y
-0.2
0
0.2
0.4
0.6
0.8
1
1.2
x

Tutorial 2: Loading and Creating Trajectories in CASPR

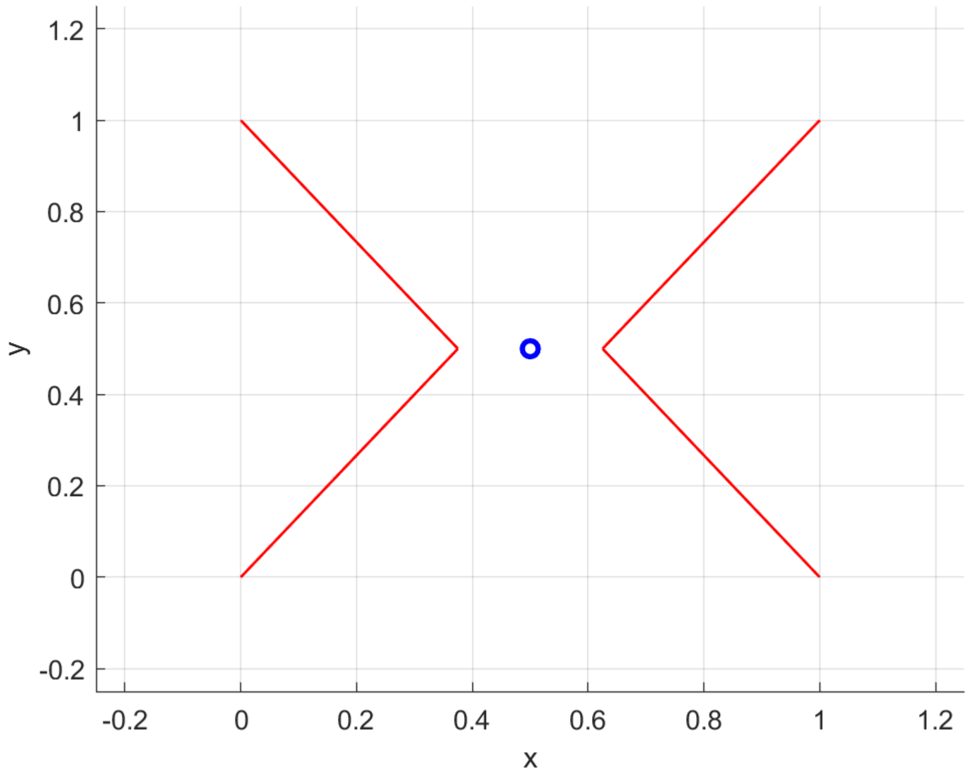
This tutorial shows how to load trajectories from XML files and also create trajectories in scripts.

```
clear;
```

Load the robot

Load a robot as shown in `T1_load_robot.mlx`

```
model_config = ModelConfig('Example planar XY');
cdpr_model = model_config.getModel('basic');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



Loading a trajectory through XML trajectory file

Trajectories of predefined types, such as linear, cubic or quintic etc., can be defined within the trajectory XML filed

```
trajectory_linear = model_config.getJointTrajectory('example_linear');
```

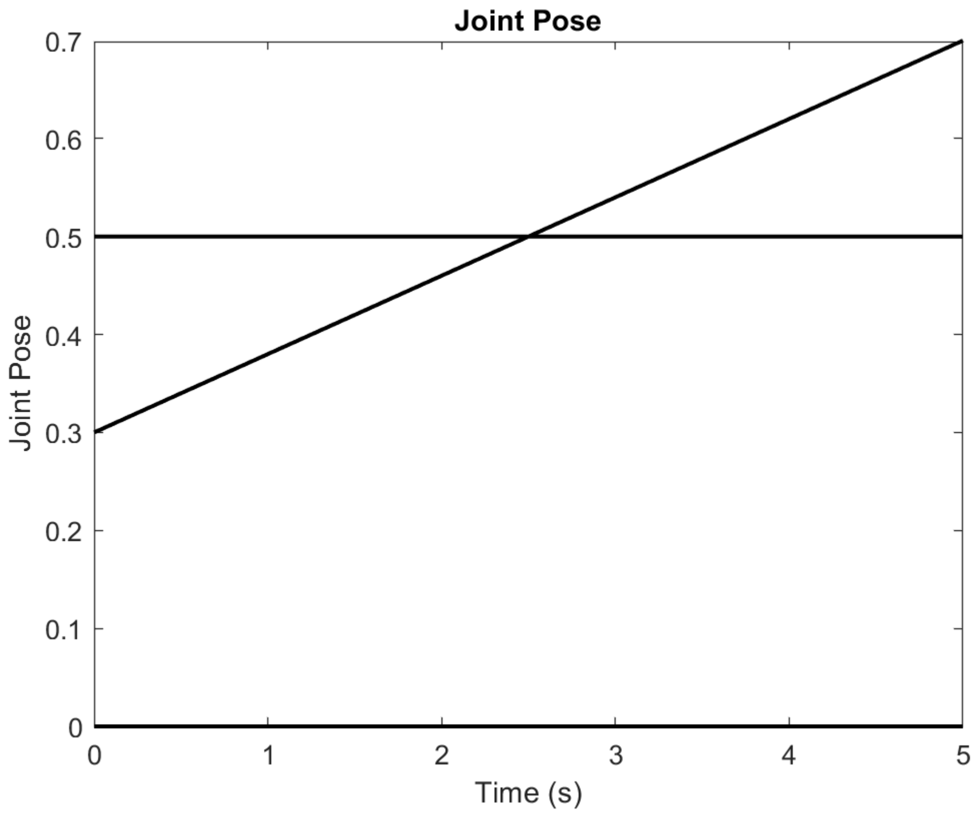
Within the XML file a linear trajectory can be defined as

1. `<linear_spline_trajectory id="example_linear" time_definition="absolute" time_step="0.01">`
2. `<points>`

```
<point>
<q>0.3 0.5 0.0</q>
</point>
<point time="5.0">
<q>0.7 0.5 0.0</q>
</point>
</points>
</linear_spline_trajectory>
```

The trajectory can be plotted through the `plotJointPose()` function

```
trajectory_linear.plotJointPose();
```



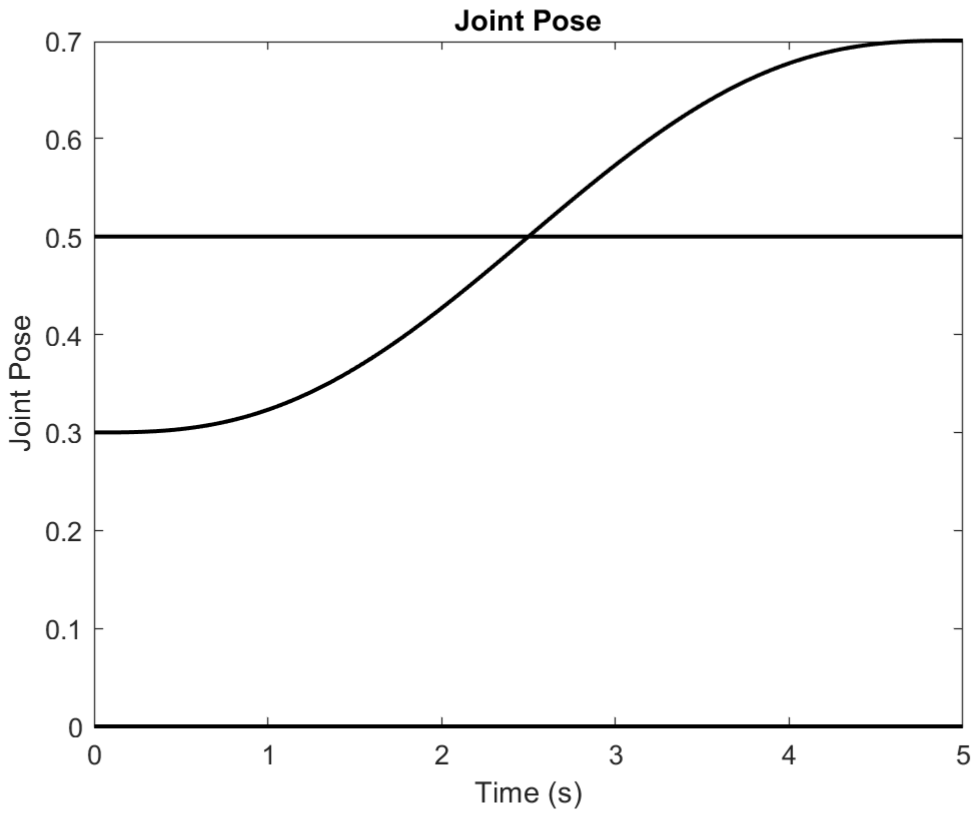
The set of trajectory names can be seen within the XML file or also within the property

```
joint_trajectory_names = model_config.jointTrajectoryNamesList
```

```
joint_trajectory_names = 1×13 cell
'example_linear'    'example_cubic'    'example_quintic'    'example_infeasible' ···
```

Another trajectory can be loaded in a similar manner

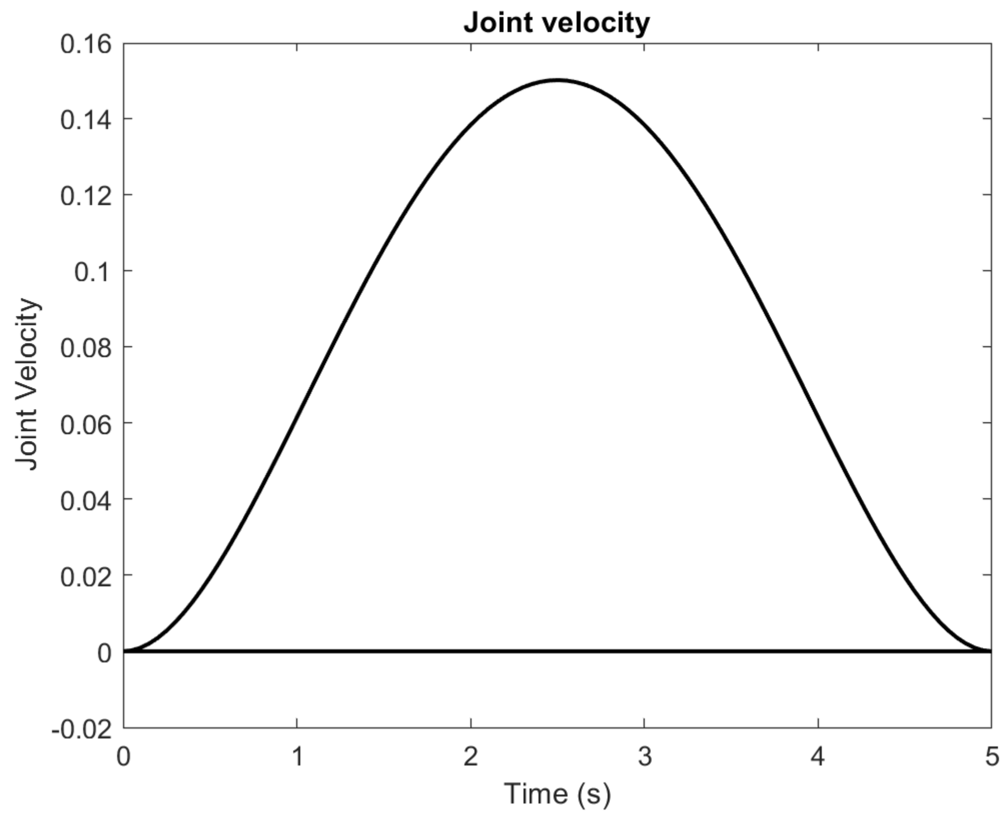
```
trajectory_quintic = model_config.getJointTrajectory('example_quintic');
trajectory_quintic.plotJointPose();
```



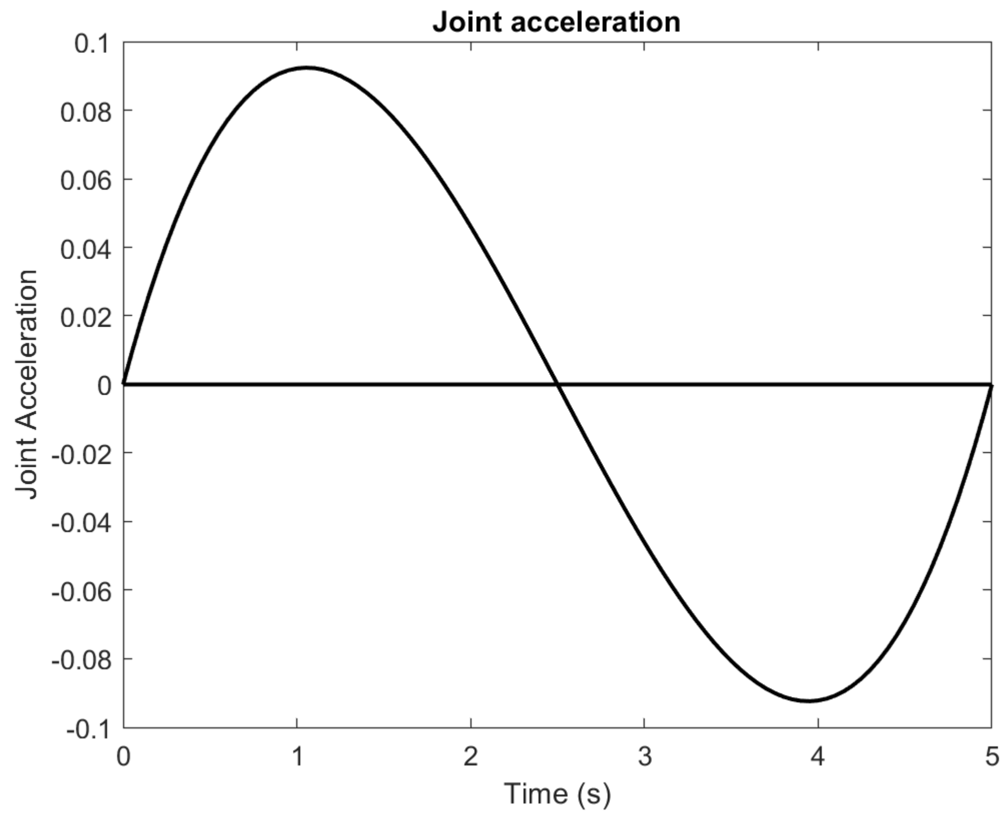
Plotting trajectories

In addition to plotting the joint space position, the velocity and acceleration can also be plotted

```
trajectory_quintic.plotJointVelocity();
```

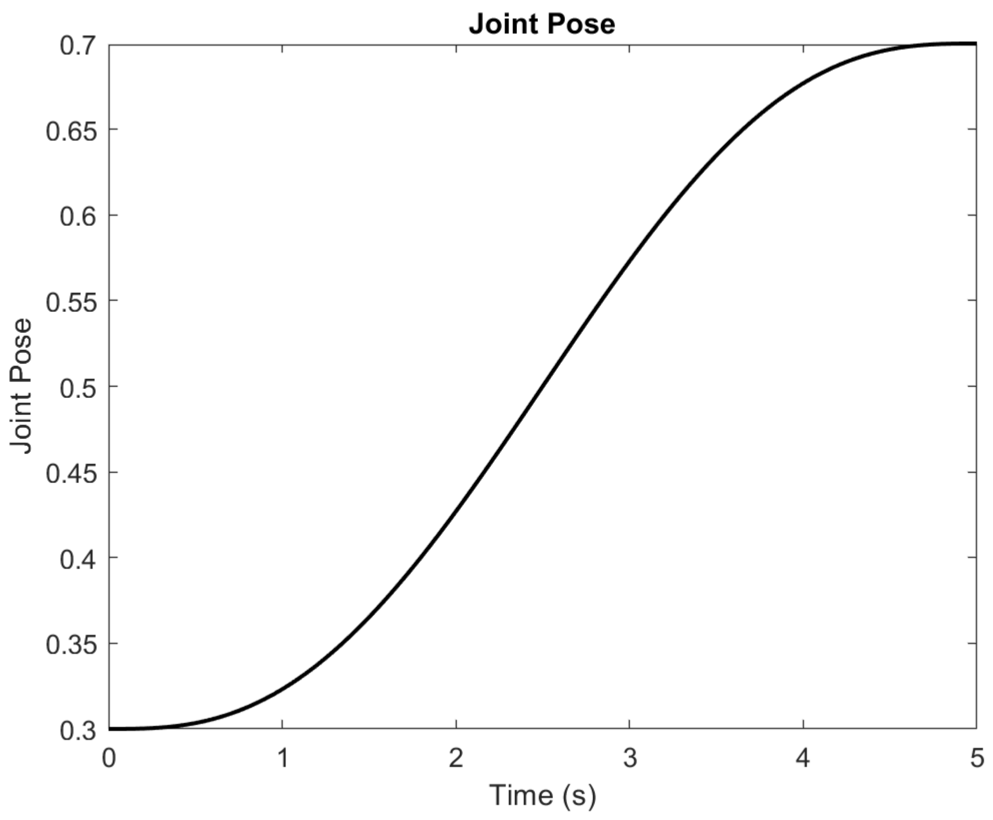


```
trajectory_quintic.plotJointAcceleration();
```



Aside from plotting all joint variables simultaneously, one or more joint variables can be independently plotted

```
vars_to_plot = [1]; % Array of indices of variables to plot
trajectory_quintic.plotJointPose(vars_to_plot);
```



Loading a trajectory through .traj files

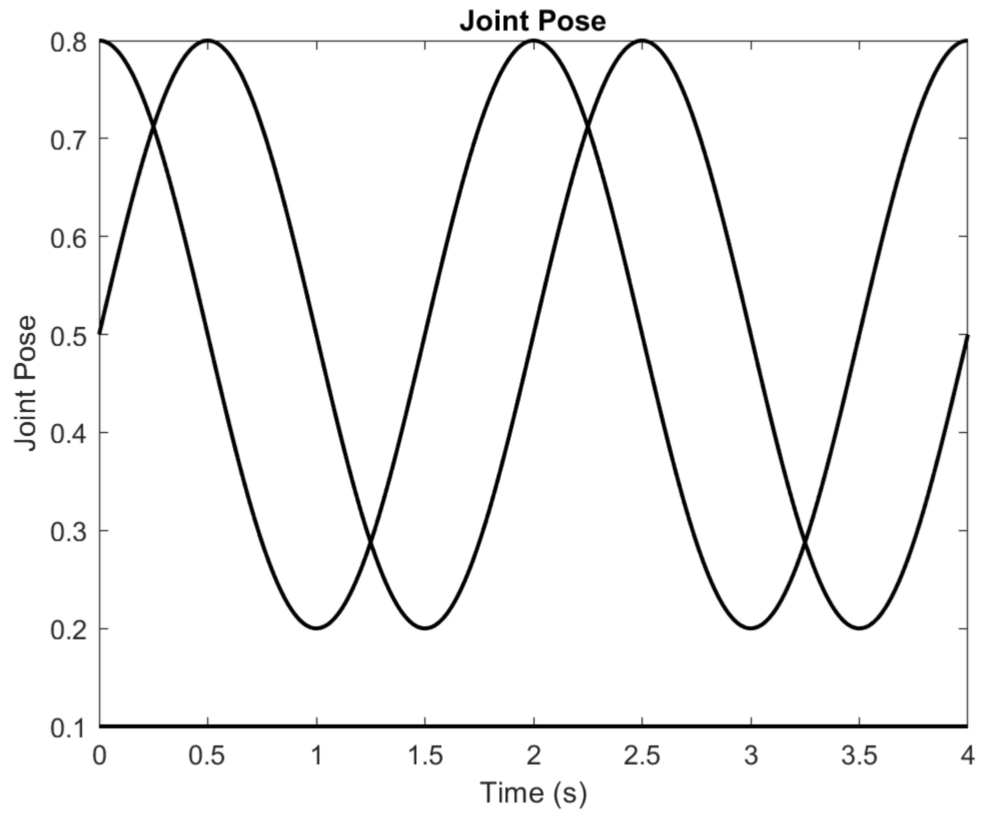
Another way to load trajectories more generically is to store trajectories within a text file in the form of

1. `time_step,0.010000`
2. `q,0.800000,0.500000,0.100000,v,-0.000000,0.942478,0.000000,a,-2.960881,-0.000000,0.000000`
3. `q,0.799852,0.509423,0.100000,v,-0.029604,0.942013,0.000000,a,-2.959420,-0.093004,0.000000`
4. `...`

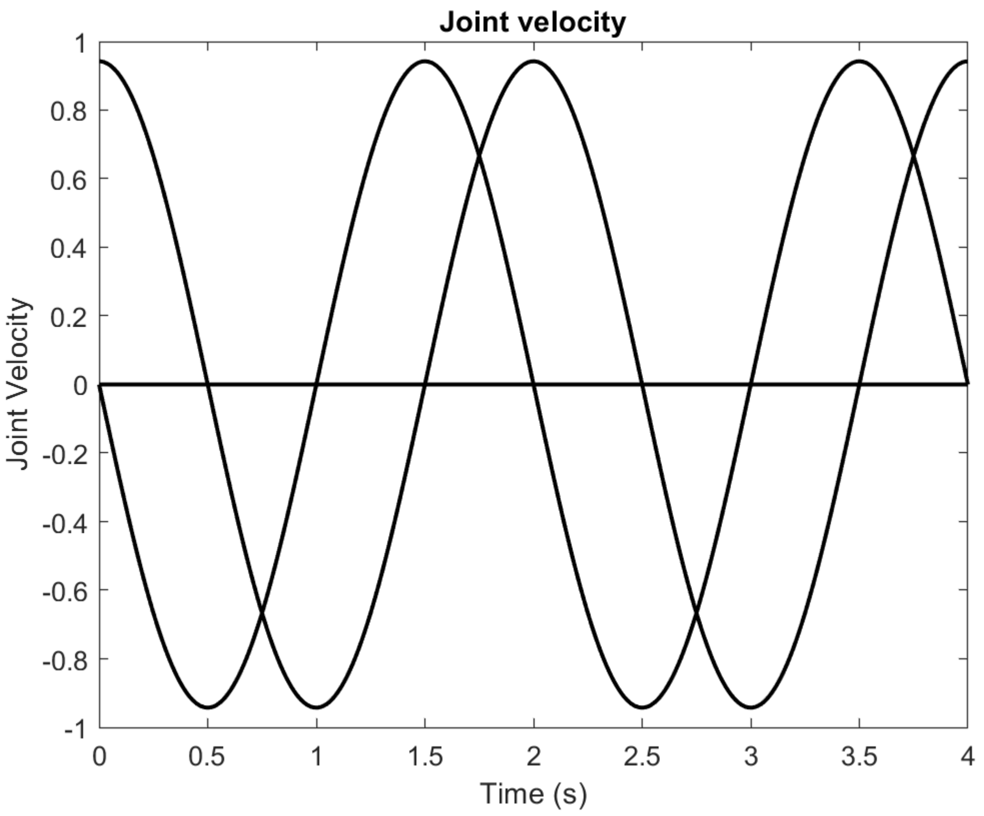
The file trajectory can then be defined within the XML as

```
<file_trajectory id="example_file"
filename="example_planar_XY_trajectory_file_example.traj" />
```

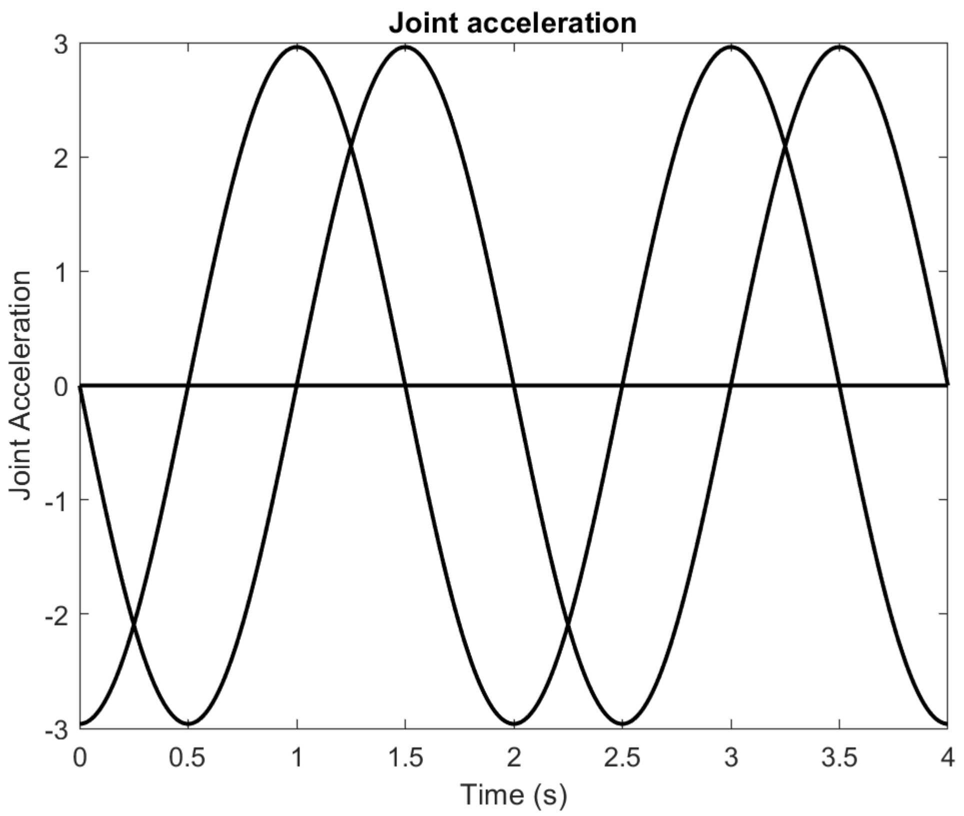
```
trajectory_file = model_config.getJointTrajectory('example_file');
trajectory_file.plotJointPose();
```



```
trajectory_file.plotJointVelocity();
```



```
trajectory_file.plotJointAcceleration();
```



Saving trajectories

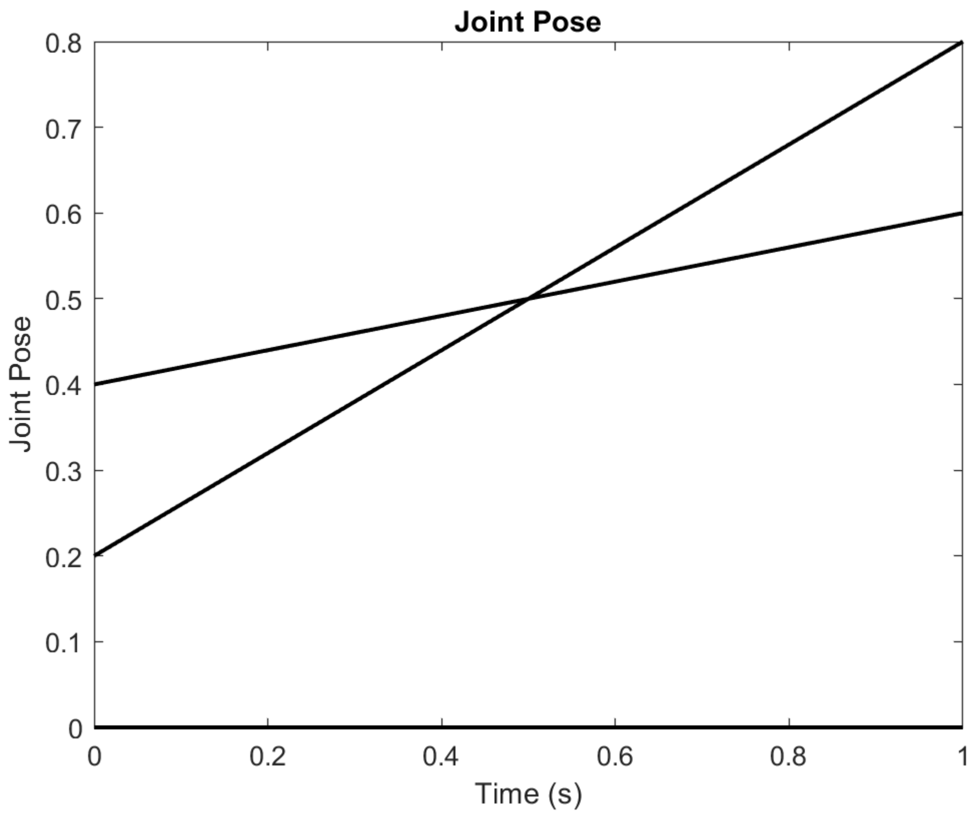
Trajectories that have been created can also be saved using `saveJointTrajectoryFile(file_path)`, for example `trajectory_quintic.saveJointTrajectoryFile('C:\test_quintic.traj')`

Creating trajectories in code

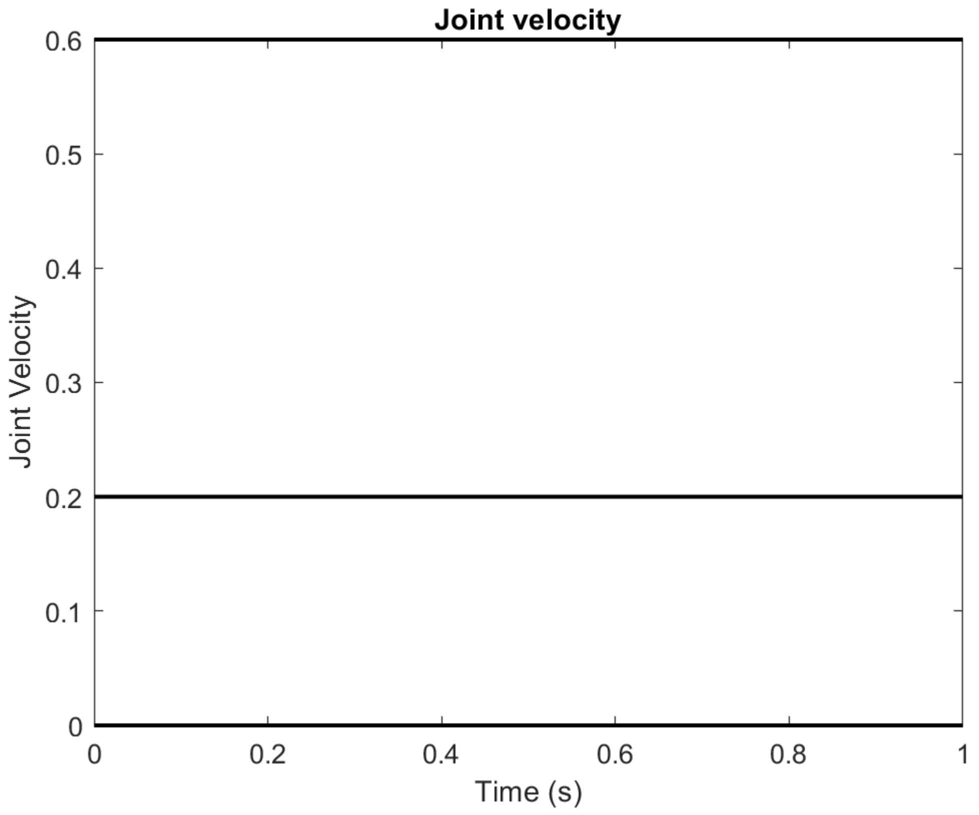
In addition to specifying trajectories in XML files with the predefined formats, or in a file, users can also dynamically create trajectories using a for-loop in code according any functional form:

```
traj_code = JointTrajectory();
traj_code.timeVector = 0:0.01:1.0;
traj_code.q = cell(1, length(traj_code.timeVector));
traj_code.q_dot = cell(1, length(traj_code.timeVector));
traj_code.q_ddot = cell(1, length(traj_code.timeVector));
for i=1:length(traj_code.timeVector)
    % Generate a linear trajectory for x = 0.2 to 0.8, y = 0.4 to 0.6 and
    % theta = 0 to 0 in 1 second
    x = 0.2 + 0.6*traj_code.timeVector(i);
    y = 0.4 + 0.2*traj_code.timeVector(i);
    th = 0;
    traj_code.q{i} = [x; y; th];
    traj_code.q_dot{i} = [0.6; 0.2; 0];
    traj_code.q_ddot{i} = [0; 0; 0];
end
```

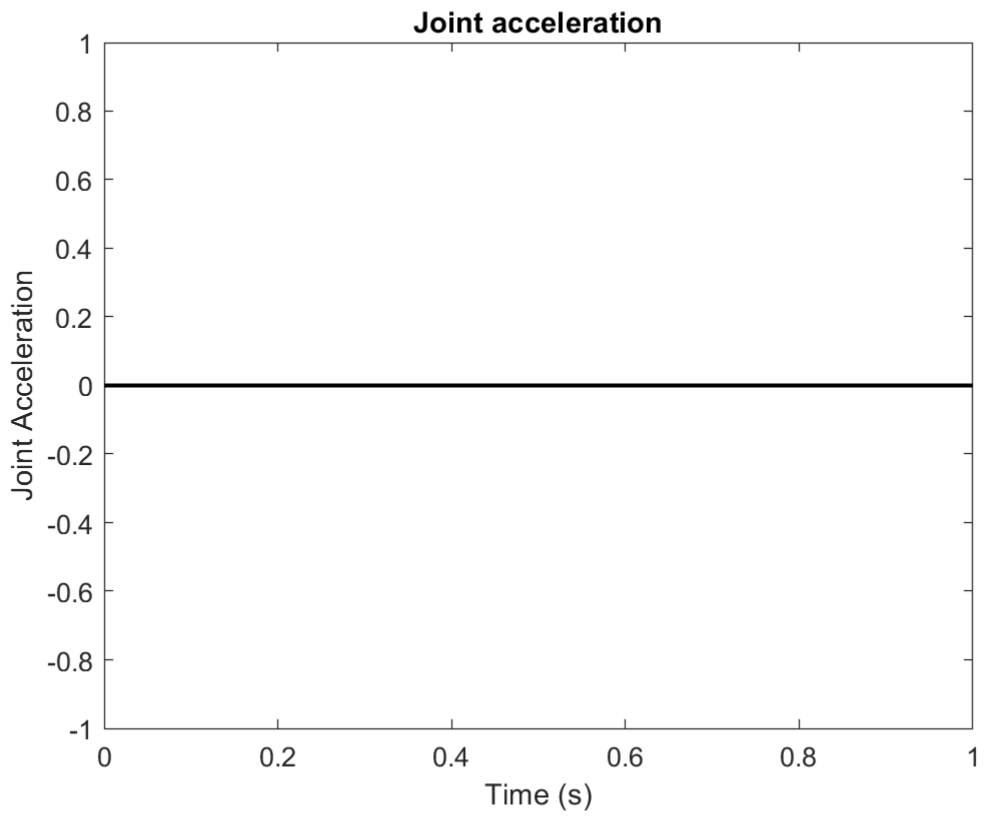
```
traj_code.plotJointPose();
```



```
traj_code.plotJointVelocity();
```



```
traj_code.plotJointAcceleration();
```



Tutorial 3: Running Inverse Kinematics in CASPR

Inverse kinematics for cable-driven robots (CDRs) refer to the determination of cable lengths for a given robot pose.

This tutorial shows how to run inverse kinematics through two approaches:

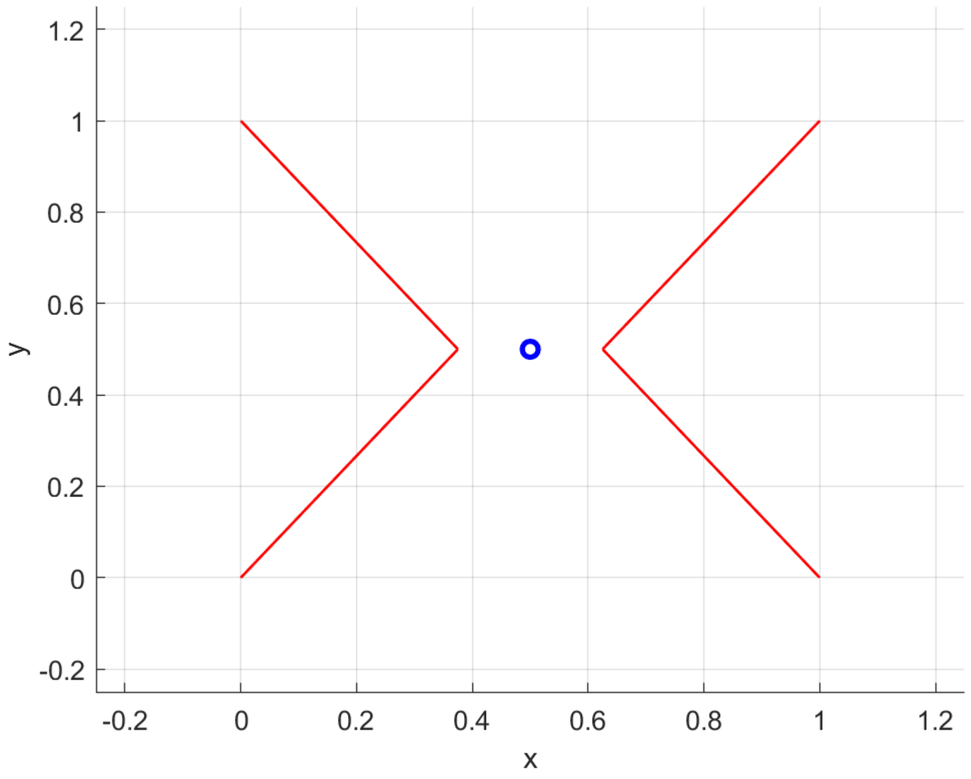
1. InverseKinematicsSimulator
2. Writing your own time loop

```
clear;
```

Load the robot and trajectory (required for both approaches)

Load a robot as shown in `T1_load_robot.mlx` and `T2_load_trajectory.mlx`

```
model_config = ModelConfig('Example planar XY');
cdpr_model = model_config.getModel('basic');
trajectory = model_config.getJointTrajectory('example_quintic');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.viewA
```



1) Solve using the InverseKinematicsSimulator

The `InverseKinematicsSimulator` provides the simplest way to run a "standard" inverse kinematics simulation for a defined trajectory.

First, create the simulator object using the robot model

```
ik_sim = InverseKinematicsSimulator(cdpr_model);
```

Next, run the simulator on a particular trajectory

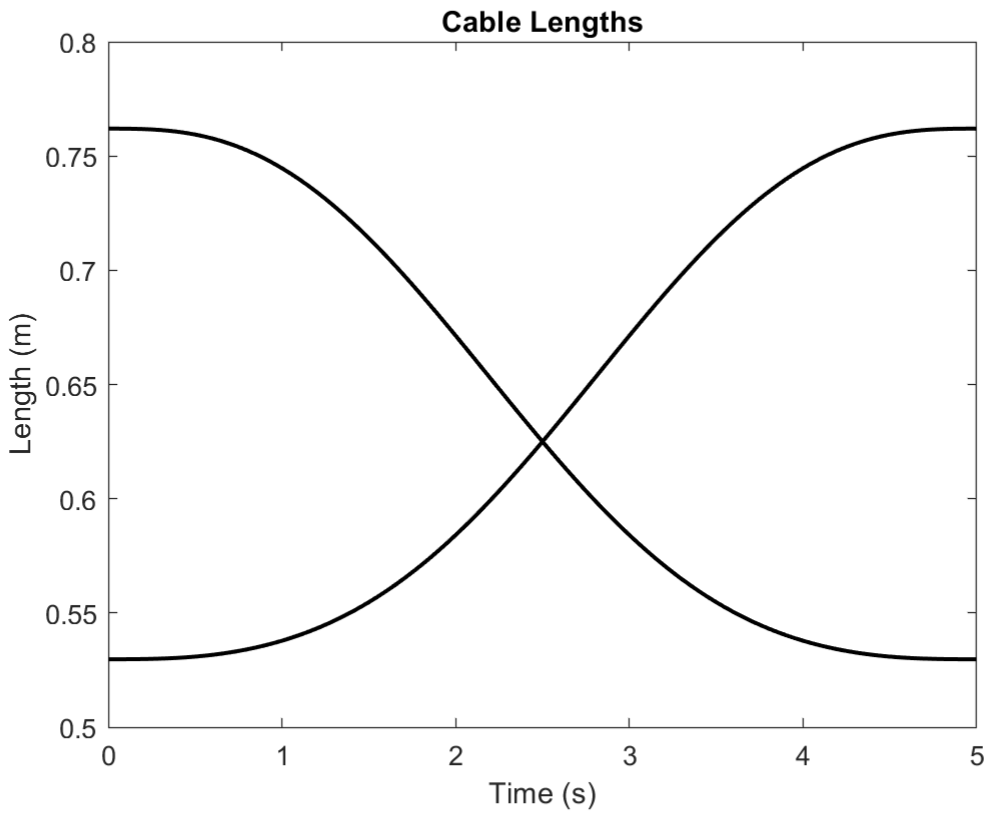
```
ik_sim.run(trajectory);
```

```
[INFO] Begin inverse kinematics simulator run...
[INFO] Time : 0.000000
[INFO] Time : 0.050000
[INFO] Time : 0.100000
[INFO] Time : 0.150000
[INFO] Time : 0.200000
[INFO] Time : 0.250000
[INFO] Time : 0.300000
[INFO] Time : 0.350000
[INFO] Time : 0.400000
[INFO] Time : 0.450000
[INFO] Time : 0.500000
[INFO] Time : 0.550000
[INFO] Time : 0.600000
[INFO] Time : 0.650000
[INFO] Time : 0.700000
[INFO] Time : 0.750000
[INFO] Time : 0.800000
[INFO] Time : 0.850000
[INFO] Time : 0.900000
[INFO] Time : 0.950000
[INFO] Time : 1.000000
[INFO] Time : 1.050000
[INFO] Time : 1.100000
[INFO] Time : 1.150000
[INFO] Time : 1.200000
[INFO] Time : 1.250000
[INFO] Time : 1.300000
[INFO] Time : 1.350000
[INFO] Time : 1.400000
[INFO] Time : 1.450000
[INFO] Time : 1.500000
[INFO] Time : 1.550000
[INFO] Time : 1.600000
[INFO] Time : 1.650000
[INFO] Time : 1.700000
[INFO] Time : 1.750000
[INFO] Time : 1.800000
[INFO] Time : 1.850000
[INFO] Time : 1.900000
[INFO] Time : 1.950000
[INFO] Time : 2.000000
[INFO] Time : 2.050000
[INFO] Time : 2.100000
[INFO] Time : 2.150000
[INFO] Time : 2.200000
[INFO] Time : 2.250000
[INFO] Time : 2.300000
[INFO] Time : 2.350000
[INFO] Time : 2.400000
[INFO] Time : 2.450000
[INFO] Time : 2.500000
[INFO] Time : 2.550000
[INFO] Time : 2.600000
[INFO] Time : 2.650000
[INFO] Time : 2.700000
[INFO] Time : 2.750000
```

```
[INFO] Time : 2.800000
[INFO] Time : 2.850000
[INFO] Time : 2.900000
[INFO] Time : 2.950000
[INFO] Time : 3.000000
[INFO] Time : 3.050000
[INFO] Time : 3.100000
[INFO] Time : 3.150000
[INFO] Time : 3.200000
[INFO] Time : 3.250000
[INFO] Time : 3.300000
[INFO] Time : 3.350000
[INFO] Time : 3.400000
[INFO] Time : 3.450000
[INFO] Time : 3.500000
[INFO] Time : 3.550000
[INFO] Time : 3.600000
[INFO] Time : 3.650000
[INFO] Time : 3.700000
[INFO] Time : 3.750000
[INFO] Time : 3.800000
[INFO] Time : 3.850000
[INFO] Time : 3.900000
[INFO] Time : 3.950000
[INFO] Time : 4.000000
[INFO] Time : 4.050000
[INFO] Time : 4.100000
[INFO] Time : 4.150000
[INFO] Time : 4.200000
[INFO] Time : 4.250000
[INFO] Time : 4.300000
[INFO] Time : 4.350000
[INFO] Time : 4.400000
[INFO] Time : 4.450000
[INFO] Time : 4.500000
[INFO] Time : 4.550000
[INFO] Time : 4.600000
[INFO] Time : 4.650000
[INFO] Time : 4.700000
[INFO] Time : 4.750000
[INFO] Time : 4.800000
[INFO] Time : 4.850000
[INFO] Time : 4.900000
[INFO] Time : 4.950000
[INFO] Time : 5.000000
```

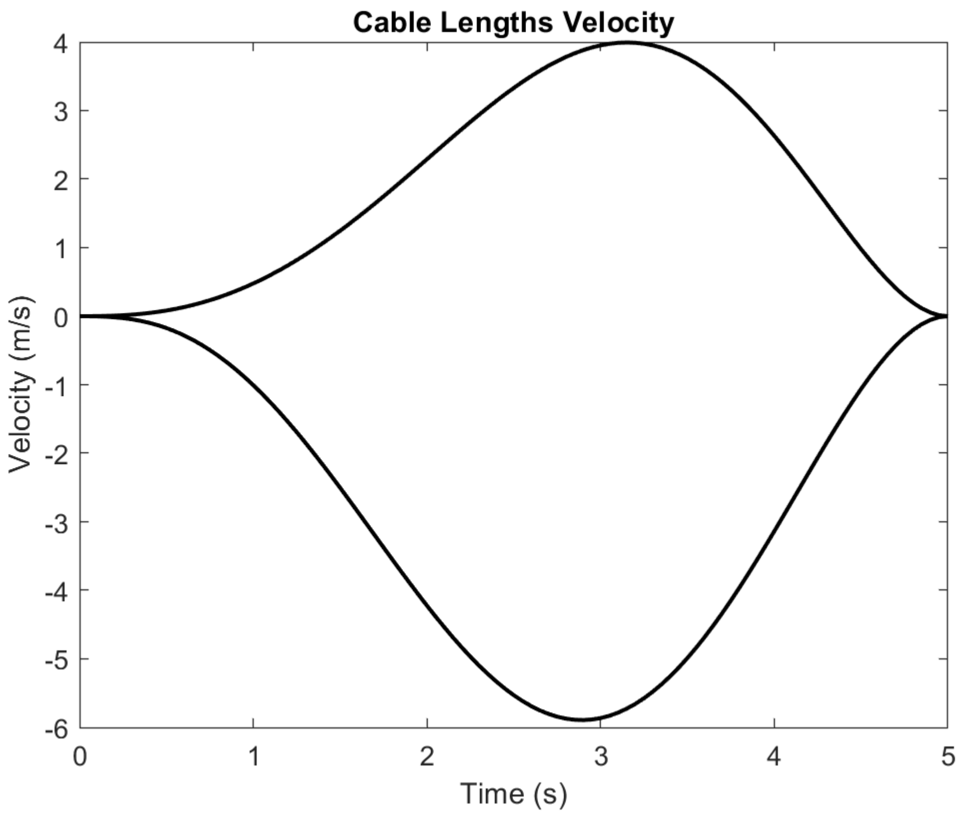
Finally, the simulator can plot the results, such as the cable lengths:

```
ik_sim.plotCableLengths();
```



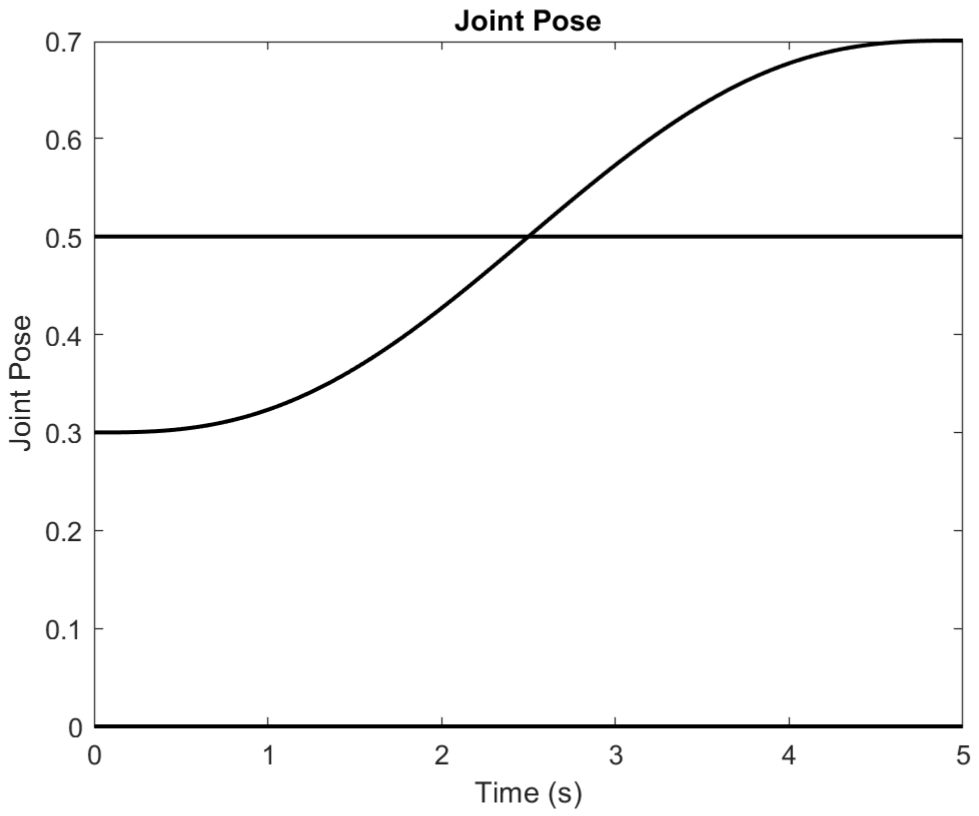
Or the velocity of the cable lengths:

```
ik_sim.plotCableLengthsVelocity();
```

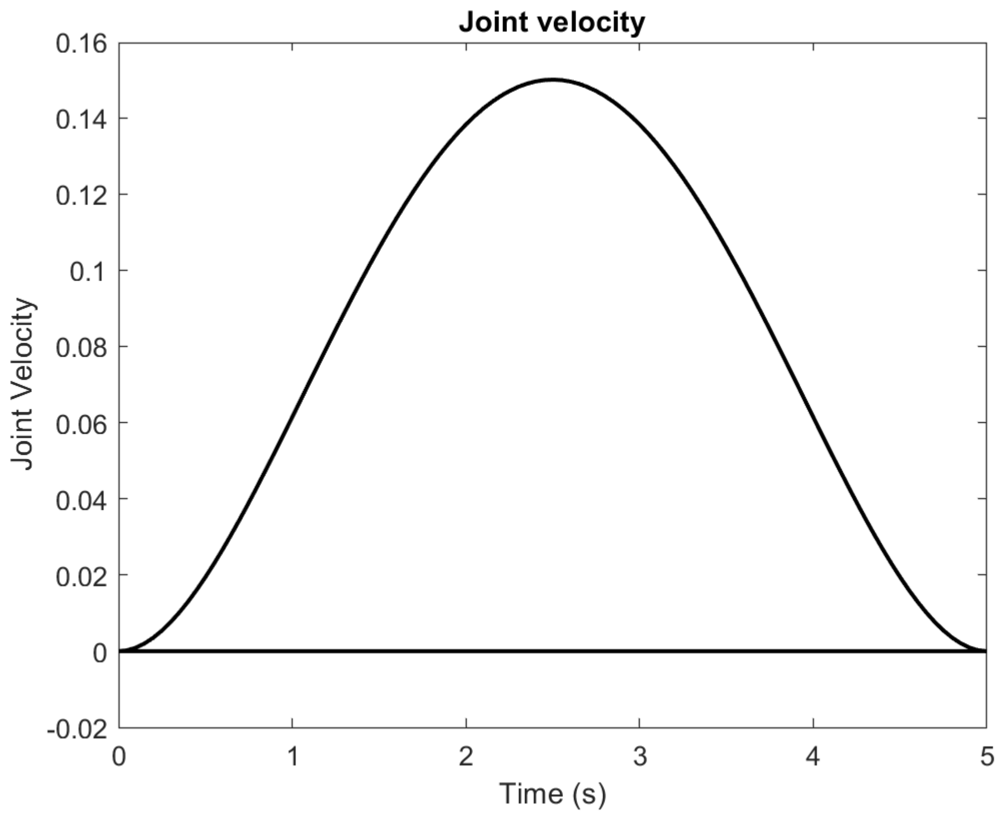


It can also plot the joint space information (like trajectories):

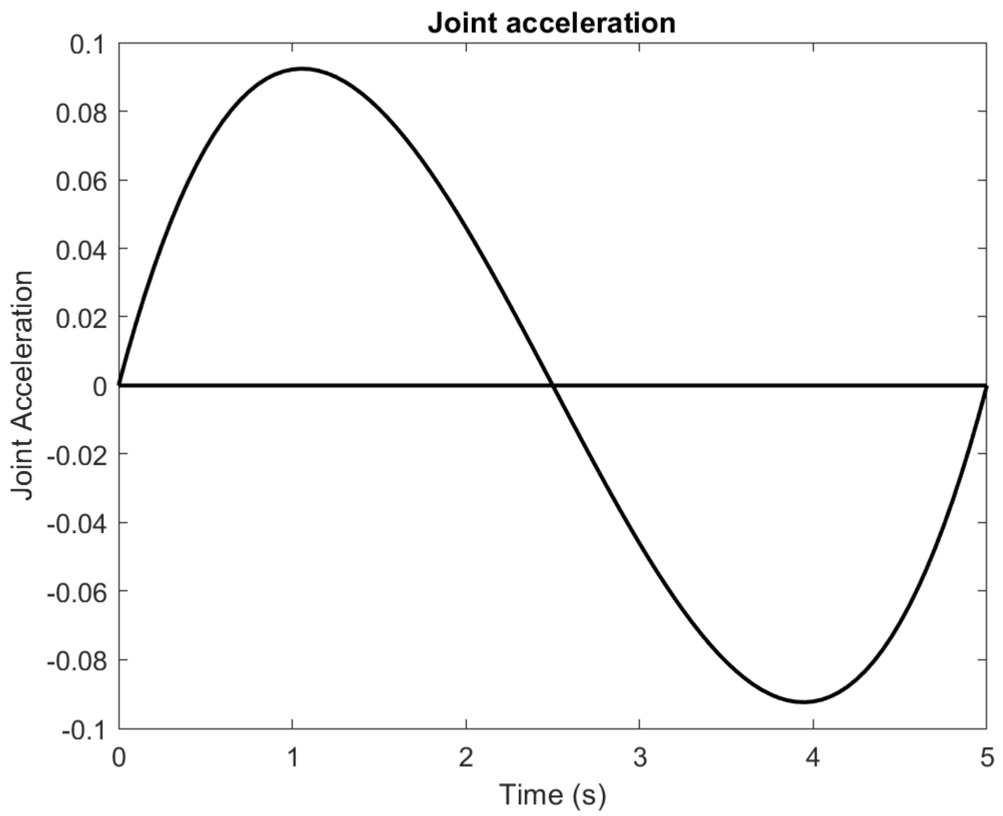
```
ik_sim.plotJointPose();
```



```
ik_sim.plotJointVelocity();
```

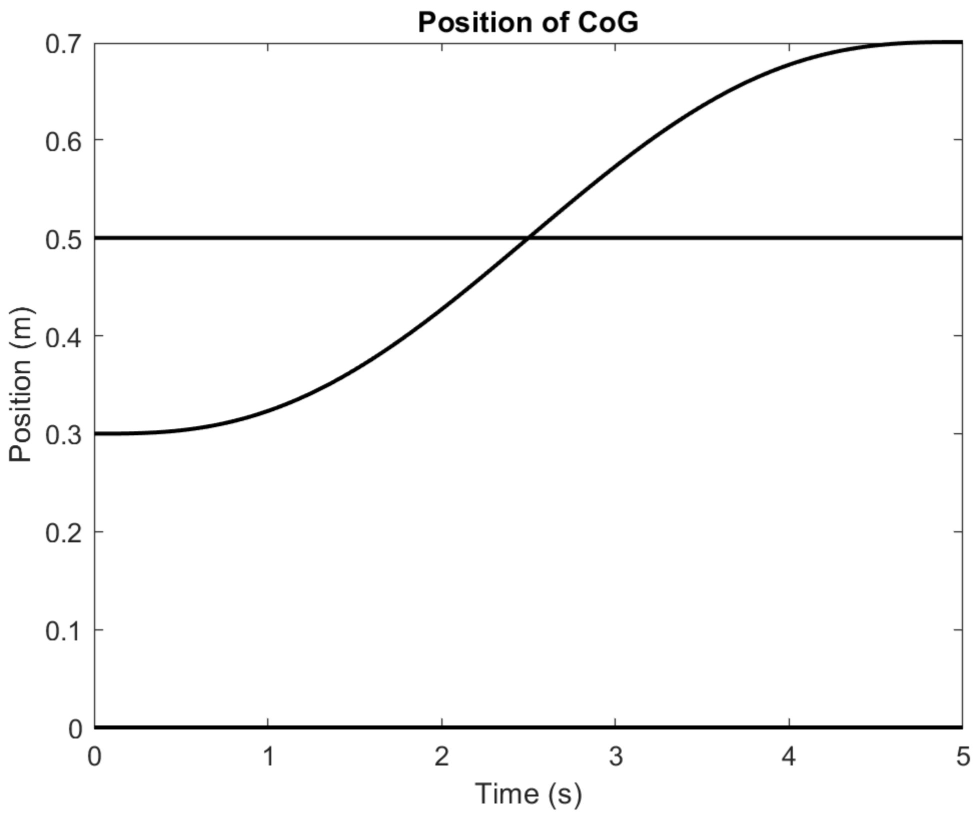


```
ik_sim.plotJointAcceleration();
```

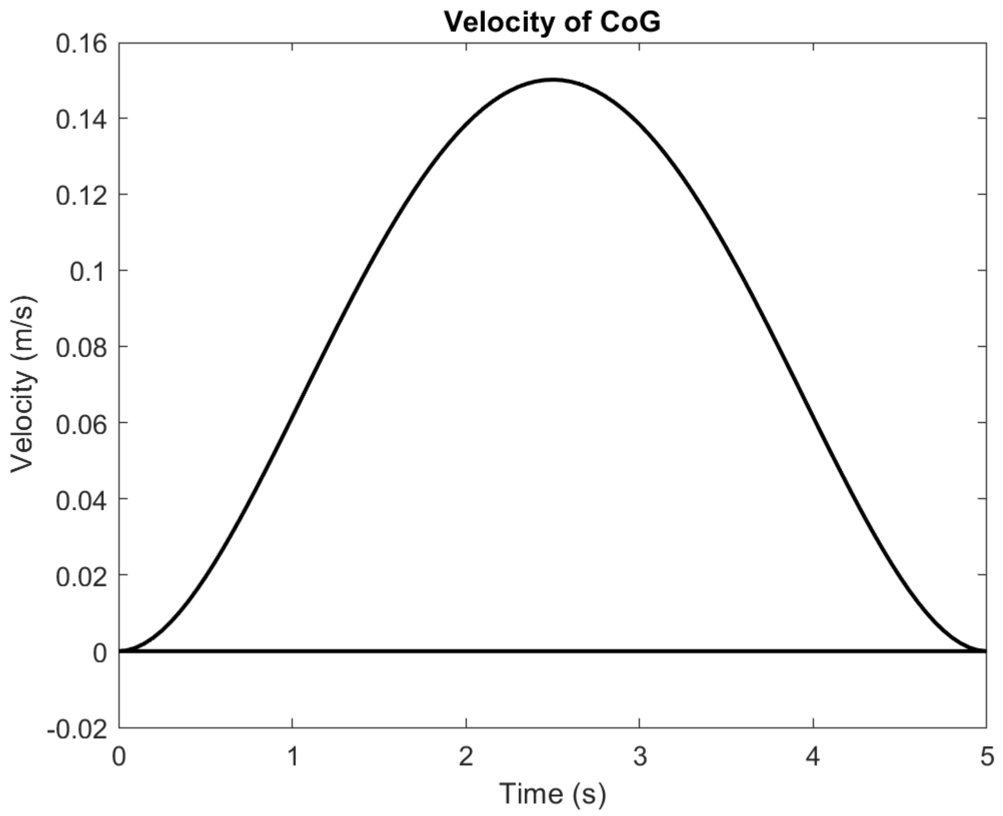


And even the position, velocity and acceleration of the centre of gravity (CoG):

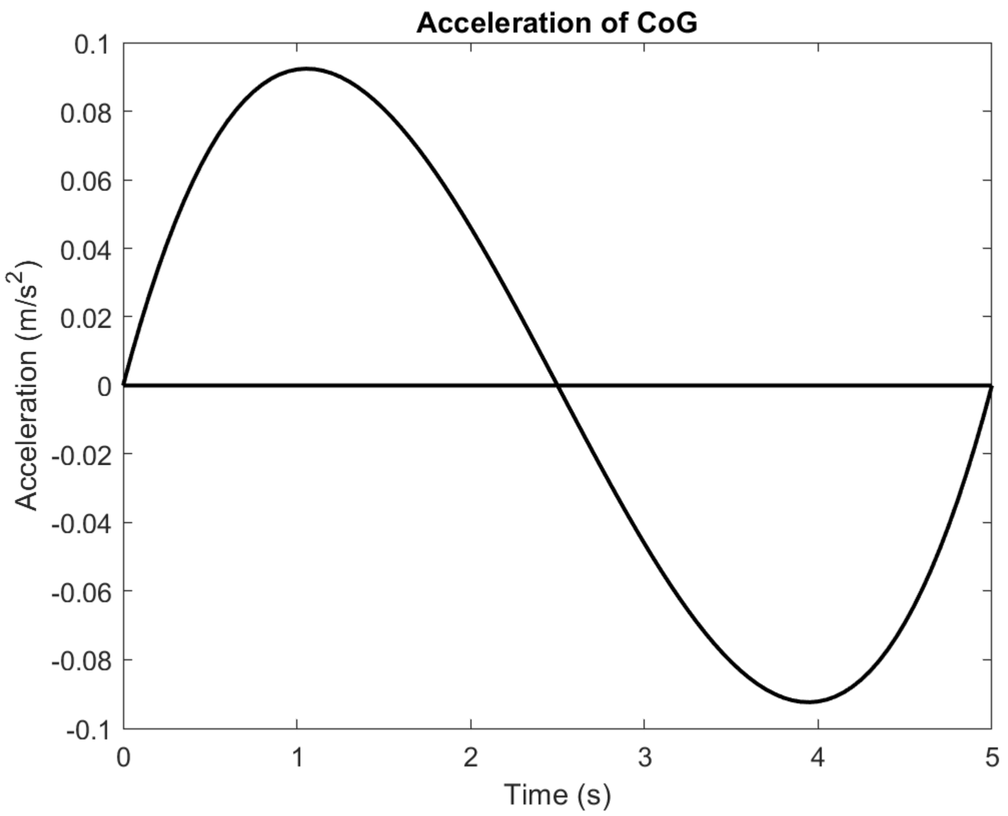
```
ik_sim.plotBodyCoG();
```



```
ik_sim.plotBodyCoGVelocity();
```



```
ik_sim.plotBodyCoGAcceleration();
```



2) Writing your own time loop

If you want more control over what is happening or customise the inner-workings during the simulation, it is recommended for you to write your own time loop to perform the inverse kinematics. The following simple example is basically what happens within the `InverseKinematicsSimulator`.

First, create the variables (pre-allocate memory) required to store the solution from the inverse kinematics (cable lengths and cable lengths velocity):

```
ik_lengths = zeros(cdpr_model.numCables, length(trajectory.timeVector)); % Each column
ik_lengths_vel = zeros(cdpr_model.numCables, length(trajectory.timeVector));
```

Next, loop through the trajectory as a function of time:

```
for t = 1:length(trajectory.timeVector)
```

The main step within the time loop is to update the model using the joint trajectory's pose, velocity and acceleration at this time instance (and assuming the external wrench is zero)

```
    cdpr_model.update(trajectory.q{t}, trajectory.q_dot{t}, trajectory.q_ddot{t}, zeros
```

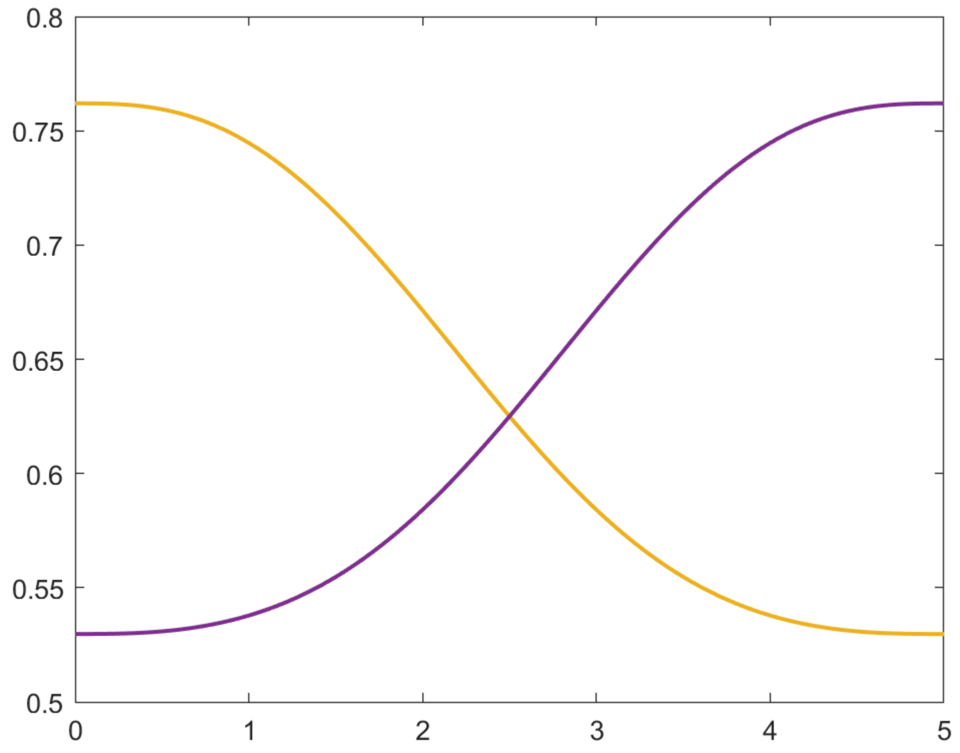
Note that in addition to using `JointTrajectory` objects, any value of `q, q_dot, q_ddot, w_ext` (joint pose, joint velocity, joint acceleration and external wrench, respectively) can be defined as input to `cdpr_model.update(q, q_dot, q_ddot, w_ext)`.

Next, the model information, including the cable lengths and cable lengths velocity, are automatically updated and hence can be stored.

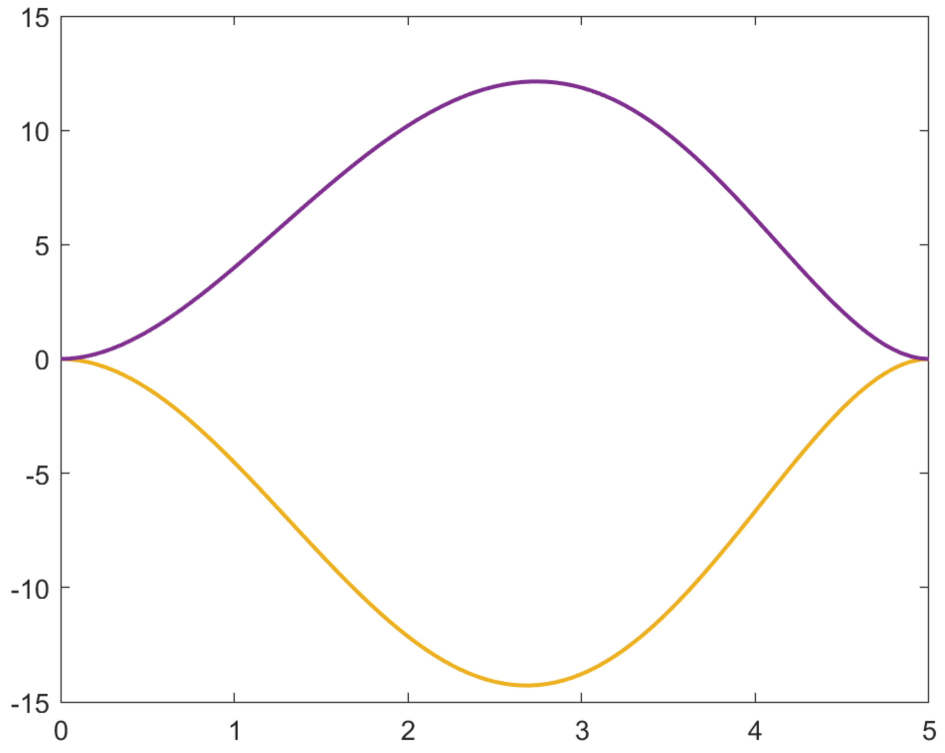
```
    ik_lengths(:,t) = cdpr_model.cableLengths;
    ik_lengths_vel(:,t) = cdpr_model.cableLengthsDot;
end
```

The results can be plotted the way you desire yourself.

```
figure;
plot(trajectory.timeVector, ik_lengths, 'LineWidth', 1.5);
```



```
figure;
plot(trajectory.timeVector, ik_lengths_vel, 'LineWidth', 1.5);
```


15
10
5
0
-5
-10
-15
0
1
2
3
4
5

Tutorial 4: Running Forward Kinematics in CASPR

Forward kinematics for cable-driven robots (CDRs) refer to the determination of the robot pose for a given set of cable lengths.

This tutorial shows how to run inverse kinematics through two approaches:

1. ForwardKinematicsSimulator
2. Writing your own time loop

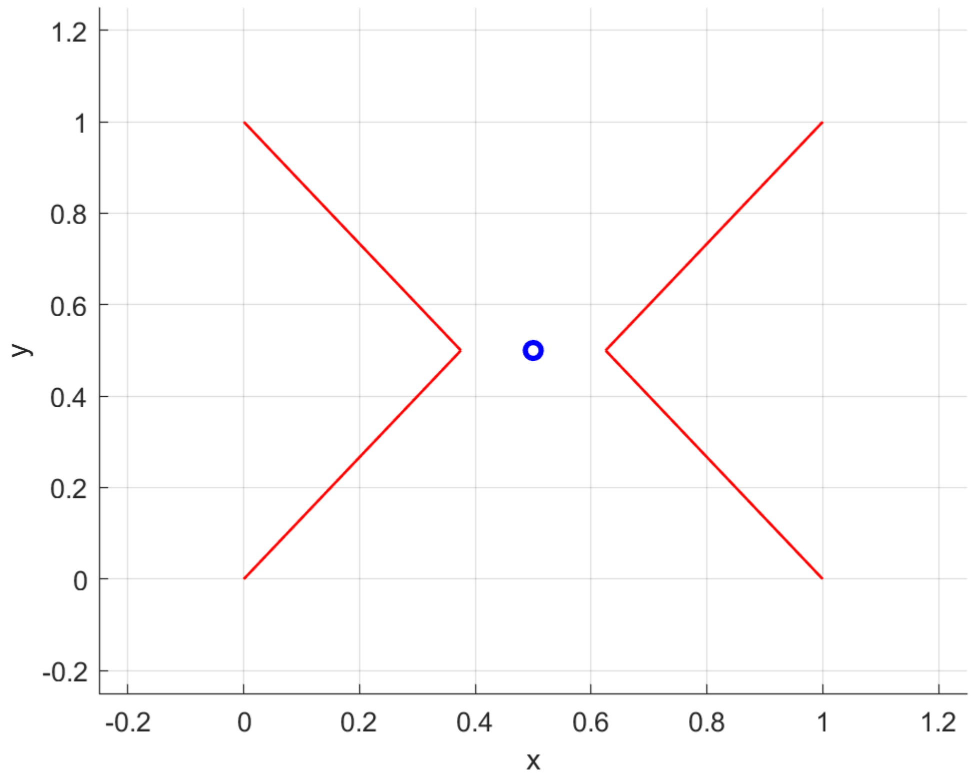
```
clear;
```

Setup (required for both approaches)

Load the robot and trajectory

Load a robot as shown in `T1_load_robot.mlx` and `T2_load_trajectory.mlx`

```
model_config = ModelConfig('Example planar XY');
cdpr_model = model_config.getModel('basic');
trajectory = model_config.getJointTrajectory('example_quintic');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



Run inverse kinematics for input to the forward kinematics

In practice, the cable lengths may be obtained from the robot hardware. However, this simulation example, the inverse kinematics cable lengths solution will be used as input to the forward kinematics.

The steps are the same as that from `tutorial_IK.mlx`

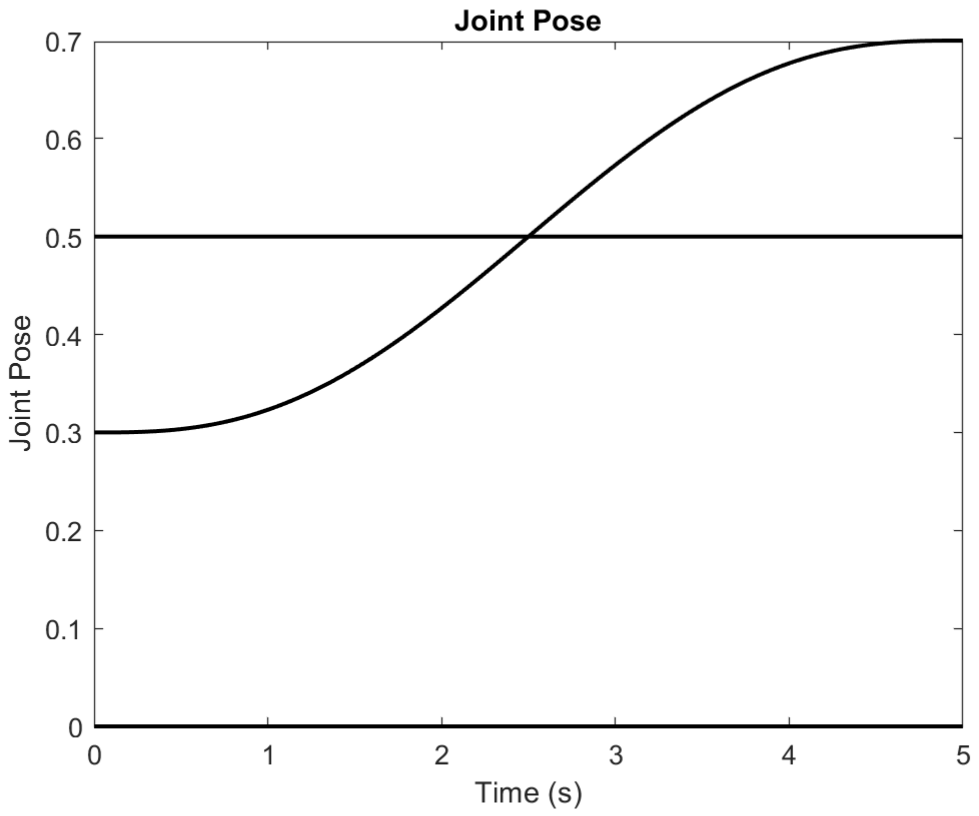
```
ik_sim = InverseKinematicsSimulator(cdpr_model);
ik_sim.run(trajectory);
```

```
[INFO] Begin inverse kinematics simulator run...
[INFO] Time : 0.000000
[INFO] Time : 0.050000
[INFO] Time : 0.100000
[INFO] Time : 0.150000
[INFO] Time : 0.200000
[INFO] Time : 0.250000
[INFO] Time : 0.300000
[INFO] Time : 0.350000
[INFO] Time : 0.400000
[INFO] Time : 0.450000
[INFO] Time : 0.500000
[INFO] Time : 0.550000
[INFO] Time : 0.600000
[INFO] Time : 0.650000
[INFO] Time : 0.700000
[INFO] Time : 0.750000
[INFO] Time : 0.800000
[INFO] Time : 0.850000
[INFO] Time : 0.900000
[INFO] Time : 0.950000
[INFO] Time : 1.000000
[INFO] Time : 1.050000
[INFO] Time : 1.100000
[INFO] Time : 1.150000
[INFO] Time : 1.200000
[INFO] Time : 1.250000
[INFO] Time : 1.300000
[INFO] Time : 1.350000
[INFO] Time : 1.400000
[INFO] Time : 1.450000
[INFO] Time : 1.500000
[INFO] Time : 1.550000
[INFO] Time : 1.600000
[INFO] Time : 1.650000
[INFO] Time : 1.700000
[INFO] Time : 1.750000
[INFO] Time : 1.800000
[INFO] Time : 1.850000
[INFO] Time : 1.900000
[INFO] Time : 1.950000
[INFO] Time : 2.000000
[INFO] Time : 2.050000
[INFO] Time : 2.100000
[INFO] Time : 2.150000
[INFO] Time : 2.200000
[INFO] Time : 2.250000
[INFO] Time : 2.300000
[INFO] Time : 2.350000
[INFO] Time : 2.400000
[INFO] Time : 2.450000
[INFO] Time : 2.500000
[INFO] Time : 2.550000
[INFO] Time : 2.600000
[INFO] Time : 2.650000
[INFO] Time : 2.700000
[INFO] Time : 2.750000
[INFO] Time : 2.800000
```

```
[INFO] Time : 2.850000
[INFO] Time : 2.900000
[INFO] Time : 2.950000
[INFO] Time : 3.000000
[INFO] Time : 3.050000
[INFO] Time : 3.100000
[INFO] Time : 3.150000
[INFO] Time : 3.200000
[INFO] Time : 3.250000
[INFO] Time : 3.300000
[INFO] Time : 3.350000
[INFO] Time : 3.400000
[INFO] Time : 3.450000
[INFO] Time : 3.500000
[INFO] Time : 3.550000
[INFO] Time : 3.600000
[INFO] Time : 3.650000
[INFO] Time : 3.700000
[INFO] Time : 3.750000
[INFO] Time : 3.800000
[INFO] Time : 3.850000
[INFO] Time : 3.900000
[INFO] Time : 3.950000
[INFO] Time : 4.000000
[INFO] Time : 4.050000
[INFO] Time : 4.100000
[INFO] Time : 4.150000
[INFO] Time : 4.200000
[INFO] Time : 4.250000
[INFO] Time : 4.300000
[INFO] Time : 4.350000
[INFO] Time : 4.400000
[INFO] Time : 4.450000
[INFO] Time : 4.500000
[INFO] Time : 4.550000
[INFO] Time : 4.600000
[INFO] Time : 4.650000
[INFO] Time : 4.700000
[INFO] Time : 4.750000
[INFO] Time : 4.800000
[INFO] Time : 4.850000
[INFO] Time : 4.900000
[INFO] Time : 4.950000
[INFO] Time : 5.000000
```

Original joint pose (to be solved using FK):

```
ik_sim.plotJointPose();
```



Setup forward kinematics solver

Since there are many potential ways to solve for the forward kinematics, CASPR offer different solvers that can be used for the simulator. For example, using the least squares method

```
fk_solver = FKLeastSquares(cdpr_model, FK_LS_ApproxOptionType.FIRST_ORDER_INTEGRATE_PS
```

Setup some initial values for use later:

```
init_q = ik_sim.trajectory.q{1}; % Initial q for the solver
init_q_dot = ik_sim.trajectory.q_dot{1}; % Initial q_dot for the solver
```

1) Solve using the ForwardKinematicsSimulator

The `ForwardKinematicsSimulator` provides the simplest way to run a "standard" forward kinematics simulation for a defined trajectory.

First, create the simulator object using the robot model and the specified solver:

```
fk_sim = ForwardKinematicsSimulator(cdpr_model, fk_solver);
```

Next, run the simulator on the resulting cable lengths and cable lengths velocity over the specified time points with the initial joint pose and velocity:

```
fk_sim.run(ik_sim.cableLengths, ik_sim.cableLengthsDot, ik_sim.timeVector, init_q, ini
[INFO] Begin forward kinematics simulator run...
[INFO] Time : 0.000000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 1
[INFO] Time : 0.050000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[INFO] Time : 0.100000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[INFO] Time : 0.150000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[INFO] Time : 0.200000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[INFO] Time : 0.250000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[INFO] Time : 0.300000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000001. Number of function calls: 4
[INFO] Time : 0.350000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000002. Number of function calls: 4
[INFO] Time : 0.400000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000004. Number of function calls: 4
[INFO] Time : 0.450000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000008. Number of function calls: 4
[INFO] Time : 0.500000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000015. Number of function calls: 5
[INFO] Time : 0.550000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000026. Number of function calls: 5
[INFO] Time : 0.600000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000042. Number of function calls: 5
[INFO] Time : 0.650000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000065. Number of function calls: 5
[INFO] Time : 0.700000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000098. Number of function calls: 5
[INFO] Time : 0.750000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000143. Number of function calls: 5
[INFO] Time : 0.800000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000203. Number of function calls: 5
[INFO] Time : 0.850000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000282. Number of function calls: 6
[INFO] Time : 0.900000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000384. Number of function calls: 6
[INFO] Time : 0.950000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000513. Number of function calls: 6
[INFO] Time : 1.000000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000673. Number of function calls: 6
[INFO] Time : 1.050000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000870. Number of function calls: 6
[INFO] Time : 1.100000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.001110. Number of function calls: 6
[INFO] Time : 1.150000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.001398. Number of function calls: 6
[INFO] Time : 1.200000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.001741. Number of function calls: 6
[INFO] Time : 1.250000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.002145. Number of function calls: 6
[INFO] Time : 1.300000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.002617. Number of function calls: 6
[INFO] Time : 1.350000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.003163. Number of function calls: 6
[INFO] Time : 1.400000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.003791. Number of function calls: 6
[INFO] Time : 1.450000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.004508. Number of function calls: 6
[INFO] Time : 1.500000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.005321. Number of function calls: 6
[INFO] Time : 1.550000
```

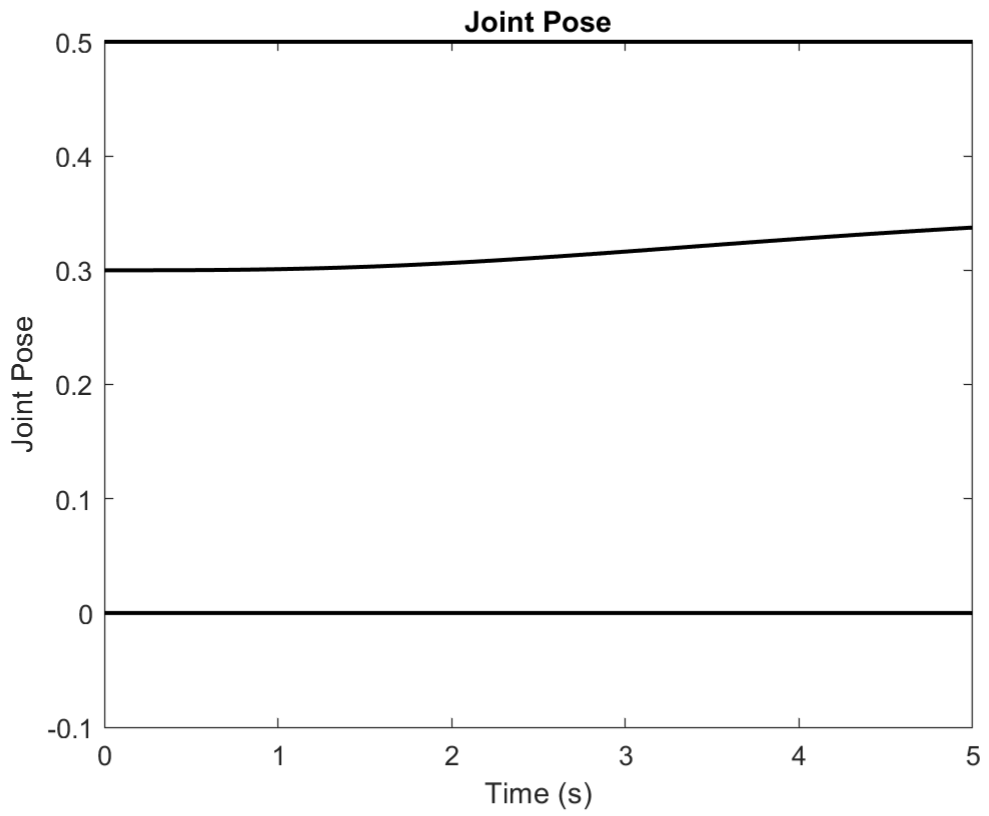
```
[DEBUG] Function lsqnonlin completed. Fitting error: 0.006237. Number of function calls: 6
[INFO] Time : 1.600000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.007263. Number of function calls: 6
[INFO] Time : 1.650000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.008406. Number of function calls: 6
[INFO] Time : 1.700000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.009673. Number of function calls: 6
[INFO] Time : 1.750000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.011068. Number of function calls: 6
[INFO] Time : 1.800000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.012599. Number of function calls: 6
[INFO] Time : 1.850000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.014270. Number of function calls: 7
[INFO] Time : 1.900000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.016086. Number of function calls: 7
[INFO] Time : 1.950000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.018052. Number of function calls: 7
[INFO] Time : 2.000000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.020171. Number of function calls: 7
[INFO] Time : 2.050000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.022445. Number of function calls: 7
[INFO] Time : 2.100000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.024876. Number of function calls: 7
[INFO] Time : 2.150000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.027465. Number of function calls: 7
[INFO] Time : 2.200000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.030212. Number of function calls: 7
[INFO] Time : 2.250000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.033117. Number of function calls: 7
[INFO] Time : 2.300000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.036177. Number of function calls: 7
[INFO] Time : 2.350000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.039389. Number of function calls: 7
[INFO] Time : 2.400000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.042749. Number of function calls: 7
[INFO] Time : 2.450000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.046253. Number of function calls: 7
[INFO] Time : 2.500000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.049893. Number of function calls: 7
[INFO] Time : 2.550000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.053664. Number of function calls: 7
[INFO] Time : 2.600000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.057557. Number of function calls: 7
[INFO] Time : 2.650000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.061564. Number of function calls: 7
[INFO] Time : 2.700000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.065674. Number of function calls: 7
[INFO] Time : 2.750000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.069877. Number of function calls: 7
[INFO] Time : 2.800000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.074161. Number of function calls: 7
[INFO] Time : 2.850000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.078515. Number of function calls: 7
[INFO] Time : 2.900000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.082925. Number of function calls: 7
[INFO] Time : 2.950000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.087379. Number of function calls: 7
[INFO] Time : 3.000000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.091863. Number of function calls: 7
[INFO] Time : 3.050000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.096363. Number of function calls: 7
[INFO] Time : 3.100000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.100865. Number of function calls: 7
[INFO] Time : 3.150000
```

```
[DEBUG] Function lsqnonlin completed. Fitting error: 0.105354. Number of function calls: 7
[INFO] Time : 3.200000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.109817. Number of function calls: 7
[INFO] Time : 3.250000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.114239. Number of function calls: 7
[INFO] Time : 3.300000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.118605. Number of function calls: 7
[INFO] Time : 3.350000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.122902. Number of function calls: 7
[INFO] Time : 3.400000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.127116. Number of function calls: 7
[INFO] Time : 3.450000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.131235. Number of function calls: 7
[INFO] Time : 3.500000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.135245. Number of function calls: 7
[INFO] Time : 3.550000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.139135. Number of function calls: 7
[INFO] Time : 3.600000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.142894. Number of function calls: 7
[INFO] Time : 3.650000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.146509. Number of function calls: 7
[INFO] Time : 3.700000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.149973. Number of function calls: 7
[INFO] Time : 3.750000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.153275. Number of function calls: 7
[INFO] Time : 3.800000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.156409. Number of function calls: 7
[INFO] Time : 3.850000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.159365. Number of function calls: 7
[INFO] Time : 3.900000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.162140. Number of function calls: 7
[INFO] Time : 3.950000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.164727. Number of function calls: 7
[INFO] Time : 4.000000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.167122. Number of function calls: 7
[INFO] Time : 4.050000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.169324. Number of function calls: 7
[INFO] Time : 4.100000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.171329. Number of function calls: 7
[INFO] Time : 4.150000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.173138. Number of function calls: 7
[INFO] Time : 4.200000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.174751. Number of function calls: 7
[INFO] Time : 4.250000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.176171. Number of function calls: 7
[INFO] Time : 4.300000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.177399. Number of function calls: 7
[INFO] Time : 4.350000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.178442. Number of function calls: 7
[INFO] Time : 4.400000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.179304. Number of function calls: 7
[INFO] Time : 4.450000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.179992. Number of function calls: 7
[INFO] Time : 4.500000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.180514. Number of function calls: 7
[INFO] Time : 4.550000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.180880. Number of function calls: 7
[INFO] Time : 4.600000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.181100. Number of function calls: 7
[INFO] Time : 4.650000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.181187. Number of function calls: 7
[INFO] Time : 4.700000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.181152. Number of function calls: 7
[INFO] Time : 4.750000
```

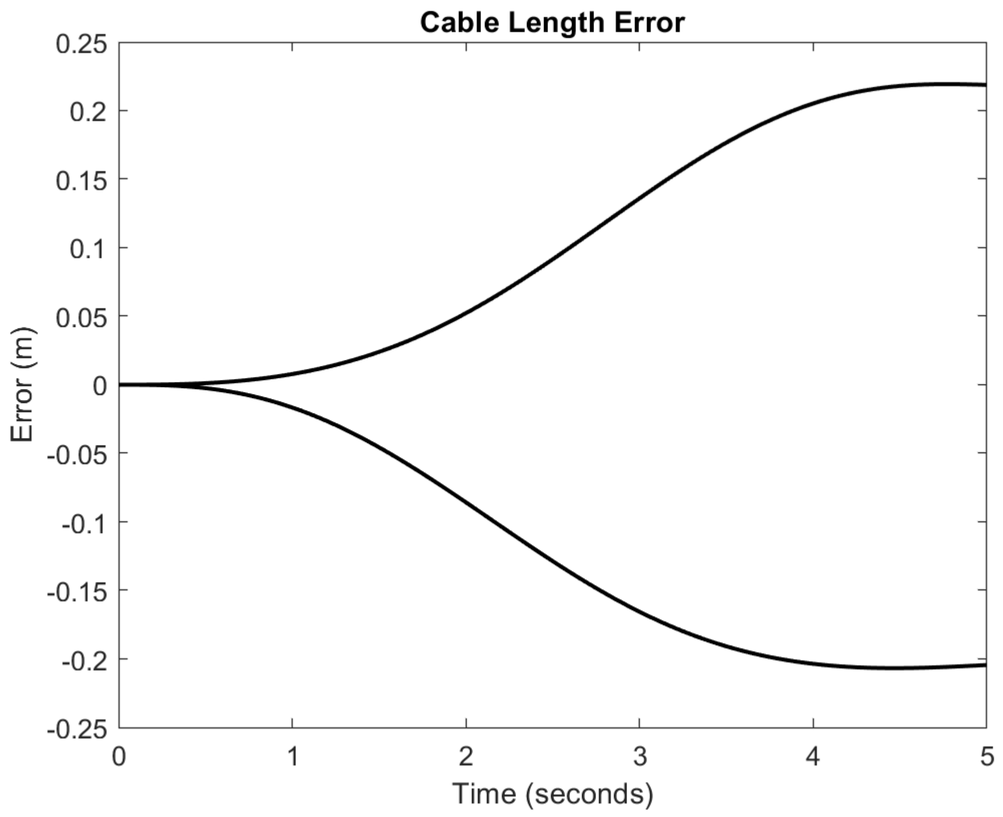
```
[DEBUG] Function lsqnonlin completed. Fitting error: 0.181012. Number of function calls: 7
[INFO] Time : 4.800000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.180781. Number of function calls: 7
[INFO] Time : 4.850000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.180475. Number of function calls: 7
[INFO] Time : 4.900000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.180114. Number of function calls: 7
[INFO] Time : 4.950000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.179715. Number of function calls: 7
[INFO] Time : 5.000000
[DEBUG] Function lsqnonlin completed. Fitting error: 0.179301. Number of function calls: 7
```

The forward kinematics simulator can plot the joint pose, but also resulting cable length error resulting from the joint pose results

```
fk_sim.plotJointPose();
```



```
fk_sim.plotCableLengthError();
```



The simulator also stores some useful information:

```
fprintf('Total time: %f seconds\n', fk_sim.compTimeTotal);
```

```
Total time: 6.012046 seconds
```

```
fprintf('Total length error: %f\n', fk_sim.lengthErrorTotal);
```

```
Total length error: 22.059590
```

2) Writing your own time loop

If you want more control over what is happening or customise the inner-workings during the simulation, it is recommended for you to write your own time loop. The following simple example is basically what happens within the ForwardKinematicsSimulator.

First, create the variables (pre-allocate memory) required to store the solution from the inverse kinematics (cable lengths and cable lengths velocity):

```
fk_solution = zeros(cdpr_model.numDofs, length(trajectory.timeVector)); % Each column
comp_time = zeros(length(trajectory.timeVector), 1);
```

Setup some initial variables, previous length, previous joint pose and previous joint velocity. This is required by the forward kinematics solvers.

```
len_prev = ik_sim.cableLengths{1};          % Initialise the previous length as the fi
q_prev_leastsquares = init_q;
q_dot_prev_leastsquares = init_q_dot;
```

Next, loop through the trajectory as a function of time:

```
for t = 1:length(trajectory.timeVector)
```

The main step within the time loop is to call the solver to resolve the forward kinematics:

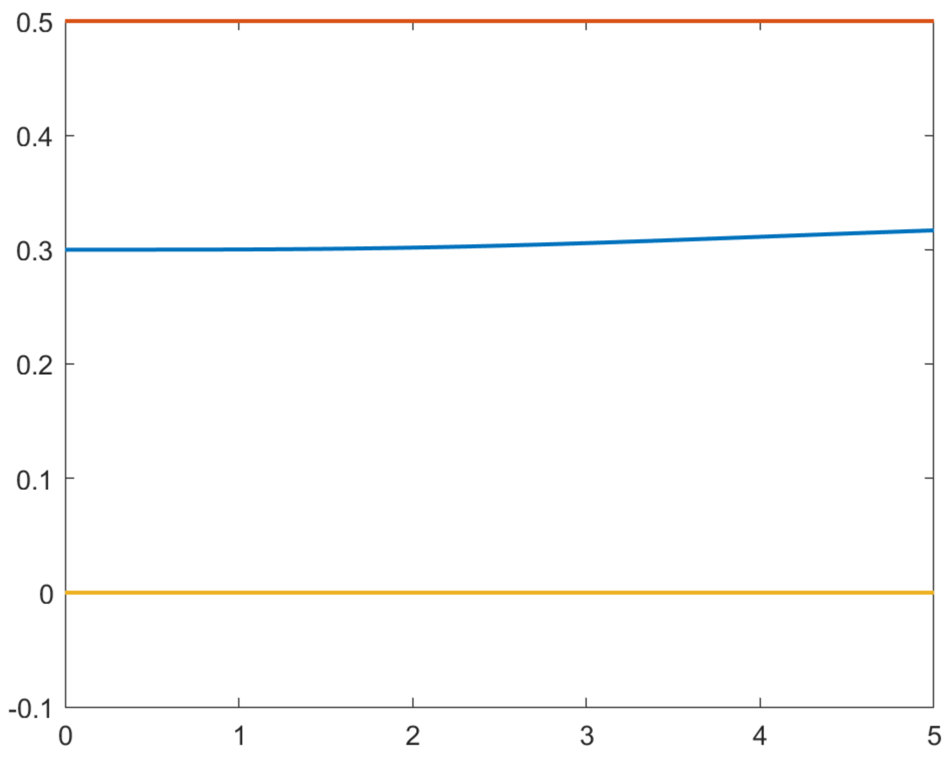
```
    % Compute for the least squares method forward kinematics solver
    [q_sol, q_dot_sol, comp_time(t)] = fk_solver.compute(ik_sim.cableLengths{t}, len_p
    fk_solution(:, t) = q_sol;
    q_prev_leastsquares = q_sol;
    q_dot_prev_leastsquares = q_dot_sol;

    % Store cable length for next loop
    len_prev = ik_sim.cableLengths{t};
end
```

```
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 1
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000000. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000001. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000002. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000004. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000009. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000016. Number of function calls: 3
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000027. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000044. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000069. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000104. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000152. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000217. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000301. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000410. Number of function calls: 4
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000548. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000720. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.000932. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.001190. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.001499. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.001868. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.002302. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.002809. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.003396. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.004072. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.004844. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.005719. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.006705. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.007810. Number of function calls: 5
[DEBUG] Function lsqnonlin completed. Fitting error: 0.009041. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.010406. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.011910. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.013560. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.015362. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.017321. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.019441. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.021726. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.024179. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.026802. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.029596. Number of function calls: 6
```

```
[DEBUG] Function lsqnonlin completed. Fitting error: 0.032561. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.035696. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.038999. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.042467. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.046096. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.049881. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.053815. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.057891. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.062101. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.066434. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.070881. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.075431. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.080071. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.084788. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.089570. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.094403. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.099271. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.104160. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.109055. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.113941. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.118803. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.123625. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.128393. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.133092. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.137707. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.142224. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.146630. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.150913. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.155060. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.159060. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.162901. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.166576. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.170074. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.173389. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.176513. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.179441. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.182169. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.184694. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.187013. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.189126. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.191033. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.192735. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.194237. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.195542. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.196656. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.197586. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.198340. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.198927. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199359. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199648. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199807. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199852. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199798. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199662. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199464. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.199224. Number of function calls: 6
[DEBUG] Function lsqnonlin completed. Fitting error: 0.198963. Number of function calls: 6
```

The results can be plotted the way you desire yourself.

```
figure;
plot(trajectory.timeVector, fk_solution, 'LineWidth', 1.5);
```



```
fprintf('Total time: %f seconds\n', sum(comp_time));
```

```
Total time: 3.869684 seconds
```

Running another forward kinematics solver

Another forward kinematics solver, `FKDifferential`, will be shown to compare with the `FKLeastSquares` solver. Refer to `~/src/Analysis/ForwardKinematics/` folder for more solvers that you can use.

```
fk_diff_solver = FKDifferential(cdpr_model); % Refer to ~/src/Analysis/ForwardKinemati
fk_sim_diff = ForwardKinematicsSimulator(cdpr_model, fk_diff_solver);
fk_sim_diff.run(ik_sim.cableLengths, ik_sim.cableLengthsDot, ik_sim.timeVector, init_q
```

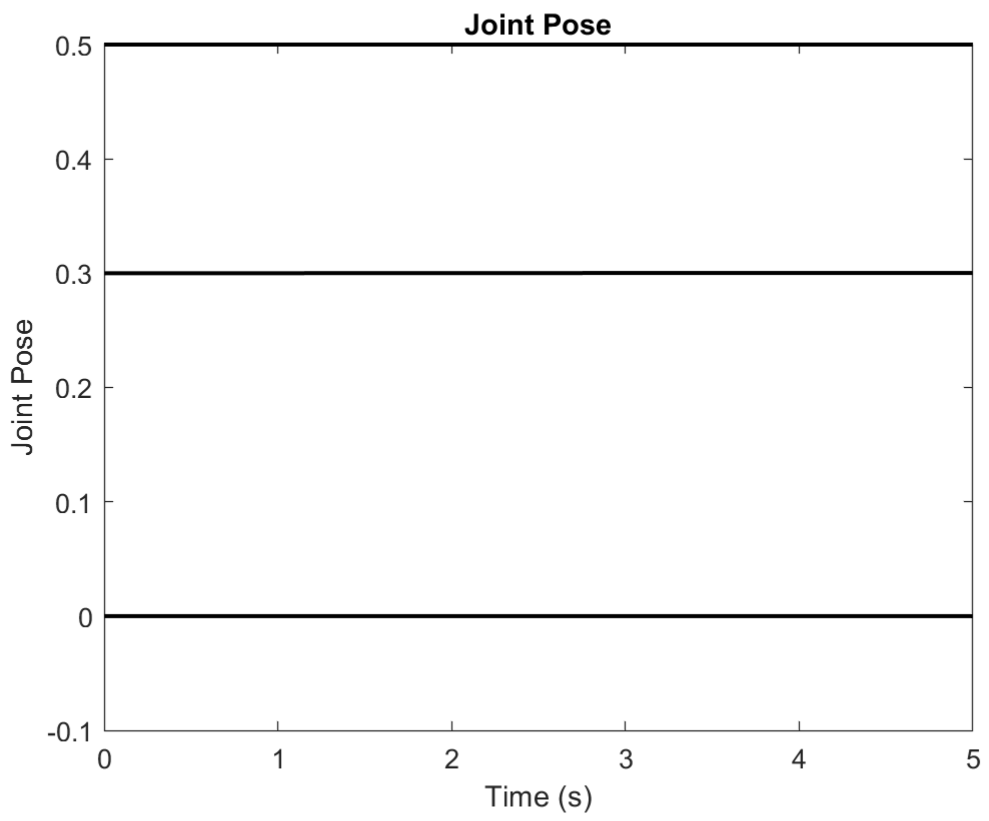
```
[INFO] Begin forward kinematics simulator run...
[INFO] Time : 0.000000
[INFO] Time : 0.050000
[INFO] Time : 0.100000
[INFO] Time : 0.150000
[INFO] Time : 0.200000
[INFO] Time : 0.250000
[INFO] Time : 0.300000
[INFO] Time : 0.350000
[INFO] Time : 0.400000
[INFO] Time : 0.450000
[INFO] Time : 0.500000
[INFO] Time : 0.550000
[INFO] Time : 0.600000
[INFO] Time : 0.650000
[INFO] Time : 0.700000
[INFO] Time : 0.750000
```

```
[INFO] Time : 0.800000
[INFO] Time : 0.850000
[INFO] Time : 0.900000
[INFO] Time : 0.950000
[INFO] Time : 1.000000
[INFO] Time : 1.050000
[INFO] Time : 1.100000
[INFO] Time : 1.150000
[INFO] Time : 1.200000
[INFO] Time : 1.250000
[INFO] Time : 1.300000
[INFO] Time : 1.350000
[INFO] Time : 1.400000
[INFO] Time : 1.450000
[INFO] Time : 1.500000
[INFO] Time : 1.550000
[INFO] Time : 1.600000
[INFO] Time : 1.650000
[INFO] Time : 1.700000
[INFO] Time : 1.750000
[INFO] Time : 1.800000
[INFO] Time : 1.850000
[INFO] Time : 1.900000
[INFO] Time : 1.950000
[INFO] Time : 2.000000
[INFO] Time : 2.050000
[INFO] Time : 2.100000
[INFO] Time : 2.150000
[INFO] Time : 2.200000
[INFO] Time : 2.250000
[INFO] Time : 2.300000
[INFO] Time : 2.350000
[INFO] Time : 2.400000
[INFO] Time : 2.450000
[INFO] Time : 2.500000
[INFO] Time : 2.550000
[INFO] Time : 2.600000
[INFO] Time : 2.650000
[INFO] Time : 2.700000
[INFO] Time : 2.750000
[INFO] Time : 2.800000
[INFO] Time : 2.850000
[INFO] Time : 2.900000
[INFO] Time : 2.950000
[INFO] Time : 3.000000
[INFO] Time : 3.050000
[INFO] Time : 3.100000
[INFO] Time : 3.150000
[INFO] Time : 3.200000
[INFO] Time : 3.250000
[INFO] Time : 3.300000
[INFO] Time : 3.350000
[INFO] Time : 3.400000
[INFO] Time : 3.450000
[INFO] Time : 3.500000
[INFO] Time : 3.550000
[INFO] Time : 3.600000
[INFO] Time : 3.650000
[INFO] Time : 3.700000
[INFO] Time : 3.750000
[INFO] Time : 3.800000
[INFO] Time : 3.850000
[INFO] Time : 3.900000
[INFO] Time : 3.950000
```

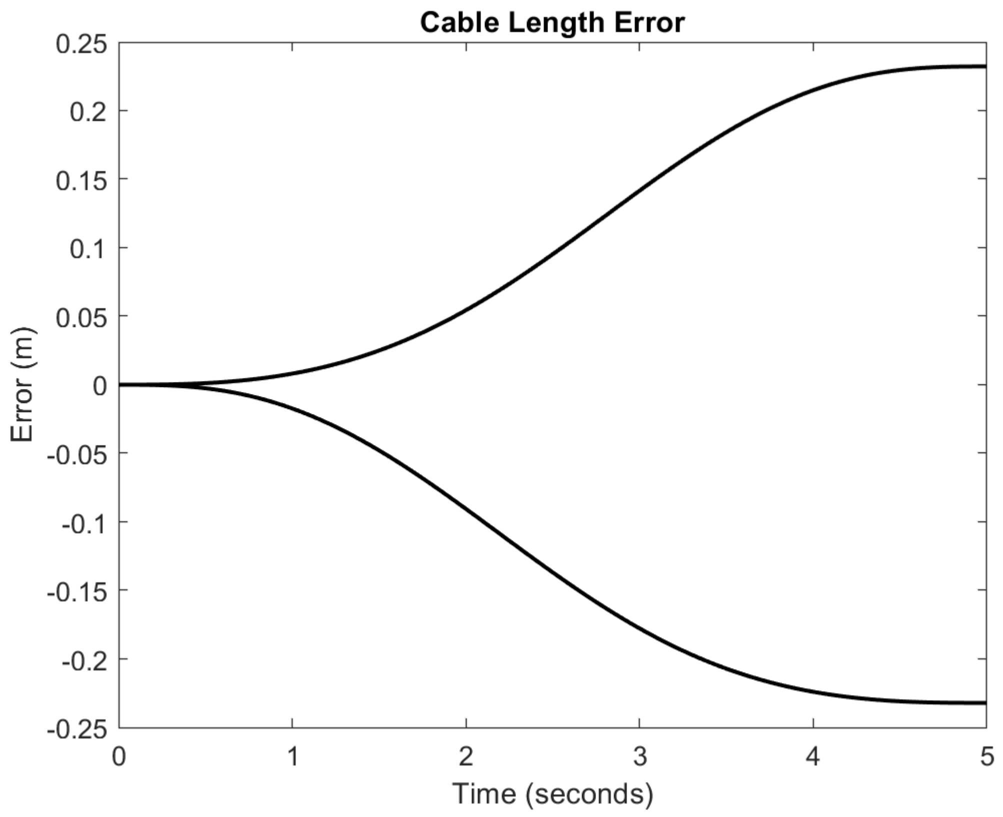
```
[INFO] Time : 4.000000
[INFO] Time : 4.050000
[INFO] Time : 4.100000
[INFO] Time : 4.150000
[INFO] Time : 4.200000
[INFO] Time : 4.250000
[INFO] Time : 4.300000
[INFO] Time : 4.350000
[INFO] Time : 4.400000
[INFO] Time : 4.450000
[INFO] Time : 4.500000
[INFO] Time : 4.550000
[INFO] Time : 4.600000
[INFO] Time : 4.650000
[INFO] Time : 4.700000
[INFO] Time : 4.750000
[INFO] Time : 4.800000
[INFO] Time : 4.850000
[INFO] Time : 4.900000
[INFO] Time : 4.950000
[INFO] Time : 5.000000
```

Again, the performance of the solver can be observed by plotting the resulting joint poses and the cable length error:

```
fk_sim_diff.plotJointPose();
```



```
fk_sim_diff.plotCableLengthError();
```



It can be seen that the least squares method is more accurate (lower error) but is more time consuming compared with the differential method.

```
% Print information related to the least squares method simulation
fprintf('For least squares method:\n');
```

```
For least squares method:
```

```
fprintf('Total time: %f seconds\n', fk_sim.compTimeTotal);
```

```
Total time: 6.012046 seconds
```

```
fprintf('Total length error: %f\n', fk_sim.lengthErrorTotal);
```

```
Total length error: 22.059590
```

```
% Print information related to the differential method simulation
fprintf('For differential method:\n');
```

```
For differential method:
```

```
fprintf('Total time: %f seconds\n', fk_sim_diff.compTimeTotal);
```

```
Total time: 0.191308 seconds
```

```
fprintf('Total length error: %f\n', fk_sim_diff.lengthErrorTotal);
```

```
Total length error: 23.609543
```

Tutorial 5: Running Inverse Dynamics in CASPR

Inverse dynamics for cable-driven robots (CDRs) refer to the determination of the set of cable forces for a given desired motion and wrench to be executed.

This tutorial shows how to run inverse dynamics through two approaches:

1. InverseDynamicsSimulator
2. Writing your own time loop

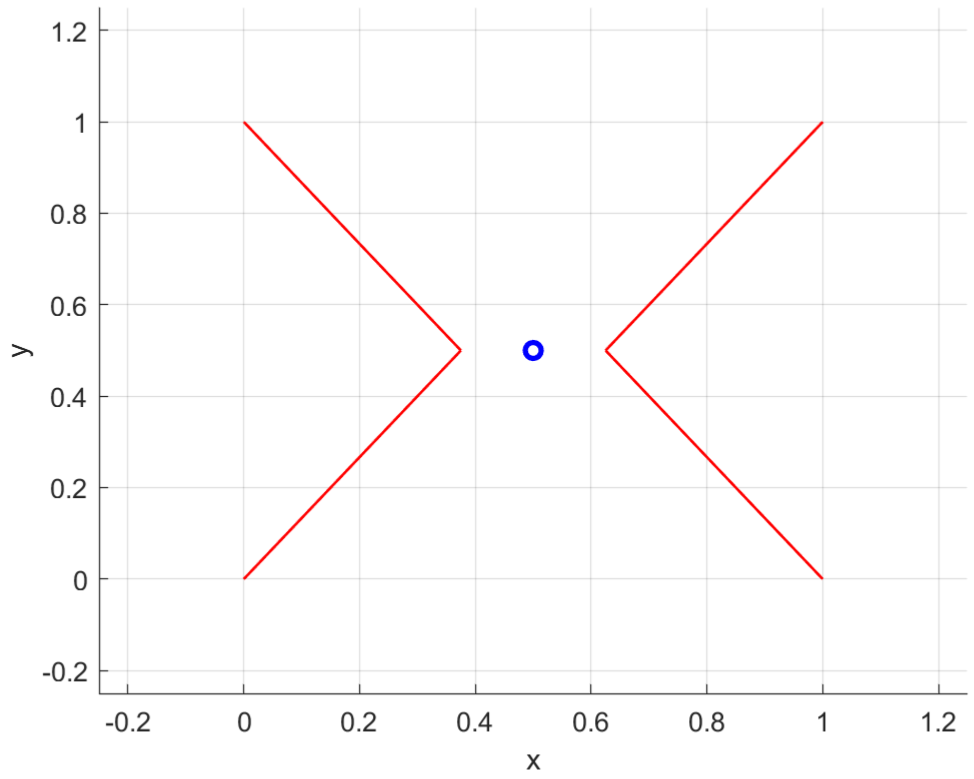
```
clear;
```

Setup (required for both approaches)

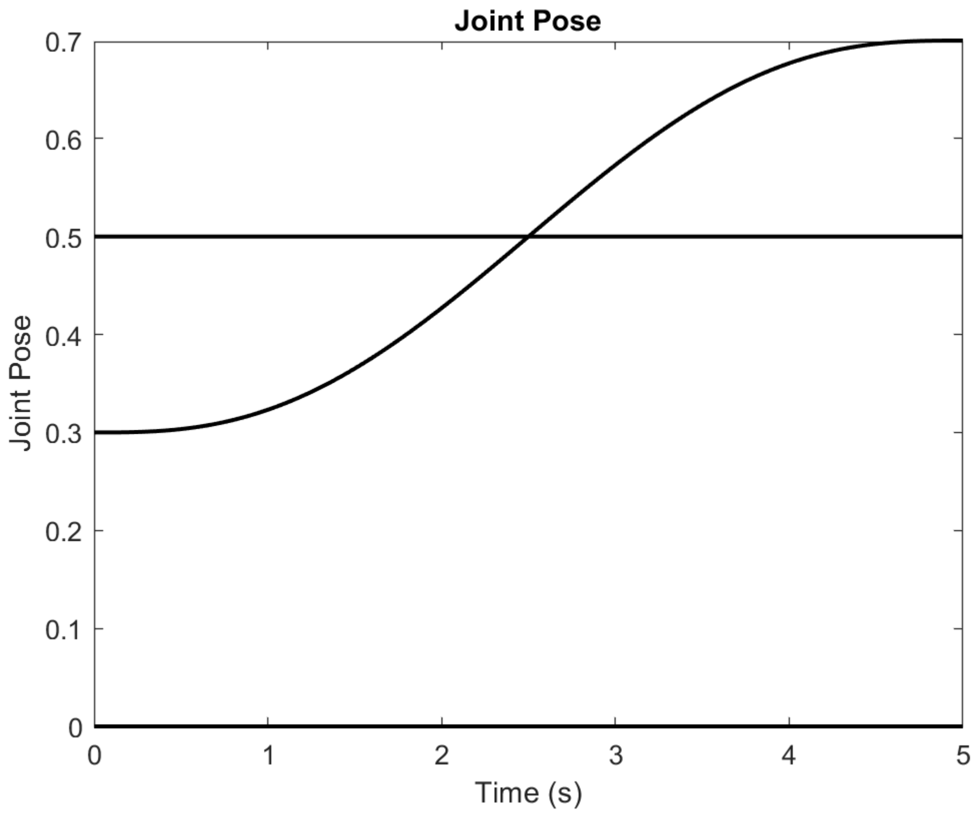
Load the robot and trajectory

Load a robot as shown in `T1_load_robot.mlx` and `T2_load_trajectory.mlx`

```
model_config = ModelConfig('Example planar XY');
cdpr_model = model_config.getModel('basic');
trajectory = model_config.getJointTrajectory('example_quintic');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



```
trajectory.plotJointPose();
```



Setup inverse dynamics solver

Since there are many different methods to solve for the inverse dynamics, different solvers can be setup to be used in CASPR. For the sake of example, the most common quadratic programming (QP) method with an objective function of minimising the sum of the squared of cable forces will be shown in this example:

```
min_forces_objective = IDObjectiveMinQuadCableForce(ones(cdpr_model.numActuatorsActive
id_solver = IDSolverQuadProg(cdpr_model, min_forces_objective, ID_QP_SolverType.MATLAB
```

1) Using the InverseDynamicsSimulator

The `InverseDynamicsSimulator` provides the simplest way to run a "standard" inverse dynamics simulation for a defined trajectory.

First, create the simulator object using the robot model and the specified solver:

```
id_sim = InverseDynamicsSimulator(cdpr_model, id_solver);
```

Next, run the simulator on a particular trajectory

```
id_sim.run(trajectory);
```

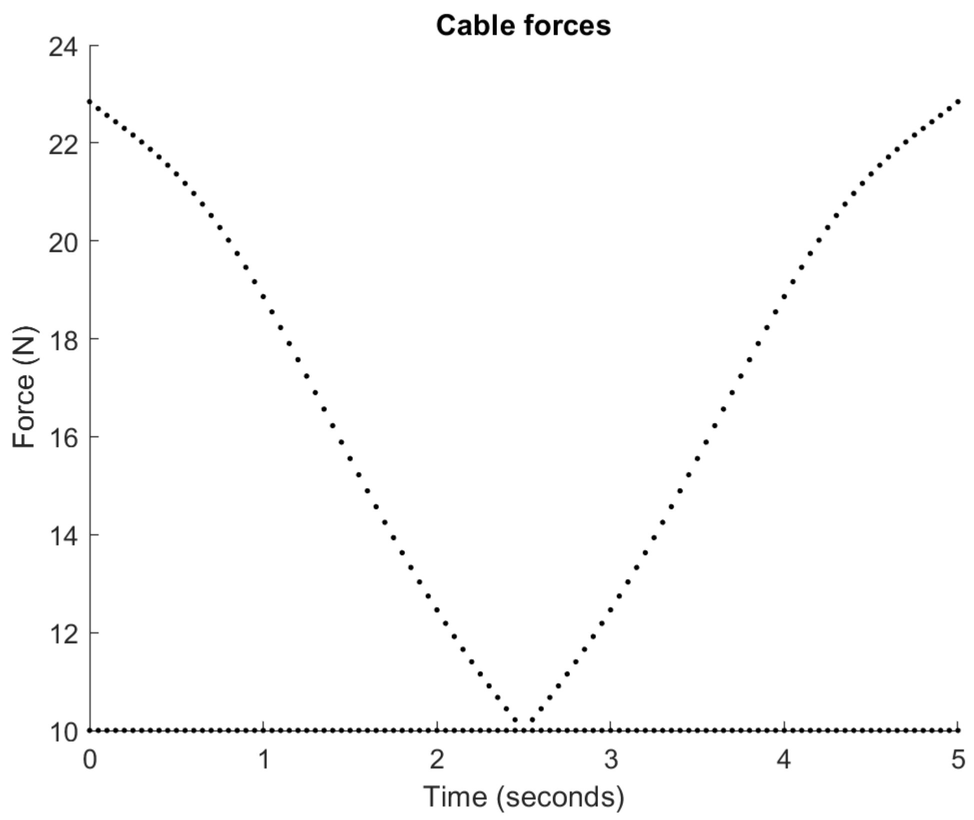
```
[INFO] Time : 0.000000
[INFO] Time : 0.050000
[INFO] Time : 0.100000
[INFO] Time : 0.150000
[INFO] Time : 0.200000
[INFO] Time : 0.250000
```

```
[INFO] Time : 0.300000
[INFO] Time : 0.350000
[INFO] Time : 0.400000
[INFO] Time : 0.450000
[INFO] Time : 0.500000
[INFO] Time : 0.550000
[INFO] Time : 0.600000
[INFO] Time : 0.650000
[INFO] Time : 0.700000
[INFO] Time : 0.750000
[INFO] Time : 0.800000
[INFO] Time : 0.850000
[INFO] Time : 0.900000
[INFO] Time : 0.950000
[INFO] Time : 1.000000
[INFO] Time : 1.050000
[INFO] Time : 1.100000
[INFO] Time : 1.150000
[INFO] Time : 1.200000
[INFO] Time : 1.250000
[INFO] Time : 1.300000
[INFO] Time : 1.350000
[INFO] Time : 1.400000
[INFO] Time : 1.450000
[INFO] Time : 1.500000
[INFO] Time : 1.550000
[INFO] Time : 1.600000
[INFO] Time : 1.650000
[INFO] Time : 1.700000
[INFO] Time : 1.750000
[INFO] Time : 1.800000
[INFO] Time : 1.850000
[INFO] Time : 1.900000
[INFO] Time : 1.950000
[INFO] Time : 2.000000
[INFO] Time : 2.050000
[INFO] Time : 2.100000
[INFO] Time : 2.150000
[INFO] Time : 2.200000
[INFO] Time : 2.250000
[INFO] Time : 2.300000
[INFO] Time : 2.350000
[INFO] Time : 2.400000
[INFO] Time : 2.450000
[INFO] Time : 2.500000
[INFO] Time : 2.550000
[INFO] Time : 2.600000
[INFO] Time : 2.650000
[INFO] Time : 2.700000
[INFO] Time : 2.750000
[INFO] Time : 2.800000
[INFO] Time : 2.850000
[INFO] Time : 2.900000
[INFO] Time : 2.950000
[INFO] Time : 3.000000
[INFO] Time : 3.050000
[INFO] Time : 3.100000
[INFO] Time : 3.150000
[INFO] Time : 3.200000
[INFO] Time : 3.250000
[INFO] Time : 3.300000
[INFO] Time : 3.350000
[INFO] Time : 3.400000
[INFO] Time : 3.450000
```

```
[INFO] Time : 3.500000
[INFO] Time : 3.550000
[INFO] Time : 3.600000
[INFO] Time : 3.650000
[INFO] Time : 3.700000
[INFO] Time : 3.750000
[INFO] Time : 3.800000
[INFO] Time : 3.850000
[INFO] Time : 3.900000
[INFO] Time : 3.950000
[INFO] Time : 4.000000
[INFO] Time : 4.050000
[INFO] Time : 4.100000
[INFO] Time : 4.150000
[INFO] Time : 4.200000
[INFO] Time : 4.250000
[INFO] Time : 4.300000
[INFO] Time : 4.350000
[INFO] Time : 4.400000
[INFO] Time : 4.450000
[INFO] Time : 4.500000
[INFO] Time : 4.550000
[INFO] Time : 4.600000
[INFO] Time : 4.650000
[INFO] Time : 4.700000
[INFO] Time : 4.750000
[INFO] Time : 4.800000
[INFO] Time : 4.850000
[INFO] Time : 4.900000
[INFO] Time : 4.950000
[INFO] Time : 5.000000
```

Finally, the simulator can plot the results, such as the cable forces:

```
id_sim.plotCableForces();
```



2) Writing your own time loop

If you want more control over what is happening or customise the inner-workings during the simulation, it is recommended for you to write your own time loop to perform the inverse kinematics. The following simple example is basically what happens within the `InverseDynamicsSimulator`.

First, create the variables (pre-allocate memory) required to store the solution from the inverse kinematics (cable lengths and cable lengths velocity):

```
id_forces = zeros(cdpr_model.numCables, length(trajectory.timeVector)); % Each column
```

Next, loop through the trajectory as a function of time:

```
for t = 1:length(trajectory.timeVector)
```

The main step within the time loop is to compute the inverse dynamics using the solver's `resolve(q, q_dot, q_ddot, w_ext)` function, which requires the current pose, velocity and acceleration, and also external wrench, respectively.

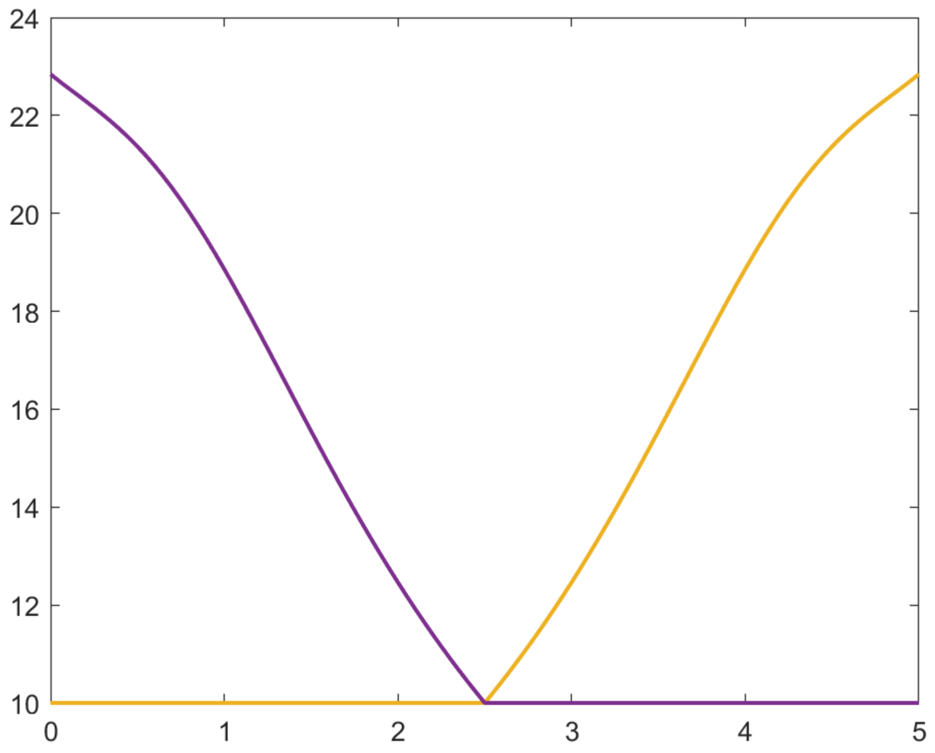
```
    [force_solution, cdpr_model, id_time_cost, id_exit_type, id_comp_time] = ...
        id_solver.resolve(trajectory.q{t}, trajectory.q_dot{t}, trajectory.q_ddot{t},
    id_forces(:,t) = force_solution;
end
```

The function outputs:

1. The solution forces vector
2. Updated robot model with solution of cable forces and other related properties
3. Computational time
4. Exit condition for the solver: no error (success), infeasible, maximum iteration reached (for iterative solvers), solver specific error or other error. Refer to `IDSolverExitType.m` for more details

The results can be plotted the way you desire yourself.

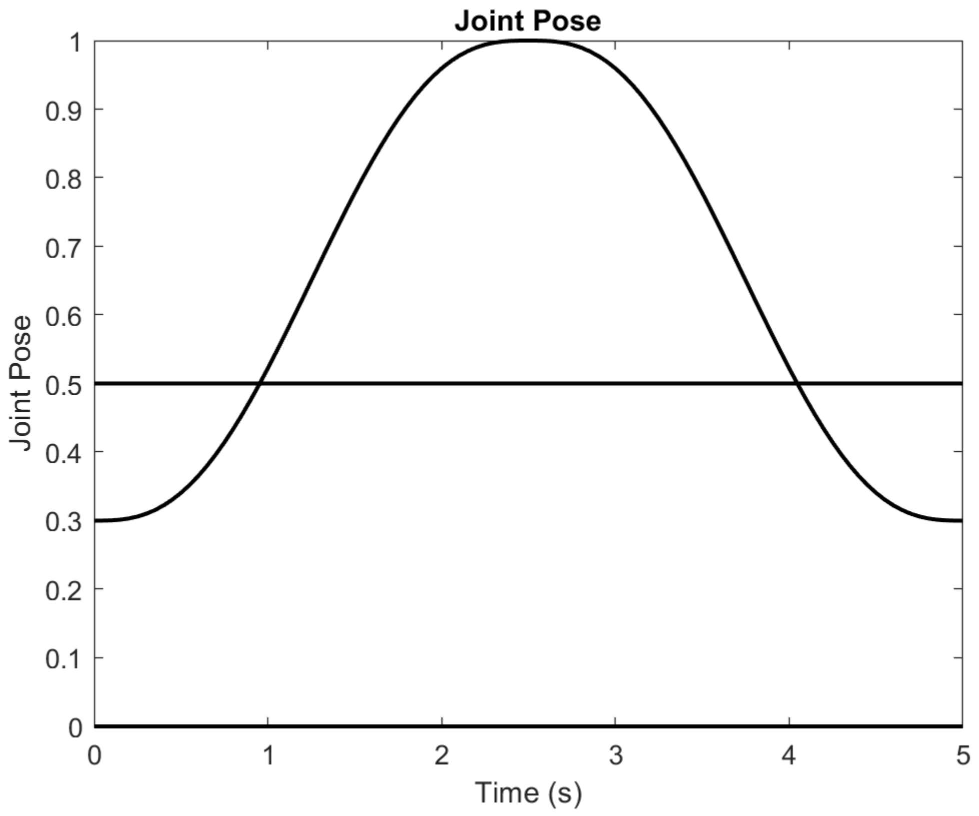
```
figure;
plot(trajectory.timeVector, id_forces, 'LineWidth', 1.5);
```



Simulation on an infeasible trajectory

When a trajectory is infeasible, the `resolve` function will return -1 for the cable force (shown in red in the cable forces plot) and exit type of `IDSolverExitType.INFEASIBLE`.

```
trajectory_infeasible = model_config.getJointTrajectory('example_infeasible');
trajectory_infeasible.plotJointPose();
```



```
id_sim.run(trajectory_infeasible);
```

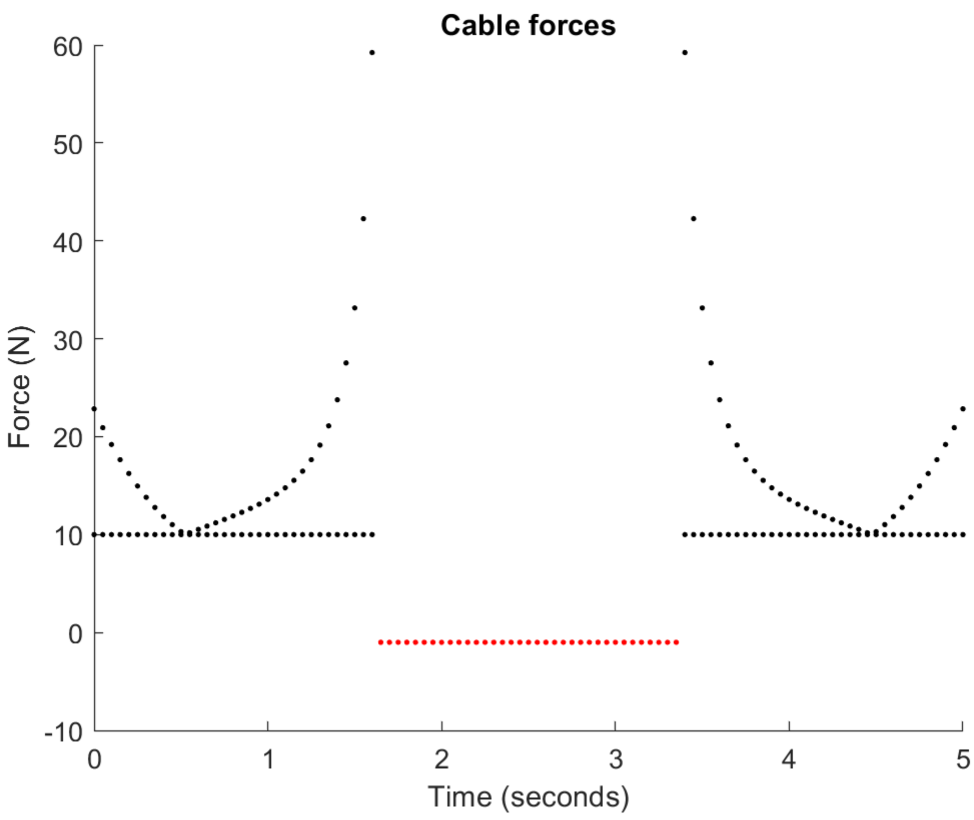
```
[INFO] Time : 0.000000
[INFO] Time : 0.050000
[INFO] Time : 0.100000
[INFO] Time : 0.150000
[INFO] Time : 0.200000
[INFO] Time : 0.250000
[INFO] Time : 0.300000
[INFO] Time : 0.350000
[INFO] Time : 0.400000
[INFO] Time : 0.450000
[INFO] Time : 0.500000
[INFO] Time : 0.550000
[INFO] Time : 0.600000
[INFO] Time : 0.650000
[INFO] Time : 0.700000
[INFO] Time : 0.750000
[INFO] Time : 0.800000
[INFO] Time : 0.850000
[INFO] Time : 0.900000
[INFO] Time : 0.950000
[INFO] Time : 1.000000
[INFO] Time : 1.050000
[INFO] Time : 1.100000
[INFO] Time : 1.150000
[INFO] Time : 1.200000
[INFO] Time : 1.250000
[INFO] Time : 1.300000
[INFO] Time : 1.350000
[INFO] Time : 1.400000
[INFO] Time : 1.450000
[INFO] Time : 1.500000
```

```
[INFO] Time : 1.550000
[INFO] Time : 1.600000
[INFO] Time : 1.650000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 1.700000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 1.750000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 1.800000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 1.850000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 1.900000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 1.950000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.000000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.050000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.100000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.150000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.200000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.250000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.300000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.350000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
```

```
[INFO] Time : 2.400000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.450000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.500000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.550000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.600000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.650000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.700000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.750000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.800000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.850000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.900000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 2.950000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.000000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.050000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.100000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.150000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
```

```
[INFO] Time : 3.200000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.250000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.300000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.350000
[INFO] Problem infeasible
[WARNING] No feasible solution for the ID
Warning: No feasible solution for the ID
[INFO] Time : 3.400000
[INFO] Time : 3.450000
[INFO] Time : 3.500000
[INFO] Time : 3.550000
[INFO] Time : 3.600000
[INFO] Time : 3.650000
[INFO] Time : 3.700000
[INFO] Time : 3.750000
[INFO] Time : 3.800000
[INFO] Time : 3.850000
[INFO] Time : 3.900000
[INFO] Time : 3.950000
[INFO] Time : 4.000000
[INFO] Time : 4.050000
[INFO] Time : 4.100000
[INFO] Time : 4.150000
[INFO] Time : 4.200000
[INFO] Time : 4.250000
[INFO] Time : 4.300000
[INFO] Time : 4.350000
[INFO] Time : 4.400000
[INFO] Time : 4.450000
[INFO] Time : 4.500000
[INFO] Time : 4.550000
[INFO] Time : 4.600000
[INFO] Time : 4.650000
[INFO] Time : 4.700000
[INFO] Time : 4.750000
[INFO] Time : 4.800000
[INFO] Time : 4.850000
[INFO] Time : 4.900000
[INFO] Time : 4.950000
[INFO] Time : 5.000000
[WARNING] At least one point on the trajectory resulted in no feasible solution for the ID
Warning: At least one point on the trajectory resulted in no feasible solution for the ID
```

```
id_sim.plotCableForces();
```



Running another inverse dynamics solver

Another inverse dynamics solver, `IDSolverClosedForm`, will be shown to compare with the QP solver. Refer to `~/src/Analysis/InverseDynamics/` folder for more solvers that you can use.

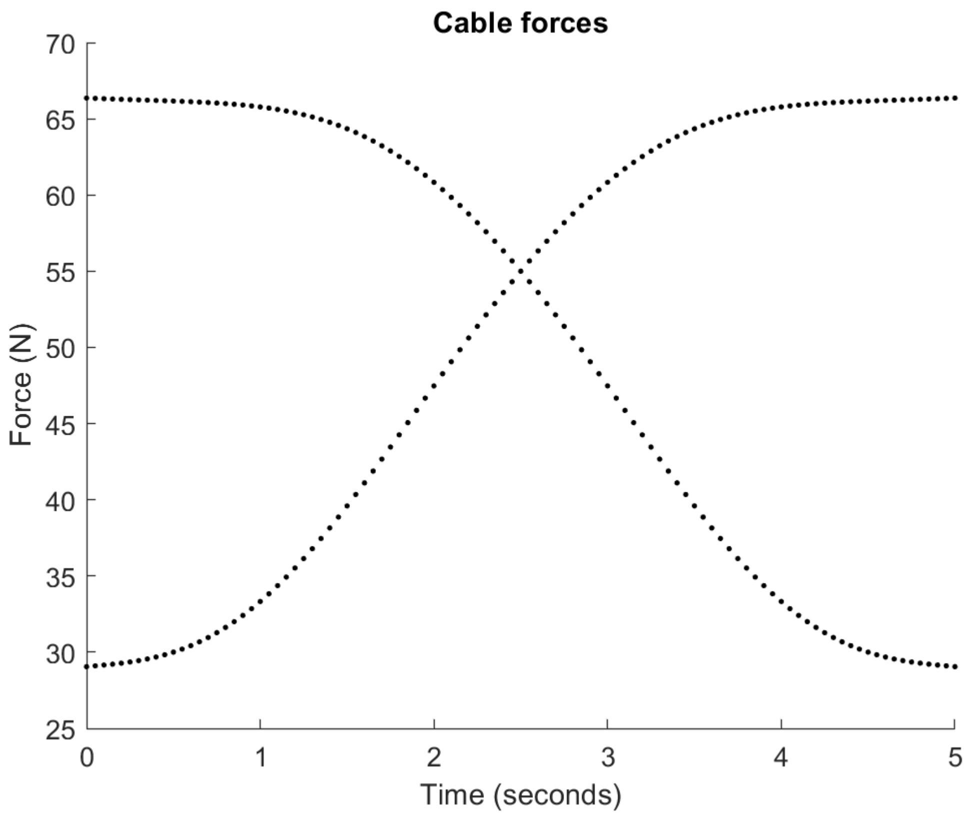
```
% Setup the solver and simulator
id_solver_cf = IDSolverClosedForm(cdpr_model, ID_CF_SolverType.IMPROVED_CLOSED_FORM);
id_sim_cf = InverseDynamicsSimulator(cdpr_model, id_solver_cf);
% Run the simulator
id_sim_cf.run(trajectory);
```

```
[INFO] Time : 0.000000
[INFO] Time : 0.050000
[INFO] Time : 0.100000
[INFO] Time : 0.150000
[INFO] Time : 0.200000
[INFO] Time : 0.250000
[INFO] Time : 0.300000
[INFO] Time : 0.350000
[INFO] Time : 0.400000
[INFO] Time : 0.450000
[INFO] Time : 0.500000
[INFO] Time : 0.550000
[INFO] Time : 0.600000
[INFO] Time : 0.650000
[INFO] Time : 0.700000
[INFO] Time : 0.750000
[INFO] Time : 0.800000
[INFO] Time : 0.850000
[INFO] Time : 0.900000
[INFO] Time : 0.950000
```

```
[INFO] Time : 1.000000
[INFO] Time : 1.050000
[INFO] Time : 1.100000
[INFO] Time : 1.150000
[INFO] Time : 1.200000
[INFO] Time : 1.250000
[INFO] Time : 1.300000
[INFO] Time : 1.350000
[INFO] Time : 1.400000
[INFO] Time : 1.450000
[INFO] Time : 1.500000
[INFO] Time : 1.550000
[INFO] Time : 1.600000
[INFO] Time : 1.650000
[INFO] Time : 1.700000
[INFO] Time : 1.750000
[INFO] Time : 1.800000
[INFO] Time : 1.850000
[INFO] Time : 1.900000
[INFO] Time : 1.950000
[INFO] Time : 2.000000
[INFO] Time : 2.050000
[INFO] Time : 2.100000
[INFO] Time : 2.150000
[INFO] Time : 2.200000
[INFO] Time : 2.250000
[INFO] Time : 2.300000
[INFO] Time : 2.350000
[INFO] Time : 2.400000
[INFO] Time : 2.450000
[INFO] Time : 2.500000
[INFO] Time : 2.550000
[INFO] Time : 2.600000
[INFO] Time : 2.650000
[INFO] Time : 2.700000
[INFO] Time : 2.750000
[INFO] Time : 2.800000
[INFO] Time : 2.850000
[INFO] Time : 2.900000
[INFO] Time : 2.950000
[INFO] Time : 3.000000
[INFO] Time : 3.050000
[INFO] Time : 3.100000
[INFO] Time : 3.150000
[INFO] Time : 3.200000
[INFO] Time : 3.250000
[INFO] Time : 3.300000
[INFO] Time : 3.350000
[INFO] Time : 3.400000
[INFO] Time : 3.450000
[INFO] Time : 3.500000
[INFO] Time : 3.550000
[INFO] Time : 3.600000
[INFO] Time : 3.650000
[INFO] Time : 3.700000
[INFO] Time : 3.750000
[INFO] Time : 3.800000
[INFO] Time : 3.850000
[INFO] Time : 3.900000
[INFO] Time : 3.950000
[INFO] Time : 4.000000
[INFO] Time : 4.050000
[INFO] Time : 4.100000
[INFO] Time : 4.150000
```

```
[INFO] Time : 4.200000
[INFO] Time : 4.250000
[INFO] Time : 4.300000
[INFO] Time : 4.350000
[INFO] Time : 4.400000
[INFO] Time : 4.450000
[INFO] Time : 4.500000
[INFO] Time : 4.550000
[INFO] Time : 4.600000
[INFO] Time : 4.650000
[INFO] Time : 4.700000
[INFO] Time : 4.750000
[INFO] Time : 4.800000
[INFO] Time : 4.850000
[INFO] Time : 4.900000
[INFO] Time : 4.950000
[INFO] Time : 5.000000
```

```
% Plot the results
id_sim_cf.plotCableForces();
```



Tutorial 6: Running Forward Dynamics in CASPR

Forward dynamics for cable-driven robots (CDRs) refer to the determination of the robot pose for a given set of cable forces to simulate the motion of the real robot.

This tutorial shows how to run inverse kinematics through two approaches:

1. ForwardDynamicsSimulator
2. Writing your own time loop

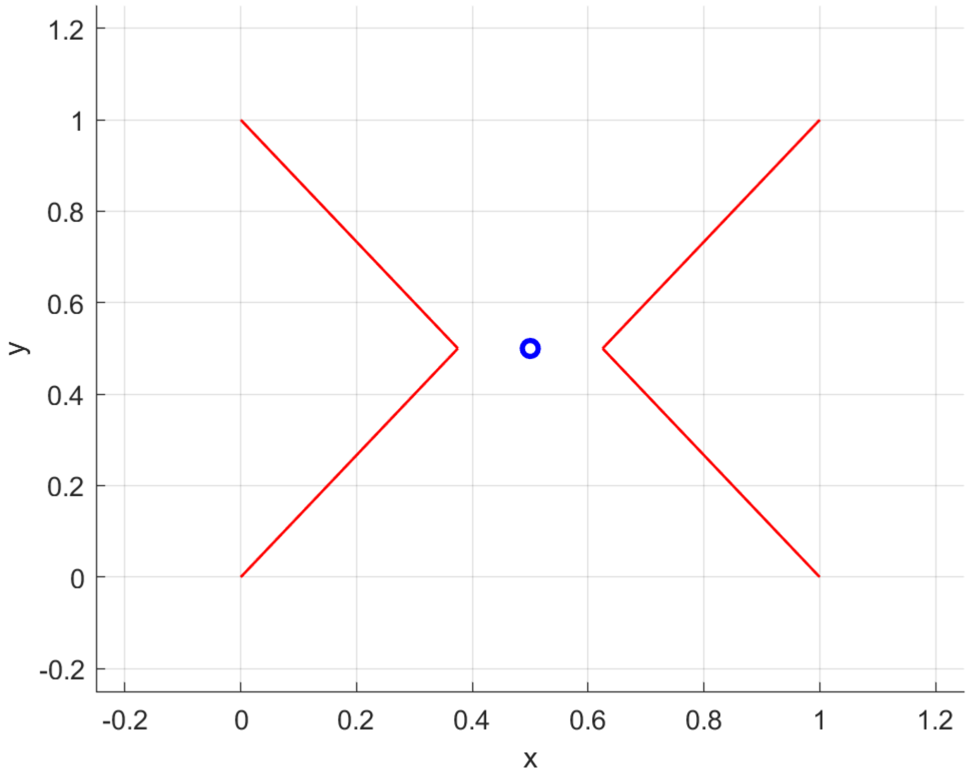
```
clear;
```

Setup (required for both approaches)

Load the robot and trajectory

Load a robot as shown in `T1_load_robot.mlx` and `T2_load_trajectory.mlx`

```
model_config = ModelConfig('Example planar XY');
cdpr_model = model_config.getModel('basic');
trajectory = model_config.getJointTrajectory('example_quintic');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



Run inverse dynamics for input to the forward dynamics

The steps are the same as that from `T5_inverse_dynamics.mlx`

```
% Setup solver
```

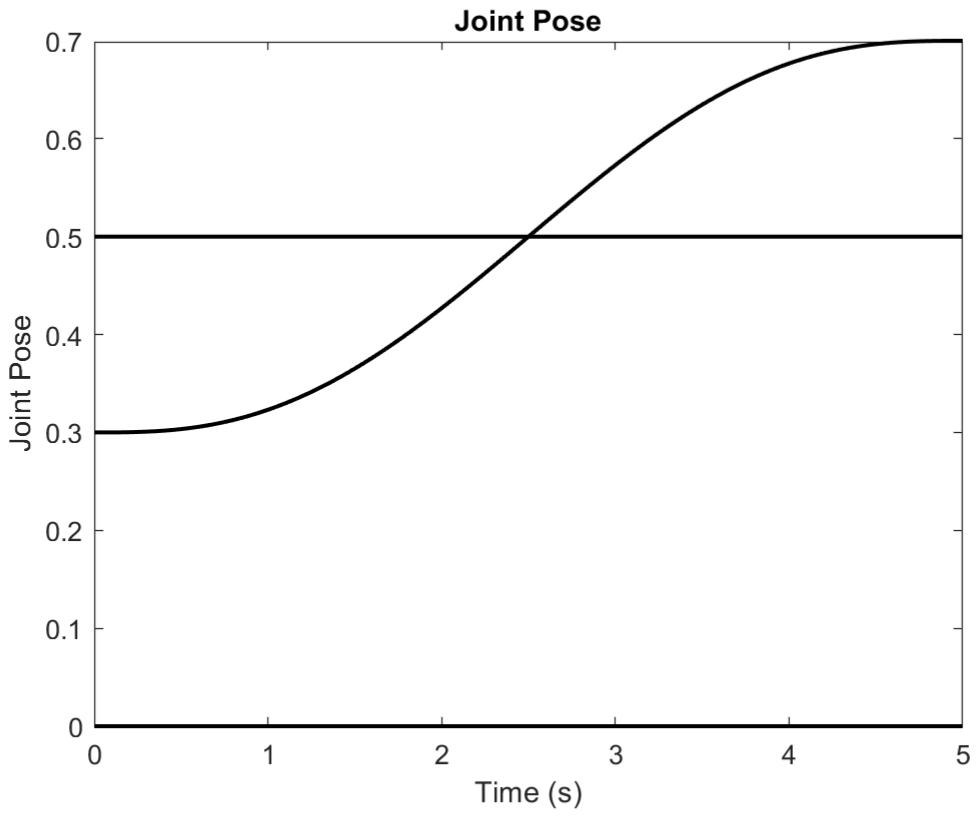
```
min_forces_objective = IDObjectiveMinQuadCableForce(ones(cdpr_model.numActuatorsActive
id_solver = IDSolverQuadProg(cdpr_model, min_forces_objective, ID_QP_SolverType.MATLAB
% Run inverse dynamics
id_sim = InverseDynamicsSimulator(cdpr_model, id_solver);
id_sim.run(trajectory);
```

```
[INFO] Time : 0.000000
[INFO] Time : 0.050000
[INFO] Time : 0.100000
[INFO] Time : 0.150000
[INFO] Time : 0.200000
[INFO] Time : 0.250000
[INFO] Time : 0.300000
[INFO] Time : 0.350000
[INFO] Time : 0.400000
[INFO] Time : 0.450000
[INFO] Time : 0.500000
[INFO] Time : 0.550000
[INFO] Time : 0.600000
[INFO] Time : 0.650000
[INFO] Time : 0.700000
[INFO] Time : 0.750000
[INFO] Time : 0.800000
[INFO] Time : 0.850000
[INFO] Time : 0.900000
[INFO] Time : 0.950000
[INFO] Time : 1.000000
[INFO] Time : 1.050000
[INFO] Time : 1.100000
[INFO] Time : 1.150000
[INFO] Time : 1.200000
[INFO] Time : 1.250000
[INFO] Time : 1.300000
[INFO] Time : 1.350000
[INFO] Time : 1.400000
[INFO] Time : 1.450000
[INFO] Time : 1.500000
[INFO] Time : 1.550000
[INFO] Time : 1.600000
[INFO] Time : 1.650000
[INFO] Time : 1.700000
[INFO] Time : 1.750000
[INFO] Time : 1.800000
[INFO] Time : 1.850000
[INFO] Time : 1.900000
[INFO] Time : 1.950000
[INFO] Time : 2.000000
[INFO] Time : 2.050000
[INFO] Time : 2.100000
[INFO] Time : 2.150000
[INFO] Time : 2.200000
[INFO] Time : 2.250000
[INFO] Time : 2.300000
[INFO] Time : 2.350000
[INFO] Time : 2.400000
[INFO] Time : 2.450000
[INFO] Time : 2.500000
[INFO] Time : 2.550000
[INFO] Time : 2.600000
[INFO] Time : 2.650000
[INFO] Time : 2.700000
[INFO] Time : 2.750000
[INFO] Time : 2.800000
```

```
[INFO] Time : 2.850000
[INFO] Time : 2.900000
[INFO] Time : 2.950000
[INFO] Time : 3.000000
[INFO] Time : 3.050000
[INFO] Time : 3.100000
[INFO] Time : 3.150000
[INFO] Time : 3.200000
[INFO] Time : 3.250000
[INFO] Time : 3.300000
[INFO] Time : 3.350000
[INFO] Time : 3.400000
[INFO] Time : 3.450000
[INFO] Time : 3.500000
[INFO] Time : 3.550000
[INFO] Time : 3.600000
[INFO] Time : 3.650000
[INFO] Time : 3.700000
[INFO] Time : 3.750000
[INFO] Time : 3.800000
[INFO] Time : 3.850000
[INFO] Time : 3.900000
[INFO] Time : 3.950000
[INFO] Time : 4.000000
[INFO] Time : 4.050000
[INFO] Time : 4.100000
[INFO] Time : 4.150000
[INFO] Time : 4.200000
[INFO] Time : 4.250000
[INFO] Time : 4.300000
[INFO] Time : 4.350000
[INFO] Time : 4.400000
[INFO] Time : 4.450000
[INFO] Time : 4.500000
[INFO] Time : 4.550000
[INFO] Time : 4.600000
[INFO] Time : 4.650000
[INFO] Time : 4.700000
[INFO] Time : 4.750000
[INFO] Time : 4.800000
[INFO] Time : 4.850000
[INFO] Time : 4.900000
[INFO] Time : 4.950000
[INFO] Time : 5.000000
```

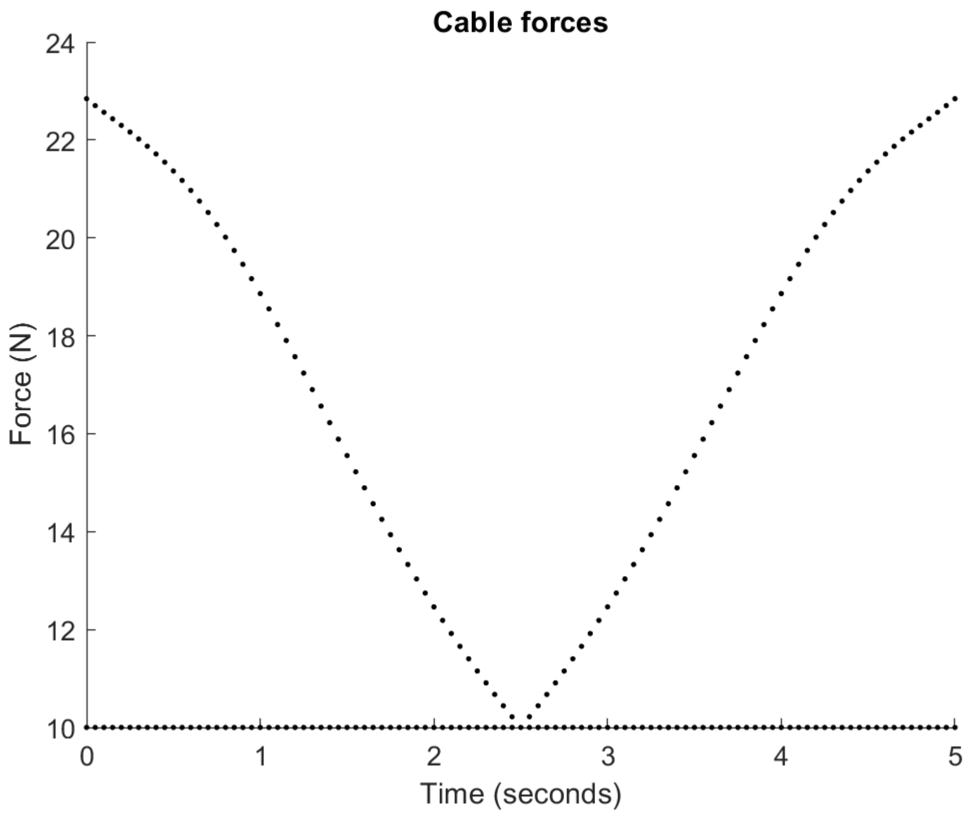
Original joint pose (to be solved using forward dynamics):

```
id_sim.plotJointPose();
```



The set of cable forces that will be the input to the forward dynamics:

```
id_sim.plotCableForces();
```



Setup some initial values for use later:

```
init_q = ik_sim.trajectory.q{1}; % Initial q for the solver
init_q_dot = ik_sim.trajectory.q_dot{1}; % Initial q_dot for the solver
```

1) Solve using the ForwardDynamicsSimulator

The Forward`KinematicsSimulator` provides the simplest way to run a "standard" forward kinematics simulation for a defined trajectory.

First, create the simulator object using the robot model and the specified solver:

```
fd_sim = ForwardDynamicsSimulator(cdpr_model, FDSolverType.ODE113);
```

Next, run the simulator on the resulting cable lengths and cable lengths velocity over the specified time points with the initial joint pose and velocity:

```
fd_sim.run(id_sim.cableForces, id_sim.cableIndicesActive, id_sim.timeVector, init_q, i
```

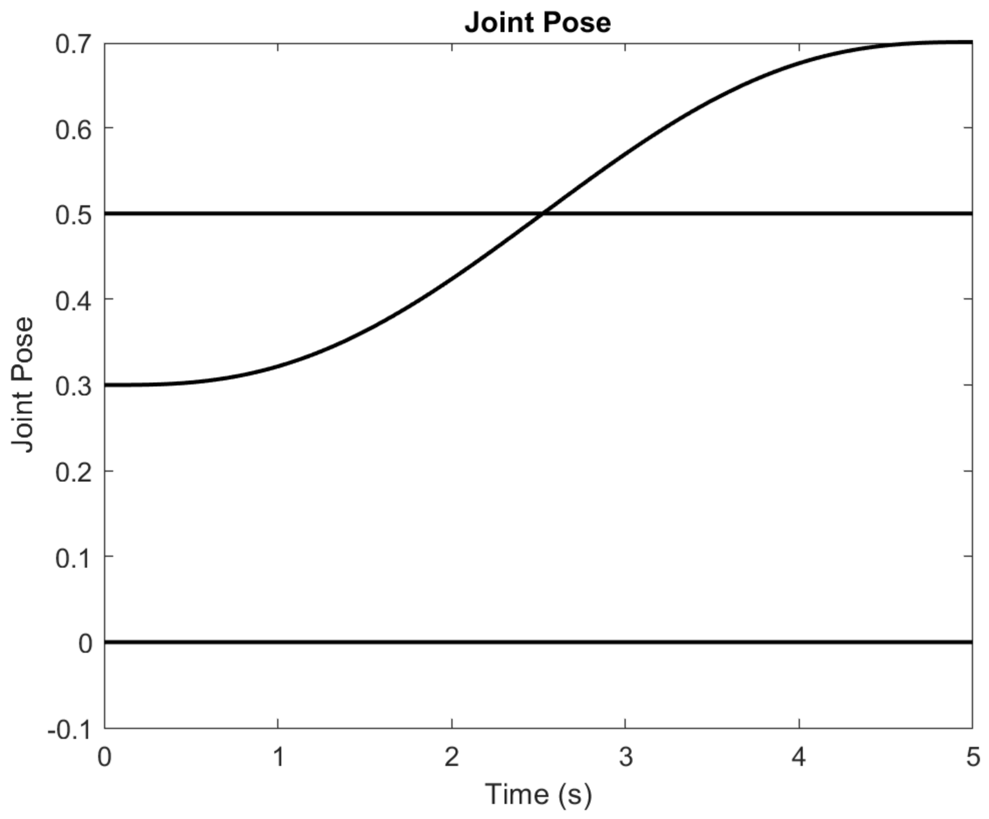
```
[INFO] Simulation time : 0.050000
[INFO] Simulation time : 0.100000
[INFO] Simulation time : 0.150000
[INFO] Simulation time : 0.200000
[INFO] Simulation time : 0.250000
[INFO] Simulation time : 0.300000
[INFO] Simulation time : 0.350000
[INFO] Simulation time : 0.400000
[INFO] Simulation time : 0.450000
[INFO] Simulation time : 0.500000
```

```
[INFO] Simulation time : 0.550000
[INFO] Simulation time : 0.600000
[INFO] Simulation time : 0.650000
[INFO] Simulation time : 0.700000
[INFO] Simulation time : 0.750000
[INFO] Simulation time : 0.800000
[INFO] Simulation time : 0.850000
[INFO] Simulation time : 0.900000
[INFO] Simulation time : 0.950000
[INFO] Simulation time : 1.000000
[INFO] Simulation time : 1.050000
[INFO] Simulation time : 1.100000
[INFO] Simulation time : 1.150000
[INFO] Simulation time : 1.200000
[INFO] Simulation time : 1.250000
[INFO] Simulation time : 1.300000
[INFO] Simulation time : 1.350000
[INFO] Simulation time : 1.400000
[INFO] Simulation time : 1.450000
[INFO] Simulation time : 1.500000
[INFO] Simulation time : 1.550000
[INFO] Simulation time : 1.600000
[INFO] Simulation time : 1.650000
[INFO] Simulation time : 1.700000
[INFO] Simulation time : 1.750000
[INFO] Simulation time : 1.800000
[INFO] Simulation time : 1.850000
[INFO] Simulation time : 1.900000
[INFO] Simulation time : 1.950000
[INFO] Simulation time : 2.000000
[INFO] Simulation time : 2.050000
[INFO] Simulation time : 2.100000
[INFO] Simulation time : 2.150000
[INFO] Simulation time : 2.200000
[INFO] Simulation time : 2.250000
[INFO] Simulation time : 2.300000
[INFO] Simulation time : 2.350000
[INFO] Simulation time : 2.400000
[INFO] Simulation time : 2.450000
[INFO] Simulation time : 2.500000
[INFO] Simulation time : 2.550000
[INFO] Simulation time : 2.600000
[INFO] Simulation time : 2.650000
[INFO] Simulation time : 2.700000
[INFO] Simulation time : 2.750000
[INFO] Simulation time : 2.800000
[INFO] Simulation time : 2.850000
[INFO] Simulation time : 2.900000
[INFO] Simulation time : 2.950000
[INFO] Simulation time : 3.000000
[INFO] Simulation time : 3.050000
[INFO] Simulation time : 3.100000
[INFO] Simulation time : 3.150000
[INFO] Simulation time : 3.200000
[INFO] Simulation time : 3.250000
[INFO] Simulation time : 3.300000
[INFO] Simulation time : 3.350000
[INFO] Simulation time : 3.400000
[INFO] Simulation time : 3.450000
[INFO] Simulation time : 3.500000
[INFO] Simulation time : 3.550000
[INFO] Simulation time : 3.600000
[INFO] Simulation time : 3.650000
[INFO] Simulation time : 3.700000
```

```
[INFO] Simulation time : 3.750000
[INFO] Simulation time : 3.800000
[INFO] Simulation time : 3.850000
[INFO] Simulation time : 3.900000
[INFO] Simulation time : 3.950000
[INFO] Simulation time : 4.000000
[INFO] Simulation time : 4.050000
[INFO] Simulation time : 4.100000
[INFO] Simulation time : 4.150000
[INFO] Simulation time : 4.200000
[INFO] Simulation time : 4.250000
[INFO] Simulation time : 4.300000
[INFO] Simulation time : 4.350000
[INFO] Simulation time : 4.400000
[INFO] Simulation time : 4.450000
[INFO] Simulation time : 4.500000
[INFO] Simulation time : 4.550000
[INFO] Simulation time : 4.600000
[INFO] Simulation time : 4.650000
[INFO] Simulation time : 4.700000
[INFO] Simulation time : 4.750000
[INFO] Simulation time : 4.800000
[INFO] Simulation time : 4.850000
[INFO] Simulation time : 4.900000
[INFO] Simulation time : 4.950000
[INFO] Simulation time : 5.000000
```

The forward dynamics simulator can plot the resulting joint pose

```
fd_sim.plotJointPose();
```



2) Writing your own time loop

If you want more control over what is happening or customise the inner-workings during the simulation, it is recommended for you to write your own time loop. The following simple example is basically what happens within the ForwardDynamicsSimulator.

First, create the variables (pre-allocate memory) required to store the solution from the inverse kinematics (cable lengths and cable lengths velocity):

```
fd_solution_q = zeros(cdpr_model.numDofs, length(trajectory.timeVector)); % Each colum
```

Setup some initial variables, previous length, previous joint pose and previous joint velocity. This is required by the forward kinematics solvers.

```
fd_solution_q(:, 1) = trajectory.q{1};
fd_solution_q_dot(:, 1) = trajectory.q_dot{1};
fd_solution_q_ddot(:, 1) = trajectory.q_ddot{1};

fd_solver = ForwardDynamics(FDSolverType.ODE113);
```

Next, loop through the trajectory as a function of time starting from index 2 since the initial pose is already known:

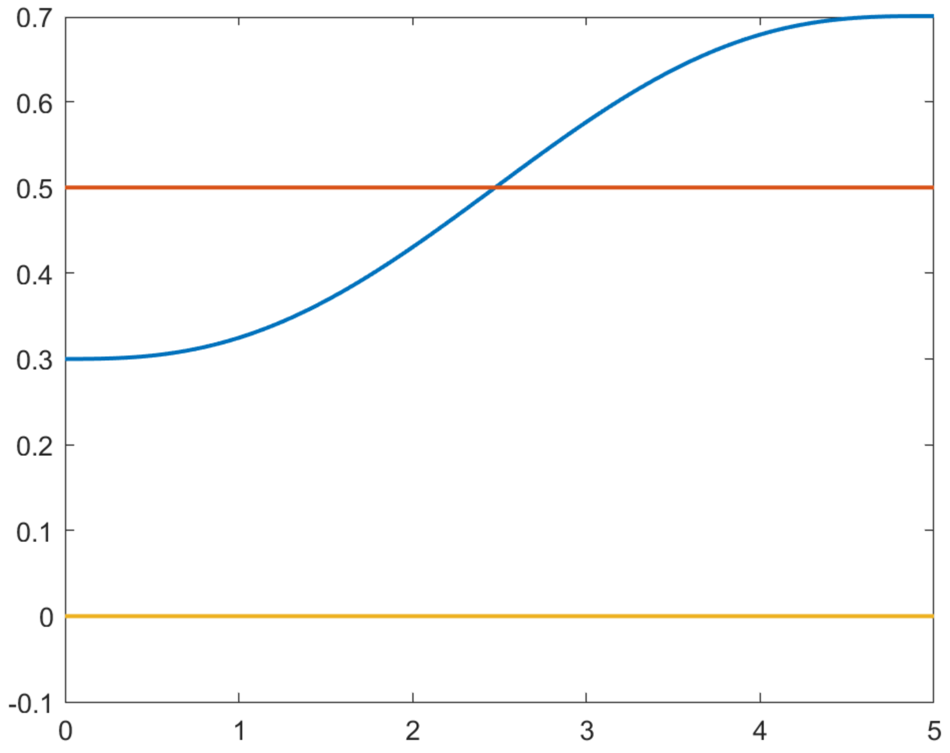
```
for t = 2:length(trajectory.timeVector)
```

The main step within the time loop is to call the solver to resolve the forward kinematics:

```
    [fd_solution_q(:,t), fd_solution_q_dot(:,t), fd_solution_q_ddot(:,t), cdpr_model] =
        fd_solver.compute(fd_solution_q(:, t-1), fd_solution_q_dot(:, t-1), id_sim.cab
end
```

The results can be plotted the way you desire yourself.

```
figure;
plot(trajectory.timeVector, fd_solution_q, 'LineWidth', 1.5);
```



0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
0
1
2
3
4
5

Tutorial 7: Control Simulations in CASPR

This tutorial will show how a simulation for testing controllers can be ran in CASPR for cable-driven robots. This tutorial will use the ideas from the following previous tutorials:

- `T1_load_robot.mlx`
- `T2_load_trajectory.mlx`
- `T5_inverse_dynamics.mlx`
- `T6_forward_dynamics.mlx`

This tutorial shows how to run controllers through two approaches:

1. ControllerSimulator
2. Writing your own time loop

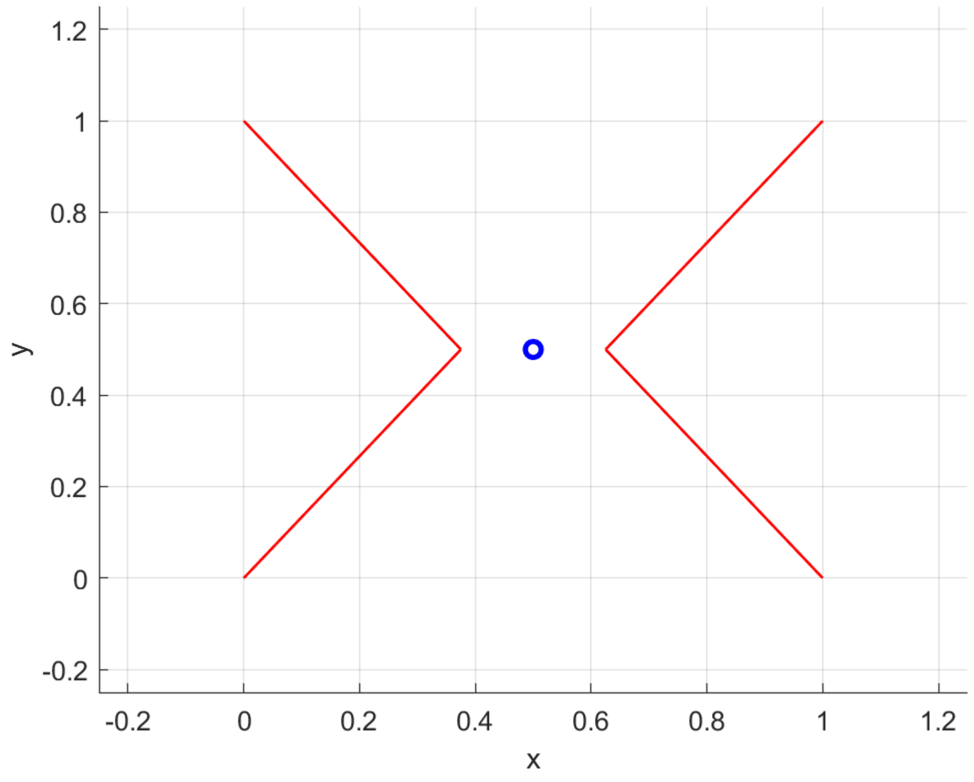
```
clear;
```

Setup (required for both approaches)

Load the robot and trajectory

Load a robot as shown in `T1_load_robot.mlx` and `T2_load_trajectory.mlx`

```
model_config = ModelConfig('Example planar XY');
cdpr_model = model_config.getModel('basic');
trajectory_ref = model_config.getJointTrajectory('example_quintic');
MotionSimulatorBase.PlotFrame(cdpr_model, model_config.displayRange, model_config.view
```



Create InverseDynamics solver

The inverse dynamics solver is required to resolve the cable force commands to be executed on the robot. In this example, a simple quadratic program (QP) solver will be used (refer to `T5_inverse_dynamics.mlx`).

```
id_objective = IDObjectiveMinQuadCableForce(ones(cdpr_model.numCables,1));
id_solver = IDSolverQuadProg(cdpr_model, id_objective, ID_QP_SolverType.MATLAB);
```

Create ForwardDynamics solver

This is needed in order to simulate the effects of the controller on a robot model. In practise, the forward dynamics would be replaced by a real robot.

```
fd_solver = ForwardDynamics(FDSolverType.ODE113);
```

Setup controller

Since there are many different methods to solve for the control problem, different controller can be setup to be used in CASPR. For the sake of example, the most common computed torque controller (CTC) with a quadratic program (QP) inverse dynamics solver will be shown in this example:

```
Kp =25 ;
Kd =10 ; % Suggested to be Kd = 2*sqrt(Kp) for critical damping
Kp_gains = Kp*eye(cdpr_model.numDofs); % Note that the same Kp gains will be applied fo
Kd_gains = Kd*eye(cdpr_model.numDofs); % Note that the same Kd gains will be applied fo
controller = ComputedTorqueController(cdpr_model, id_solver, Kp_gains, Kd_gains);
```

Setup initial conditions

Set the initial conditions for the scenario

```
x_initial_error = -0.1;
y_initial_error = 0.2;
theta_initial_error = 0.1;
q_init = trajectory_ref.q{1} + [x_initial_error; y_initial_error; theta_initial_error]
q_dot_init = trajectory_ref.q_dot{1};
q_ddot_init = trajectory_ref.q_ddot{1};
```

1) Solve using the ControllerSimulator

The ForwardKinematicsSimulator provides the simplest way to run a "standard" forward kinematics simulation for a defined trajectory.

First, create the simulator object using the robot model and the specified solver:

```
control_sim = ControllerSimulator(cdpr_model, controller, fd_solver, [], [], [], [], [
```

```
Default options will be applied.
```

Next, run the simulator on the resulting cable lengths and cable lengths velocity over the specified time points with the initial joint pose and velocity:

```
control_sim.run(trajectory_ref, q_init, q_dot_init, q_ddot_init);
```

```
[INFO] Time : 0.000000
[INFO] Completion Percentage: 0.99%
[INFO] Time : 0.050000
[INFO] Completion Percentage: 1.98%
[INFO] Time : 0.100000
[INFO] Completion Percentage: 2.97%
[INFO] Time : 0.150000
[INFO] Completion Percentage: 3.96%
[INFO] Time : 0.200000
[INFO] Completion Percentage: 4.95%
[INFO] Time : 0.250000
[INFO] Completion Percentage: 5.94%
[INFO] Time : 0.300000
[INFO] Completion Percentage: 6.93%
[INFO] Time : 0.350000
[INFO] Completion Percentage: 7.92%
[INFO] Time : 0.400000
[INFO] Completion Percentage: 8.91%
[INFO] Time : 0.450000
[INFO] Completion Percentage: 9.90%
[INFO] Time : 0.500000
[INFO] Completion Percentage: 10.89%
[INFO] Time : 0.550000
[INFO] Completion Percentage: 11.88%
[INFO] Time : 0.600000
[INFO] Completion Percentage: 12.87%
[INFO] Time : 0.650000
[INFO] Completion Percentage: 13.86%
[INFO] Time : 0.700000
[INFO] Completion Percentage: 14.85%
[INFO] Time : 0.750000
[INFO] Completion Percentage: 15.84%
```

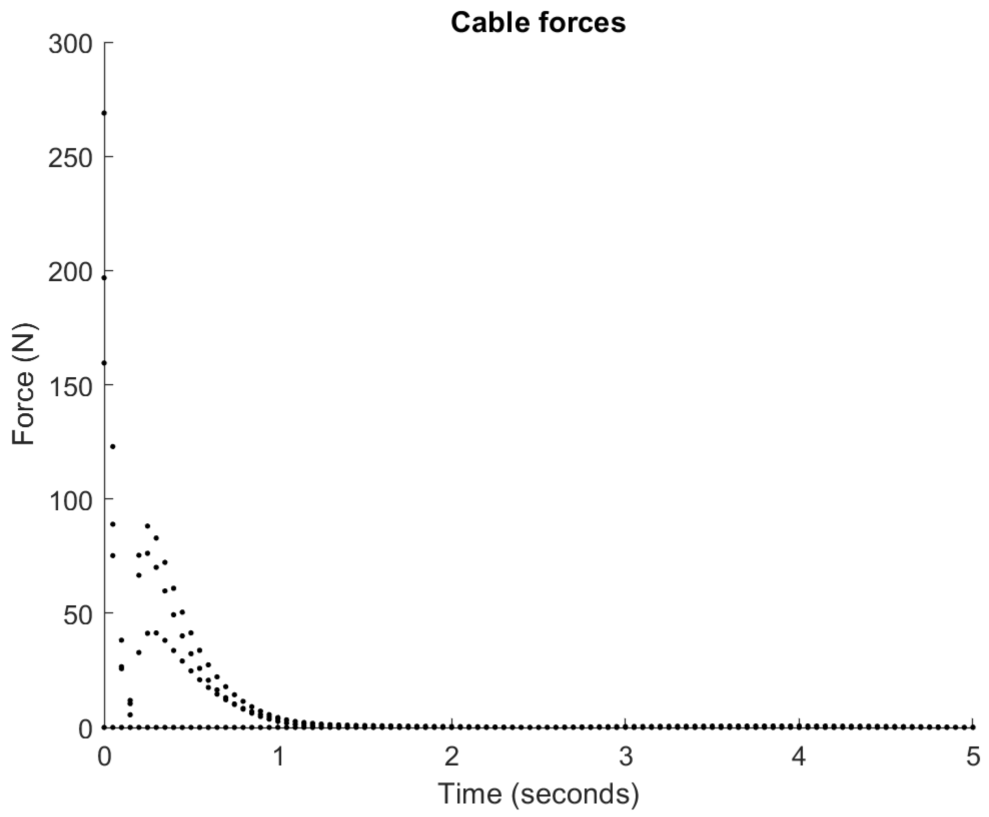
```
[INFO] Time : 0.800000
[INFO] Completion Percentage: 16.83%
[INFO] Time : 0.850000
[INFO] Completion Percentage: 17.82%
[INFO] Time : 0.900000
[INFO] Completion Percentage: 18.81%
[INFO] Time : 0.950000
[INFO] Completion Percentage: 19.80%
[INFO] Time : 1.000000
[INFO] Completion Percentage: 20.79%
[INFO] Time : 1.050000
[INFO] Completion Percentage: 21.78%
[INFO] Time : 1.100000
[INFO] Completion Percentage: 22.77%
[INFO] Time : 1.150000
[INFO] Completion Percentage: 23.76%
[INFO] Time : 1.200000
[INFO] Completion Percentage: 24.75%
[INFO] Time : 1.250000
[INFO] Completion Percentage: 25.74%
[INFO] Time : 1.300000
[INFO] Completion Percentage: 26.73%
[INFO] Time : 1.350000
[INFO] Completion Percentage: 27.72%
[INFO] Time : 1.400000
[INFO] Completion Percentage: 28.71%
[INFO] Time : 1.450000
[INFO] Completion Percentage: 29.70%
[INFO] Time : 1.500000
[INFO] Completion Percentage: 30.69%
[INFO] Time : 1.550000
[INFO] Completion Percentage: 31.68%
[INFO] Time : 1.600000
[INFO] Completion Percentage: 32.67%
[INFO] Time : 1.650000
[INFO] Completion Percentage: 33.66%
[INFO] Time : 1.700000
[INFO] Completion Percentage: 34.65%
[INFO] Time : 1.750000
[INFO] Completion Percentage: 35.64%
[INFO] Time : 1.800000
[INFO] Completion Percentage: 36.63%
[INFO] Time : 1.850000
[INFO] Completion Percentage: 37.62%
[INFO] Time : 1.900000
[INFO] Completion Percentage: 38.61%
[INFO] Time : 1.950000
[INFO] Completion Percentage: 39.60%
[INFO] Time : 2.000000
[INFO] Completion Percentage: 40.59%
[INFO] Time : 2.050000
[INFO] Completion Percentage: 41.58%
[INFO] Time : 2.100000
[INFO] Completion Percentage: 42.57%
[INFO] Time : 2.150000
[INFO] Completion Percentage: 43.56%
[INFO] Time : 2.200000
[INFO] Completion Percentage: 44.55%
[INFO] Time : 2.250000
[INFO] Completion Percentage: 45.54%
[INFO] Time : 2.300000
[INFO] Completion Percentage: 46.53%
[INFO] Time : 2.350000
[INFO] Completion Percentage: 47.52%
```

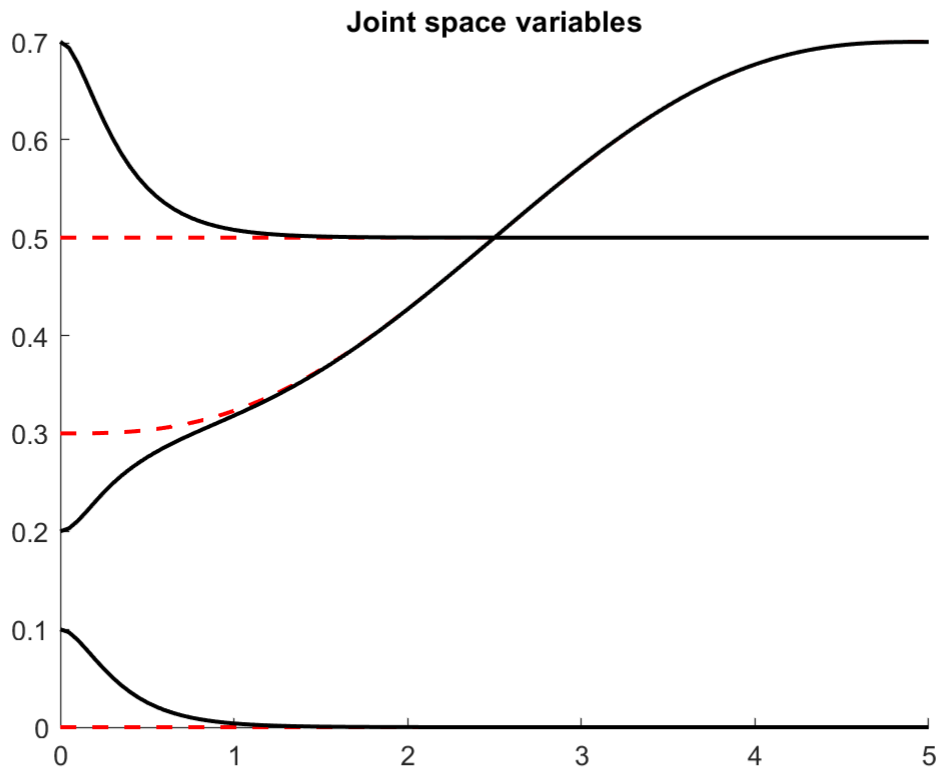
```
[INFO] Time : 2.400000
[INFO] Completion Percentage: 48.51%
[INFO] Time : 2.450000
[INFO] Completion Percentage: 49.50%
[INFO] Time : 2.500000
[INFO] Completion Percentage: 50.50%
[INFO] Time : 2.550000
[INFO] Completion Percentage: 51.49%
[INFO] Time : 2.600000
[INFO] Completion Percentage: 52.48%
[INFO] Time : 2.650000
[INFO] Completion Percentage: 53.47%
[INFO] Time : 2.700000
[INFO] Completion Percentage: 54.46%
[INFO] Time : 2.750000
[INFO] Completion Percentage: 55.45%
[INFO] Time : 2.800000
[INFO] Completion Percentage: 56.44%
[INFO] Time : 2.850000
[INFO] Completion Percentage: 57.43%
[INFO] Time : 2.900000
[INFO] Completion Percentage: 58.42%
[INFO] Time : 2.950000
[INFO] Completion Percentage: 59.41%
[INFO] Time : 3.000000
[INFO] Completion Percentage: 60.40%
[INFO] Time : 3.050000
[INFO] Completion Percentage: 61.39%
[INFO] Time : 3.100000
[INFO] Completion Percentage: 62.38%
[INFO] Time : 3.150000
[INFO] Completion Percentage: 63.37%
[INFO] Time : 3.200000
[INFO] Completion Percentage: 64.36%
[INFO] Time : 3.250000
[INFO] Completion Percentage: 65.35%
[INFO] Time : 3.300000
[INFO] Completion Percentage: 66.34%
[INFO] Time : 3.350000
[INFO] Completion Percentage: 67.33%
[INFO] Time : 3.400000
[INFO] Completion Percentage: 68.32%
[INFO] Time : 3.450000
[INFO] Completion Percentage: 69.31%
[INFO] Time : 3.500000
[INFO] Completion Percentage: 70.30%
[INFO] Time : 3.550000
[INFO] Completion Percentage: 71.29%
[INFO] Time : 3.600000
[INFO] Completion Percentage: 72.28%
[INFO] Time : 3.650000
[INFO] Completion Percentage: 73.27%
[INFO] Time : 3.700000
[INFO] Completion Percentage: 74.26%
[INFO] Time : 3.750000
[INFO] Completion Percentage: 75.25%
[INFO] Time : 3.800000
[INFO] Completion Percentage: 76.24%
[INFO] Time : 3.850000
[INFO] Completion Percentage: 77.23%
[INFO] Time : 3.900000
[INFO] Completion Percentage: 78.22%
[INFO] Time : 3.950000
[INFO] Completion Percentage: 79.21%
```

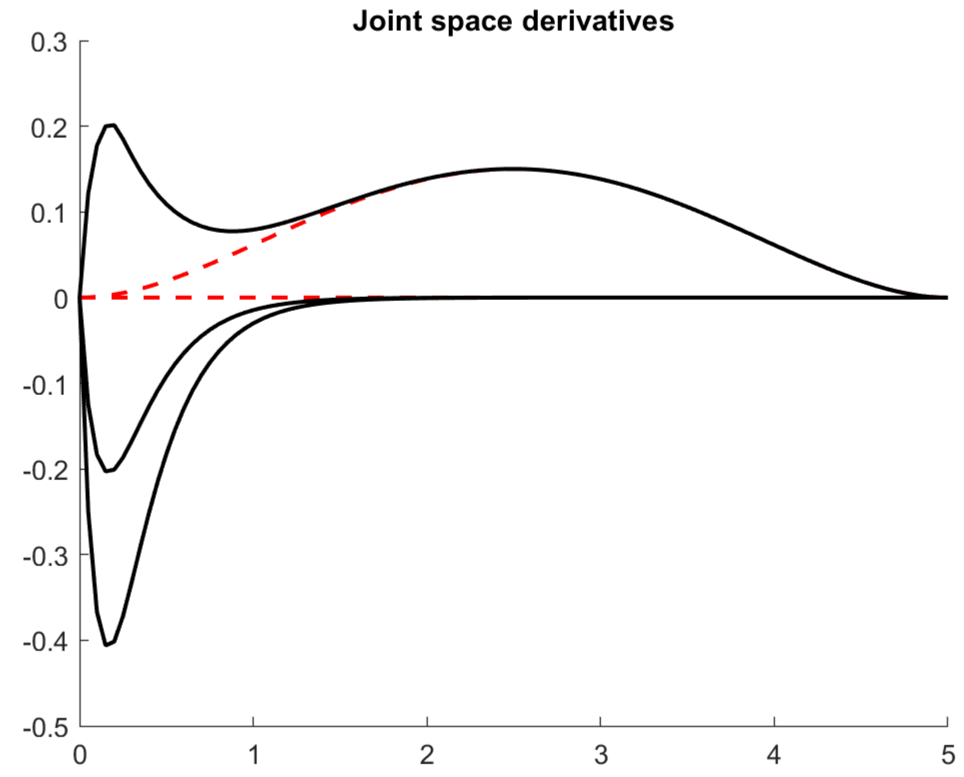
```
[INFO] Time : 4.000000
[INFO] Completion Percentage: 80.20%
[INFO] Time : 4.050000
[INFO] Completion Percentage: 81.19%
[INFO] Time : 4.100000
[INFO] Completion Percentage: 82.18%
[INFO] Time : 4.150000
[INFO] Completion Percentage: 83.17%
[INFO] Time : 4.200000
[INFO] Completion Percentage: 84.16%
[INFO] Time : 4.250000
[INFO] Completion Percentage: 85.15%
[INFO] Time : 4.300000
[INFO] Completion Percentage: 86.14%
[INFO] Time : 4.350000
[INFO] Completion Percentage: 87.13%
[INFO] Time : 4.400000
[INFO] Completion Percentage: 88.12%
[INFO] Time : 4.450000
[INFO] Completion Percentage: 89.11%
[INFO] Time : 4.500000
[INFO] Completion Percentage: 90.10%
[INFO] Time : 4.550000
[INFO] Completion Percentage: 91.09%
[INFO] Time : 4.600000
[INFO] Completion Percentage: 92.08%
[INFO] Time : 4.650000
[INFO] Completion Percentage: 93.07%
[INFO] Time : 4.700000
[INFO] Completion Percentage: 94.06%
[INFO] Time : 4.750000
[INFO] Completion Percentage: 95.05%
[INFO] Time : 4.800000
[INFO] Completion Percentage: 96.04%
[INFO] Time : 4.850000
[INFO] Completion Percentage: 97.03%
[INFO] Time : 4.900000
[INFO] Completion Percentage: 98.02%
[INFO] Time : 4.950000
[INFO] Completion Percentage: 99.01%
[INFO] Time : 5.000000
[INFO] Completion Percentage: 100.00%
```

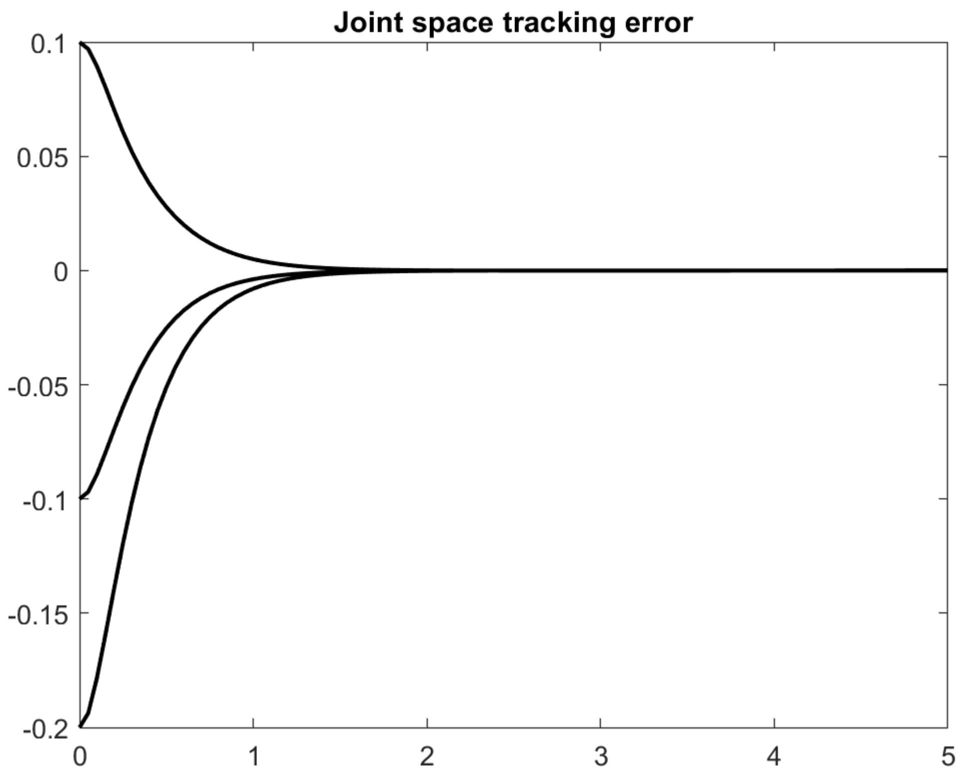
The forward dynamics simulator can plot the resulting joint pose

```
if control_sim.controllerExitType == ControllerExitType.NO_ERROR
    control_sim.plotCableForces();
    control_sim.plotJointSpaceTracking();
    control_sim.plotJointTrackingError();
else
    CASPR_log.Info('The controller simulation did not complete and hence nothing will
end
```


Cable forces
Force (N)
Time (seconds)
300
250
200
150
100
50
0
0
1
2
3
4
5


Joint space variables
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1
2
3
4
5





2) Writing your own time loop

If you want more control over what is happening or customise the inner-workings during the simulation, it is recommended for you to write your own time loop. The following simple example is basically what happens within the `ControllerSimulator`.

First, create the variables (pre-allocate memory) required to store the solution from the inverse kinematics (cable lengths and cable lengths velocity):

```
q_soln = zeros(cdpr_model.numDofs, length(trajectory_ref.timeVector)); % Each column i
q_dot_soln = zeros(cdpr_model.numDofs, length(trajectory_ref.timeVector)); % Each colum
q_ddot_soln = zeros(cdpr_model.numDofs, length(trajectory_ref.timeVector)); % Each col
force_soln = zeros(cdpr_model.numActuators, length(trajectory_ref.timeVector)); % Each
```

Setup some initial variables for the joint space:

```
q_soln(:, 1) = q_init;
q_dot_soln(:, 1) = q_dot_init;
q_ddot_soln(:, 1) = q_ddot_init;
```

Next, loop through the trajectory as a function of time:

```
for t = 1:length(trajectory_ref.timeVector)-1
```

Execute the controller with the current joint states and reference trajectory (note that zero disturbance is applied) to determine the cable force command:

```
    [force_soln(:, t), ~, ~, controller_exit_type] = controller.execute(q_soln(:, t), 
```

If controller exit type is not ControllerExitType.NO_ERROR, then it means that the controller didn't run to completion and hence should quit from the time loop.

```
    if controller_exit_type ~= ControllerExitType.NO_ERROR
        CASPR_log.Info('The controller simulation did not complete and hence nothing w
        break;
    end
```

And if controller was successful, then execute the forward dynamics with the cable force solution from the controller to determine the next joint states (for the next iteration):

```
    [q_soln(:,t+1), q_dot_soln(:,t+1), q_ddot_soln(:,t+1), cdpr_model] = ...
        fd_solver.compute(q_soln(:, t), q_dot_soln(:, t), force_soln(:, t), cdpr_model
end
```

The results can be plotted the way you desire yourself if there was no error. Starting with the reference and solution joint trajectory:

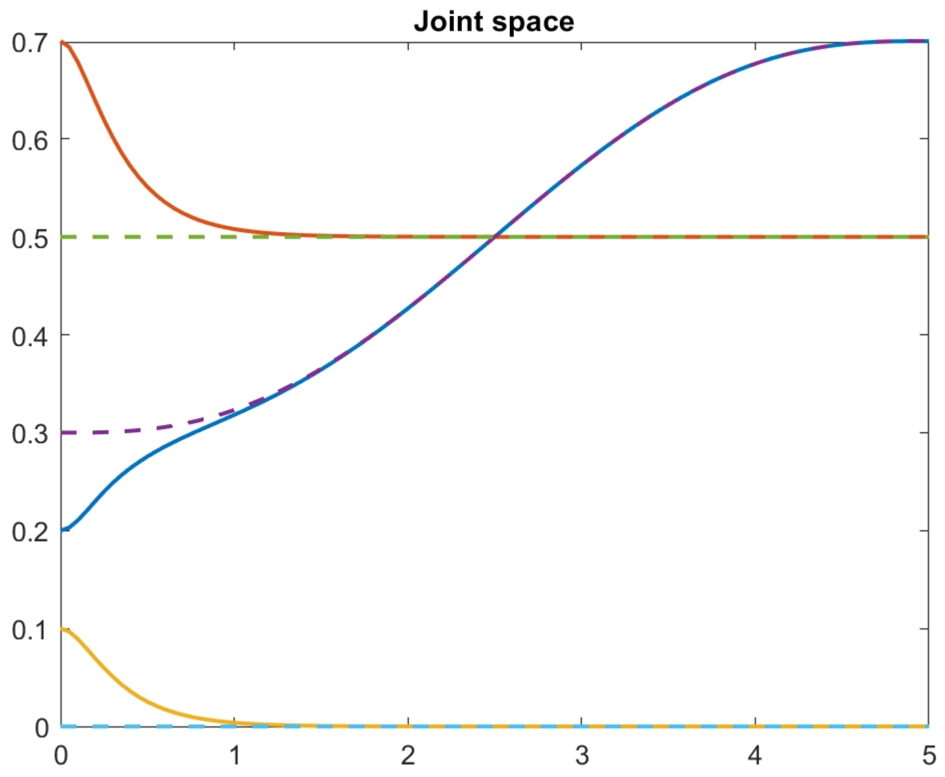
```
if controller_exit_type == ControllerExitType.NO_ERROR
    figure;
    plot(trajectory_ref.timeVector, q_soln, 'LineWidth', 1.5);
    hold on;
    plot(trajectory_ref.timeVector, cell2mat(trajectory_ref.q), '--', 'LineWidth', 1.5
    hold off;
    title('Joint space');
```

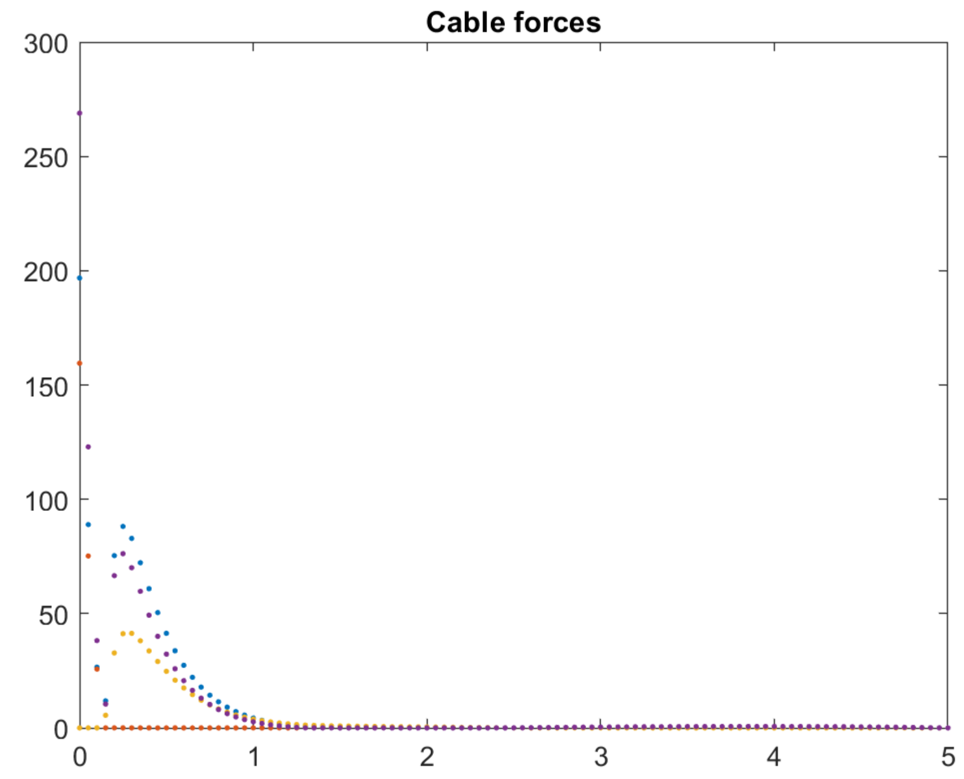
Then the cable force solution from the controller:

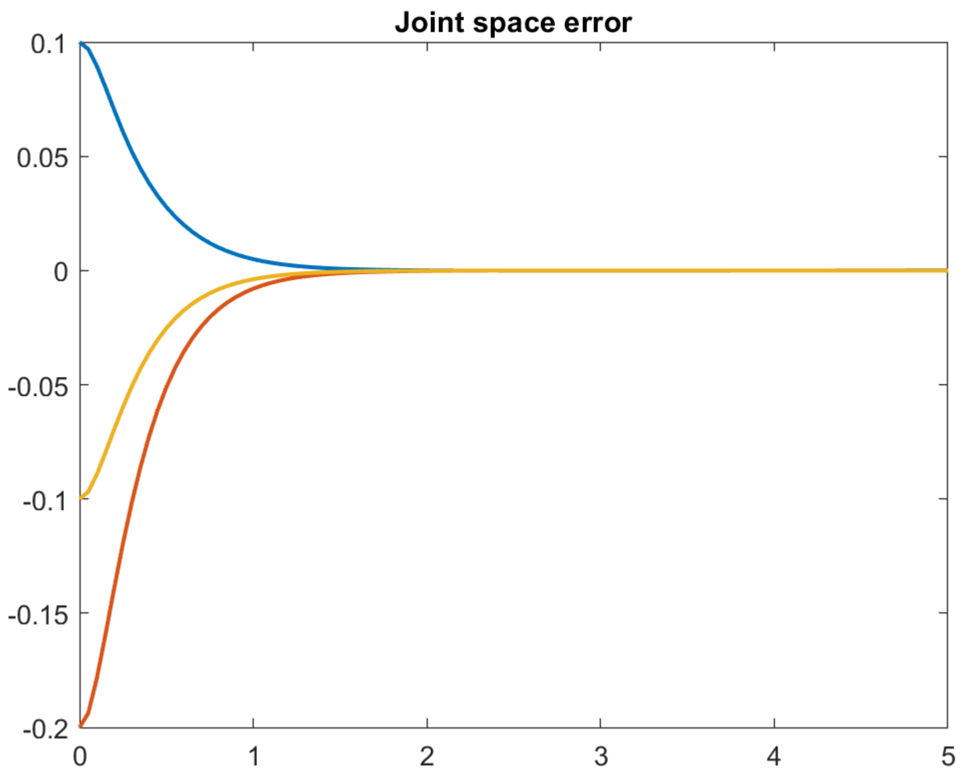
```
    figure;
    plot(trajectory_ref.timeVector, force_soln, '.', 'LineWidth', 1.5);
    title('Cable forces');
```

Finally the tracking error:

```
    figure;
    plot(trajectory_ref.timeVector, cell2mat(trajectory_ref.q) - q_soln, 'LineWidth', 
    title('Joint space error');
end
```




Cable forces
300
250
200
150
100
50
0
0
1
2
3
4
5


Joint space error
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
1
2
3
4
5