

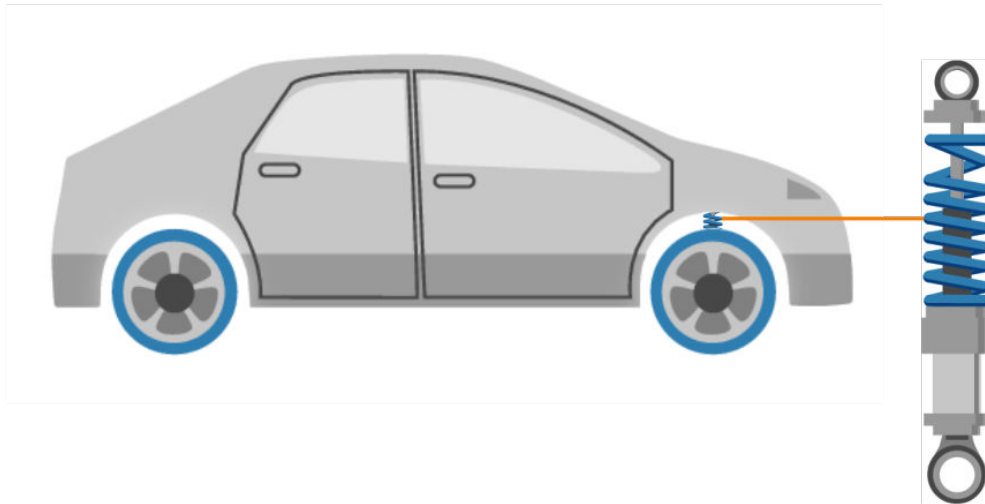
Dynamics

22320630

Zhu Chenhao

How do you design a car to ensure a smooth ride?

Many parts of a vehicle contribute to ensuring a smooth ride for passengers. The suspension system contains key components: tires, shock absorbers, and linkages. Together, these cushion the vehicle's response to bumps and dips in the road.



A shock absorber helps to provide a smooth ride and reduces wear on vehicle components

Shock absorbers use hydraulic fluid to dissipate mechanical energy, reducing the vehicle's response to a shock. How can you choose the right shock damping to smooth out the ride while preserving vehicle control? The answer lies in dynamic modeling. In this live script, you will

1. examine the mass-spring-damper model,
2. solve for its solutions in Simulink, and
3. apply it to tune the damping of a suspension system to meet design requirements.

The Mass-Spring-Damper

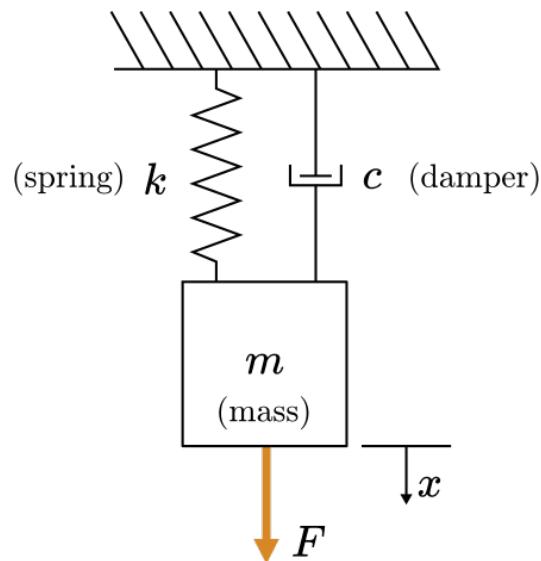
Table of Contents

Dynamics.....	1
22320630.....	1
Zhu Chenhao.....	1
How do you design a car to ensure a smooth ride?.....	1
The Mass-Spring-Damper.....	1
1. The mass-spring-damper model.....	2
Modeling vehicle suspension as a mass-spring-damper.....	3
YOUR ANSWER.....	5
2. Manipulate the mass-spring-damper equation using symbolic operations.....	6
3. Solve a first-order ODE in Simulink.....	7
4. Solve the mass-spring-damper model in Simulink.....	8
5. Tune suspension damping using the mass-spring-damper model.....	11

```
clear % This clears the MATLAB workspace each time you run the script
addModelPaths % This helper adds the model paths
```

1. The mass-spring-damper model

The mass-spring-damper model is a simple idealization that can be applied to a wide variety of vibrating physical systems. As the name suggests, the mass-spring-damper system contains three idealized components: a mass, a spring, and a damper.



Schematic of the mass-spring-damper system

Several parameters determine the behavior of this mechanical system:

- k : spring stiffness [N/m]
- c : damping coefficient [N · s/m]
- m : mass [kg]

- F : externally applied force [N] (can be constant or time-dependent: $F(t)$)

The time-dependent behavior of the system is solved for using the variables:

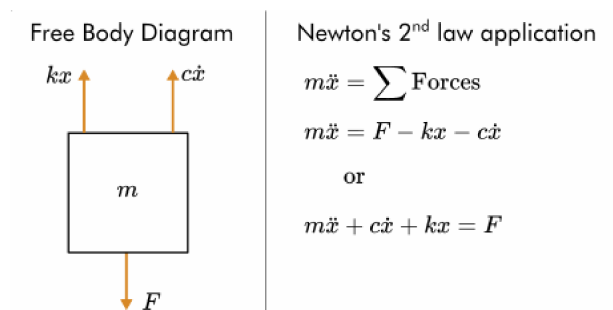
- t : time since the start of the simulation [s]
- x : time-dependent ($x = x(t)$) displacement of the mass from the reference equilibrium position [m]
- \dot{x} : mass velocity [m/s]
- \ddot{x} : mass acceleration [m/s²]

When a force is applied to the system, the mass will begin to vibrate. Over time, the vibration diminishes as mechanical energy is converted to heat by the dashpot.

- draw a free body diagram of the forces acting on the mass m .
- After drawing the free body diagram, apply Newton's second law to derive the equation of motion:

$$m\ddot{x} = \sum \text{Forces}.$$

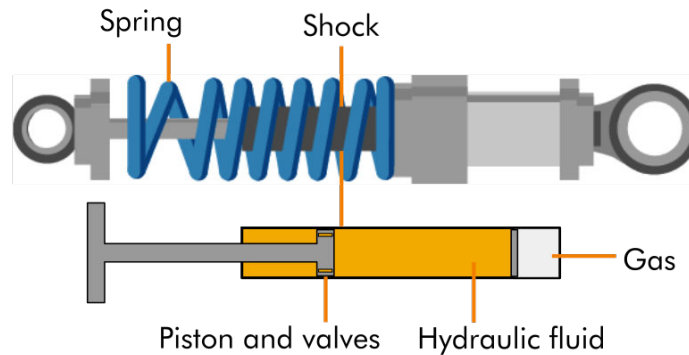
```
showSolutionFBD = true; % Click the checkbox to view the solution
if(showSolutionFBD)
    I = imread("images/mass_spring_damper_FBD.png");
    imshow(I);
end
```



Modeling vehicle suspension as a mass-spring-damper

The mass-spring-damper is suitable for modeling a shock absorber as the components of the model correspond directly to the mechanical components. The model spring and damper represent the physical spring and shock, respectively. These two components serve complementary functions in bringing the vehicle smoothly back to equilibrium.

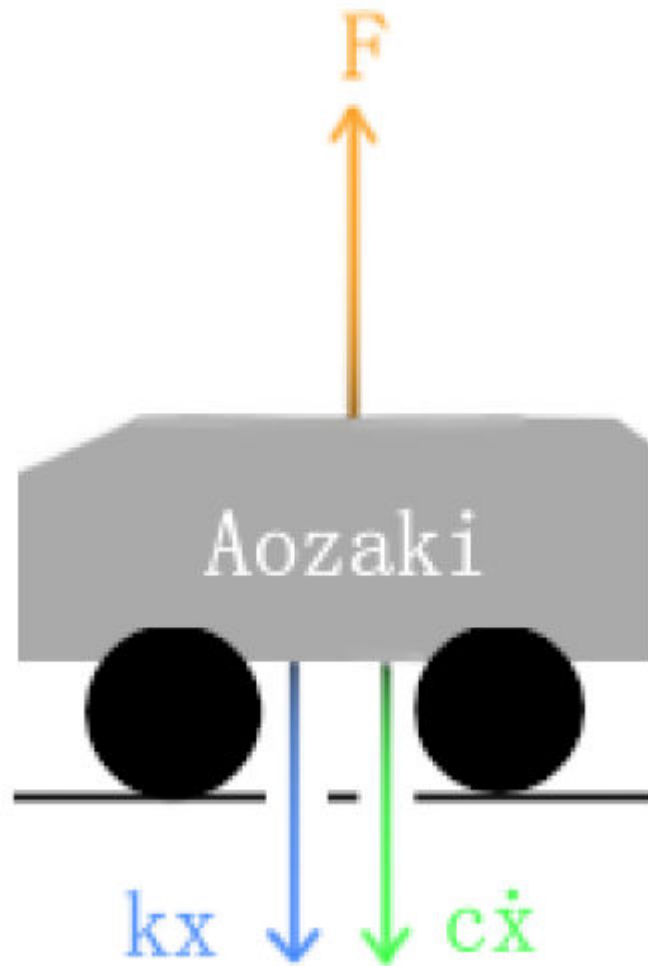
1. The shock component dissipates mechanical energy, which is given off as heat. This reduces the overall motion of the vehicle after contacting a bump.
2. The spring spreads the response to an impact out over time. The initial impact compresses the spring, followed by regular oscillations. These oscillations give time for the shock to dissipate mechanical energy.



Simplified schematic of a shock/spring system. Internally, the shock has a piston that moves through hydraulic fluid. The viscous resistance of the fluid dissipates mechanical energy as heat.

In the simple idealization used in this script, the vehicle's entire suspension system (rather than a single shock absorber) is modeled as a single mass-spring-damper with stiffness, k , and damping coefficient, c . The mass, m , is the mass of the vehicle itself. Drive over a pothole and you've applied an impulse force, F .

- Try drawing a free body diagram for a vehicle with a suspension described by a single mass-spring-damper system that has driven over a speed bump.



- What major features are left out of the model by assuming a single mass-spring-damper system for the entire suspension?

While the single mass-spring-damper model provides a useful starting point for understanding basic suspension dynamics, it oversimplifies the real-world behavior of a vehicle's suspension system. To capture more realistic behavior, additional degrees of freedom, nonlinearities, and subsystem interactions(e.g., tires, suspension geometry, and adaptive components) must be incorporated into the model. Advanced models often use multi-body dynamics simulations to account for these complexities.


YOUR ANSWER

2. Manipulate the mass-spring-damper equation using symbolic operations

The mass-spring-damper differential equation that you derived in the previous section is

$$m\ddot{x} + c\dot{x} + kx = F.$$

When solving an ordinary differential equation (ODE) in Simulink, it helps to have it in the form: $\ddot{x} = \dots$. This form makes constructing the solution model simpler. Only two simple algebraic steps are required to solve for \ddot{x} here. But, in more realistic model systems, there are typically several equations that need to be solved simultaneously, making the algebra tedious and error-prone. To prepare yourself for those problems, you can use symbolic computations to solve for \ddot{x} in the mass-spring-damper ODE.

 **Exercise.** In this exercise, you will define the equation in symbolic notation and then solve for \ddot{x} .

Task 1. Solve for \ddot{x} on paper.

$$\begin{aligned} m\ddot{x} + c\dot{x} + kx &= F \\ m\ddot{x} &= F - c\dot{x} - kx \\ \ddot{x} &= \frac{F - c\dot{x} - kx}{m} \end{aligned}$$

Task 2. Define seven symbolic variables in the space provided: x , \dot{x} , \ddot{x} , m , c , k , and F . You can define multiple symbolic variables in MATLAB using the command syntax:

```
syms x y
```

```
% Call the syms command here
syms x x_dot x_ddot m c k F
checkSyms() % This checks if you defined the symbolic variables
```

```
Success! x is symbolic.
Success! x_dot is symbolic.
Success! x_ddot is symbolic.
Success! m is symbolic.
Success! c is symbolic.
Success! k is symbolic.
Success! F is symbolic.
```

Task 3. Write the mass-spring-damper equation in terms of symbolic variables and assign it to msdEq.

When you write a symbolic equation, you assign the whole equation to a MATLAB variable using $=$. This means that the equals sign in the symbolic equation must be written using $==$. For example:

```
syms y m x b
linearEq = y == m*x + b
```

```
msdEq = m*x_ddot + c*x_dot + k*x == F; % Replace NaN with your equation
checkMSDeq(msdEq) % This helper function (located at the bottom of the script) checks your equation
```

Correct!

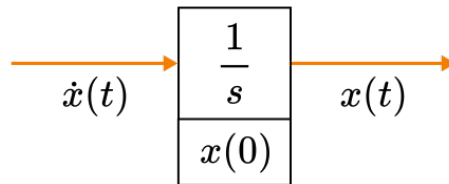
Task 4. Solve for x_{ddot} in the mass-spring-damper equation using `solve(equation, variable)` and assign the result to `sltnMSD`. In this syntax, `equation` is the symbolic equation to be solved and `variable` is the symbolic variable to be solved for.

```
sltnMSD = solve(msdEq, x_ddot); % Replace NaN with a call to solve
checkMSDSolve(sltnMSD) % This helper function (located at the bottom of the script) checks your
```

Correct!

3. Solve a first-order ODE in Simulink

To investigate the dynamics of the mass-spring-damper, you can solve for its solutions in Simulink. Before you solve the second-order mass-spring-damper ODE, you can solve a simpler first-order ODE. In Simulink, ODEs are solved using integrator blocks.



Simplified schematic for solving a first-order ODE in Simulink. The integrator block (labeled $1/s$) integrates once, reducing the derivative order by one. The integrator block also provides the initial condition, $x(0)$.

 **Exercise.** Consider the ODE:

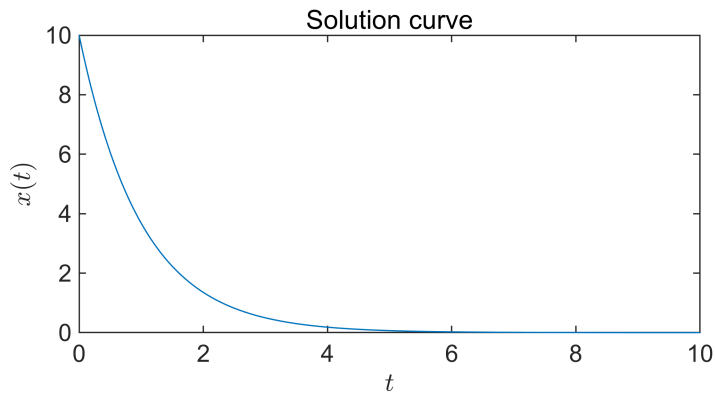
$$\dot{x} = -x$$

subject to $x(0) = 10$

(a) Solve for the analytic solution by hand. Type your solution below in terms of t .

```
syms t;
xSltn = 10*exp(-t); % Replace NaN with your solution in terms of t. Use exp for the exponential
checkFirstOrder(xSltn)
```

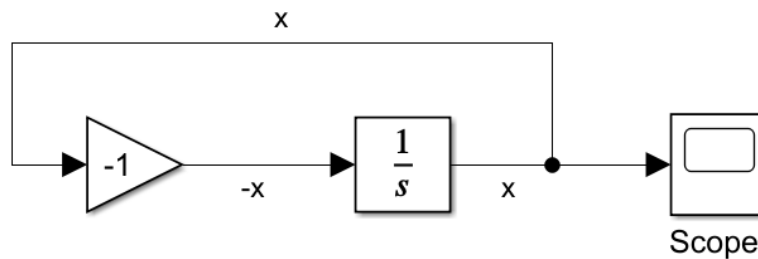
Correct!



(b) build solution model shown below by performing the following:

1. Add an **integrator** block and set the initial condition of the integrator block to $x(0) = 10$ (double-click the block to access the dialogue).
2. Add a **gain** block with value -1.
3. Add a **scope** block.
4. Connect as shown.
5. Run the model, and double click the scope to verify that the output matches the solution found in (a).

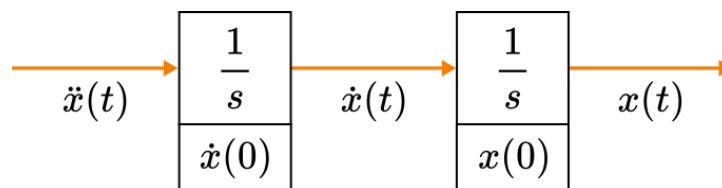
Double-click anywhere in the Simulink canvas and type the name of the block you want to add.



Simulink ODE solver for $\dot{x} = -x$.

4. Solve the mass-spring-damper model in Simulink

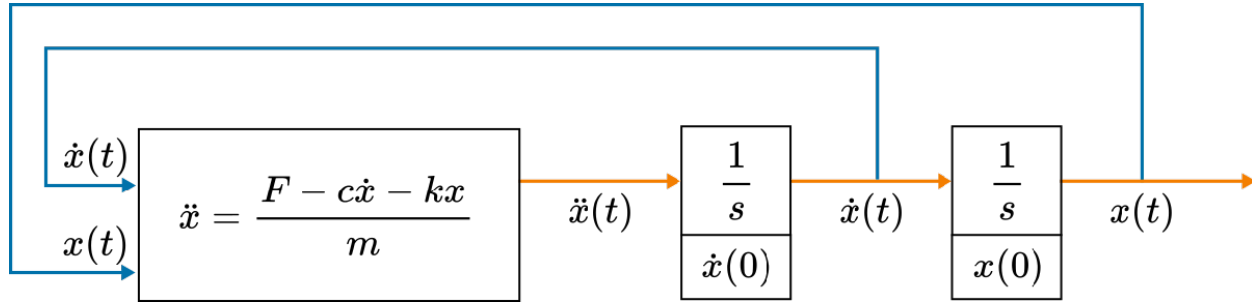
In this section, you will solve the mass-spring-damper 2nd order ODE using Simulink.



Solving a second-order ODE in Simulink requires two integrator blocks.


To provide the input to the first integrator block, you can use the expression for \ddot{x} :

$$\ddot{x} = \frac{F - c\dot{x} - kx}{m}.$$



The inputs (\dot{x} and x) required to compute the expression for \ddot{x} are obtained from the outputs of the integrator blocks.


Earlier, you solved for \ddot{x} in the equations of motion using symbolic computations in MATLAB. You can port this expression directly into Simulink by writing it into a MATLAB Function block.

 **Exercise.** Open the Simulink model [mfb.slx](#) and complete the MATLAB Function block by adding the MATLAB expression for \ddot{x} .

```
yourExpression = char(sltmMSD) % This displays your expression for x_ddot.
```

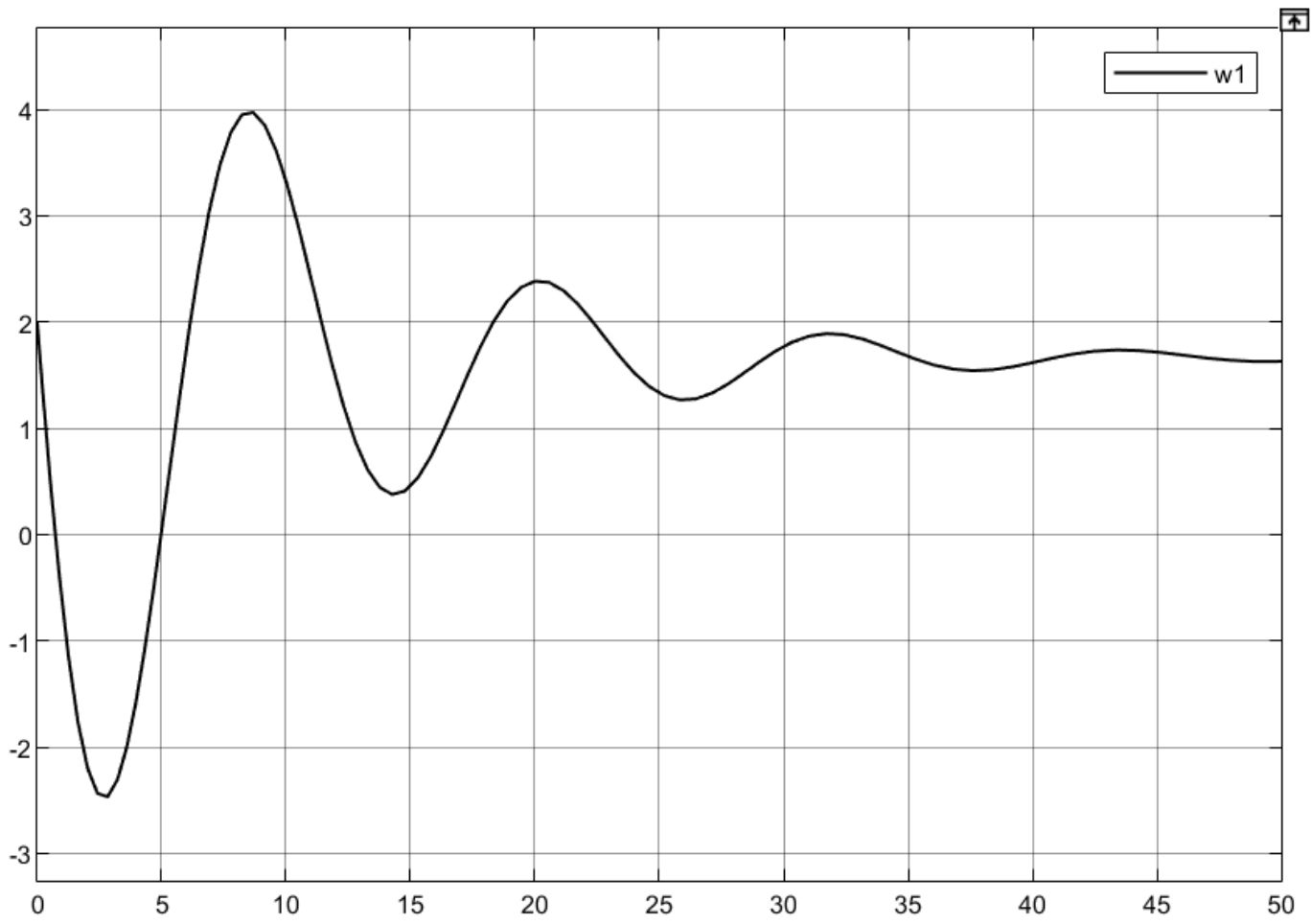
```
yourExpression =  
'-(c*x_dot - F + k*x)/m'
```

- You can use the MATLAB expression that you symbolically computed above for x_ddot inside the MATLAB Function Block.
- Double-click the [MATLAB Function block](#) to edit the MATLAB code inside.
- Run the model to check if your expression is correct.

 **Exercise.** In this exercise, you will complete a model that solves the mass-spring-damper ODE in Simulink.

To compute the solution to this second-order ODE, two integrator blocks are required.

1. Open [msd.slx](#).
2. Add two **Integrator** blocks to complete the system and connect them to the system.
3. Set the initial conditions of the Integrator blocks to $\dot{x}(0) = -3$ and $x(0) = 2$.
4. Run the model and double-click the [Scope](#) to view the output. If it matches the solution shown below, your model is correct.

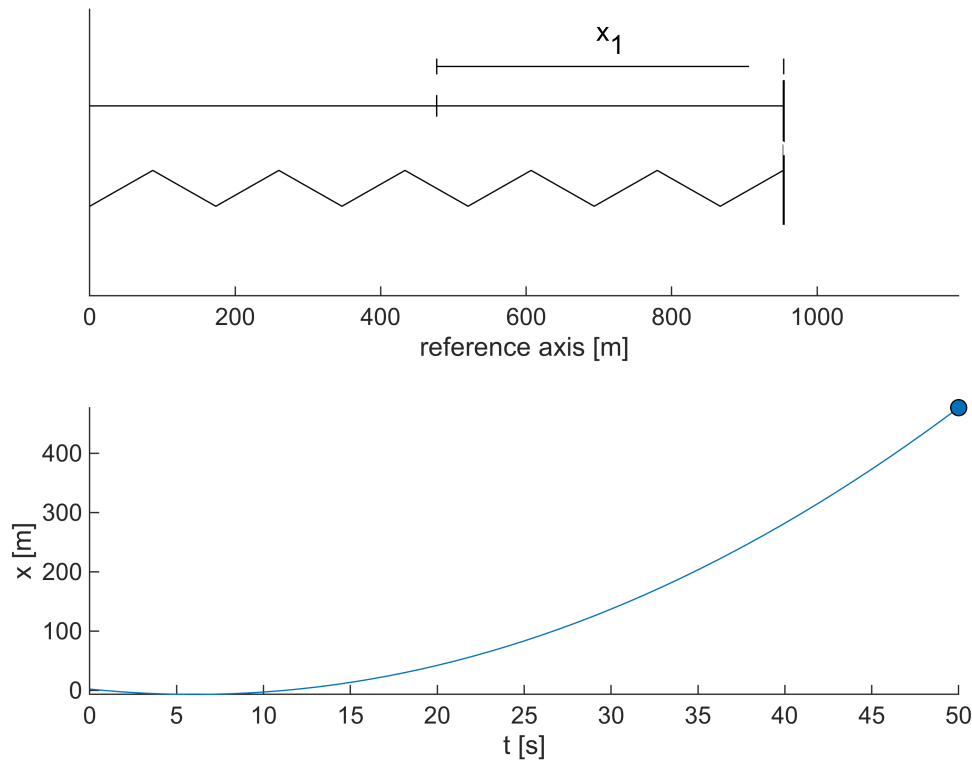


Solution to the mass-spring-damper system

Visualize the motion of the mass-spring-damper

With the model complete, you can visualize the motion. The `sim` command simulates a completed mass-spring-damper model, `msdSimulationOutput.slx`. The local function `animateMSD` creates an animation of the output from the model.

```
n = 200; % The number of steps used in the visualization
visualize = true; % Click the checkbox to show the visualization
if(visualize)
    simOut = sim("models/msd"); % Run the Simulink model
    animateMSD(simOut.tout,simOut.yout,0.5,n)
end
```



Exercise. Open the Simulink model [msd.slx](#) and then try the following.

- Set the damping to 0. Then, rerun this section of the script. How is the behavior affected?

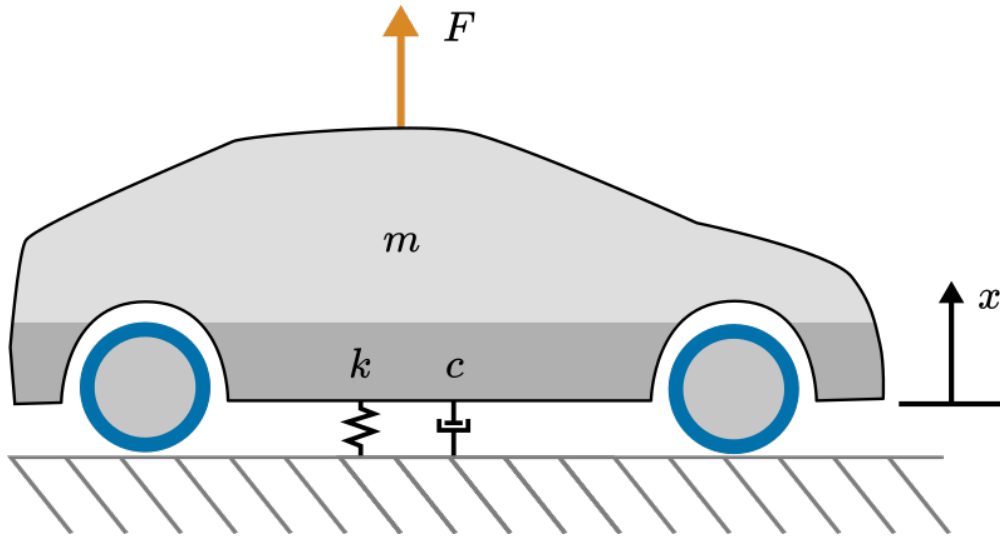
The system oscillates indefinitely without any amplitude decay. The oscillation frequency is determined by the system's stiffness k and mass m . An external force F shifts the equilibrium position, and the system oscillates around this new position.

- Lower the spring stiffness in the model and then rerun the animation here.
- Adjust the initial conditions in the model. First, change the initial position $x(0)$, then change the initial velocity $\dot{x}(0)$. Do these have different or similar effects?

initial Position $x(0)$ and Initial Velocity $\dot{x}(0)$ have different effects on the system's behavior. $x(0)$ primarily affects the amplitude and phase of oscillation, while $\dot{x}(0)$ affects the amplitude through kinetic energy and the initial slope of the oscillation. Both initial conditions contribute to the total energy of the system, but they influence the system's dynamics in distinct ways.

5. Tune suspension damping using the mass-spring-damper model

Now that you've solved the ODE in Simulink, you can apply it to tune the suspension of a vehicle modeled as a single mass-spring-damper.



Schematic of the mass-spring-damper model applied to a vehicle. The entire suspension has a stiffness, k , and damping, c . At rest, the vertical position of the vehicle's undercarriage is $x = 0$.

Your goal in this section is to tune the suspension so that it meets design requirements.

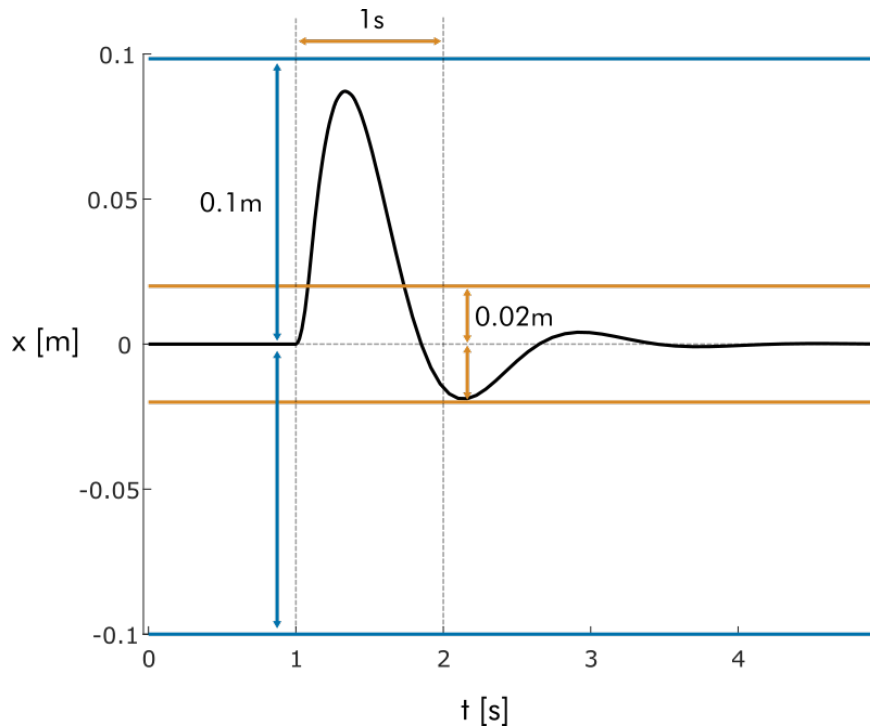
 **Exercise.** Consider a vehicle with the following properties:

- Vehicle mass: 1360 kg
- Suspension stiffness: 26000 N/m


Your task is to select a damping, c , that meets the following design requirements. After encountering a bump in the road, which is modeled by a 9000 N force applied for 0.1 seconds, the vehicle should:

1. have a maximum absolute vertical displacement of less than 0.1 m after impact,
2. after 1 second, the absolute vertical displacement should be less than 0.02 m, and
3. the damping coefficient should be as small as possible while still meeting requirements 1 and 2.

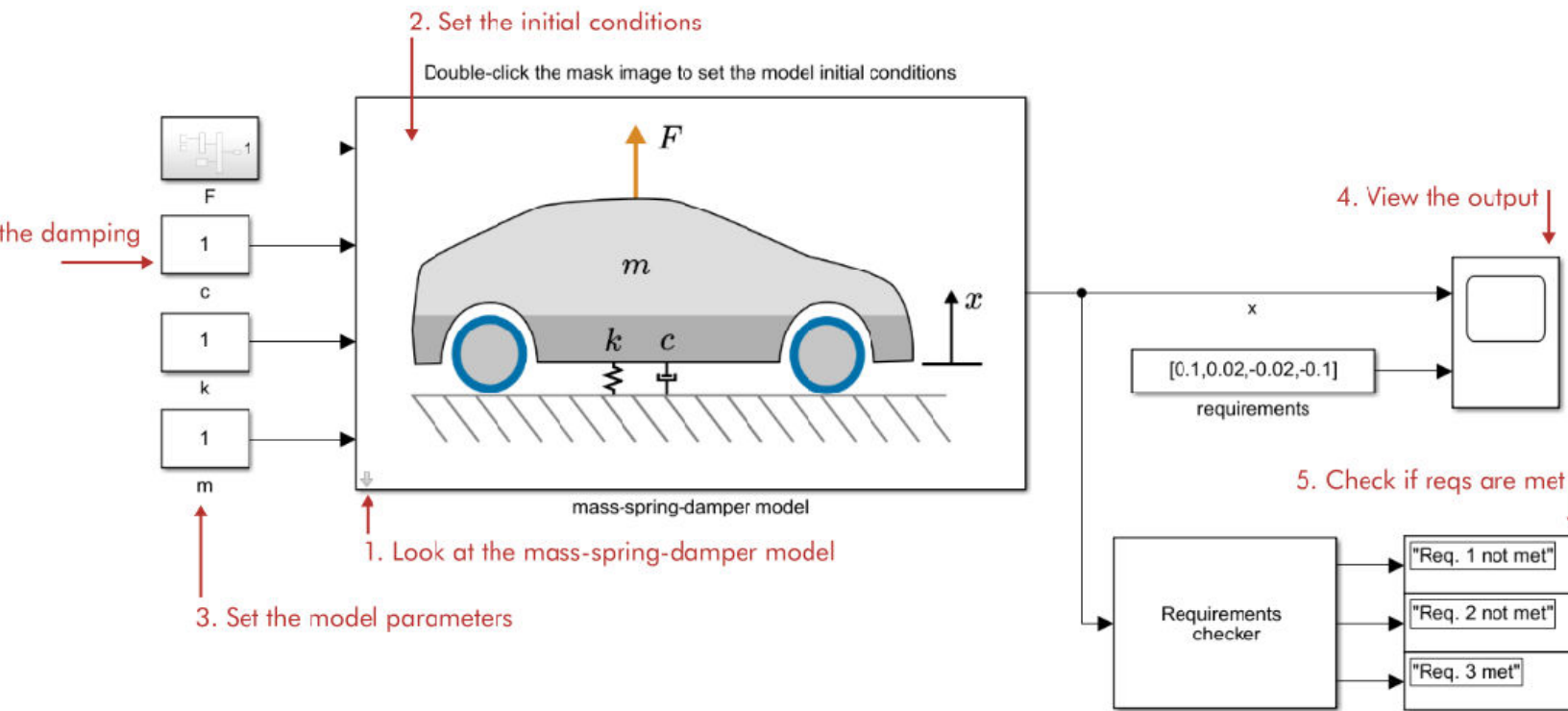
Assume that, initially, the vehicle has no motion or displacement in the vertical direction.



Schematic of the design requirements. The initial vertical displacement should be less than 0.1 m. Subsequent oscillations should have a magnitude of less than 0.02 m.

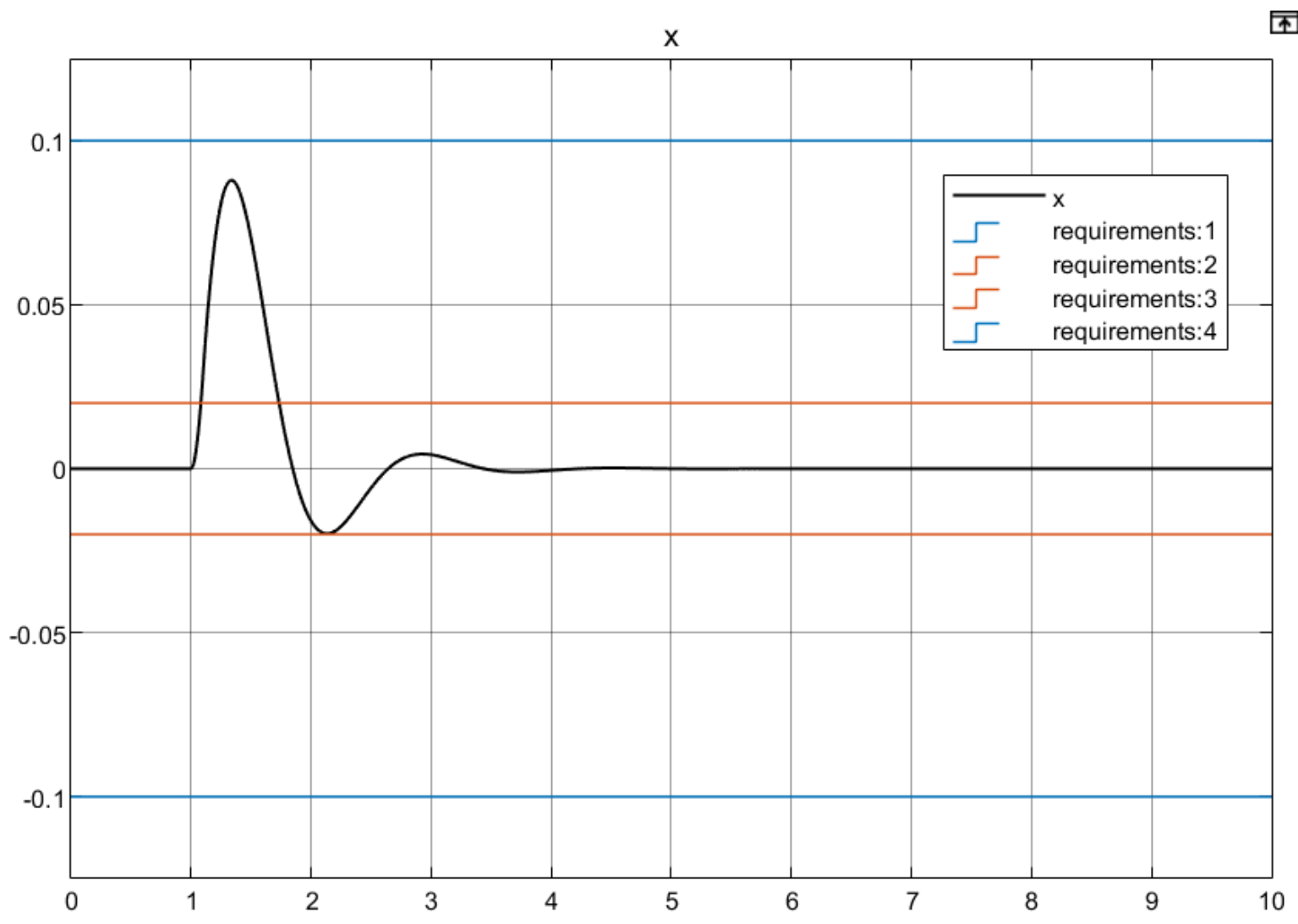
 **Exercise.** To adjust the design so that it meets the requirements, perform the following.

1. Open the model [msdTune.slx](#). This model has a masked subsystem containing the mass-spring-damper model you worked with in the previous section. You can look under the mask by clicking the arrow (▼) to view this model.
2. Double-click the [mask](#) of the model and set the initial conditions of the displacement $x(0)$ and velocity $\dot{x}(0)$.
3. Set the fixed parameters of the model: k and m . The model already applies the correct impulse force F , so do not modify the "F" subsystem.
4. Set a damping coefficient, c , and run the model. View the vehicle's bouncing behavior in the [scope](#).
5. Tune the damping until all three requirements have been met.

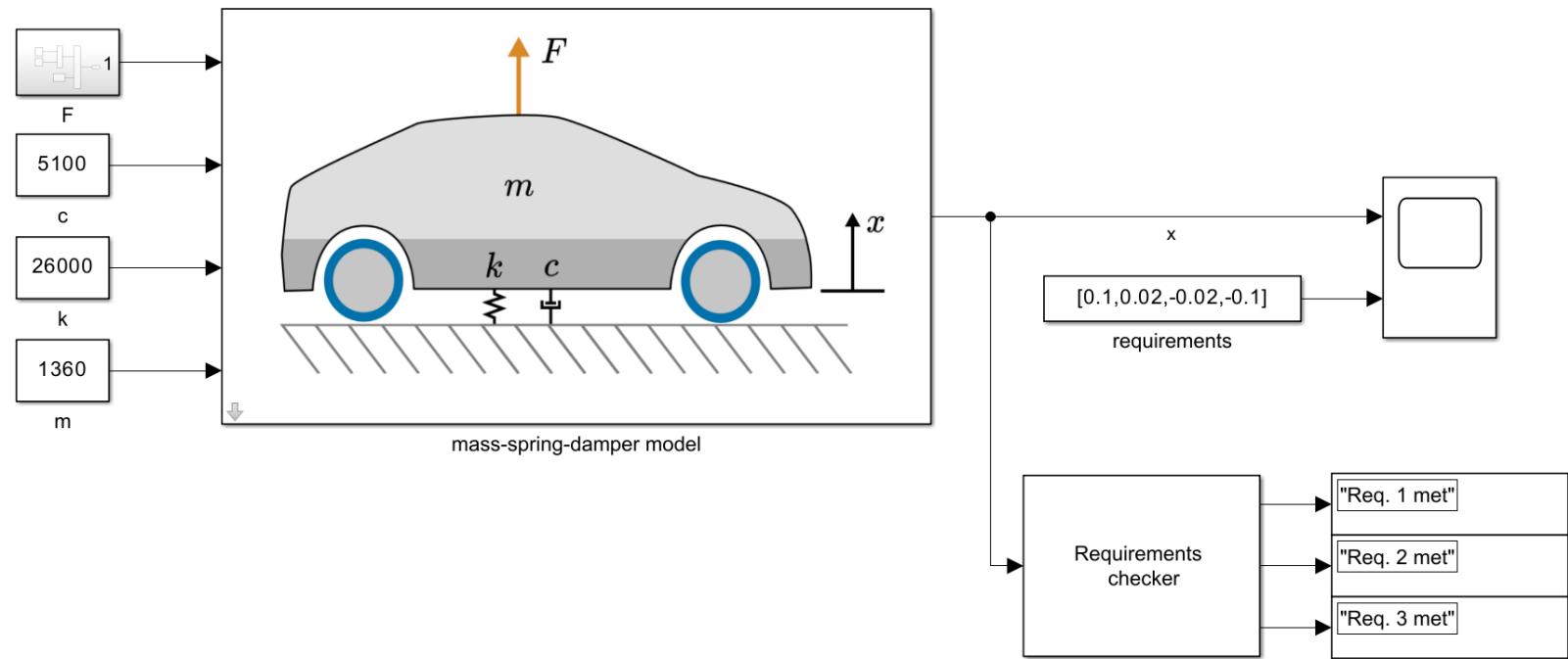


Schematic of the Simulink model and where you should perform the tasks

Exercise.



Double-click the mask image to set the model initial conditions



Helper functions

These functions check answers and generate animations.

```
function checkSyms
    varList = ["x","x_dot","x_ddot","m","c","k","F"];
    for k = 1:length(varList)
        varExists = evalin("caller",strcat("exist('",varList(k), "','var')"));
        if(varExists)
            varSym = evalin("caller",strcat("isequal(class(",varList(k),"),""sym'')"));
            if(varSym)
                disp(strcat("Success! ",varList(k)," is symbolic."))
            else
                warning(strcat(varList(k)," exists but is not symbolic."))
            end
        else
            warning(strcat(varList(k)," does not exist!"))
        end
    end
end

function correct = isEqualCheck(userSubmission,correctAnswer)
    correct = true;
    if( isequal(userSubmission,correctAnswer) )
        disp("Correct!")
    else
        warning("Your answer does not match the expected result.")
    end
end
```



```

        correct = false;
    end
end

function checkMSDeq(msdEqUser)
    syms x x_dot x_ddot m c k F
    msdEq = m*x_ddot + c*x_dot + k*x == F;
    isEqualCheck(solve(msdEqUser,x_ddot), solve(msdEq,x_ddot));
end

function checkMSDSolve(sltnUser)
    syms x x_dot x_ddot m c k F
    msdEq = m*x_ddot + c*x_dot + k*x == F;
    sltn = solve(msdEq,x_ddot);
    isEqualCheck(sltnUser,sltn);
end

function checkFirstOrder(xSltnUser)
    syms t;
    xSltn = 10*exp(-t);
    correct = isEqualCheck(xSltnUser,xSltn);
    if(correct)
        tvec = linspace(0,10,500);
        figure("position",[1 1 400 200])
        plot(tvec,10*exp(-tvec))
        xlabel("$t$", "Interpreter", "latex")
        ylabel("$x(t)$", "Interpreter", "latex")
        title("Solution curve")
    end
end

function animateMSD(tin,xin,spacingMultiplier,steps)

    Nmass = size(xin,2); % Number of masses

    % Options that determine how the graph will look
    colors = lines(Nmass);
    w = 1; % Mass width
    h = 2; % Height of masses
    y = 0; % Vertical position

    % Local variables
    t = linspace(tin(1),tin(end),steps);
    x = zeros(steps,Nmass);
    for k = 1:Nmass
        x(:,k) = interp1(tin,xin(:,k),t);
    end
    spacing = 2*max(abs(x),[], "all")*spacingMultiplier;
    xmax = spacing*2.5 + 2;

```

```

% Loop to create the animation
figure;
for j = 1:numel(t)

    subplot(2,1,1) % This shows the masses
    cla
    hold on
    x0k = 0;
    for k = 1:Nmass
        % Plot springs
        xc = spacing*k+x(j,k)-w/2;
        xs = linspace(x0k,xc,12);
        ys = 0.25*(-1).^(1:numel(xs));
        plot(xs,ys + 0.5,"k");
        % Plot dampers
        yd = 1.65;
        xd = x0k + (xc-x0k)*0.5+0.1;
        xd2 = x0k + (xc-x0k)*0.5;
        xd3 = x0k + (xc-x0k)*0.5 + 0.2;
        plot([x0k,xd,NaN,xd,xd],[yd,yd,NaN,yd-0.1,yd+0.1],"k")
        plot([xd2,xd3,NaN,xd2,xd3,NaN,xd3,xd3,NaN,xd3,xc],[yd-0.15,yd-0.15,NaN,yd+0.15,yd+0.15,NaN,yd+0.15,yd+0.15,NaN,yd+0.15,yd+0.15],"k")
        % Update position
        x0k = xc + w;
        % Plot masses
        x1 = spacing*k + x(j,k) - 0.5;
        rectangle("Position",[x1 y w h],"FaceColor",colors(k,:))
        text(x1 + w/2, h/2,"m_" + num2str(k),"HorizontalAlignment","center","color","w")
        % Plot x1, x2 distances
        quiver(spacing*k, y+h*1.1, x(j,k),0,"k-","MaxHeadSize",1/(x(j,k))^2)
        plot([1;1]*[spacing*k, spacing*k+x(j,k)],[0.95;1.05]*(y*[1,1]+h*1.1),"k-","markersize",10)
        text((spacing*k + spacing*k+x(j,k))/2,(y+h*1.3),"x_" + num2str(k),"HorizontalAlignment","center","color","w")
    end
    hold off
    axis([0,xmax,-1,3])
    set(gca,"ytick",[])
    xlabel("reference axis [m]")

    subplot(2,1,2) % This shows the position plots with moving markers
    cla
    hold on
    plot(t,x)
    for k = 1:Nmass
        plot(t(j),x(j,k),"ko","MarkerFaceColor",colors(k,:))
    end
    xlabel("t [s]")
    ylabel("x [m]")
    hold off
    box off
    drawnow
end

```

```
end

function addModelPaths()
    % Adds the paths for the models and images
    scriptPath = fileparts(matlab.desktop.editor.getActiveFilename);
    addpath(scriptPath,scriptPath + "/images/",scriptPath + "/models/")
end

% Suppress unused suggestions
%#ok<*UNRCH>
```