MMF062 Vehicle Dynamics
Design Task 3 - Simulation of Vertical Dynamics

Gabriel Wendel
Erik Lydig

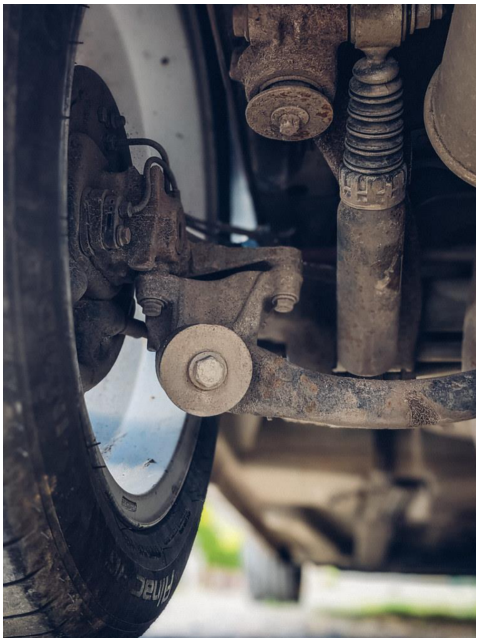


Figure 1: Suspension. [1]

Third group assignment in the course MMF062 - Vehicle Motion Engineering

MSc Mobility Engineering
Chalmers University of Technology
Sweden

Task 1: Quarter Car Model Transfer Functions in Matrix Form

Task 1.1: Model and derive equations

Figure 2 represents a free-body diagram of the system.

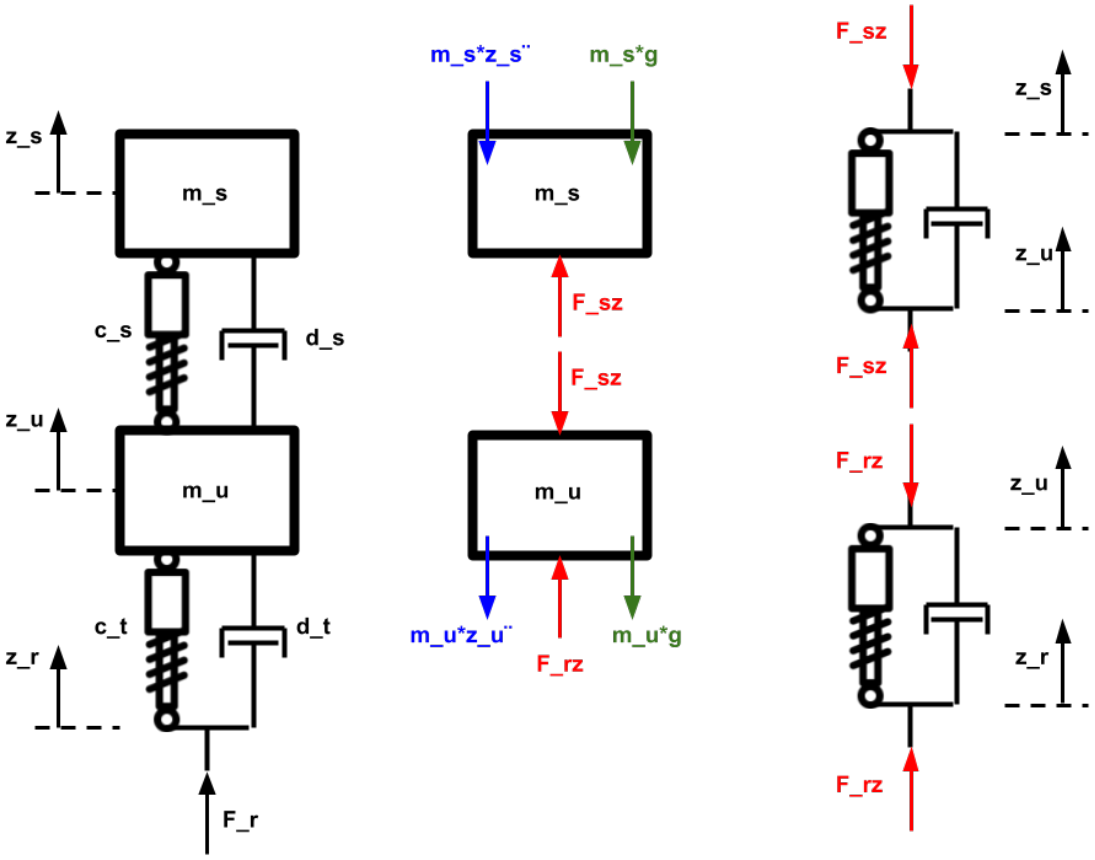


Figure 2: Freebody diagram for Quarter Car Model.

Mathematical model can be divided into equilibrium, constitution and excitation:

Equilibrium:

$$m_z \ddot{z}_s = F_{sz} - m_s g \tag{1}$$

$$m_u \ddot{z}_u = F_{rz} - F_{sz} - m_u g \tag{2}$$

Constitution:

$$F_{sz} = \underbrace{c_s(z_u - z_s) + m_s g}_{\text{linear spring } c_s} + d_s(\dot{z}_u - \dot{z}_s) \tag{3}$$

$$F_{rz} = \underbrace{c_t(z_r - z_u) + (m_s g + m_u)g}_{\text{linear spring } c_t} + d_t(\dot{z}_r - \dot{z}_u) \tag{4}$$

Excitation:

$$z_r = z_r(t) \tag{5}$$

We want to state the equations in state space form given by Equation 6.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{6}$$

Where $\mathbf{x}$ is a state vector containing displacements of sprung and unsprung mass as well as their speed, see Equation 7. Input vector $\mathbf{u}$ is road displacement, z_r.

$$\mathbf{x} = \begin{bmatrix} z_s \\ z_u \\ \dot{z}_s \\ \dot{z}_u \end{bmatrix} \tag{7}$$

From Equtions 1 and 2 we can solve for $\ddot{z}_s$ and $\ddot{z}_u$:

$$\ddot{z}_s = \frac{F_{sz} - m_s g}{m_z} \tag{8}$$

$$\ddot{z}_u = \frac{F_{rz} - F_{sz} - m_u g}{m_u} \tag{9}$$

Insert Equation 1 and 2 into Equation 8 and 9 to obtain expression for accelerations:

$$\begin{aligned} \ddot{z}_s &= \frac{c_s(z_u - z_s)m_s g + d_s(\dot{z}_u - \dot{z}_s) - m_s g}{m_s} \\ &= \frac{c_s(z_u - z_s) + d_s(\dot{z}_u - \dot{z}_s)}{m_s} \end{aligned} \tag{10}$$

$$\begin{aligned} \ddot{z}_u &= \frac{c_t(z_r - z_u) + (m_s g + m_u)g + d_t(\dot{z}_r - \dot{z}_u) - c_s(z_u - z_s) + m_s g - d_s(\dot{z}_u - \dot{z}_s) - m_u g}{m_u} \\ &= \frac{c_t(z_r - z_u) - c_s(z_u - z_s) - d_s(\dot{z}_u - \dot{z}_s)}{m_u} \end{aligned} \tag{11}$$

Thus state-space form can be written as:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \Leftrightarrow \underbrace{\begin{bmatrix} \dot{z}_u \\ \dot{z}_s \\ \ddot{z}_u \\ \ddot{z}_s \end{bmatrix}}_{\dot{\mathbf{x}}} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{(c_t + c_s)}{m_u} & \frac{c_s}{m_u} & -\frac{d_s}{m_u} & \frac{d_s}{m_u} \\ \frac{c_s}{m_s} & -\frac{c_s}{m_s} & \frac{d_s}{m_s} & -\frac{d_s}{m_s} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} z_u \\ z_s \\ \dot{z}_u \\ \dot{z}_s \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ \frac{c_t}{m_u} \\ 0 \end{bmatrix}}_{\mathbf{B}} \underbrace{z_r}_{u} \tag{12}$$

Task 1.2: Transfer function

State Equation 13 and Output Equation 14 define the linear time invarient (LTI) system.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{13}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{14}$$

Taking the Laplace tranformation of Equation 13 and 14 gives:

$$sX(s) = AX(s) + BU(s) \tag{15}$$

$$Y = CX(s) + DU(s) \tag{16}$$

Collecting terms and solving for X(s) in Equation 15:

$$(s - I_n)X(s) = BU(s) \Rightarrow X(s) = (s - I_n)^{-1}BU(s) \tag{17}$$

Inserting Equation 17 into Equation 16:

$$Y(s) = C(sI_n - A)^{-1}BU(s) + DU(s) = \left[C(sI_n - A)^{-1}B + D\right]U(s) \tag{18}$$

Thus the transfer function $H(s)$ can be expressed as:

$$H(s) = \frac{Y(s)}{U(s)} = C(sI_n - A)^{-1}B + D \tag{19}$$

Task 1.3: Plot transfer functions $\mathbf{H}(\omega)$

Three distinct transfer functions, denoted as $H(\omega)z_r \rightarrow \ddot{z}_s$, $H(\omega)z_r \rightarrow (z_u - z_s)$, and $H(\omega)z_r \rightarrow \Delta F_{rz}$ for rear and front wheel in the quarter car model, were plotted, see Figure 3 - 5. Matrices **C** and **D** in Equation 16 were determined for each of the three cases based on input and output considerations. **C** and **D** matrices are defined in Equation 20 - 22.

$H(\omega)_{z_r \rightarrow \ddot{z}_s}$:

$$\mathbf{C} = \begin{bmatrix} \frac{c_s}{m_s} & -\frac{c_s}{m_s} & \frac{d_s}{m_s} & -\frac{d_s}{m_s} \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 \end{bmatrix} \tag{20}$$

$H(\omega)_{z_r \rightarrow (z_u - z_s)}$:

$$\mathbf{C} = \begin{bmatrix} 1 & -1 & 0 & 0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 0 \end{bmatrix} \tag{21}$$

$H(\omega)_{z_r \rightarrow \Delta F_{rz}}$:

$$\mathbf{C} = \begin{bmatrix} c_t & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} c_t \end{bmatrix} \tag{22}$$

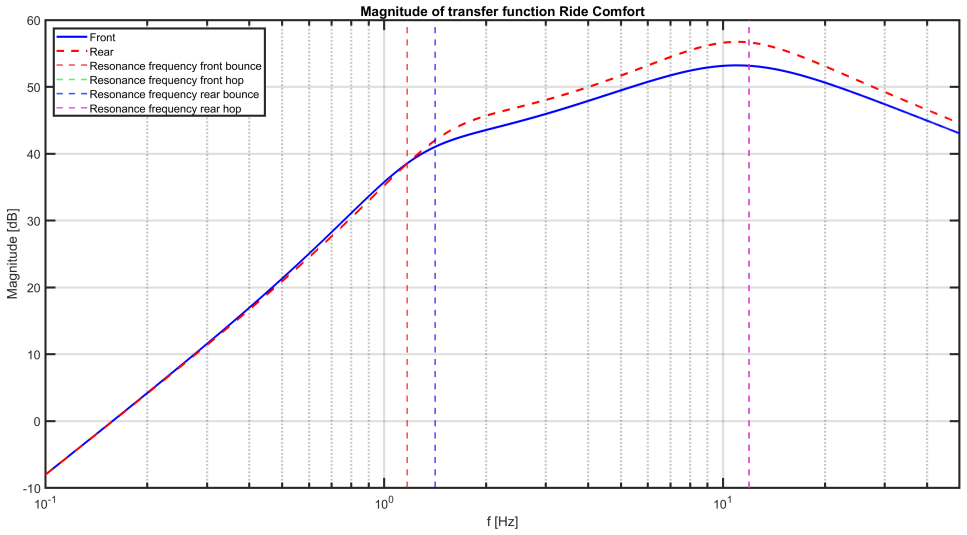


Figure 3: Transfer function $H(\omega)_{z_r \rightarrow \ddot{z}_s}$, road displacement, z_r, as input and acceleration of sprung mass $\ddot{z}_s$ as output.

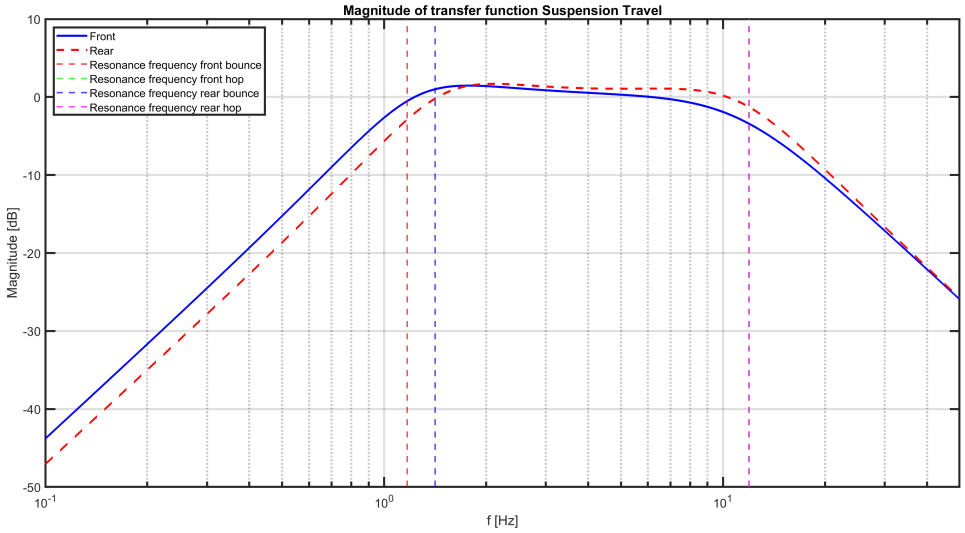


Figure 4: Transfer function $H(\omega)_{z_r \to (z_u - z_s)}$, road displacement, z_r, as input and difference in displacement of the unsprung and spring mass, $z_u - z_s$, as output.

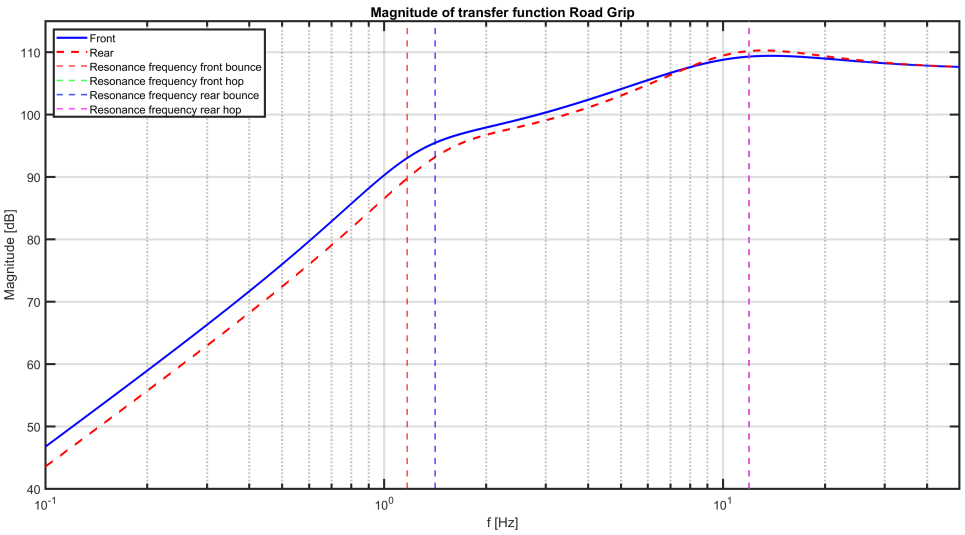


Figure 5: Transfer function $H(\omega)_{z_r \to \Delta F_{r_z}}$, road displacement, z_r, as input and tyre force (road grip, which is defined by difference in displacement of the road and the unsprung mass multiplied with tire stiffness) as output.

Task 1.4: Identify natural frequencies

Natural frequencies was calculated using Equation 23 and 24. Table 1 displays the values for each natural frequency in the four cases. The calculated natural frequencies are also plotted in Figure 3 - 5. As seen in the graphs, we have a sudden increase in magnitude due to resonance in the system which amplifies the oscillation.

$$\omega_{bounce} = \sqrt{\frac{1/\left(\frac{1}{c_s} + \frac{1}{c_t}\right)}{m_s}} \tag{23}$$

$$\omega_{WheelHop} = \sqrt{\frac{c_s + c_t}{m_u}} \tag{24}$$

Table 1: Resonance frequency for Bounce and Hop (Front and Rear tire)

Natural frequency	**Front**	**Rear**
$f_{res,bounce}$ [Hz]	1.1701	1.4145
$f_{res,hop}$ [Hz]	11.9514	11.9307

By studying Figures 3 - 5 we can see that natural frequency in bounce mode is a the lower frequency mode, while natural frequency in wheel hop mode is the higher one. In a magnitude plot the bumps corresponds to frequencies at which the system responds strongly. One can conclude that the natural frequencies (represented by the vertical lines in Figure 3 - 5) appear at the bumps, which indicates that the system responds most strongly to frequencies that match its natural frequency.

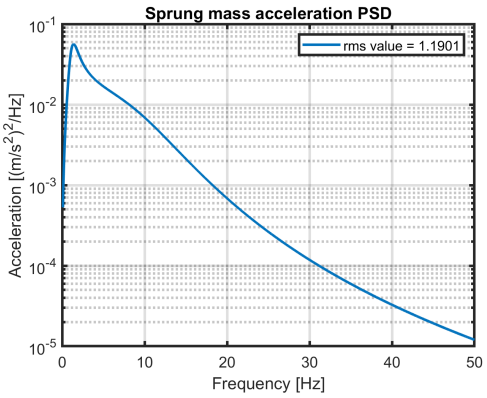
For the wheel hop model the mass is unsprung mass, m_u, and tire- and suspension stiffness c_t, c_s are parallel connected. Therefore, total stiffness becomes $c_{tot} = c_t + c_s$. Mass for bounce model is sprung mass, m_s, and stiffnesses are series connected. Which results in total stiffness, $c_{tot} = 1/((1/c_s)+(1/c_t))$. The total stiffness is larger for the parallel connected wheel hop model, which results in a higher natural frequnecy, see Equations 23 and 24.

Task 2: Study of the suspension stiffness and damping

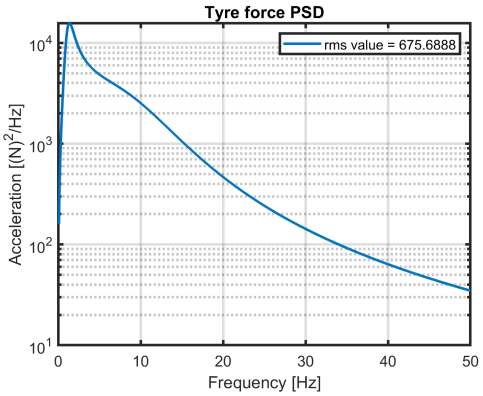
Task 2.1: Plot response spectrum and calculate rms values

Power Spectral Density (PSD) was calculated using Equation 25. PSD was calculated for acceleration of the sprung mass and tyre force (road grip) and then plotted, see Figure 6a and Figure 6b.

$$G(\omega) = |H(\omega)|^2 G_{z_r}(\omega) \tag{25}$$



(a) PSD sprung mass acceleration, $\ddot{z}_s$ vs frequency [Hz].



(b) PSD tyre force, ΔF_{rz} vs frequency [Hz].

Figure 6: PSD plots.

Root Mean Square (RMS) values were calculated using Equation 26.

$$\text{RMS} = \sqrt{\int_0^\infty G(\omega) d\omega} \approx \sqrt{\sum_i^N G(\omega_i)\Delta\omega} \tag{26}$$

Task 2.2 Balance Ride comfort and Tyre force/road grip

For each stiffness value in the given `SuspStiffVector` the damping was varied according to: `SuspDampVector = [1000 : 100 : 9000];`. RMS values for sprung mass acceleration and tyre force were calculated using Equation 26 for each setting of stiffness and damping.

RMS values were then plotted against damping, with one curve per stiffness value, see Figure 7 and Figure 8. From the Figures we can conclude that an increase in spring stiffness will result in both higher vertical tyre force, and also higher vertical acceleration. The relation for the damping coefficients is non-linear.

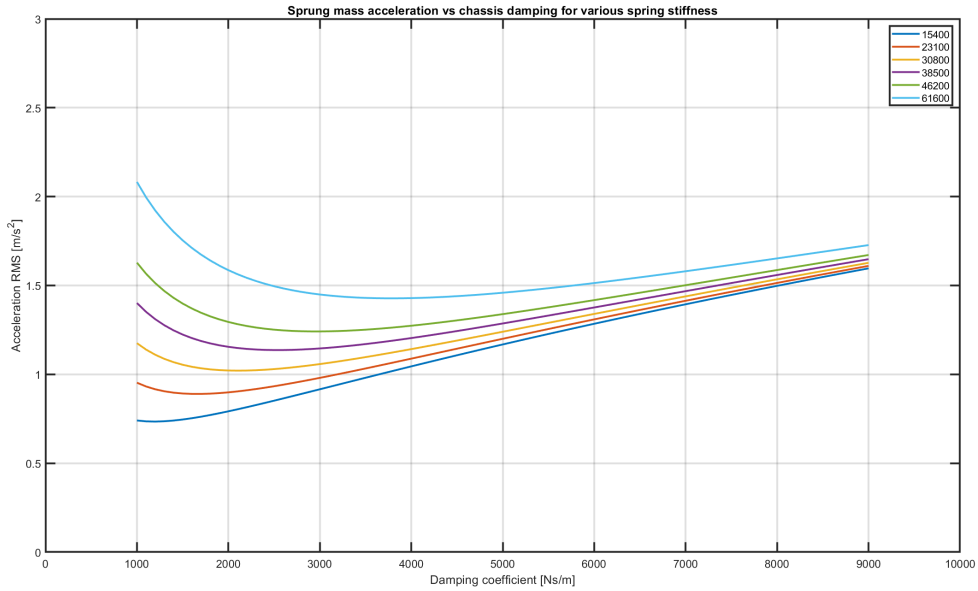


Figure 7: Sprung mass acceleration vs damping for different spring stiffness.

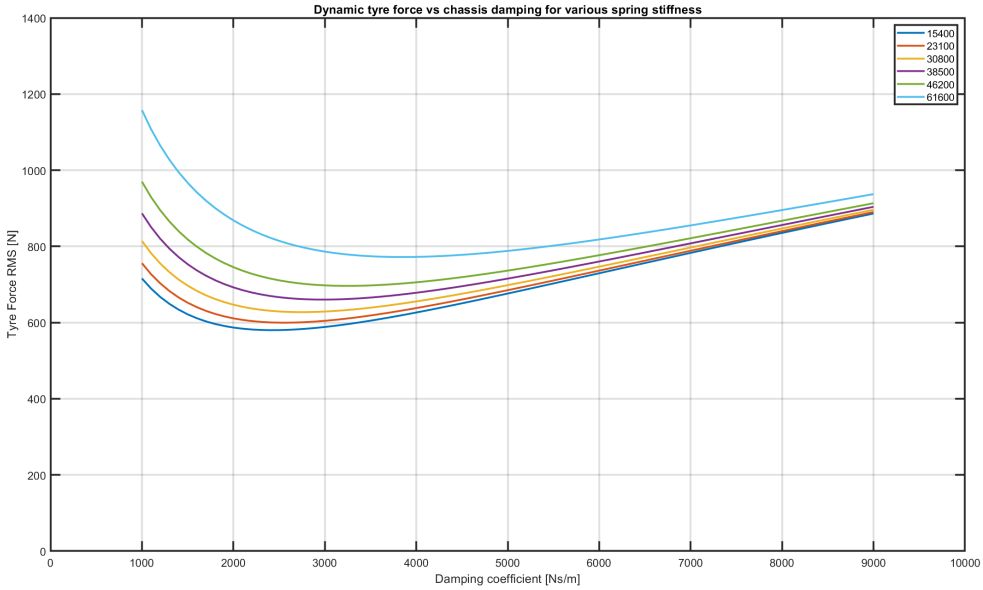


Figure 8: Dynamic tyre force vs damping for different spring stiffness.

Figure 9 illustrates optimal damping as a function of spring stiffness with regards to sprung mass acceleration and tyre force. To find a optimal damping value that takes into account both $RMS_{\ddot{z}_s}$ and $RMS_{tyreforce}$ the mean value of the two optimal was calculated. Data for optimal damping is presented in Table 2.

Table 2: Optimal damping values w.r.t sprung mass acceleration, tyre force and a mean optimal of the two for given spring stiffnesses.

$\mathbf{c_s}$ [N/m]	**Optimal w.r.t** $\ddot{z}_s$ [Ns/m]	**Optimal w.r.t** ΔF_{rz} [Ns/m]	**Mean optimal** [Ns/m]
15400	1200	2400	1800
23100	1700	2600	2150
30800	2100	2700	2400
38500	2500	3000	2750
46200	3000	3200	3100
61600	3800	3800	3800

From Figure 9 we can see that to achieve an optimal damping vs spring rate w.r.t. sprung mass acceleration and tyre force, spring stiffness should be increased as damping coefficient increases. In reality we need to consider suspension travel and having a to low spring stiffness together with a low damping value would result in hitting the bump stops.

In Figure 9, since rmsForce is above the rmsAcc we can conclude that Tyre force requires the highest damping coefficient. Meaning that prioritising the optimal Tyre force will increase the vertical acceleration, and move further away from the optimal point. Picking the mean point might not be the optimal, and it rather depends on what characteristic you want to focus on.

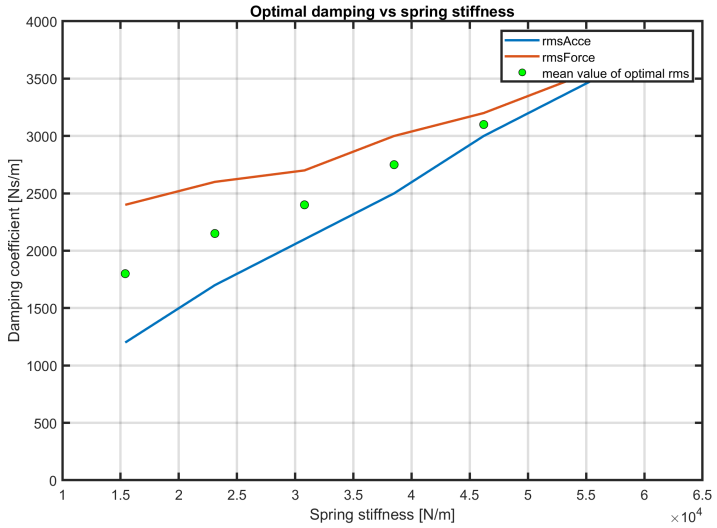


Figure 9: Optimal damping vs spring stiffness w.r.t. sprung mass acceleration and tyre force. Scatter points represent mean of the optimal damping values.

Task 3: Ride comfort and the use of ISO-2631

Task 3.1: Calculation of daily whole-body exposure values

Using Equation 27 from Compendium [5.52], the time averaged whole-body vibration exposure value $a_{w,av}$ can be calculated. This equation takes the weighted RMS Acceleration values for Smooth, Rough and Very Rough and averages them over their respective time duration.

Calculating the time averaged vibration exposure value using the values specified in Table 3, we can conclude that the driving conditions are too rough, and the value exceeds the ISO-2631 limit of 1.15 m/s^2.

Table 3: Speeds during each driving condition.

Condition	**Speed** [km/h]
Smooth	110
Rough	110
Very Rough	110

$$a_{w,av} = \sqrt{\frac{\sum a_{wi}^2 T_i}{\sum T_i}} = 1.5492 \text{m/s}^2 \tag{27}$$

Trips per day = 5.5

Task 3.2: Modification of vehicle velocity

From Table 4 we can see that if we priorities to keep each Weighted RMS Acceleration value, for each, driving condition, below 1.15 m/s^2, we achieve low Time averaged exposure value, but also a few trips per day.

Trips per day = 2.25
Time averaged vibration exposure value = 0.99 m/s^2

Table 4: Each Weighted vibration exposure value is below 1.15 m/s^2.

Condition	Speed [km/h]	Weighted RMS Acceleration [m/s^2]
Smooth	110	0.394
Rough	101	1.149
Very Rough	8.9	1.145

The next test instead priorities to achieve the maximum amount of trips, without exceeding the ISO limit. As seen in Table 5 the Weighted RMS value is above the ISO limit for Rough and Very Rough, but since it is Time averaged this results in a value which is acceptable. The idea was to first increase speed for the Rough driving condition since the distance travelled is longer than the Very Rough driving condition. And then later if possible increase the speed for the Very Rough driving condition until the Time averaged limit was reached.

Trips per day = 3.02
Time averaged vibration exposure value = 1.147 m/s^2

Table 5: Maximize Trips but keep Time averaged vibration exposure below 1.15 m/s^2.

Condition	Speed [km/h]	Weighted RMS Acceleration [m/s^2]
Smooth	110	0.394
Rough	110	1.225
Very Rough	14.5	1.463

It can therefore be concluded that to maximise trips, the Time averaged exposure value should be maximised but kept below the ISO limit. Prioritising each individual Weighted RMS value is not optimal since it will reduce the number of trips.

References

[1] Lecture Module 5:2 Road and vehicle models MMF062. Fredrik Bruzelius. 2023

Appendix 1

Listing 1: MATLAB code Task 1.1 - 1.4

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vehicle Dynamics, MMF062, 2020
% Vertical assignment, Task 1
%
% Add your own code where "%ADD YOUR CODE HERE" is stated
%
clear all;
close all;
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load parameters from file "InitParameters.m"

InitParametersSkeleton

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Task 1.3
%
% Consider one single front wheel, identify sprung mass and unsprung
    mass

% sprungMassFront = (totalSprungMass * (wheelBase -
    distanceCogToFrontAxle) / wheelBase)/2;
% unsprungMassFront = (totalUnsprungMass * (wheelBase -
    distanceCogToFrontAxle) / wheelBase)/2;

sprungMassFront = 0.5*(distanceCogToRearAxle/wheelBase)*
    totalSprungMass;
unsprungMassFront =  0.25*totalUnsprungMass;

% Identify indiviual A and B matrix
Af =  [0 0 1 0;
       0 0 0 1;
       -(tireStiff + frontWheelSuspStiff)/unsprungMassFront
           frontWheelSuspStiff/unsprungMassFront -frontWheelSuspDamp/
           unsprungMassFront frontWheelSuspDamp/unsprungMassFront;
       frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
           sprungMassFront frontWheelSuspDamp/sprungMassFront -
           frontWheelSuspDamp/sprungMassFront];
Bf =  [0; 0; tireStiff/unsprungMassFront; 0];

% Calculate transfer functions for
% 1)front wheel Zr to Ride,
% 2)front wheel Zr to Suspension travel and
% 3) front wheel Zr to Tyre force

% matrices Zr to Ride,front wheel:
% C1f = 1/sprungMassFront*[frontWheelSuspStiff -frontWheelSuspStiff
    frontWheelSuspDamp - frontWheelSuspDamp];
C1f = [frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
    sprungMassFront frontWheelSuspDamp/sprungMassFront -
    frontWheelSuspDamp/sprungMassFront];
D1f = 0;
```

```
43
44 % matrices for Zr to Suspension travel, front wheel:
45 C2f = [1 -1 0 0];
46 D2f = 0;
47
48 % matrices for Zr to Tyre force, front wheel:
49 C3f = [-tireStiff 0 0 0];
50 D3f = tireStiff;
51
52 transferFunctionFrontZrToRide = zeros(length(angularFrequencyVector)
       ,1);
53 transferFunctionFrontZrToTravel = zeros(length(angularFrequencyVector)
       ,1);
54 transferFunctionFrontZrToForce = zeros(length(angularFrequencyVector)
       ,1);
55
56 for j = 1 : length(angularFrequencyVector)
57     % Calculate H(w) not the absolut value |H(w)|
58     transferFunctionFrontZrToRide(j,:) = C1f*inv(1i*
           angularFrequencyVector(j)*eye(4) - Af)*Bf + D1f;
59     transferFunctionFrontZrToTravel(j,:) = C2f*inv(1i*
           angularFrequencyVector(j)*eye(4) - Af)*Bf + D2f;
60     transferFunctionFrontZrToForce(j,:) = C3f*inv(1i*
           angularFrequencyVector(j)*eye(4) - Af)*Bf + D3f;
61 end
62
63
64 %%%%%%%%%%%%%%%%%%%%%%%%%%%
65 % Consider one single rear wheel, identify sprung mass and unsprung
       mass
66
67 % sprungMassRear = 0.5*totalSprungMass * (wheelBase -
       distanceCogToRearAxle) / wheelBase;
68 % unsprungMassRear = 0.5*totalUnsprungMass * (wheelBase -
       distanceCogToRearAxle) / wheelBase;
69
70 sprungMassRear = 0.5*(distanceCogToFrontAxle/wheelBase)*
       totalSprungMass;
71 unsprungMassRear = 0.25*totalUnsprungMass;
72
73 % Identify indiviual A and B matrix
74 Ar =  [0 0 1 0;
75        0 0 0 1;
76        -(tireStiff + rearWheelSuspStiff)/unsprungMassRear
              rearWheelSuspStiff/unsprungMassRear -rearWheelSuspDamp/
              unsprungMassRear rearWheelSuspDamp/unsprungMassRear;
77        rearWheelSuspStiff/sprungMassRear -rearWheelSuspStiff/
              sprungMassRear rearWheelSuspDamp/sprungMassRear -
              rearWheelSuspDamp/sprungMassRear];
78 Br =  [0; 0; tireStiff/unsprungMassRear; 0];
79
80 % Calculate transfer functions for
81 % 1) rear wheel Zr to Ride,
82 % 2) rear wheel Zr to Suspension travel and
83 % 3) rear wheel Zr to Tyre force
84
```

```
% matrices Zr to Ride, rear wheel:
C1r = [rearWheelSuspStiff/sprungMassRear -rearWheelSuspStiff/
    sprungMassRear rearWheelSuspDamp/sprungMassRear -rearWheelSuspDamp/
    sprungMassRear];
D1r = 0;

% matrices for Zr to Suspension travel, rear wheel:
C2r = [1 -1 0 0];
D2r = 0;

% matrices for Zr to Zr to Tyre force, rear wheel:
C3r = [-tireStiff 0 0 0];
D3r = tireStiff;

% Rear wheel
transferFunctionRearZrToRide = zeros(length(angularFrequencyVector),1)
    ;
transferFunctionRearZrToTravel = zeros(length(angularFrequencyVector)
    ,1);
transferFunctionRearZrToForce = zeros(length(angularFrequencyVector)
    ,1);


for j = 1 : length(angularFrequencyVector)
    % Calculate H(w) not the absolut value |H(w)|
    transferFunctionRearZrToRide(j,:) = C1r*inv(1i*
        angularFrequencyVector(j)*eye(4) - Ar)*Br + D1r;
    transferFunctionRearZrToTravel(j,:) = C2r*inv(1i*
        angularFrequencyVector(j)*eye(4) - Ar)*Br + D2r;
    transferFunctionRearZrToForce(j,:) = C3r*inv(1i*
        angularFrequencyVector(j)*eye(4) - Ar)*Br + D3r;
end

%% Task 1.4
%
% Identify natural frequencies

% Front wheel
resonanceFreqFrontBounce = sqrt((1 /(1 / frontWheelSuspStiff + 1/
    tireStiff)) / sprungMassFront);
resonanceFreqFrontHop = sqrt((frontWheelSuspStiff + tireStiff) /
    unsprungMassFront);

% Rear wheel
resonanceFreqRearBounce = sqrt((1 /(1 / rearWheelSuspStiff + 1/
    tireStiff)) / sprungMassRear);
resonanceFreqRearHop = sqrt((rearWheelSuspStiff + tireStiff) /
    unsprungMassRear);

% Convert from rad to Hertz
resonanceFreqFrontBounceInHertz = (resonanceFreqFrontBounce)/(2*pi)
resonanceFreqFrontHopInHertz = (resonanceFreqFrontHop)/(2*pi)
resonanceFreqRearBounceInHertz = (resonanceFreqRearBounce)/(2*pi)
resonanceFreqRearHopInHertz = (resonanceFreqRearHop)/(2*pi)

% Plot the transfer functions and natural frequencies
```

```
figure(1)
semilogx(frequencyVector,db(abs(transferFunctionFrontZrToRide)),'-b',...
    frequencyVector,db(abs(transferFunctionRearZrToRide)),'--r');
axis([0 50 -10 60]);grid
legend('Front','Rear','Location','northwest');
ylabel('Magnitude [dB]');%ADD YOUR CODE HERE
xlabel('f [Hz]');%ADD YOUR CODE HERE
title('Magnitude of transfer function Ride Comfort');
hold on
% add vertical lines to plot
xline(resonanceFreqFrontBounceInHertz,'r--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency front bounce');
xline(resonanceFreqFrontHopInHertz,'g--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency front hop');
xline(resonanceFreqRearBounceInHertz,'b--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency rear bounce');
xline(resonanceFreqRearHopInHertz,'m--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency rear hop');

figure(2)
semilogx(frequencyVector,db(abs(transferFunctionFrontZrToTravel)),'-b'
    ,...
    frequencyVector,db(abs(transferFunctionRearZrToTravel)),'--r');
axis([0 50 -50 10]);grid
legend('Front','Rear','Location','northwest');
ylabel('Magnitude [dB]');%ADD YOUR CODE HERE
xlabel('f [Hz]');%ADD YOUR CODE HERE
title('Magnitude of transfer function Suspension Travel');
hold on
% add vertical lines to plot
xline(resonanceFreqFrontBounceInHertz,'r--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency front bounce');
xline(resonanceFreqFrontHopInHertz,'g--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency front hop');
xline(resonanceFreqRearBounceInHertz,'b--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency rear bounce');
xline(resonanceFreqRearHopInHertz,'m--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency rear hop');

figure(3)
semilogx(frequencyVector,db(abs(transferFunctionFrontZrToForce)),'-b',...
    frequencyVector,db(abs(transferFunctionRearZrToForce)),'--r');
axis([0 50 40 115]);grid
legend('Front','Rear','Location','northwest');
ylabel('Magnitude [dB]');%ADD YOUR CODE HERE
xlabel('f [Hz]');%ADD YOUR CODE HERE
title('Magnitude of transfer function Road Grip');
hold on
% add vertical lines to plot
xline(resonanceFreqFrontBounceInHertz,'r--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency front bounce');
xline(resonanceFreqFrontHopInHertz,'g--', 'LineWidth', 1.5, '
    DisplayName', 'Resonance frequency front hop');
```

```
171 xline(resonanceFreqRearBounceInHertz,'b--', 'LineWidth', 1.5, '
       DisplayName', 'Resonance frequency rear bounce');
172 xline(resonanceFreqRearHopInHertz,'m--', 'LineWidth', 1.5, '
       DisplayName', 'Resonance frequency rear hop');
```

Appendix 2

Listing 2: MATLAB code Task 2.1 - 2.2

```
1
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % Vehicle Dynamics, MMF062, 2020
4  % Vertical assignment, Task 2
5  %
6  %
7  clear all;
8  close all;
9  clc;
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 %% Load parameters from file "InitParameters.m"
12
13 InitParametersSkeleton
14
15 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16 %% Task 2.1
17 %
18 % Front wheel
19 %
20 % Calculate road spectrum
21 roadSpectrum = zeros(length(angularFrequencyVector),1);
22
23 for i = 1 : length(angularFrequencyVector)
24     % Equation 5.29 from the compendium
25     roadSpectrum(i,:) = vehicleVelocity^(roadWaviness-1)*roadSeverity*
          angularFrequencyVector(i)^(-roadWaviness);
26 end
27
28 % Calculate transfer functions for front wheel Zr to Ride and Tyre
       force
29 % You can use the code from Task 1
30
31 % Consider one single front wheel (same expressions as Task 1)
32 sprungMassFront = 0.5*(distanceCogToRearAxle/wheelBase)*
       totalSprungMass;
33 unsprungMassFront =  0.25*totalUnsprungMass;
34
35 % Identify A and B matrix (same matrices as task 1)
36
37 Af =  [0 0 1 0;
38        0 0 0 1;
39        -(tireStiff + frontWheelSuspStiff)/unsprungMassFront
             frontWheelSuspStiff/unsprungMassFront -frontWheelSuspDamp/
             unsprungMassFront frontWheelSuspDamp/unsprungMassFront;
40        frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
             sprungMassFront frontWheelSuspDamp/sprungMassFront -
             frontWheelSuspDamp/sprungMassFront];
```

```
Bf =  [0; 0; tireStiff/unsprungMassFront; 0];

% Calculate transfer functions for
% 1)front wheel Zr to Ride,
% 3) front wheel Zr to Tyre force
% similar to Task 1 define C and D matrices for both cases
C1f = [frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
    sprungMassFront frontWheelSuspDamp/sprungMassFront -
    frontWheelSuspDamp/sprungMassFront];
D1f = 0;

C3f = [-tireStiff 0 0 0];
D3f = tireStiff;

transferFunctionFrontZrToRide = zeros(length(angularFrequencyVector)
    ,1);
transferFunctionFrontZrToForce = zeros(length(angularFrequencyVector)
    ,1);

for j = 1 : length(angularFrequencyVector)
    % Calculate H(w) not the absolut value |H(w)| (same as task 1)
    transferFunctionFrontZrToRide(j,:) = C1f*inv(1i*
        angularFrequencyVector(j)*eye(4) - Af)*Bf + D1f;
    transferFunctionFrontZrToForce(j,:) = C3f*inv(1i*
        angularFrequencyVector(j)*eye(4) - Af)*Bf + D3f;
end

% Calculate acceleration and tyre force response spectrum
psdAcceleration = zeros(length(angularFrequencyVector),1);
psdForce = zeros(length(angularFrequencyVector),1);

for m = 1 : length(angularFrequencyVector)
    % Equation 5.30 from compendium
    psdAcceleration(m,:) = abs(transferFunctionFrontZrToRide(m))^2*
        roadSpectrum(m);
    psdForce(m,:) = abs(transferFunctionFrontZrToForce(m))^2*
        roadSpectrum(m);
end

% Calculate rms values of acceleration and tyre force
msAcceleration = 0;
msForce = 0;

for n = 1 : length(angularFrequencyVector)
    % Equation 3.51 from compendium
    msAcceleration = msAcceleration + psdAcceleration(n)*
        deltaAngularFrequency;
    msForce = msForce + psdForce(n)*deltaAngularFrequency;
end

rmsAcceleration = sqrt(msAcceleration);
rmsForce = sqrt(msForce);

figure(100);
semilogy(frequencyVector,psdAcceleration);grid
xlabel('Frequency [Hz]');
```

```
ylabel('Acceleration [(m/s^2)^2/Hz]');
title('Sprung mass acceleration PSD');
legend(['rms value = ',num2str(rmsAcceleration)])

figure;
semilogy(frequencyVector,psdForce);grid
xlabel('Frequency [Hz]');
ylabel('Acceleration [(N)^2/Hz]');
title('Tyre force PSD');
legend(['rms value = ',num2str(rmsForce)])

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Task 2.2
%
% Front wheel
rmsAcceleration = zeros(length(frontWheelSuspStiffVector),length(
    frontWheelSuspDampVector));
rmsForce = zeros(length(frontWheelSuspStiffVector),length(
    frontWheelSuspDampVector));

for ind1 = 1 : length(frontWheelSuspStiffVector)

    for ind2 = 1 : length(frontWheelSuspDampVector)

        % Update suspension stiffness and damping
        frontWheelSuspStiff = frontWheelSuspStiffVector(ind1);
        frontWheelSuspDamp = frontWheelSuspDampVector(ind2);
        % Update A and C matrices
        Af =  [0 0 1 0;
               0 0 0 1;
               -(tireStiff + frontWheelSuspStiff)/unsprungMassFront
                  frontWheelSuspStiff/unsprungMassFront -
                  frontWheelSuspDamp/unsprungMassFront
                  frontWheelSuspDamp/unsprungMassFront;
               frontWheelSuspStiff/sprungMassFront -
                  frontWheelSuspStiff/sprungMassFront
                  frontWheelSuspDamp/sprungMassFront -
                  frontWheelSuspDamp/sprungMassFront];

        C1f = [frontWheelSuspStiff/sprungMassFront -
           frontWheelSuspStiff/sprungMassFront frontWheelSuspDamp/
           sprungMassFront -frontWheelSuspDamp/sprungMassFront];
        C3f = [-tireStiff 0 0 0];

        % Calculate transfer functions for front wheel Zr to Ride and
           Tyre force

            transferFunctionFrontZrToRide = zeros(length(
               angularFrequencyVector),1);
            transferFunctionFrontZrToForce = zeros(length(
               angularFrequencyVector),1);

            for j = 1 : length(angularFrequencyVector)
                % Calculate H(w) not the absolut value |H(w)|
                transferFunctionFrontZrToRide(j,:) = C1f*inv(1i*
                   angularFrequencyVector(j)*eye(4) - Af)*Bf + D1f;
```

```
                transferFunctionFrontZrToForce(j,:) = C3f*inv(1i*
                    angularFrequencyVector(j)*eye(4) - Af)*Bf + D3f;
            end


        % Calculate acceleration and tyre force response spectrum
        psdAcceleration = zeros(length(angularFrequencyVector),1);
        psdForce = zeros(length(angularFrequencyVector),1);

        for m = 1 : length(angularFrequencyVector)
            psdAcceleration(m,:) = abs(transferFunctionFrontZrToRide(m
                ))^2*roadSpectrum(m);
            psdForce(m,:) = abs(transferFunctionFrontZrToForce(m))^2*
                roadSpectrum(m);
        end

        % Calculate rms values of acceleration and tyre force
        msAcceleration = 0;
        msForce = 0;

        for n = 1 : length(angularFrequencyVector)
            msAcceleration = msAcceleration + psdAcceleration(n)*
                deltaAngularFrequency;
            msForce = msForce + psdForce(n)*deltaAngularFrequency;
        end

        rmsAcceleration(ind1,ind2) = sqrt(msAcceleration);
        rmsForce(ind1,ind2) = sqrt(msForce);

    end
end

% Plot rms values vs damping
figure;
plot(frontWheelSuspDampVector,rmsAcceleration);grid
legend(num2str(frontWheelSuspStiffVector'));
xlabel('Damping coefficient [Ns/m]');
ylabel('Acceleration RMS [m/s^2]');
title('Sprung mass acceleration vs chassis damping for various spring
    stiffness');
axis([0 10000 0 3]);

figure;
plot(frontWheelSuspDampVector,rmsForce);grid
legend(num2str(frontWheelSuspStiffVector'));
xlabel('Damping coefficient [Ns/m]');
ylabel('Tyre Force RMS [N]');
title('Dynamic tyre force vs chassis damping for various spring
    stiffness');
axis([0 10000 0 1400]);

% Identify optimal damping values for each stiffness value
[optimalRmsAcceleration,iOptimalRmsAcceleration] = min(rmsAcceleration
    ,[],2);
[optimalRmsForce,iOptimalRmsForce] = min(rmsForce,[],2);

```

```
% Plot optimal damping for Ride and Tyre force vs Stiffness
figure;
% Use optimal rms acc/force index
plot(frontWheelSuspStiffVector, frontWheelSuspDampVector(
    iOptimalRmsAcceleration));
hold on
plot(frontWheelSuspStiffVector, frontWheelSuspDampVector(
    iOptimalRmsForce));
grid

frontWheelSuspDamp_optimal_points = frontWheelSuspDampVector(
    iOptimalRmsAcceleration);
frontWheelSuspStiff_optimal_points = frontWheelSuspDampVector(
    iOptimalRmsForce);

% Plot optimal damping coefficient as a mean
for i=1:length(frontWheelSuspDampVector(iOptimalRmsForce))
     mean_optimal_damping(i) = (frontWheelSuspDamp_optimal_points(i) +
         frontWheelSuspStiff_optimal_points(i))/2;
end
scatter(frontWheelSuspStiffVector,mean_optimal_damping,'o','filled', '
    MarkerEdgeColor', 'k', 'MarkerFaceColor', 'g', 'SizeData', 50)
xlabel('Spring stiffness [N/m]');
ylabel('Damping coefficient [Ns/m]');
title('Optimal damping vs spring stiffness');
legend('rmsAcce','rmsForce','mean value of optimal rms');
axis([10000 65000 0 4000]);
```

Appendix 3

Listing 3: MATLAB code Task 3.1 - 3.2

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vehicle Dynamics, MMF062, 2020
% Vertical assignment, Task 3

clear all;
close all;
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load parameters from file "InitParameters.m"

InitParametersSkeleton

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Task 3.1

numberOfTripsPerDay = totalDrivingTime/...
    (distanceSmoothRoad/vehicleVelocitySmooth+distanceRoughRoad/
        vehicleVelocityRough+distanceVeryRoughRoad/
        vehicleVelocityVeryRough);

% i)
% Calculate road spectrum
roadSpectrumSmooth = zeros(length(angularFrequencyVector),1);
```

```
roadSpectrumRough = zeros(length(angularFrequencyVector),1);
roadSpectrumVeryRough = zeros(length(angularFrequencyVector),1);

for j = 1 : length(angularFrequencyVector)
    roadSpectrumSmooth(j,:) = vehicleVelocitySmooth^(
        roadWavinessSmooth-1)*roadSeveritySmooth*angularFrequencyVector
        (j)^(-roadWavinessSmooth);
    roadSpectrumRough(j,:) = vehicleVelocityRough^(roadWavinessRough
        -1)*roadSeverityRough*angularFrequencyVector(j)^(-
        roadWavinessRough);
    roadSpectrumVeryRough(j,:) = vehicleVelocityVeryRough^(
        roadWavinessVeryRough-1)*roadSeverityVeryRough*
        angularFrequencyVector(j)^(-roadWavinessVeryRough);
end

% Calculate transfer functions for front wheel Zr to Ride
% You can use result from Task 1

sprungMassFront = 0.5*(distanceCogToRearAxle/wheelBase)*
    totalSprungMass;
unsprungMassFront =  0.25*totalUnsprungMass;

Af =  [0 0 1 0;
       0 0 0 1;
       -(tireStiff + frontWheelSuspStiff)/unsprungMassFront
           frontWheelSuspStiff/unsprungMassFront -frontWheelSuspDamp/
           unsprungMassFront frontWheelSuspDamp/unsprungMassFront;
       frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
           sprungMassFront frontWheelSuspDamp/sprungMassFront -
           frontWheelSuspDamp/sprungMassFront];
Bf =  [0; 0; tireStiff/unsprungMassFront; 0];

C1f = [frontWheelSuspStiff/sprungMassFront -frontWheelSuspStiff/
    sprungMassFront frontWheelSuspDamp/sprungMassFront -
    frontWheelSuspDamp/sprungMassFront];
D1f = 0;

transferFunctionFrontZrToRide = zeros(length(angularFrequencyVector)
    ,1);

for j = 1 : length(angularFrequencyVector)
    transferFunctionFrontZrToRide(j,:) = C1f*inv(1i*
        angularFrequencyVector(j)*eye(4) - Af)*Bf + D1f;
end

% Calculate acceleration response spectrum for all roads

psdAccelerationSmooth = zeros(length(angularFrequencyVector),1);
psdAccelerationRough = zeros(length(angularFrequencyVector),1);
psdAccelerationVeryRough = zeros(length(angularFrequencyVector),1);

for j = 1 : length(angularFrequencyVector)
    psdAccelerationSmooth(j,:) = abs(transferFunctionFrontZrToRide(j))
        ^2*roadSpectrumSmooth(j);
    psdAccelerationRough(j,:) = abs(transferFunctionFrontZrToRide(j))
        ^2*roadSpectrumRough(j);
```

```
    psdAccelerationVeryRough(j,:) = abs(transferFunctionFrontZrToRide(j))^2*roadSpectrumVeryRough(j);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ii)
% Calculate rms values weighted according to ISO2631
[weightedRmsAccelerationSmooth] = CalculateIsoWeightedRms(frequencyVector,psdAccelerationSmooth)
[weightedRmsAccelerationRough] = CalculateIsoWeightedRms(frequencyVector,psdAccelerationRough)
[weightedRmsAccelerationVeryRough] = CalculateIsoWeightedRms(frequencyVector,psdAccelerationVeryRough)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% iii)
% Calculate time averaged vibration exposure value for 8h period

timeOnSmoothRoad = (distanceSmoothRoad / vehicleVelocitySmooth)*numberOfTripsPerDay;
timeOnRoughRoad = (distanceRoughRoad / vehicleVelocityRough)*numberOfTripsPerDay;
timeOnVeryRoughRoad = (distanceVeryRoughRoad / vehicleVelocityVeryRough)*numberOfTripsPerDay;

timeWeightedMsAcceleration = (weightedRmsAccelerationSmooth^2*timeOnSmoothRoad + ...
    weightedRmsAccelerationRough^2*timeOnRoughRoad + weightedRmsAccelerationVeryRough^2*timeOnVeryRoughRoad) ...
    / (timeOnSmoothRoad + timeOnRoughRoad + timeOnVeryRoughRoad);

timeWeightedRmsAcceleration = sqrt(timeWeightedMsAcceleration);

disp(['timeWeightedRmsAcceleration =' num2str(timeWeightedRmsAcceleration),' m/s2 (1.15)',...
    ', numberOfTripsPerDay = ' num2str(numberOfTripsPerDay)]);
disp(['vehicleVelocitySmooth = ' num2str(vehicleVelocitySmooth*3.6),' km/h',...
    ', vehicleVelocityRough = ' num2str(vehicleVelocityRough*3.6),' km/h',...
    ', vehicleVelocityVeryRough = ' num2str(vehicleVelocityVeryRough*3.6),' km/h'])
```