Midnight Sun Solar Car Team
University of Waterloo

MSXII Suspension Parameters

Prepared for Project 3 of ME321 Winter 2017

Prepared by:
Devon Copeland

April 3, 2017

Table 1: Important Vehicle Parameters

Mass	500	kg
Distance from Front Wheel to CoG	1.32	m
Moment of Inertia	550	kgm^2
Wheelbase	1.6	m

1 Background

1.1 Vehicle Design

Midnight Sun Twelve (MSXII) is a cruiser class solar electric vehicle being designed with the goal of competing in the 2018 American Solar Challenge (ASC 2018) and the 2019 World Solar Challenge (WSC 2019). Cruiser class vehicles are must be multi-occupant and are designed with the intent of being more practical than a typical solar car. Because of the unique requirements of this class, the spring rates and target damping coefficients on MSXII's suspension must be selected to optimize for efficient while still keeping driver comfort in mind. This report proposes and analytical technique for determining the response of the suspension system to a variety of road conditions.

1.2 Important Vehicle Parameters

Table 1 lists predicted vehicle parameters that are of importance to the design and analysis of the suspension.

1.3 Front Suspension

MSXII's front suspension comprises of a double wishbone linkage with an outboard coilover. Important dimensions required for subsequent analysis are shown in Figure **??**:

1.4 Rear Suspension

MSXII's rear suspension comprises of independent trailing arms allowing for zero scrub and thus less rolling resistance. Dimensions required for subsequent analysis are shown in Figure **??**:

1.5 Selecting Spring Rates

2 Suspension Model

2.1 Overview

MSXII's suspension is modelled using a point mass connected to two vertical, linear spring mass damper systems at the two wheels. This half car model assumes infinite tire stiffness and that the vehicle pitches about it's center of gravity. Figure **??** describes the proposed model where:

θ is the pitch angle about positive x axis
x_0 is the displacement of the center of gravity in the vertical axis
x_1 & x_2 are the displacements of the mounting points for the front and rear wheels respectively
y_1 & y_2 is the change in road elevation below the front and rear wheels respectively
M is the mass of the vehicle
I is the moment of inertia normal to the half car plane at the center of gravity
$k1, k2, c1$ & c_2 are the spring and damping coefficients
a & b are the distances from center of gravity to the front and rear wheels respectively

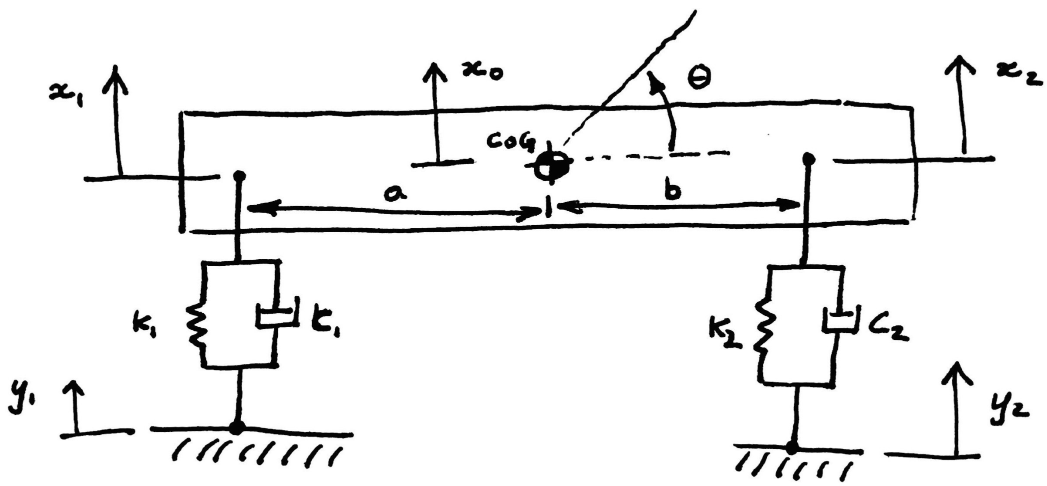

Figure 1: Half car model with infinite tire stiffness

2.2 Equations of Motion

The spring mass damper system shown in Figure **??** can be described by the folowing two differential equations:

$$M\ddot{x} + c_1(\dot{x}_1 - \dot{y}_1) + c_2(\dot{x}_2 - \dot{y}_2) + k_1(x_1 - y_1) + k_2(x_2 - y_2) = 0 \tag{1}$$

$$I\ddot{\theta} - ac_1(\dot{x}_1 - \dot{y}_1) + bc_2(\dot{x}_2 - \dot{y}_2) - ak_1(x_1 - y_1) + bk_2(x_2 - y_2) = 0 \tag{2}$$

Rearranging to move the inputs y_1 and y_2 to right hand side:

$$M\ddot{x} + c_1\dot{x}_1 + c_2\dot{x}_2 + k_1x_1 + k_2x_2 = c_1\dot{y}_1 + c_2\dot{y}_2 + k_1y_1 + k_2y_2 \tag{3}$$

$$I\ddot{\theta} - ac_1\dot{x}_1 + bc_2\dot{x}_2 - ak_1x_1 + bk_2x_2 = -ac_1\dot{y}_1 + bc_2\dot{y}_2 - ak_1y_1 + bk_2y_2 \tag{4}$$

By assuming small pitch angles, θ, the angle between the spring mass damper system becomes approximately perpendicular and the displacements x_1 and x_2 can be approximated as follows:

$$x_1 = x_0 - a\theta \tag{5}$$

$$x_2 = x_0 + b\theta \tag{6}$$

It follows that the time derivatives of the above two equations are:

$$\dot{x}_1 = x_0 - a\dot{\theta} \tag{7}$$

$$\dot{x}_2 = x_0 + b\dot{\theta} \tag{8}$$

Substituting into the original differential equations:

$$M\ddot{x} + \dot{x}_0(c_2 + c_1) + \dot{\theta}(bc_2 - ac_1) + x_0(k_2 + k_1) + \theta(bk_2 - ak_1) = c_1\dot{y}_1 + c_2\dot{y}_2 + k_1 y_1 + k_2 y_2 \tag{9}$$

$$I\ddot{\theta} + \dot{x}_0(bc_2 - ac_1) + \dot{\theta}(b^2c_2 - a^2c_1) + x_0(bk_2 - ak_1) + \theta(b^2k_2 + a^2k_1) = -ac_1\dot{y}_1 + bc_2\dot{y}_2 - ak_1y_1 + bk_2y_2 \tag{10}$$

By taking the unilateral laplace transform of the above two equations with all initial conditions set to zero, the following s domain equations are obtained:

$$\left[Ms^2 + (c_2 + c_1)s + (k_2 + k_1)\right] X_0 + \left[(bc_2 - ac_1)s + (bk_2 + ak_1)\right] \Theta = c_1sY_1 + c_2sY_2 + k_1Y_1 + k_2Y_2 \tag{11}$$

$$\left[(bc_2 - ac_1)s + (bk_2 - ak_1)\right] X_0 + \left[Is^2 + (b^2c_2 + a^2c_1)s + (b^2k_2 + a^2k_1)\right] \Theta = c_1sY_1 + c_2sY_2 + k_1Y_1 + k_2Y_2 \tag{12}$$

The above two equations can be expressed in matrix form as follows:

$$\begin{bmatrix} Q_{Ax}(s) & Q_{A\Theta}(s) \\ Q_{Bx}(s) & Q_{B\Theta}(s) \end{bmatrix} \begin{bmatrix} X_0 \\ \Theta \end{bmatrix} = \begin{bmatrix} P_{A1}(s) \\ P_{B1}(s) \end{bmatrix} Y_1 + \begin{bmatrix} P_{A2}(s) \\ P_{B2}(s) \end{bmatrix} Y_2 \tag{13}$$

If it is assumed that the rear tire of the car experiences the exact same road conditions as the front tire, $Y1$ and $Y2$ can be expressed as functions of the same input, Y where $Y2$ is simply a phase shifted version of $Y1$:

$$\begin{matrix} y_1(t) = y(t) \\ y_2(t) = y(t - t_0) \end{matrix} \implies \begin{matrix} Y_1(s) = Y(s) \\ Y_2(s) = Y(s)e^{-st_0} \end{matrix} \tag{14}$$

Equation 15 can then be rewritten as follows:

$$Q \begin{bmatrix} X_0 \\ \Theta \end{bmatrix} = \left([P_1] + [P_2]\, e^{-st_0}\right) Y \tag{15}$$

Since the system is linear linear, the above matrix equation can be solved in two parts to obtain the transfer functions for X_0 and Θ:

$$\frac{\begin{bmatrix} X_0 \\ \Theta \end{bmatrix}}{Y} = [H] = [H_1] + [H_2] = [Q^{-1}][P_1] + [Q^{-1}][P_1]e^{-st_0} \tag{16}$$

3 Analysis

3.1 Natural Frequency

Using the MATLAB script show in Appendix A, the transfer function $H(s)$ is solved for symbolically and the parameters for I, M, k_1, k_2, a and b are substituted into the equation while

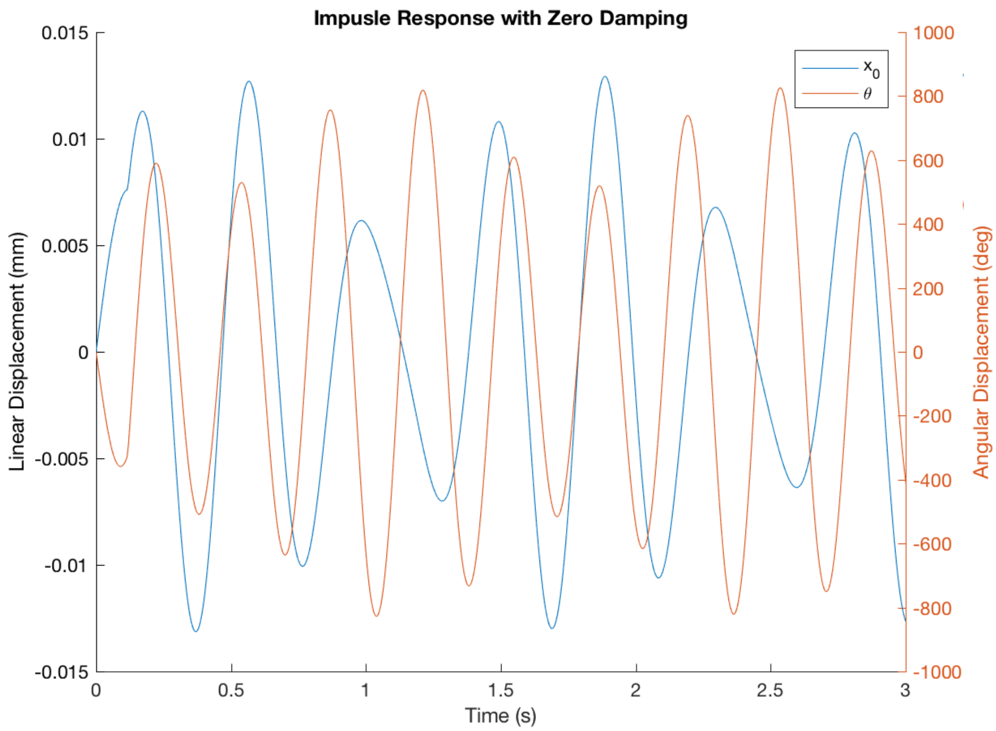

Figure 2: Undamped Impulse Response

the damping coefficients c_1 and c_2 are set to zero. The inverse laplace transform is then taken to obtain the impulse response as shown in Figure **??**.

By performing an Fast Fourier Transform on the undamped system the harmonics of the system can be approximated as shown in Figure **??**. A sampling frequency of 200Hz and and a window of 30s was used for the FFT. From figure **??**, the largest harmonics of the system occur at Hz for translational displacement and Hz for angular displacement.

3.2 Critical Damping

3.3 Force Response

3.4 Worst Case Total Response

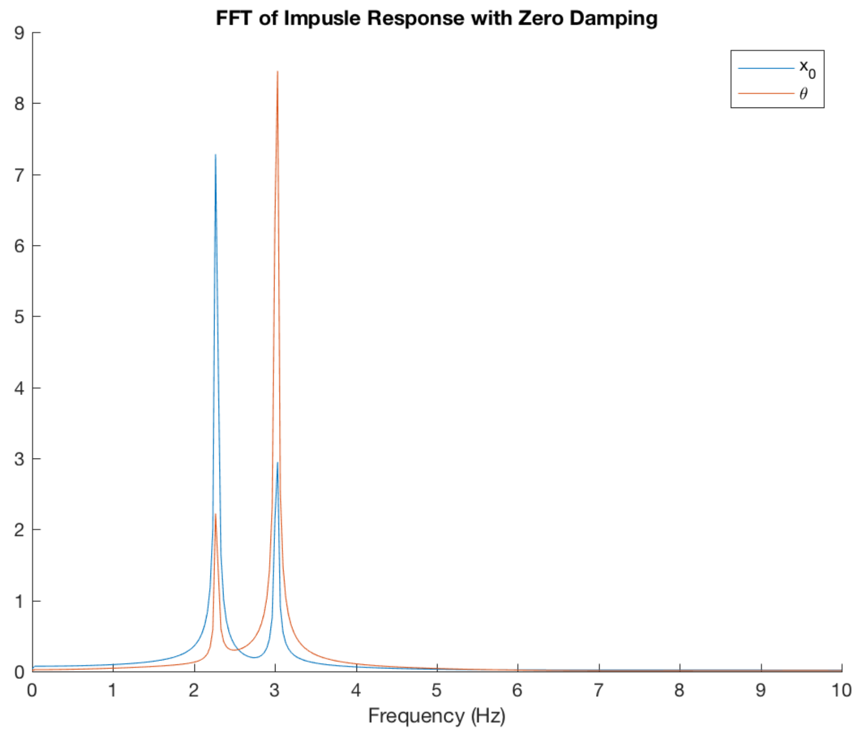

Figure 3: FFT of the Undamped Impulse Response

A MATLAB Source Code

```
% ------------------------------------------------------------------------ %
%
% Suspension Spring Rates Project - createTransferFuncitons.m
% Function setup transfer functions for a half car model to determine the
% reponse of MSXII to disturbances in the road conditions
%
% Author: Devon Copeland
%
%   Midnight Sun 2017
%
%
% Notes:
%
%   -
% ------------------------------------------------------------------------ %

load_system('halfCarModel');
close all;

% ------------------------------ Constants ------------------------------- %

c1_ = 0;      % Front damping coefficient (Ns/m)
c2_ = 0;      % Rear damping coefficient (Ns/m)
```

```
k1_ = 50000;     % Front sring rate (N/m)
k2_ = 60000;     % Rear sring rate (N/m)
m_ = 500;        % Mass of the vehicle (kg)
I_ = 550;        % Moment of interial about global x axis (kg*m^2)
a_ = 1.2;        % Distance from COG to front tire (m)
b_ = 1.4;        % Distance from COG to rear tire (m)

% ---------------------- Forcing Function Parameters ------------------- %

amplitude = 0.015;                % Amplitude of bumps on road surface
bumpSparation = 2;                % Distance between peaks on the road surface (m)
speed = 22.2;                     % Cruising speed (m/s)
wb = 2*pi*speed/bumpSparation;    % Frequency of base excitation (rad/sec)
timeDelay = (a_+b_)/speed;          % delay between input striking front and
                                  % rear wheels (s)

% Update simulink parameters
% set_param('halfCarModel/Road','Frequency',mat2str(wb),'Amplitude',mat2str(
    amplitude));
set_param('halfCarModel/DelayA','DelayTime',mat2str(timeDelay));
set_param('halfCarModel/DelayB','DelayTime',mat2str(timeDelay));

% ----------------- Create Symbolic Transfer Functions ----------------- %

syms t s c1 c2 k1 k2 m I a b t0 A w0

QA1 = m*s^2 + (c1+c2)*s + (k1+k2); % X terms of ODE (A)
QA2 = (b*c2-a*c1)*s + (b*k2-a*k1); % Theta terms of ODE (A)
QB1 = (b*c2-a*c1)*s + (b*k2-a*k1); % X terms of ODE (B)
QB2 = I*s^2 + (b^2*c2+a^2*c1)*s + (b^2*k2+a^2*k1); % Theta terms of ODE (B)

PA1 = (c1*s+k1); % Non time shifted componet of P for ODE (A) (front tire)
PA2 = (c2*s+k2); % Time shifted componet of P for ODE (A) (rear tire)
PB1 = (-a*c1*s-a*k1); % Non time shifted componet of P for ODE (B) (front tire)
PB2 = (b*c2*s+b*k2); % % Time shifted componet of P for ODE (B) (rear tire)

% Create linear systems in to solve for transfer functions

Q = [QA1, QA2;
     QB1, QB2];

P1 = [PA1;
      PB1];
P2 = [PA2;
      PB2];

H1 = linsolve(Q,P1);
H2 = linsolve(Q,P2);

% Input in frequency domain
phaseShift = exp(-s*timeDelay); % Time shift to be applied to P_2
input = (amplitude*wb)/(s^2+wb^2); % Laplace transform of input L{A*sin(wb*t))}

% -------------- Solve for response with zero damping ------------------ %

simplifiedH1 = simplify(collect(subs(H1, {k1 k2 c1 c2 m I a b}, {k1_ k2_ 0 0 m_
```

```
I_ a_ b_})));
simplifiedH2 = simplify(collect(subs(H2, {k1 k2 c1 c2 m I a b}, {k1_ k2_ 0 0 m_ I_ a_ b_})));
impulseResponse = vpa(simplify(ilaplace(simplifiedH1) + ilaplace(simplifiedH2*phaseShift)),3);

Fs = 200; % sampling frequency in Hz
T = 1/Fs; % sampling period
t_ = 0:T:30; % time values 0 to 30 seconds
L = length(t_);

% Numerical impulse response with zero damping
numericalImpulseResponse = double(subs(impulseResponse,t,t_));
figure;
hold on;
plot(t_,numericalImpulseResponse(1,:)./1000,'DisplayName','x_0');
ylabel('Linear Displacement (mm)');
yyaxis right
plot(t_,numericalImpulseResponse(2,:).*(180/pi),'DisplayName','\theta');
ylabel('Angular Displacement (deg)');
xlabel('Time (s)');
xlim([0 3]);
title('Impusle Response with Zero Damping');
legend show;
saveas(gcf,'impResp','png');

% Compute the first harmonic of the displacement
fftOut = fft(numericalImpulseResponse(1,:));
p2 = abs(fftOut/L);
p1 = p2(1:L/2+1);
p1(2:end-1) = 2*p1(2:end-1);
f = Fs*(0:(L/2))/L;
figure;
hold on
plot(f,p1,'DisplayName','x_0');

% FourierTransform
fftOut = fft(numericalImpulseResponse(2,:));
p2 = abs(fftOut/L);
p1 = p2(1:L/2+1);
p1(2:end-1) = 2*p1(2:end-1);
f = Fs*(0:(L/2))/L;
plot(f,p1,'DisplayName','\theta');

legend show
xlabel('Frequency (Hz)');
title('FFT of Impusle Response with Zero Damping');
xlim([0 10]);
saveas(gcf,'fftImpResp','png');

% -------------- Find Criticl Damping Coefficients -------------------- %

simplifiedH1 = simplify(collect(subs(H1, {k1 k2 m I a b}, {k1_ k2_ m_ I_ a_ b_})));
simplifiedH2 = simplify(collect(subs(H2, {k1 k2 m I a b}, {k1_ k2_ m_ I_ a_ b_})));
```

```
Fs = 50; % sampling frequency in Hz
T = 1/Fs; % sampling period
t_ = 0:T:5; % time values 0 to 5 seconds
L = length(t_);

% Spring rates to test:
dampingOptions = (100:500:5000); % N/m
[c1Options, c2Options] = meshgrid(dampingOptions, dampingOptions);

% Preallocate outputs:
pitchSettlingTime = zeros(size(c2Options));
xSettlingTime = pitchSettlingTime;

for i = 1:size(c1Options,1)
    for j = 1:size(c1Options,2)

        x1 = subs(simplify(collect(simplifiedH1(1))),{c1 c2},{c1Options(i,j) c2Options(i,j)});
        x2 = subs(simplify(collect(simplifiedH2(1))),{c1 c2},{c1Options(i,j) c2Options(i,j)});
        xWithSpring = vpa(simplify(simplify(ilaplace(simplify(x1))) + simplify(ilaplace(simplify(x2)))),3);

        pitch1 = subs(simplify(collect(simplifiedH1(2))),{c1 c2},{c1Options(i,j) c2Options(i,j)});
        pitch2 = subs(simplify(collect(simplifiedH2(2))),{c1 c2},{c1Options(i,j) c2Options(i,j)});
        pitchWithSpring = vpa(simplify(simplify(ilaplace(simplify(pitch1))) + simplify(ilaplace(simplify(pitch2.*phaseShift)))),3);

        % Compute response in the time domain
        xResponse = double(subs(xWithSpring,t,t_));
        pitchResponse = double(subs(pitchWithSpring,t,t_));

        % Record settling time
        xStepInfo = stepinfo(xResponse,t_,0);
        pitchStepInfo = stepinfo(pitchResponse,t_,0);
        xSettlingTime(i,j) = xStepInfo.SettlingTime;
        pitchSettlingTime(i,j) = pitchStepInfo.SettlingTime;
    end
end

figure;
surf(c1Options, c2Options, real(xSettlingTime));
figure;
surf(c1Options, c2Options, real(pitchSettlingTime));


% % --- Plot Magnitude of First Harmonic as a Funciton of Spring Rates --- %
%
% simplifiedH1 = subs(H1, {c1 c2 m I a b}, {0 0 m_ I_ a_ b_});
% simplifiedH2 = subs(H2, {c1 c2 m I a b}, {0 0 m_ I_ a_ b_});
%
% Fs = 200; % sampling frequency in Hz
% T = 1/Fs; % sampling period
```

```
% t_ = 0:T:5; % time values 0 to 5 seconds
% L = length(t_);
%
% % Spring rates to test:
% springRateOptions = (25000:20000:115000); % N/m
% [k1Options, k2Options] = meshgrid(springRateOptions, springRateOptions);
%
% % Preallocate outputs:
% pitchHarmonicFrequency = zeros(size(k2Options));
% xHarmonicFrequency = pitchHarmonicFrequency;
% pitchHarmonicMagnitude = pitchHarmonicFrequency;
% xHarmonicMagnitude = pitchHarmonicFrequency;
%
% for i = 1:size(k1Options,1)
%     for j = 1:size(k1Options,2)
%
%         x1 = subs(simplify(collect(simplifiedH1(1)*input)),{k1 k2},{k1Options(i,j) k2Options(i,j)});
%         x2 = subs(simplify(collect(simplifiedH2(1)*input)),{k1 k2},{k1Options(i,j) k2Options(i,j)});
%         xWithSpring = vpa(simplify(simplify(ilaplace(simplify(x1))) + simplify(ilaplace(simplify(x2)))),3);
%
%         pitch1 = subs(simplify(collect(simplifiedH1(2)*input)),{k1 k2},{k1Options(i,j) k2Options(i,j)});
%         pitch2 = subs(simplify(collect(simplifiedH2(2)*input)),{k1 k2},{k1Options(i,j) k2Options(i,j)});
%         pitchWithSpring = vpa(simplify(simplify(ilaplace(simplify(pitch1))) + simplify(ilaplace(simplify(pitch2)))),3);
%
%         % Compute response in the time domain
%         xResponse = double(subs(xWithSpring,t,t_));
%         pitchResponse = double(subs(pitchWithSpring,t,t_));
%
%         % Compute the first harmonic of the displacement
%         fftOut = fft(xResponse);
%         p2 = abs(fftOut/L);
%         p1 = p2(1:L/2+1);
%         p1(2:end-1) = 2*p1(2:end-1);
%         f = Fs*(0:(L/2))/L;
%         [magnitude, harmonicIndex] = max(p1);
%         xHarmonicFrequency(i,j) = f(harmonicIndex);
%         xHarmonicMagnitude(i,j) = magnitude;
%
%         hold on
%         plot(f,p1);
%
%         % Compute the first harmonic of the pitch
%         fftOut = fft(pitchResponse);
%         p2 = abs(fftOut/L);
%         p1 = p2(1:L/2+1);
%         p1(2:end-1) = 2*p1(2:end-1);
%         f = Fs*(0:(L/2))/L;
%         [magnitude, harmonicIndex] = max(p1);
%         pitchHarmonicFrequency(i,j) = f(harmonicIndex);
%         pitchHarmonicMagnitude(i,j) = magnitude;
```

```
%
%     end
% end
%
% figure;
% surf(k1Options, k2Options, xHarmonicFrequency);
% figure;
% surf(k1Options, k2Options, pitchHarmonicFrequency);
% figure;
% surf(k1Options, k2Options, xHarmonicMagnitude);
% figure;
% surf(k1Options, k2Options, pitchHarmonicMagnitude);

% ------------------------------ Simulink ---------------------------- %

numericalH1 = subs(H1, {c1 c2 k1 k2 m I a b}, {c1_ c2_ k1_ k2_ m_ I_ a_ b_});
numericalH2 = subs(H2, {c1 c2 k1 k2 m I a b}, {c1_ c2_ k1_ k2_ m_ I_ a_ b_});

[numH1, denomH1] = numden(numericalH1);
[numH2, denomH2] = numden(numericalH2);

transFuncA1 = tf(sym2poly(numH1(1)),sym2poly(denomH1(1)));
set_param('halfCarModel/HA1','Numerator',mat2str(sym2poly(numH1(1)))...
                            ,'Denominator',mat2str(sym2poly(denomH1(1))));

transFuncA2 = tf(sym2poly(numH2(1)),sym2poly(denomH2(1)));
set_param('halfCarModel/HA2','Numerator',mat2str(sym2poly(numH2(1)))...
                            ,'Denominator',mat2str(sym2poly(denomH2(1))));

transFuncB1 = tf(sym2poly(numH1(2)),sym2poly(denomH1(2)));
set_param('halfCarModel/HB1','Numerator',mat2str(sym2poly(numH1(2)))...
                            ,'Denominator',mat2str(sym2poly(denomH1(2))));

transFuncB2 = tf(sym2poly(numH2(2)),sym2poly(denomH2(2)));
set_param('halfCarModel/HB2','Numerator',mat2str(sym2poly(numH2(2)))...
                            ,'Denominator',mat2str(sym2poly(denomH2(2))));

% --------------- Solving for natural resonance frequency -------------- %
```