Mechanical Simulation Technical Memo

755 Phoenix Drive, Ann Arbor MI, 48108, USA
Phone: 734 668-2930 • Fax: 734 668-2877 • Email: info@carsim.com carsim.com

Linear Analysis in VehicleSim Models

Yukio Watanabe

VehicleSim (VS) products support a scripting capability through VS commands. This tech memo shows how to use VS commands to linearize the vehicle model in order to apply frequency-domain analyses such as bode and root locus plots. The general approach is illustrated using examples that are part of the BikeSim 3.2 database; this approach can also be used with CarSim 8.2 and TruckSim 8.2.

The VS linearization commands produce text files with linearized descriptions of the state of the math model. You will need the MATLAB software from The Math Works to analyze the linearized model. You will also need the MATLAB *Control System Toolbox* to view bode (frequency response) plots.

This memo assumes that you are familiar with the basic use of your VS product and with MATLAB. At a minimum, you should have gone through the Quick Start Guide. It is also helpful if you have some familiarity with the way VS solvers work, as documented in the VS Solvers Manual. The detailed reference material for VS commands is provided in the VS commands Reference Manual.

Linearization Overview

Prior to the availability of high-speed digital computers, researchers and engineers typically studied dynamic systems by analytic frequency-domain methods such as eigenvalues and bode plotting. These methods have been used in automotive dynamic analyses since the 1950's, and in motorcycle dynamic analyses since the 1970's. The classical frequency-domain methods are all based on system models with linear equations of motion.

Although modern nonlinear vehicle dynamics models such as those in BikeSim, CarSim, and TruckSim run much faster than real time, the linear analysis capability is still valuable for engineers, especially for motorcycle studies. This memo illustrates how VS solvers can linearize the system in support of classical analysis methods that are supported in MATLAB.

A *linear system* is one in which the amplitude of the output is linearly proportional to the amplitude of the input: twice the input makes twice the output; triple the input makes triple the output, etc. For example, a spring with stiffness K that produces a force F = K•x, where x is the spring deflection, is linear because the output F is always proportional to the input x.

Now consider a dynamic math model where the variables of interest are organized into three arrays: **x** is an array of all *n* state variables in the system, **y** is an array of *m* output variables, and **u** is an array of *r* control inputs. A linear dynamic system described with ordinary differential equations is typically represented with equations of the form:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)\end{aligned} \tag{1}$$

where **A**, **B**, **C**, and **D** are matrices whose dimensions are matched to the arrays: **A** is *n* x *n*, **B** is *n* rows x *r* columns, **C** is *m* x *n*, and **D** is *m* x *r*.

Note Other VS documents use different nomenclature for the arrays of variables in the math model. A convention in multibody dynamics is to break the state variables into two groups: generalized coordinates **q** and generalized speeds **u**. For this tech memo, the nomenclature of traditional system control is used, with all state variables in the array **x** and control inputs in the array **u**.

Traditional control theory defines frequency analysis and stability criteria that are inherent properties of linear systems. MATLAB has built-in functions for calculating frequency response characteristics when provided **A**, **B**, **C**, and **D** matrices from Eq. 1.

The equations of motion for VS vehicle math models are not linear. They contain nonlinear force equations, nonlinear motions involving trigonometric functions of angles, friction and other hysteresis, lock conditions, and many other physically realistic characteristics that are nonlinear. Multibody models such as those used to represent vehicle dynamics in VS products have nonlinear equations of motion expressed as:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}\{\mathbf{x}(t), \mathbf{u}(t)\} \\ \mathbf{y}(t) &= \mathbf{g}\{\mathbf{x}(t), \mathbf{u}(t)\}\end{aligned} \tag{2}$$

where **f** and **g** are nonlinear array functions of time, as well as being the model variables that are used to calculate the *n* derivatives of the state variables and the *m* output variables. Although the full model equations are nonlinear, the behavior of the full system can be approximated as a linear system at any time during a simulation run using *perturbation* (see the Appendix for details on the calculations). At any time during the simulation run, the `LINEARIZE` VS command can be used to apply a built-in perturbation algorithm to create **A**, **B**, **C**, and **D** matrices, and write them into a MATLAB M-file. After the simulation run has finished, M-files generated during the run can be loaded into MATLAB for analysis, as will be shown in following sections.

Example: Wobble, Weave, and Chatter of Motorcycle

Stability criteria and linear frequency response plots are widely used for characterizing motorcycle behavior. Given the natural ability of a motorcycle to fall over, stability has always been a larger concern than is the case with four-wheeled vehicles.

Most motorcycles have three vibration modes that have been identified in the literature:

1. **Wobble** is a vibration on the steer bar with a frequency of about 12 Hz. The wobble mode is typically the least stable at speeds around 70 km/h.
2. **Weave** is yaw motion like weaving with a frequency of about 3 Hz, which happens with high speed (> 200 km/h). It is more likely to be a problem with a heavier rear load, such as tandem or carrying heavy luggage.
3. **Chatter** is a vibration on the entire front fork in combination of steer, twist, and bending directions of motions that occurs with a frequency of about 25 Hz. Chatter is mainly seen at high speed with high leaning during cornering (> 200 km/h, > 50° lean angle) and can pose a serious stability issue for racing bikes such as MotoGP.

Note If you are already familiar with these vibration modes, you can skip to page 6: “Linearization with VS commands.”

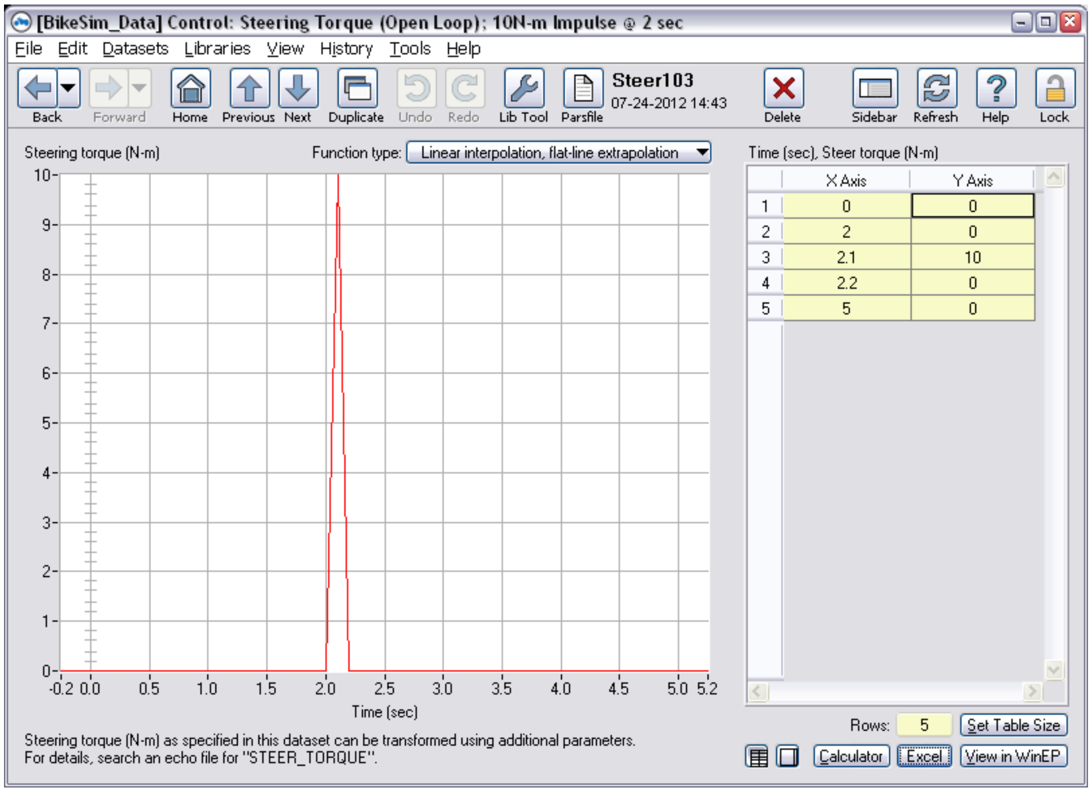
Wobble

Wobble motions can be seen in the example run **** Wobble w/Light Damped** in the category **Handling and Stability Tests**, available from the BikeSim Run Control screen. The bike has a lightly damped steering system and runs on a flat surface with a constant speed of 70 km/h. A spike torque is applied to the steer bar at 2 sec (Figure 1). Figure 2 shows that this causes an oscillation of the steer bar at about 13 Hz that continues until the simulation ends.

Figure 3 compares the wobble motion for four different vehicle speeds: 40, 70, 100 and 130 km/h, showing that the wobble motion at 70 km/h is unstable for this bike.

Weave

Weave motions occur in the example run **** Weave w/ Payload** in the category **Handling and Stability Tests**. The test maneuver is similar to the previous wobble test but the speed is higher (200 km/h) and the bike is set up to carry a 60 kg payload on the back seat. After a spike steer torque is applied at 2 sec, the oscillation of the steer bar diverges and entire chassis yaws with an oscillation at about 3 Hz (Figure 4).


[BikeSim_Data] Control: Steering Torque (Open Loop); 10N-m Impulse @ 2 sec
File Edit Datasets Libraries View History Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
Steer103
07-24-2012 14:43
Delete Sidebar Refresh Help Lock
Steering torque (N-m)
Time (sec)
Rows: 5 Set Table Size
Calculator Excel View in WinEP

Function type: Linear interpolation, flat-line extrapolation

Time (sec), Steer torque (N-m)

	X Axis	Y Axis
1	0	0
2	2	0
3	2.1	10
4	2.2	0
5	5	0

Steering torque (N-m) as specified in this dataset can be transformed using additional parameters.
For details, search an echo file for "STEER_TORQUE".

Figure 1. Spike steer torque to initiate the excitation of the steer vibration.

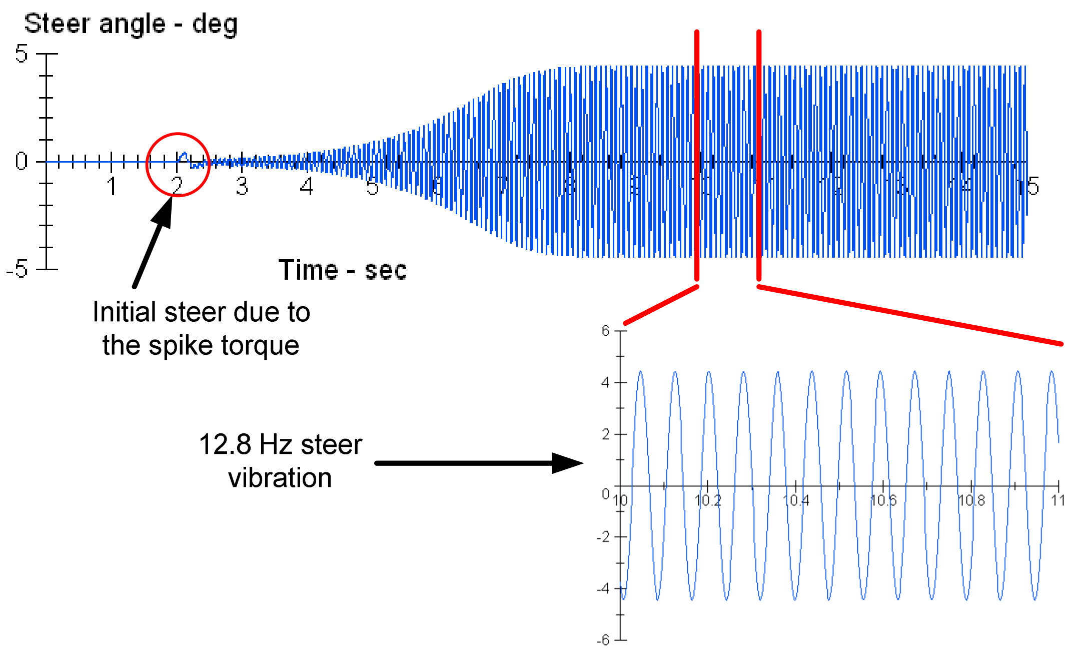


Figure 2. Wobble steer vibration (about 13 Hz) at 70 km/h.

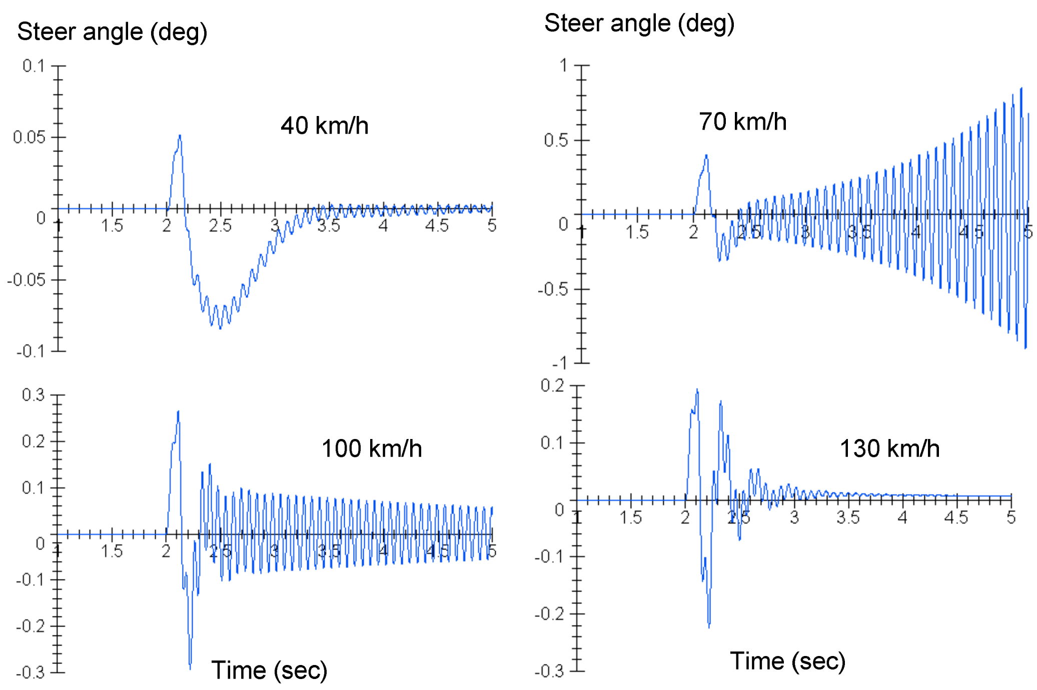


Figure 3. Wobble steer vibration at different vehicle speeds.

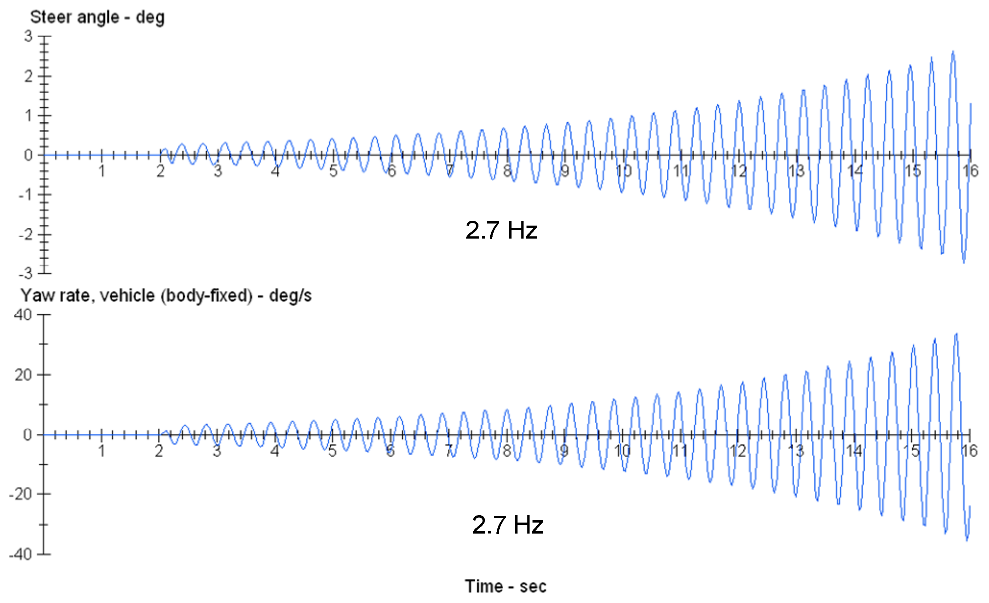


Figure 4. Weave motion (about 3 Hz) at 200 km/h.

Chatter

Chatter motions appear in the example run **** Chatter w/ Soft Frame** in the category **Handling and Stability Tests**. The frame stiffness and corresponding damping at the steering head were reduced for this simulation; the steer damping was also reduced for this example to demonstrate an instability in chatter. The test maneuver is more aggressive than the previous wobble/weave test – the bike runs on a flat surface at high speed (216 km/h) with a high lean angle of 53° and the lateral acceleration is about 1.1 g. In this test run, a spike torque is not applied to the steer bar like the previous tests, but instead the steer torque is controlled by a closed-loop rider model to maintain the lean angle.

In this maneuver, the entire front fork oscillates in three rotational directions (steer, twist and bend). These chattering oscillations occur at about 25 Hz (Figure 5).

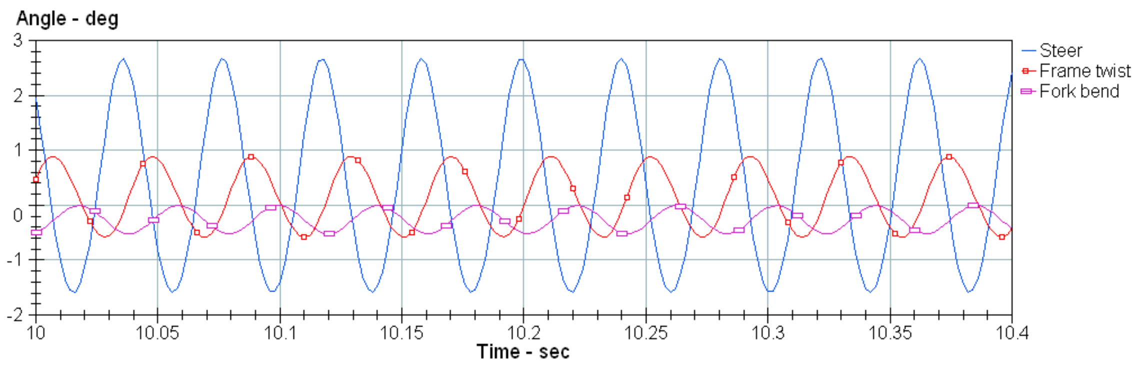


Figure 5. Chatter motion (about 25 Hz) at 216 km/h with 53° lean angle.

Linearization with VS Commands

Generally, there are two ways to obtain frequency-domain results from the VS vehicle model:

1. **Spectral analysis of time-domain data**. This is the method used for physically measured test data. Time histories of measured input and output variables are processed using numerical Fourier Transform tools. VS products include a basic spectrum analyzer. Alternatively, users can transfer simulation time histories into other software (e.g., MATLAB) and process the data the same as would be done with test results.

Linearization of the model. The VS model is linearized at points of interest during a run, and M-files are written for calculating the linear matrices **A**, **B**, **C**, and **D**. The M-files are later loaded into MATLAB to use built-in tools.

The second method is illustrated in the remainder of this memo.

Linearization options are controlled with four VS commands (Table 1), along with a parameter `OPT_LINEARIZATION`.

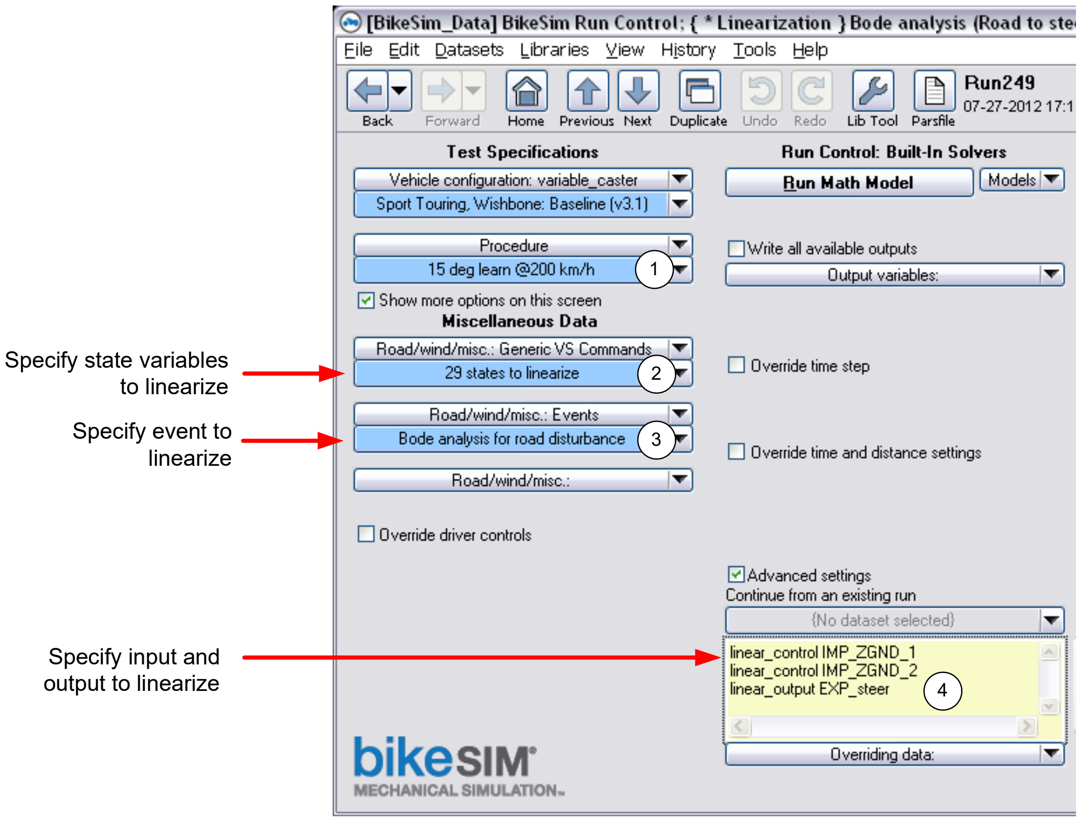
Table 1. VS commands used in linearization.

Command	Action
`LINEARIZE`	Trigger the linearization and write matrices to M-File.
`LINEAR_SV`	Specify a state variable to perturb and linearize.
`LINEAR_CONTROL`	Specify a control variable to perturb and linearize.
`LINEAR_OUTPUT`	Specify an output variable for linearization.

Recall that the linear matrices **A**, **B**, **C**, and **D** are defined based on three sets of variables: *n* state variables in the array **x**, *r* control inputs in the array **u**, and *m* output variables in the array **y.**

Use the command LINEAR_SV to identify variables in the math model that should be placed in the **x** array of the linearized system; use the command LINEAR_CONTROL to identify variables for the **u** array, and use the command LINEAR_OUTPUT to identify variables for the **y** array. Use the command LINEARIZE to generate a MATLAB M-file at any time during the simulation run.

The definitions of the three sets of variables (**x**, **y**, and **u**) are done for several examples in BikeSim that reproduce an analysis published by Sharp and others involving the wobble, weave, and chatter modes of vibration [1, 2]. One example is the dataset **Bode Analysis (Road to Steer)**, located in the category *** Linearization** in the **BikeSim Run Control** library (Figure 6).


Specify state variables to linearize
Specify event to linearize
Specify input and output to linearize
File Edit Datasets Libraries View History Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
Run Math Model
Models
bikeSIM MECHANICAL SIMULATION

[BikeSim_Data] BikeSim Run Control: { * Linearization } Bode analysis (Road to ste

Run249
07-27-2012 17:1

Test Specifications
Vehicle configuration: variable_caster
Sport Touring, Wishbone: Baseline (v3.1)
Procedure
15 deg lean @200 km/h (1)
[✓] Show more options on this screen
Miscellaneous Data
Road/wind/misc.: Generic VS Commands
29 states to linearize (2)
Road/wind/misc.: Events
Bode analysis for road disturbance (3)
Road/wind/misc.:
[ ] Override driver controls

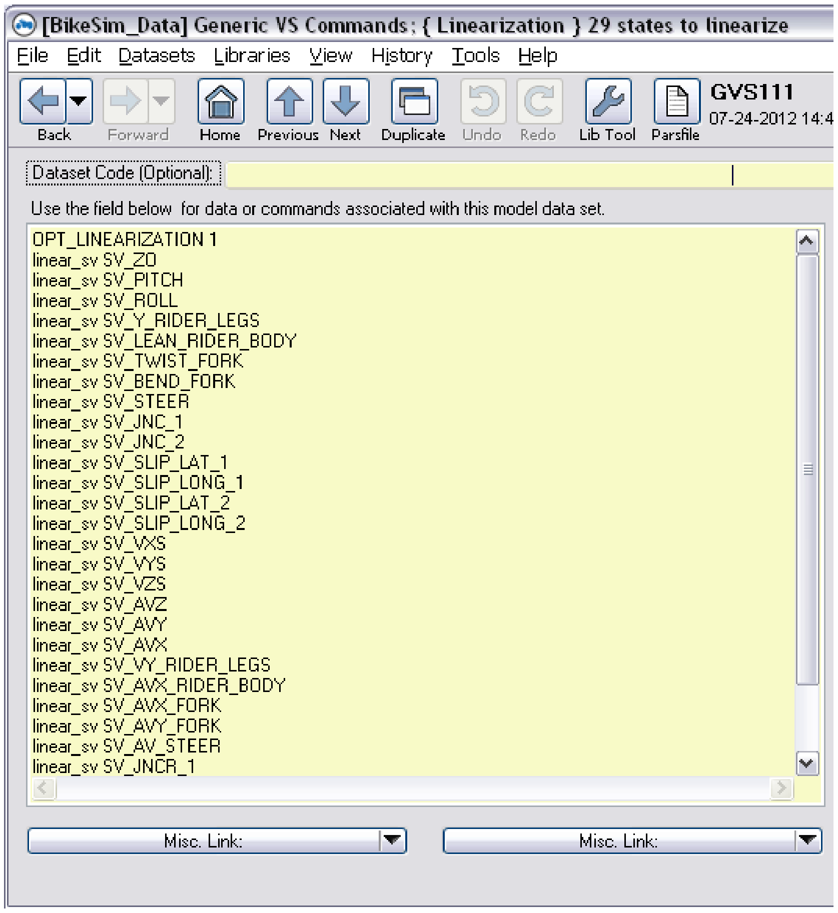
Run Control: Built-In Solvers
[ ] Write all available outputs
Output variables:
[ ] Override time step
[ ] Override time and distance settings
[✓] Advanced settings
Continue from an existing run
{No dataset selected}

```
linear_control IMP_ZGND_1
linear_control IMP_ZGND_2
linear_output EXP_steer
```
(4)
Overriding data:

Figure 6. Run control screen settings for bode analysis.

The linked procedure dataset (1) defines a test on a flat level surface with the speed controller maintaining a target speed of 200 km/h and the rider leaning 15°. A linked dataset with VS commands (2) identifies 29 state variables that will be used to define the linearized model (Figure 7).

VS solvers have two options for identifying the state variables, controlled by the parameter OPT_LINEARIZATION. The default (0) indicates that when a linearization occurs, all of the state variables of the VS math model are used as state variables for the linearized system. On the other hand, a non-zero value for OPT_LINEARIZATION specifies that only the state variables identified by the LINEAR_SV command be perturbed for the linearization. The BikeSim model includes 58 state variables, but many of them are irrelevant for the analysis of interest. For this example, the first line of input in Figure 7 enables the option to identify state variables. It is followed by 29 LINEAR_SV commands that identify the state variables of interest.


[BikeSim_Data] Generic VS Commands: { Linearization } 29 states to linearize
File Edit Datasets Libraries View History Tools Help
Back
Forward
Home
Previous
Next
Duplicate
Undo
Redo
Lib Tool
Parsfile
GVS111
07-24-2012 14:4
Dataset Code (Optional):
Use the field below for data or commands associated with this model data set.
Misc. Link:
Misc. Link:

```
OPT_LINEARIZATION 1
linear_sv SV_ZO
linear_sv SV_PITCH
linear_sv SV_ROLL
linear_sv SV_Y_RIDER_LEGS
linear_sv SV_LEAN_RIDER_BODY
linear_sv SV_TWIST_FORK
linear_sv SV_BEND_FORK
linear_sv SV_STEER
linear_sv SV_JNC_1
linear_sv SV_JNC_2
linear_sv SV_SLIP_LAT_1
linear_sv SV_SLIP_LONG_1
linear_sv SV_SLIP_LAT_2
linear_sv SV_SLIP_LONG_2
linear_sv SV_VXS
linear_sv SV_VYS
linear_sv SV_VZS
linear_sv SV_AVZ
linear_sv SV_AVY
linear_sv SV_AVX
linear_sv SV_VY_RIDER_LEGS
linear_sv SV_AVX_RIDER_BODY
linear_sv SV_AVX_FORK
linear_sv SV_AVY_FORK
linear_sv SV_AV_STEER
linear_sv SV_JNCR_1
```

Figure 7. Generic VS Commands screen specifying the state variables to linearize.

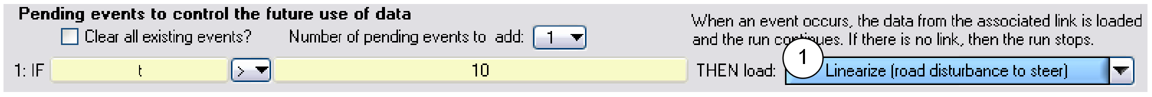
The input and output variables that will be used for the linearization are specified in the miscellaneous yellow field (4) of the **Run Control** screen (Figure 6). It contains the following commands to specify two input variables and one output variable.

```
linear_control IMP_ZGND_1
linear_control IMP_ZGND_2
linear_output EXP_steer
```

The command `LINEAR_CONTROL` is used twice to identify two import variables that are to be used as inputs for the linearization. These are the ground coordinates under each tire. The command `LINEAR_OUTPUT` identifies the steer angle as the output. (These are the same variables used by Sharp and others in the published analysis [1].)

In practice, the `LINEARIZE` command is nearly always applied when an event is triggered at a time of interest in the run. (If the command were applied at the start of the run, the linearization would occur at the initial time, when the vehicle system might not be in equilibrium.) Accordingly, all of the examples in BikeSim that demonstrate the `LINEARIZE` command also include at least one pending event. As will be seen later, some have a sequence of events to enable linearization at multiple times of interest.

For this example, a linked dataset ((3), Figure 6) defines a pending event (Figure 8). When the simulation time reaches 10 seconds, the VS solver loads a specified dataset (1).


Pending events to control the future use of data

Clear all existing events? Number of pending events to add: 1

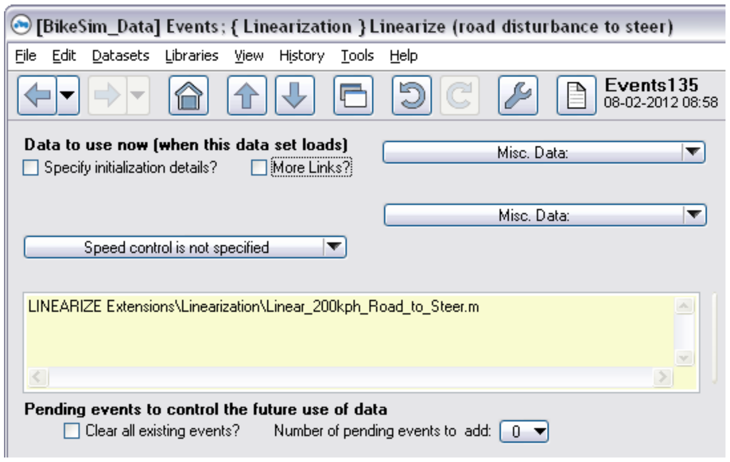
When an event occurs, the data from the associated link is loaded and the run continues. If there is no link, then the run stops.

1: IF t > 10 THEN load: Linearize (road disturbance to steer)

Figure 8. Pending Event to load a dataset at 10 seconds.

Figure 9 shows this linked dataset with a single line of input:

```
LINEARIZE Extensions\Linearization\Linear_200kph_Road_to_Steer.m
```


[BikeSim_Data] Events ; { Linearization } Linearize (road disturbance to steer)

Data to use now (when this data set loads)

Specify initialization details? More Links?

Misc. Data:

Misc. Data:

Speed control is not specified

```
LINEARIZE Extensions\Linearization\Linear_200kph_Road_to_Steer.m
```

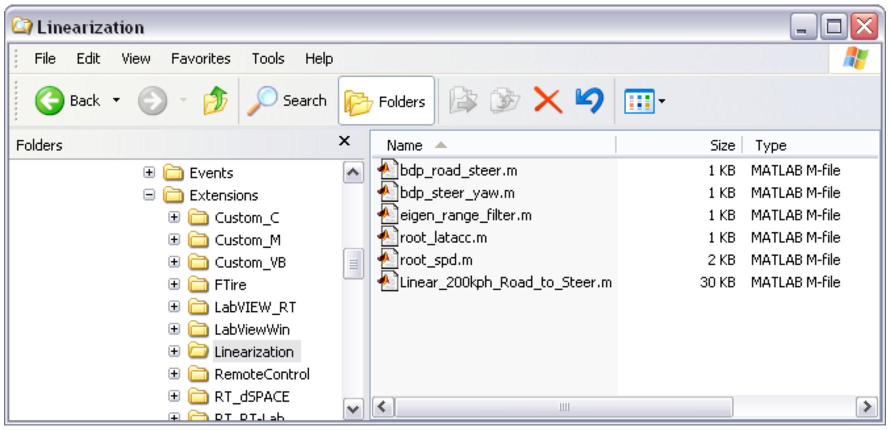
Pending events to control the future use of data

Clear all existing events? Number of pending events to add: 0

Figure 9. Events dataset with VS command to linearize the vehicle model.

The `LINEARIZE` command causes the VS solver to linearize the model and write the results into a MATLAB M-File with the specified name.

In this example, the file name is a relative pathname. The Windows working directory is set by BikeSim to be the currently active BikeSim database folder. Figure 10 shows the M-file made during the run.



Name	Size	Type
bdp_road_steer.m	1 KB	MATLAB M-file
bdp_steer_yaw.m	1 KB	MATLAB M-file
eigen_range_filter.m	1 KB	MATLAB M-file
root_latacc.m	1 KB	MATLAB M-file
root_spd.m	2 KB	MATLAB M-file
Linear_200kph_Road_to_Steer.m	30 KB	MATLAB M-file

Figure 10. Existing example files for the linear analysis and the M-Files created as a result of the linearization.

After the run is made, time-domain results can also be viewed using the plotting and animation capabilities in BikeSim. For example, Figure 11 shows the plots generated for this simulation.

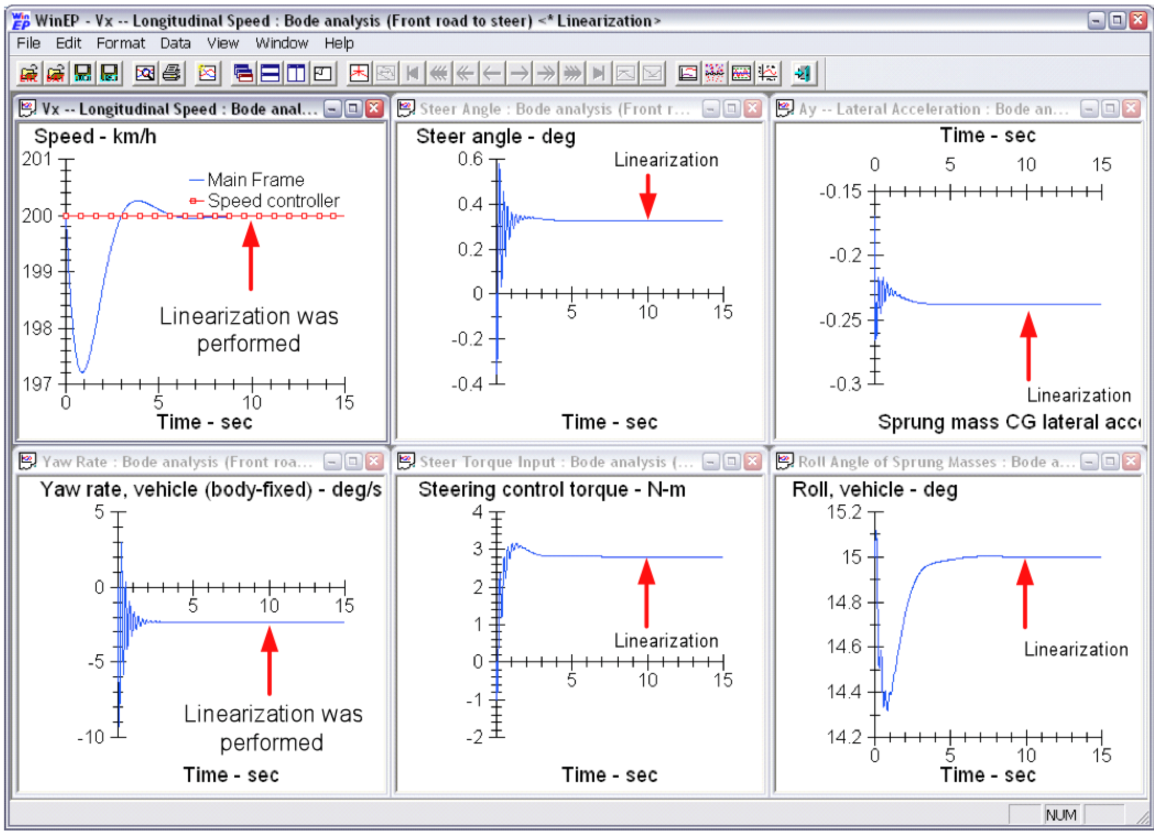

WinEP - Vx -- Longitudinal Speed : Bode analysis (Front road to steer) <* Linearization>
File Edit Format Data View Window Help
Vx -- Longitudinal Speed : Bode anal...
Speed - km/h
Main Frame
Speed controller
Linearization was performed
Time - sec
Steer Angle : Bode analysis (Front r...
Steer angle - deg
Linearization
Time - sec
Ay -- Lateral Acceleration : Bode an...
Time - sec
Linearization
Sprung mass CG lateral acc
Yaw Rate : Bode analysis (Front roa...
Yaw rate, vehicle (body-fixed) - deg/s
Linearization was performed
Time - sec
Steer Torque Input : Bode analysis (...
Steering control torque - N-m
Linearization
Time - sec
Roll Angle of Sprung Masses : Bode a...
Roll, vehicle - deg
Linearization
Time - sec
NUM

Figure 11. Time-history of the simulation with the linearization at 10 seconds.

The bike shows just an initial oscillation at the beginning of the run; it settles into a steady state after few seconds. The linearization was performed at 10 seconds (indicated by red arrows in the figure).

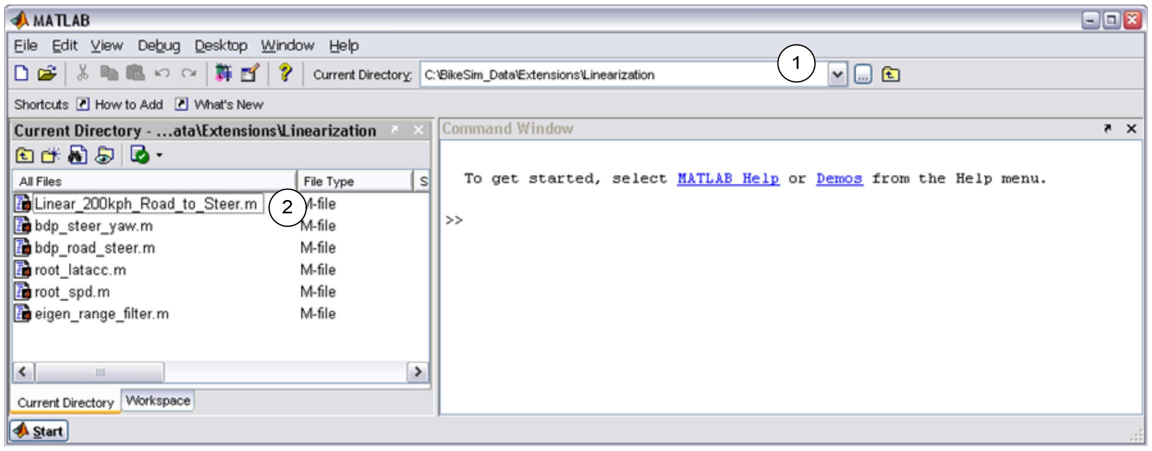
Note The linearization action involves writing the linear system matrices in a specified MATLAB M-File. It does not affect the time-domain simulation results. Therefore, you can perform the linearization at any instance during the simulation (also multiple times) without affecting time-history.

Loading the Matrices into MATLAB

The VS solver writes the instant values of the **A**, **B**, **C**, and **D** matrices for use in MATLAB.

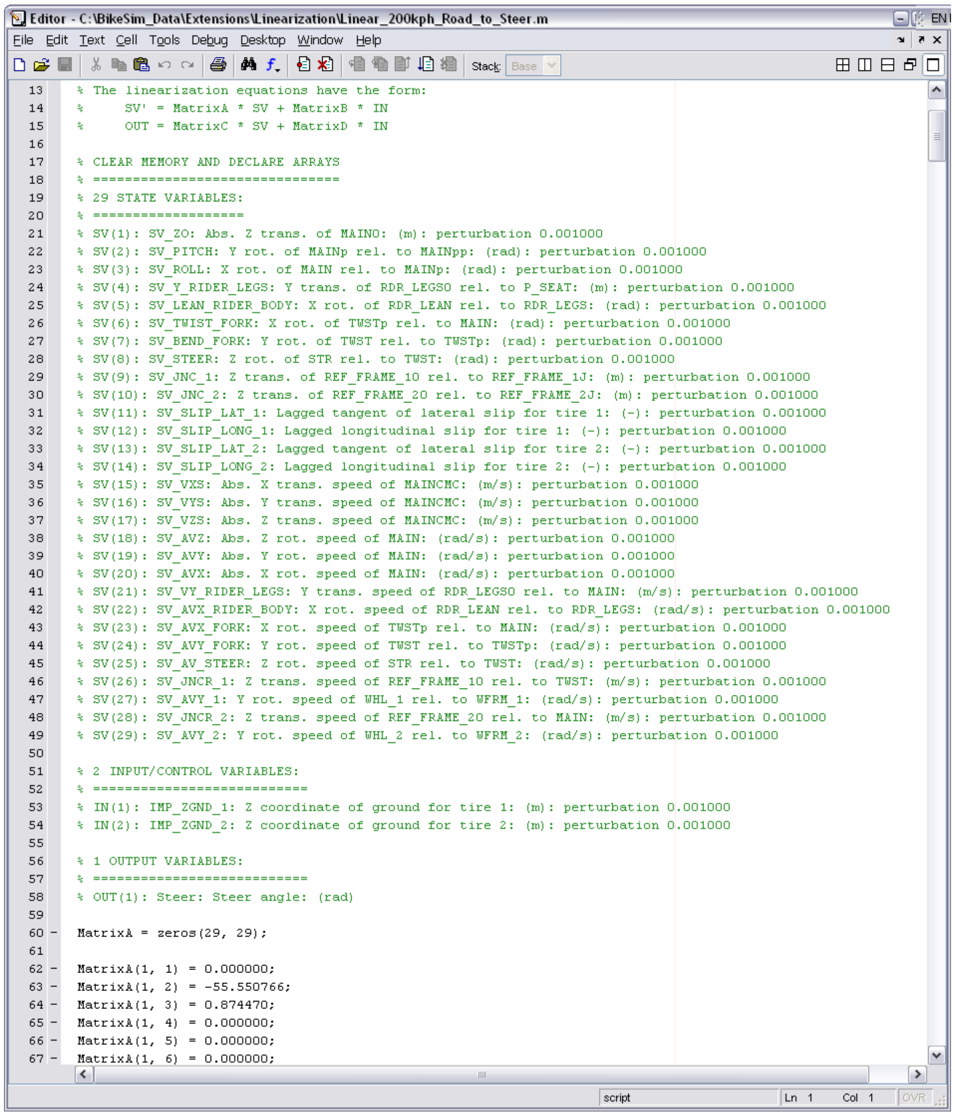
Start MATLAB and change the current directory to the `Linearization` folder where the M-file was written ① (Figure 12). You can see the contents the directory on the left-side panel ②. Double click on the name of the generated M-File (`Linear_200kph_Road_to_Steer.m`) to view it (Figure 13).

This M-File defines `MatrixA` (29×29 elements), `MatrixB` (29×2 elements), `MatrixC` (1×29 elements), and `MatrixD` (1×2). The contents of 29 state variables (`SV`), input (`IN`), and output variable (`OUT`) are described in comments at the beginning of the file.


MATLAB
File Edit View Debug Desktop Window Help
Current Directory: C:\BikeSim_Data\Extensions\Linearization
1
Shortcuts How to Add What's New
Current Directory - ...ata\Extensions\Linearization
All Files
File Type
Linear_200kph_Road_to_Steer.m
2
M-file
bdp_steer_yaw.m
M-file
bdp_road_steer.m
M-file
root_latacc.m
M-file
root_spd.m
M-file
eigen_range_filter.m
M-file
Current Directory
Workspace
Start
Command Window

```
To get started, select MATLAB Help or Demos from the Help menu.

>>
```

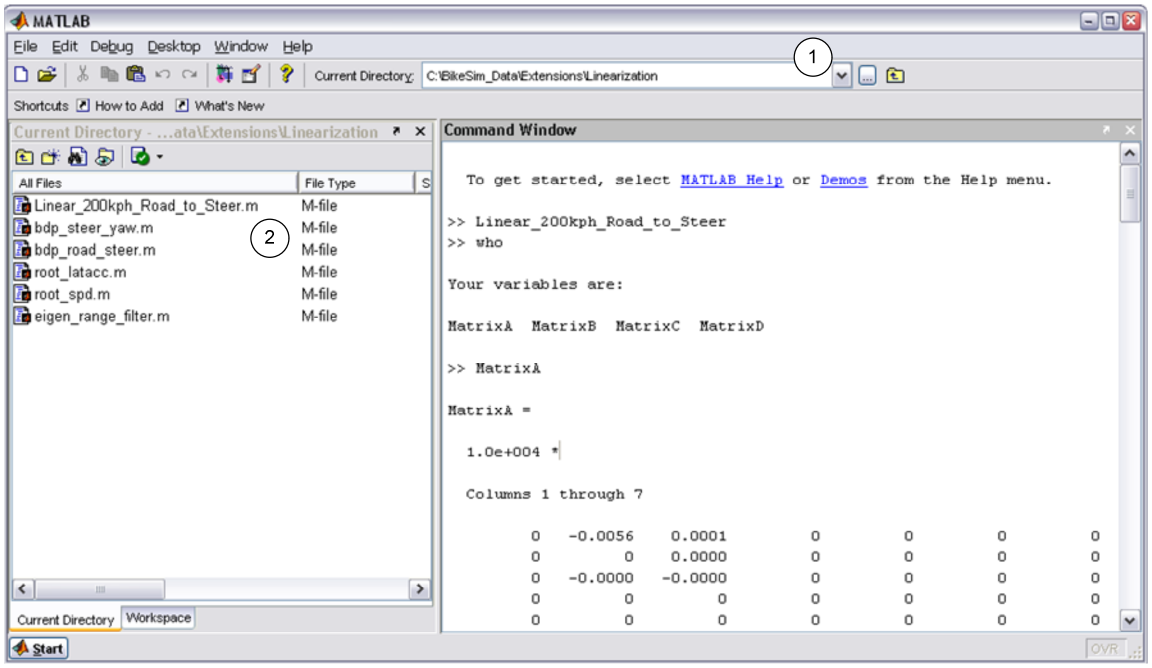
Figure 12. MATLAB command windows showing Linearization folder.


Editor - C:\BikeSim_Data\Extensions\Linearization\Linear_200kph_Road_to_Steer.m
File Edit Text Cell Tools Debug Desktop Window Help
Stack: Base
script
Ln 1
Col 1
OVR

```
% The linearization equations have the form:
%     SV' = MatrixA * SV + MatrixB * IN
%     OUT = MatrixC * SV + MatrixD * IN

% CLEAR MEMORY AND DECLARE ARRAYS
% ===============================
% 29 STATE VARIABLES:
% ===================
% SV(1): SV_ZO: Abs. Z trans. of MAINO: (m): perturbation 0.001000
% SV(2): SV_PITCH: Y rot. of MAINp rel. to MAINpp: (rad): perturbation 0.001000
% SV(3): SV_ROLL: X rot. of MAIN rel. to MAINp: (rad): perturbation 0.001000
% SV(4): SV_Y_RIDER_LEGS: Y trans. of RDR_LEGSO rel. to P_SEAT: (m): perturbation 0.001000
% SV(5): SV_LEAN_RIDER_BODY: X rot. of RDR_LEAN rel. to RDR_LEGS: (rad): perturbation 0.001000
% SV(6): SV_TWIST_FORK: X rot. of TWSTp rel. to MAIN: (rad): perturbation 0.001000
% SV(7): SV_BEND_FORK: Y rot. of TWST rel. to TWSTp: (rad): perturbation 0.001000
% SV(8): SV_STEER: Z rot. of STR rel. to TWST: (rad): perturbation 0.001000
% SV(9): SV_JNC_1: Z trans. of REF_FRAME_10 rel. to REF_FRAME_1J: (m): perturbation 0.001000
% SV(10): SV_JNC_2: Z trans. of REF_FRAME_20 rel. to REF_FRAME_2J: (m): perturbation 0.001000
% SV(11): SV_SLIP_LAT_1: Lagged tangent of lateral slip for tire 1: (-): perturbation 0.001000
% SV(12): SV_SLIP_LONG_1: Lagged longitudinal slip for tire 1: (-): perturbation 0.001000
% SV(13): SV_SLIP_LAT_2: Lagged tangent of lateral slip for tire 2: (-): perturbation 0.001000
% SV(14): SV_SLIP_LONG_2: Lagged longitudinal slip for tire 2: (-): perturbation 0.001000
% SV(15): SV_VXS: Abs. X trans. speed of MAINCMC: (m/s): perturbation 0.001000
% SV(16): SV_VYS: Abs. Y trans. speed of MAINCMC: (m/s): perturbation 0.001000
% SV(17): SV_VZS: Abs. Z trans. speed of MAINCMC: (m/s): perturbation 0.001000
% SV(18): SV_AVZ: Abs. Z rot. speed of MAIN: (rad/s): perturbation 0.001000
% SV(19): SV_AVY: Abs. Y rot. speed of MAIN: (rad/s): perturbation 0.001000
% SV(20): SV_AVX: Abs. X rot. speed of MAIN: (rad/s): perturbation 0.001000
% SV(21): SV_VY_RIDER_LEGS: Y trans. speed of RDR_LEGSO rel. to MAIN: (m/s): perturbation 0.001000
% SV(22): SV_AVX_RIDER_BODY: X rot. speed of RDR_LEAN rel. to RDR_LEGS: (rad/s): perturbation 0.001000
% SV(23): SV_AVX_FORK: X rot. speed of TWSTp rel. to MAIN: (rad/s): perturbation 0.001000
% SV(24): SV_AVY_FORK: Y rot. speed of TWST rel. to TWSTp: (rad/s): perturbation 0.001000
% SV(25): SV_AV_STEER: Z rot. speed of STR rel. to TWST: (rad/s): perturbation 0.001000
% SV(26): SV_JNCR_1: Z trans. speed of REF_FRAME_10 rel. to TWST: (m/s): perturbation 0.001000
% SV(27): SV_AVY_1: Y rot. speed of WHL_1 rel. to WFRM_1: (rad/s): perturbation 0.001000
% SV(28): SV_JNCR_2: Z trans. speed of REF_FRAME_20 rel. to MAIN: (m/s): perturbation 0.001000
% SV(29): SV_AVY_2: Y rot. speed of WHL_2 rel. to WFRM_2: (rad/s): perturbation 0.001000

% 2 INPUT/CONTROL VARIABLES:
% ==========================
% IN(1): IMP_ZGND_1: Z coordinate of ground for tire 1: (m): perturbation 0.001000
% IN(2): IMP_ZGND_2: Z coordinate of ground for tire 2: (m): perturbation 0.001000

% 1 OUTPUT VARIABLES:
% ==========================
% OUT(1): Steer: Steer angle: (rad)

MatrixA = zeros(29, 29);

MatrixA(1, 1) = 0.000000;
MatrixA(1, 2) = -55.550766;
MatrixA(1, 3) = 0.874470;
MatrixA(1, 4) = 0.000000;
MatrixA(1, 5) = 0.000000;
MatrixA(1, 6) = 0.000000;
```

Figure 13. Linear system matrices written in MATLAB M-File.

Load the file into MATLAB by using the file name as a command: `Linear_200kph_Road_to_Steer` (Figure 14). At this point, the four matrices from the linearized Bike model are loaded into MATLAB.


MATLAB
File Edit Debug Desktop Window Help
Shortcuts How to Add What's New
Current Directory - ...ata\Extensions\Linearization
Command Window
1
2
Current Directory Workspace
Start
OVR

Current Directory: C:\BikeSim_Data\Extensions\Linearization

All Files	File Type
Linear_200kph_Road_to_Steer.m	M-file
bdp_steer_yaw.m	M-file
bdp_road_steer.m	M-file
root_latacc.m	M-file
root_spd.m	M-file
eigen_range_filter.m	M-file

```
  To get started, select MATLAB Help or Demos from the Help menu.

>> Linear_200kph_Road_to_Steer
>> who

Your variables are:

MatrixA  MatrixB  MatrixC  MatrixD

>> MatrixA

MatrixA =

  1.0e+004 *

  Columns 1 through 7

         0   -0.0056    0.0001         0         0         0         0
         0         0    0.0000         0         0         0         0
         0   -0.0000   -0.0000         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
```

Figure 14. Loading system matrices on MATLAB command window.

Type `who` and press the Enter key to confirm that the matrices have been defined (Figure 14). Type `MatrixA` and press Enter to see all the numerical values of the **A** matrix elements.

Note Each element shown in the lists of the state variables and control inputs (Figure 13) are indicated with `perturbation 0.001`. This perturbation is Δx and Δu in the equations presented in the Appendix. Different perturbation amounts can be set for each variable defined by VS commands `LINEAR_SV` and `LINEAR_CONTROL`, putting the value after the variable name, e.g.,

```
linear_sv SV_ZO 0.002
linear_control IMP_ZGND_1 0.005
```

If the value is not indicated, a default value (0.001) is applied automatically.

Now that the matrices for the linearized system are loaded into MATLAB, built-in functions of MATLAB can be used to characterize the dynamics. For example, enter the command `eig(MatrixA)` to view the eigenvalues of the system.

The following sections describe the methods used to generate graphs showing dynamic properties of the bike system in the frequency domain.

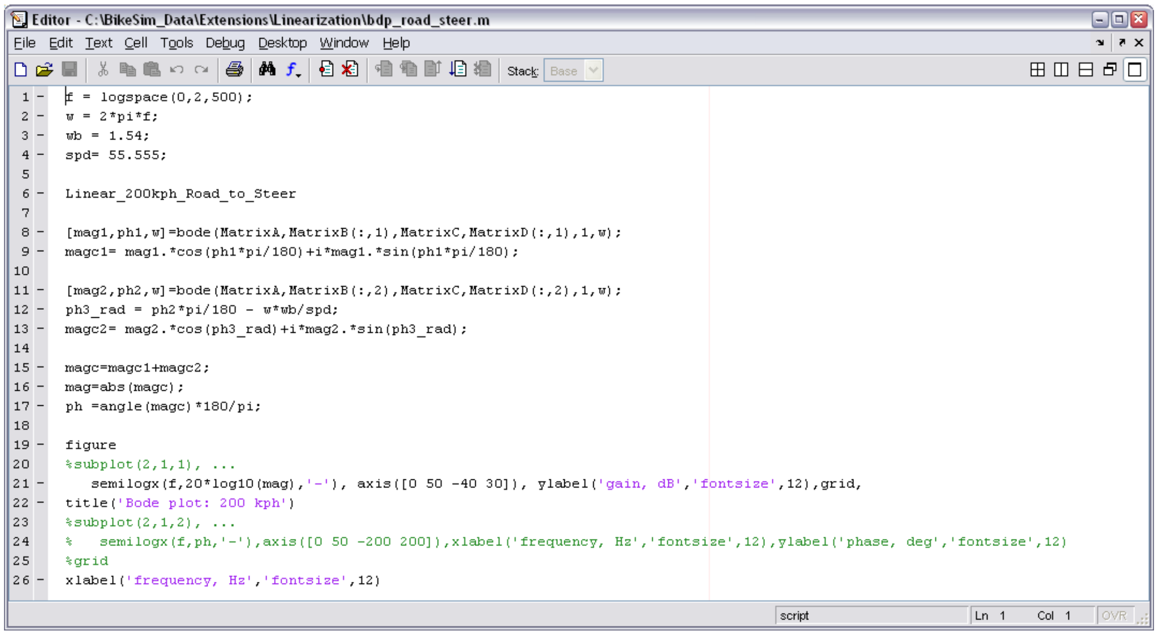
Bode Plot (Frequency Response)

Suppose an input control to a linear dynamic system is a sinusoidal wave. Because the system is linear, the output will also be a sinusoid with the same frequency, but with a different amplitude and phase. The ratio of the output sinusoidal amplitude to the input amplitude defines a *gain*, and the difference of phase angle defines a *phase lag* between input and output. Because the system is linear, the gain and phase lag do not depend on the amplitude of the input. If the frequency of the input sinusoid is changed, the gain and phase lag might change. The measures of gain and phase lag as functions of frequency are called the *frequency response*; the frequency response diagram showing gain and phase lag versus the frequency is called a *bode plot*.

Although the simulation made for this example did not have sinusoidal inputs, the **A**, **B**, **C**, and **D** matrices can be used to generate bode plots in MATLAB.

Note You will need the MATLAB *Control System Toolbox* to view frequency response plots.

The BikeSim database includes some example M-files with scripts to perform the frequency-domain analyses described in this memo (see Figure 12). For example, the file `bdp_road_steer.m` will perform the calculations needed to generate the bode plot showing the transfer function between ground elevation and steering. Double-click on the file name to view the contents (Figure 15).


Editor - C:\BikeSim_Data\Extensions\Linearization\bdp_road_steer.m
File Edit Text Cell Tools Debug Desktop Window Help
Stack: Base
script
Ln 1 Col 1
OVR

```
f = logspace(0,2,500);
w = 2*pi*f;
wb = 1.54;
spd= 55.555;

Linear_200kph_Road_to_Steer

[mag1,ph1,w]=bode(MatrixA,MatrixB(:,1),MatrixC,MatrixD(:,1),1,w);
magc1= mag1.*cos(ph1*pi/180)+i*mag1.*sin(ph1*pi/180);

[mag2,ph2,w]=bode(MatrixA,MatrixB(:,2),MatrixC,MatrixD(:,2),1,w);
ph3_rad = ph2*pi/180 - w*wb/spd;
magc2= mag2.*cos(ph3_rad)+i*mag2.*sin(ph3_rad);

magc=magc1+magc2;
mag=abs(magc);
ph =angle(magc)*180/pi;

figure
%subplot(2,1,1), ...
   semilogx(f,20*log10(mag),'-'), axis([0 50 -40 30]), ylabel('gain, dB','fontsize',12),grid,
title('Bode plot: 200 kph')
%subplot(2,1,2), ...
%   semilogx(f,ph,'-'),axis([0 50 -200 200]),xlabel('frequency, Hz','fontsize',12),ylabel('phase, deg','fontsize',12)
%grid
xlabel('frequency, Hz','fontsize',12)
```

Figure 15. Bode plotting script (MATLAB M-File).

This file loads `Linear_200kph_Road_to_Steer.m` (just in case it's not already in memory) to obtain the four system matrices and then uses the MATLAB `bode` function to calculate the magnitude and phase lag of the steer angle to the road elevation under each tire.

There is a complication when using road elevation as an input, because the elevation seen at the rear wheel is from the same ground as seen at the front wheel, but with a delay due to the wheelbase of the bike. Accounting for the tire delay with the so-called wheelbase filtering effect combines the two responses.

Run the file `bpd_road_steer` by typing the name in the command window and pressing the Enter key. This will display the bode plot (Figure 16) showing that resonance of the cornering weave is evident at 3.2 Hz; the wobble is most responsive at 12.5 Hz; and the chatter is observed at 31.8 Hz.

Recall that in the earlier subsection involving time-domain analyses showed special runs in which the bike properties were modified to illustrate each mode of vibration. In this case, the frequencies of the three modes are easily seen in the bode plot for the same test condition.

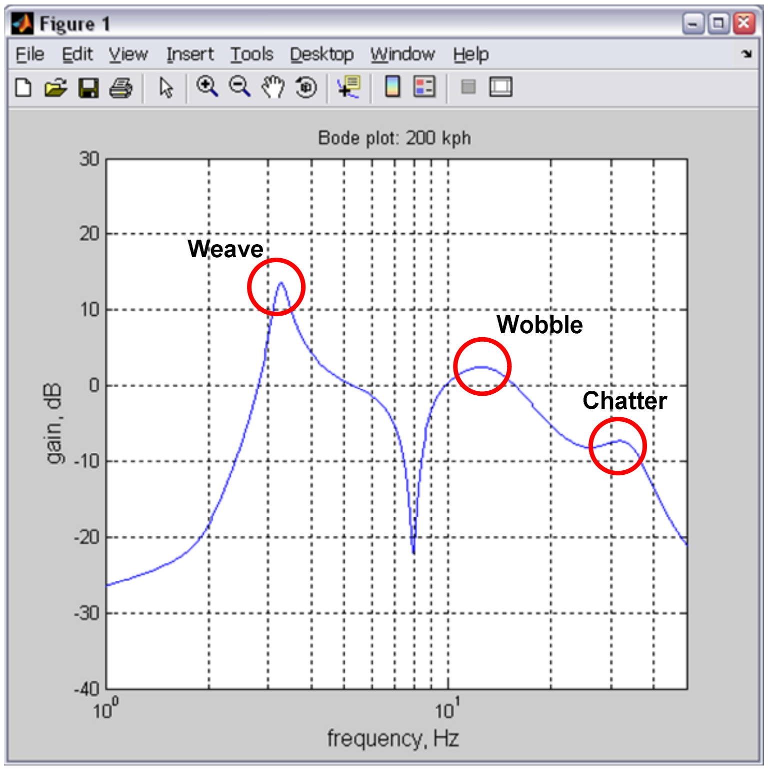


Figure 16. Observed weave, wobble and chatter on bode plot.

Eigenvalues: Root Locus for Speed Sensitivity

As noted earlier, the vibration frequencies of the system can be calculated in MATLAB with the command: `eig(MatrixA)`.

The three vibration modes of interest (wobble, weave and chatter) are speed dependent. An established characterization of motorcycle dynamics is to show how the eigenvalues vary with speed using a root locus plot.

In this section, a series of events are performed in BikeSim to generate linear matrices with different vehicle speeds. This allows eigenvalues associated with various speeds to be plotted on a complex plane using MATLAB.

> **Note** The eigenvalue calculations used in this section run in basic MATLAB; no extra toolboxes are required.

A Sequence of Steady-State Tests with Increasing Speed

In the previous example, a single linearization was done after the bike reached a steady-state condition at 10s. For the purpose of generating a speed-dependent root locus, we want to run a series of steady-state tests at increasing speeds. This is done in a single run in BikeSim using the Events capability.

The method is used for a dataset in the **Run Control** library named **Steady-State Straight (Root Locus)** in the category *** Linearization**.

The eigenvalue analysis requires only the **A** matrix. Therefore, input and output variables need not be identified. The dataset used in the previous section to identify 29 state variables is used again for this run.

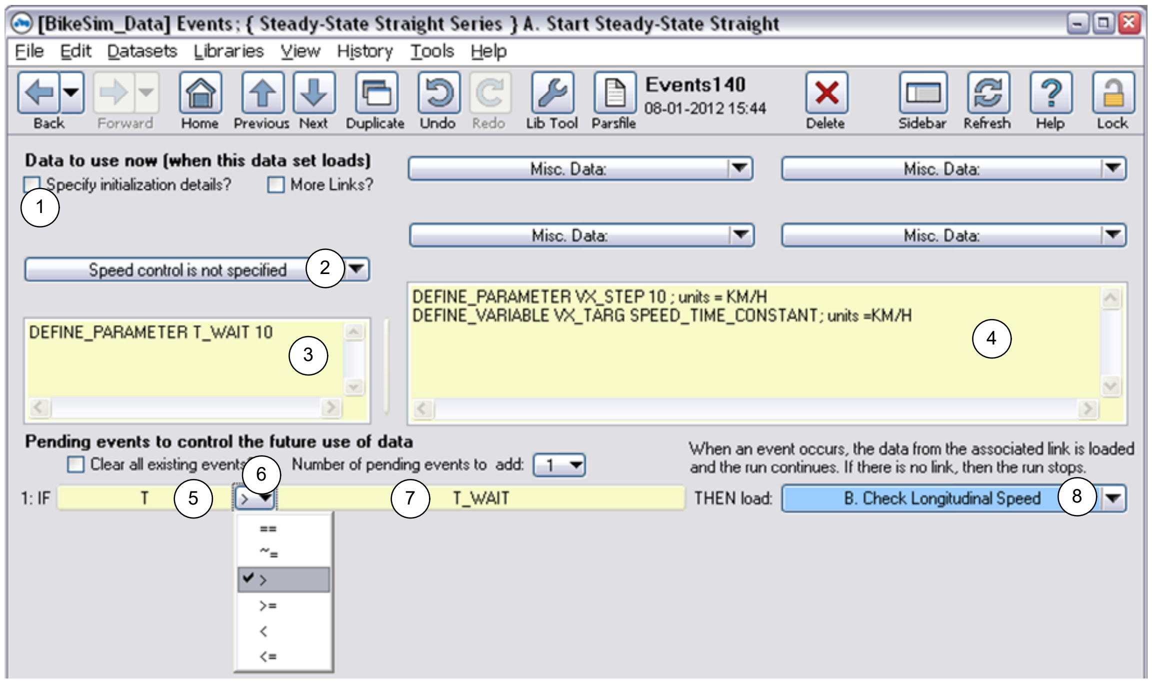
For this run, a **Procedure** dataset sets a closed-loop speed control with an initial target vehicle speed as 0 km/h, with no steer or braking. As before, the ground surface is flat and level. This dataset in turn has a link to a dataset from the **Events** library that is used to initiate the running of the steady-state straight tests (Figure 17).

This is the first of a series of **Events** datasets that will run multiple steady-state straight tests as summarized in Table 2. Note that the steps define a loop (step D goes back to step B), and that the run ends based on a condition checked in step B.

The actions summarized in this table are described in detail in the following pages.

> **Note** When a procedure is set up to use a series of **Events** datasets, it is common to name the events with a title that begins with capital letters (A, B, …) to indicate the sequence. For this example, Table 2 shows that there is a series of four datasets (A – D). All of these datasets are in the category **Steady-State Straight Series.**

The first **Events** dataset in the series (Figure 17) provides information that must be processed by the VS math model before the run starts.


[BikeSim_Data] Events: { Steady-State Straight Series } A. Start Steady-State Straight
File Edit Datasets Libraries View History Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
Events140
08-01-2012 15:44
Delete Sidebar Refresh Help Lock
1
2
3
4
5
6
7
8
==
~=
>
>=
<
<=

Data to use now (when this data set loads)

☐ Specify initialization details? ☐ More Links?

Misc. Data: | Misc. Data: | Misc. Data: | Misc. Data:

Speed control is not specified

```
DEFINE_PARAMETER T_WAIT 10
```

```
DEFINE_PARAMETER VX_STEP 10 ; units = KM/H
DEFINE_VARIABLE VX_TARG SPEED_TIME_CONSTANT; units =KM/H
```

Pending events to control the future use of data

☐ Clear all existing events Number of pending events to add: 1

When an event occurs, the data from the associated link is loaded and the run continues. If there is no link, then the run stops.

1: IF T > T_WAIT THEN load: B. Check Longitudinal Speed

Figure 17. Defining variables and waiting for 10s for the next event.

Table 2. Series of ***Events*** *datasets used to run multiple steady-state straight tests.*

Dataset Title	**Actions**
A. Start Steady-State Straight	Define new parameters and variables that will be used later. Create a pending event to monitor the simulation time to wait long enough for the vehicle to be in steady state for stopping (0 km/h.) When the simulation time reaches the specified limit (10 seconds), go to step B.
B. Check Longitudinal Speed	Compare the longitudinal speed to the target speed. Either if they are not within a tolerance or exceeding maximum speed, end the run. Otherwise, go to step C.
C. Write M-File	Linearization is performed at this point; write the linear system matrices to M-File; clean up a pending event from step B; then go to step D.
D. Go to Next Speed	Increment the target longitudinal speed. Reset a local clock and monitor the local time to wait for the vehicle to be in steady state for the new speed. When the local time reaches the specified limit, go back to step B.

The main reason for loading a dataset from the **Events** library is to set up one or more pending events. A pending event is defined with a *Boolean* expression (a logical expression that must have a value of true or false) of the form:

variable operator threshold

where *variable* is any parameter or variable in the math model that can be identified with a keyword (e.g., simulation time, T ⑤); *operator* is a comparison operator available from a pulldown list ⑥; and *threshold* is any expression that can be evaluated by the VS solver (e.g., `T_WAIT` ⑦). The *threshold* can be a number, a symbol, or an algebraic expression.

As the simulation proceeds, all pending events are evaluated at each time step. The *threshold* expression is evaluated using current values of any parameters or variables included in the expression, and the result is compared to the current value of *variable*. If the Boolean expression is false, no action is taken. However, if the Boolean expression is true, then the event is *triggered* and one of two possible actions is taken:

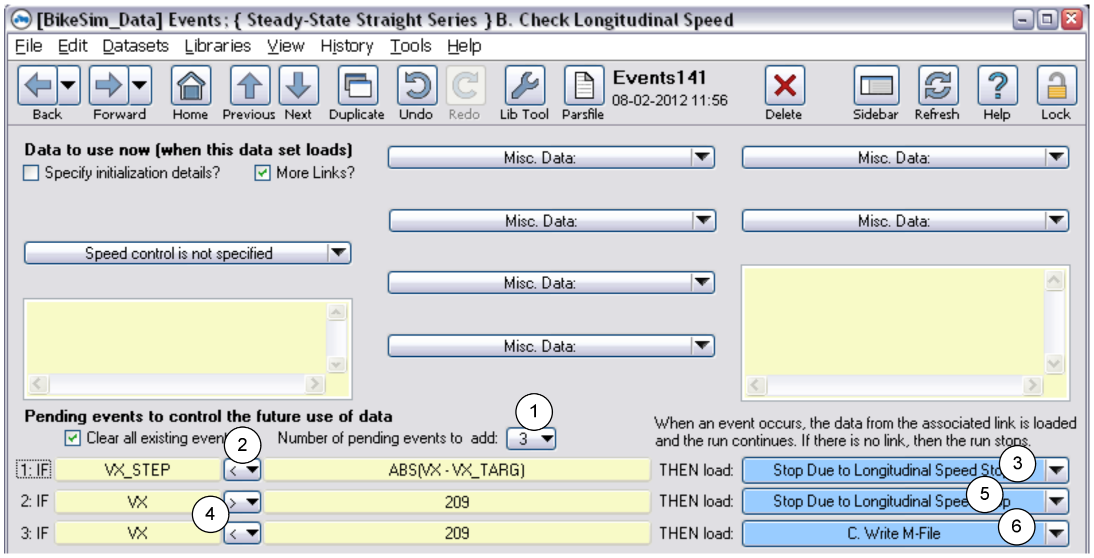
1. If another dataset is specified (e.g., **B. Check Longitudinal Speed** ⑧), then that dataset is read by the VS solver and processed, and the run continues using any changes in the vehicle properties or test conditions that are specified in the new dataset.
2. On the other hand, if there is no linked dataset associated with the pending event, then the simulation run is terminated.

The VS Commands Reference Manual provides more information about VS Events.

The parameter `T_WAIT` is not built into BikeSim, CarSim, or TruckSim; it is created in the yellow field with the command `DEFINE_PARAMETER` and given a value of 10 ③. This wait time will be used here and for all repeated tests.

When the pending event is triggered, a series of tests will be run in which the speed is increased. The speed will be set using the target speed `VX_TARG`, which will be increased by the parameter `VX_STEP`. Neither `VX_TARG` nor `VX_STEP` are built into VS models; they are defined with VS commands `DEFINE_PARAMETER` and `DEFINE_VARIABLE` ④ and will be updated using other datasets from the **Events** library that will be described shortly. The parameter `VX_STEP` is assigned units of km/h to simplify the specification of the increment to match the units used in the plots.

When the simulation time exceeds `T_WAIT` seconds, the VS solver will read the dataset file with the title **B. Check Longitudinal Speed** (Figure 18).



[BikeSim_Data] Events; { Steady-State Straight Series } B. Check Longitudinal Speed

Events141
08-02-2012 11:56

Data to use now (when this data set loads)
☐ Specify initialization details? ☑ More Links?

Speed control is not specified

Misc. Data:
Misc. Data:
Misc. Data:
Misc. Data:
Misc. Data:
Misc. Data:

Pending events to control the future use of data
☑ Clear all existing events Number of pending events to add: 3

When an event occurs, the data from the associated link is loaded and the run continues. If there is no link, then the run stops.

1: IF	VX_STEP	<	ABS(VX - VX_TARG)	THEN load:	Stop Due to Longitudinal Speed Sto
2: IF	VX	>	209	THEN load:	Stop Due to Longitudinal Spee p
3: IF	VX	<	209	THEN load:	C. Write M-File

Figure 18. Events dataset to check longitudinal speed.

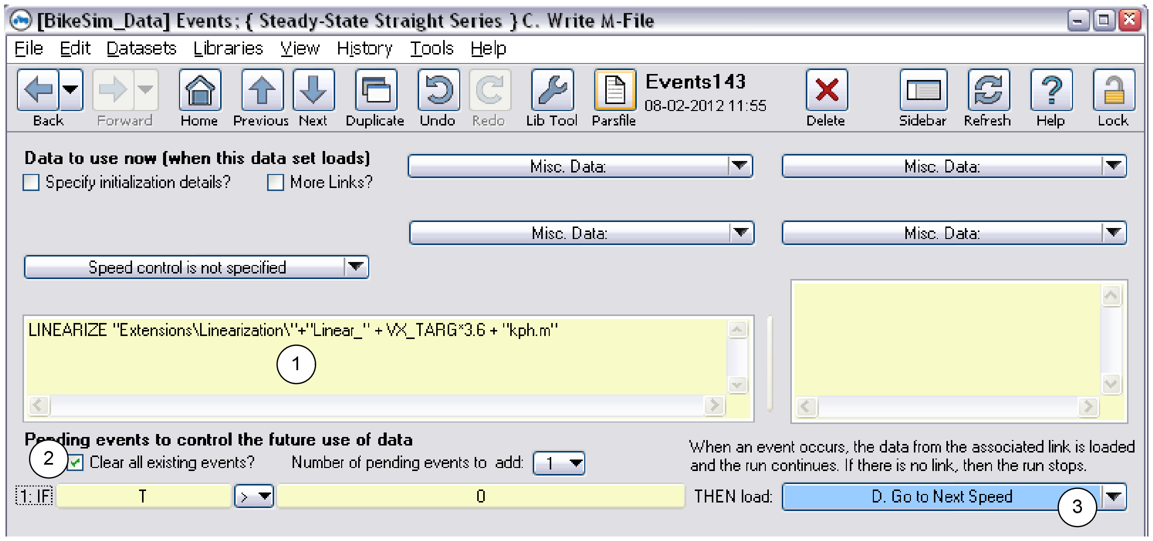
In this case, three new pending events are added. They are used here to branch the simulation into one of three mutually exclusive conditions. Either the run is ended by a tolerance ③; is ended by a speed ⑤; or we write an M-File ⑥. The sequence of the events is important because the first condition will always be checked before the second and third one.

The first condition is to check to see if the vehicle longitudinal speed `VX` is within the target `VX_TARG` using `VX_STEP` as a tolerance.

When the Boolean formula used to define an event includes functions or more than one variable, the expression must be placed into the yellow field on the right-hand side of the conditional function. In this case, the expression is `ABS(VX - VX_TARG)` ②. If the first event is triggered (the vehicle longitudinal speed is not within `VX_STEP` of the target `VX_TARG`), then a dataset is loaded that will immediately end the run. The second check is for the case where the vehicle speed exceeds 209 km/h, in which the run is ended when the dataset ⑤ is loaded. If neither of the previous events is triggered, then a dataset is loaded to perform the linearization and write the M-file ⑥.

Figure 19 shows the dataset with the `LINEARIZE` command ①. The name of the M-file is set dynamically, to allow the same dataset to be used multiple times as the run proceeds.

```
LINEARIZE "Extensions\Linearization\"+"Linear_" + VX_TARG*3.6 + "kph.m"
```


[BikeSim_Data] Events: { Steady-State Straight Series } C. Write M-File
File Edit Datasets Libraries View History Tools Help
Back
Forward
Home
Previous
Next
Duplicate
Undo
Redo
Lib Tool
Parsfile
Events143
08-02-2012 11:55
Delete
Sidebar
Refresh
Help
Lock
1
2
3
Data to use now (when this data set loads)
Specify initialization details?
More Links?
Misc. Data:
Misc. Data:
Misc. Data:
Misc. Data:
Speed control is not specified

LINEARIZE "Extensions\Linearization\"+"Linear_" + VX_TARG*3.6 + "kph.m"

Pending events to control the future use of data
✓ Clear all existing events?
Number of pending events to add: 1
When an event occurs, the data from the associated link is loaded and the run continues. If there is no link, then the run stops.
1: IF T > 0 THEN load: D. Go to Next Speed

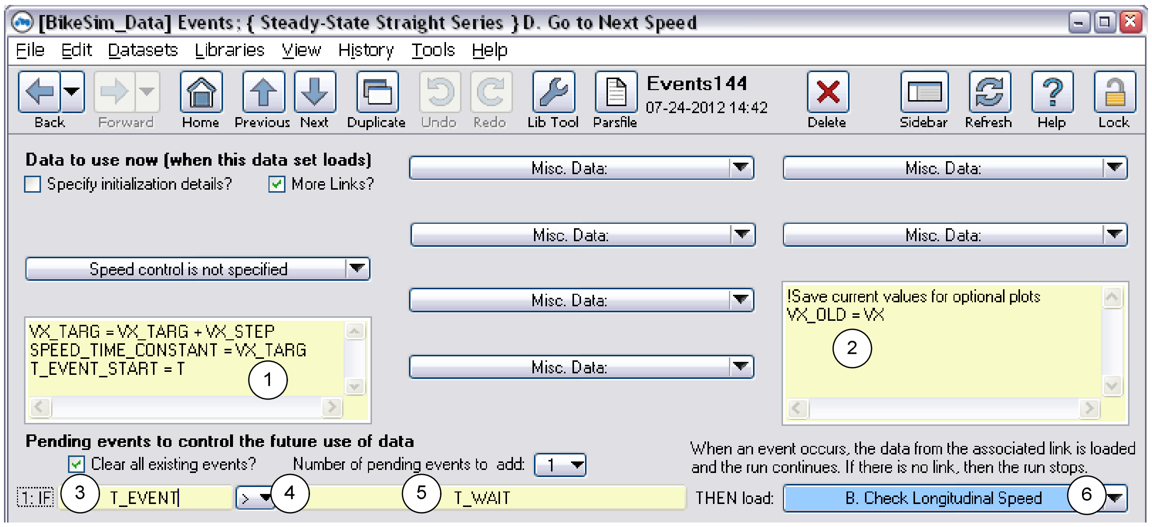
Figure 19. Events dataset to perform linearization and to write M-File.

As described earlier, `VX_TARG` (starting from 0 km/h) will be incremented by `VX_STEP` (10 km/h) each time through the series. The written M-File is given a unique name based on the speed. The file name involves the current target speed such as: `Linear_0kph.m`, `Linear_10kph.m`, …. `Linear_200kph.m`.

Note Any variable in a formula used in a VS command will have SI units. Thus, `VX_TARG` will have units of m/s. The scale factor 3.6 is included because we want to identify the file with speeds in units of km/h.

The previous **Events** dataset (Figure 18) had several pending events; they are cleared when this dataset is loaded (2).

This dataset defines a single pending event based on a condition that is guaranteed to the true (T > 0). Therefore, at the next time step, it will be triggered and the VS solver will load the **Events** dataset **D. Go to Next Speed** (3).


[BikeSim_Data] Events: { Steady-State Straight Series } D. Go to Next Speed
File Edit Datasets Libraries View History Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
Events144
07-24-2012 14:42
Delete Sidebar Refresh Help Lock
Misc. Data:
1
2
3
4
5
6

Data to use now (when this data set loads)
Specify initialization details? More Links? ✓

Speed control is not specified

```
VX_TARG = VX_TARG + VX_STEP
SPEED_TIME_CONSTANT = VX_TARG
T_EVENT_START = T
```

```
!Save current values for optional plots
VX_OLD = VX
```

Pending events to control the future use of data
✓ Clear all existing events? Number of pending events to add: 1

When an event occurs, the data from the associated link is loaded and the run continues. If there is no link, then the run stops.

1: IF T_EVENT > T_WAIT THEN load: B. Check Longitudinal Speed

Figure 20. Setting the next target speed.

Figure 20 shows the dataset that sets the next speed. When it is loaded, three program variables are given new values (1):

1. The target longitudinal speed `VX_TARG` is incremented by the amount `VX_STEP`.
2. The target speed `SPEED_TIME_CONSTANT` is given a new value of `VX_TARG`.
3. A system parameter `T_EVENT_START` is assigned the current value of the simulation time T.

Note The other field (2) updates a variable not covered in this memo that can be used for some optional plotting. If the variable `VX_OLD` has not been defined via VS commands associated with the plots, then the VS solver does not recognize it and this setting is ignored.

The variable `T_EVENT` is built into VS solvers and is defined as `T_EVENT = T – T_EVENT_START`. Therefore, setting `T_EVENT_START` to the current time T resets a relative time `T_EVENT` to zero. (The parameter `T_EVENT_START` and the associated variable `T_EVENT` are both described in the **Help** Reference Manual **System Parameters in VS Solvers**, along with other parameters that exist in all VS solver programs.)

A pending event is set to trigger when the relative time `T_EVENT` is greater than the parameter `T_WAIT` (4). When the event triggers, another **Events** dataset is loaded (6). This dataset (**B. Check Longitudinal Speed**) was described earlier (Figure 18).

The **Events** datasets B (Figure 18), C (Figure 19), and D (Figure 20) define a loop. Every time B is loaded it checks the longitudinal speed then loads C, which in turn instructs the VS solver to write to M-File for the linearization matrices. And then go to the D to increment the target speed, to wait for `T_WAIT` seconds, and go back to B dataset.

Figure 21 shows the time-history of the longitudinal speed when this series of events is used for a simulation run.

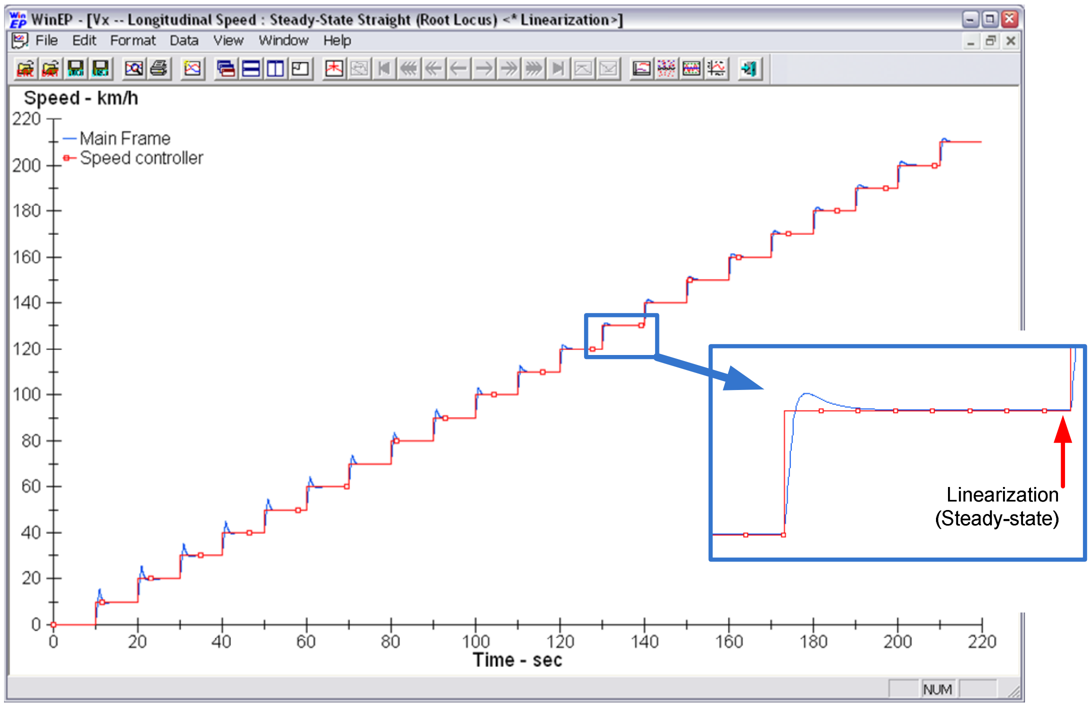


Figure 21. Time-history of the simulation during the series of event runs with various target speeds in steady-state.

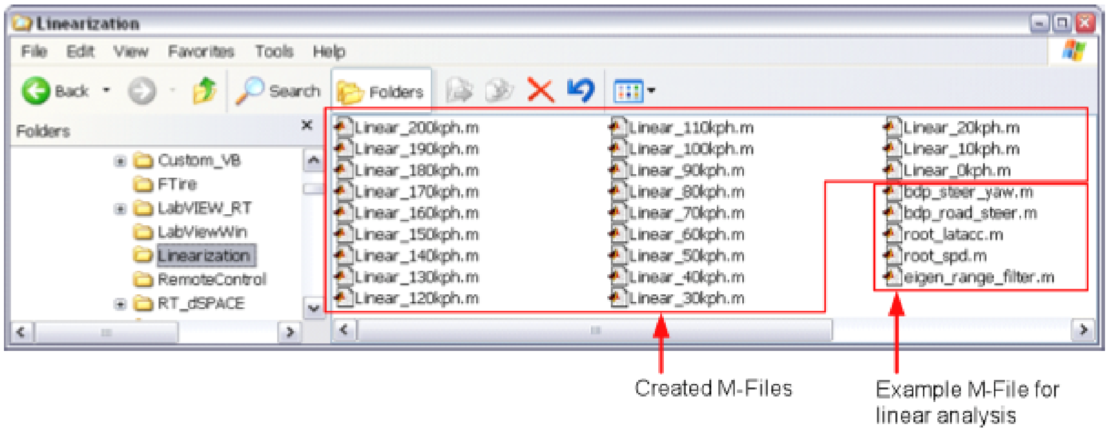
The bike runs on a flat surface starting a zero speed and goes through a series of constant-speed tests with the interval set to 10 km/h. At the start of each test, the target speed is increased and the closed-loop speed controls adjust the rear wheel drive torque. The vehicle speed shows an overshoot but it eventually settles to a steady state speed. The linearization is performed at each of the target speeds in steady state just 1 time step before the next speed adjustment.

In this example, the VS solver generates 21 new M-Files (Figure 22).

Root Locus Plot in MATLAB

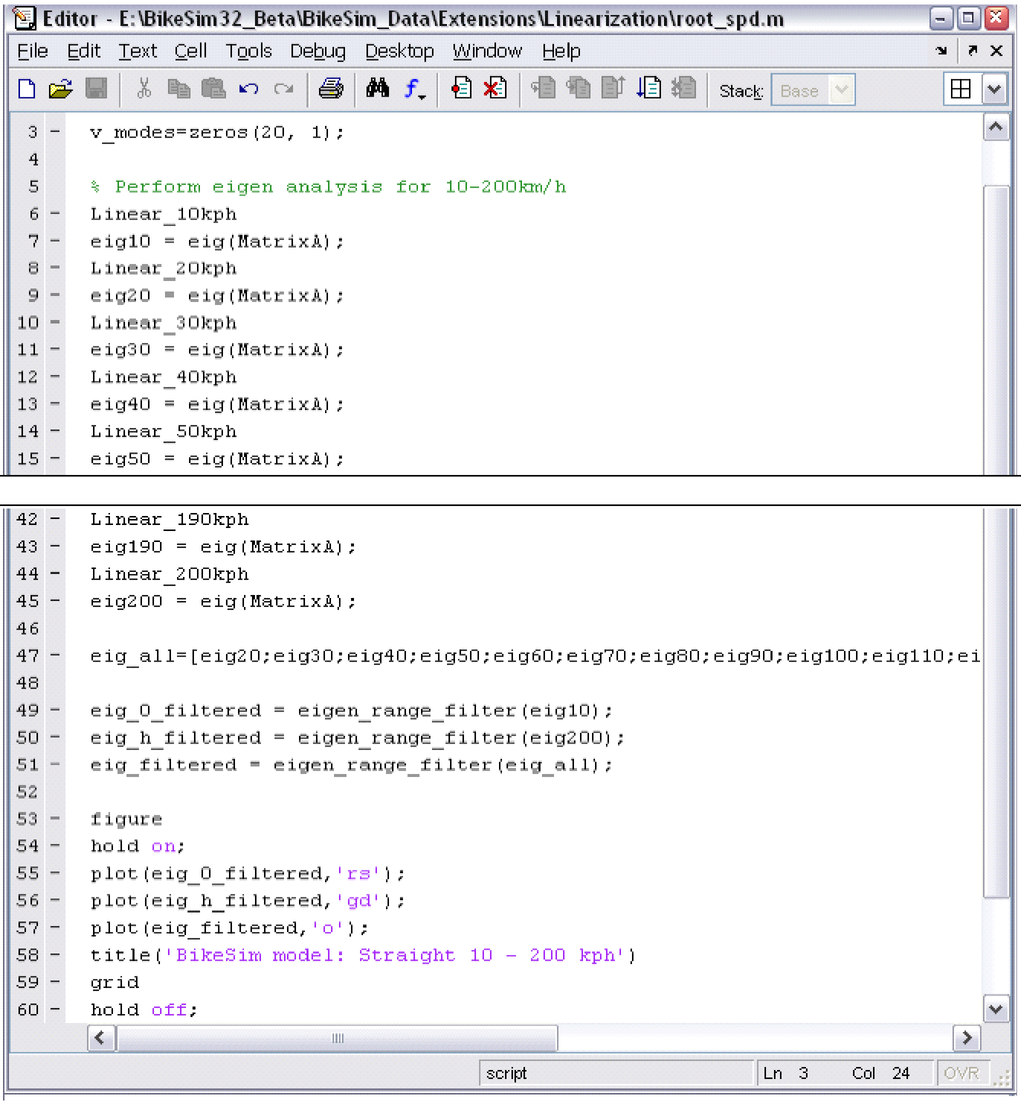
As with the previous example, start MATLAB and change the current directory to the `Linearization` folder that contains the script files for the analyses in this memo, and which was also the location specified for the M-files generated in this run (as shown earlier in Figure 12, page 11).

For this analysis, see the contents of the file `root_spd.m` (Figure 23). This script loads the M-File for each speed (`Linear_10kph.m` – `Linear_200kph.m`) and calculates the associated eigenvalues for each speed.



```
Linear_200kph.m    Linear_110kph.m    Linear_20kph.m
Linear_190kph.m    Linear_100kph.m    Linear_10kph.m
Linear_180kph.m    Linear_90kph.m     Linear_0kph.m
Linear_170kph.m    Linear_80kph.m     bdp_steer_yaw.m
Linear_160kph.m    Linear_70kph.m     bdp_road_steer.m
Linear_150kph.m    Linear_60kph.m     root_latacc.m
Linear_140kph.m    Linear_50kph.m     root_spd.m
Linear_130kph.m    Linear_40kph.m     eigen_range_filter.m
Linear_120kph.m    Linear_30kph.m
```

Figure 22. Created M-Files after the series of event runs.


Editor - E:\BikeSim32_Beta\BikeSim_Data\Extensions\Linearization\root_spd.m
File Edit Text Cell Tools Debug Desktop Window Help
Stack: Base
script
Ln 3
Col 24
OVR

```
v_modes=zeros(20, 1);

% Perform eigen analysis for 10-200km/h
Linear_10kph
eig10 = eig(MatrixA);
Linear_20kph
eig20 = eig(MatrixA);
Linear_30kph
eig30 = eig(MatrixA);
Linear_40kph
eig40 = eig(MatrixA);
Linear_50kph
eig50 = eig(MatrixA);
```

```
Linear_190kph
eig190 = eig(MatrixA);
Linear_200kph
eig200 = eig(MatrixA);

eig_all=[eig20;eig30;eig40;eig50;eig60;eig70;eig80;eig90;eig100;eig110;ei

eig_0_filtered = eigen_range_filter(eig10);
eig_h_filtered = eigen_range_filter(eig200);
eig_filtered = eigen_range_filter(eig_all);

figure
hold on;
plot(eig_0_filtered,'rs');
plot(eig_h_filtered,'gd');
plot(eig_filtered,'o');
title('BikeSim model: Straight 10 - 200 kph')
grid
hold off;
```

Figure 23. Part of the root locus script (MATLAB M-File).

For plotting purposes, the eigenvalues of the minimum and maximum speeds are separated and the eigenvalues between 20 km/h and 190 km/h are gathered into one matrix (`eig_all`). Those values are filtered in range by another M-File (`eigen_range_filter.m`) and plotted in the complex plane (Figure 24). To perform the calculations and view the plot in MATLAB, type the file name as a command: `root_spd`.

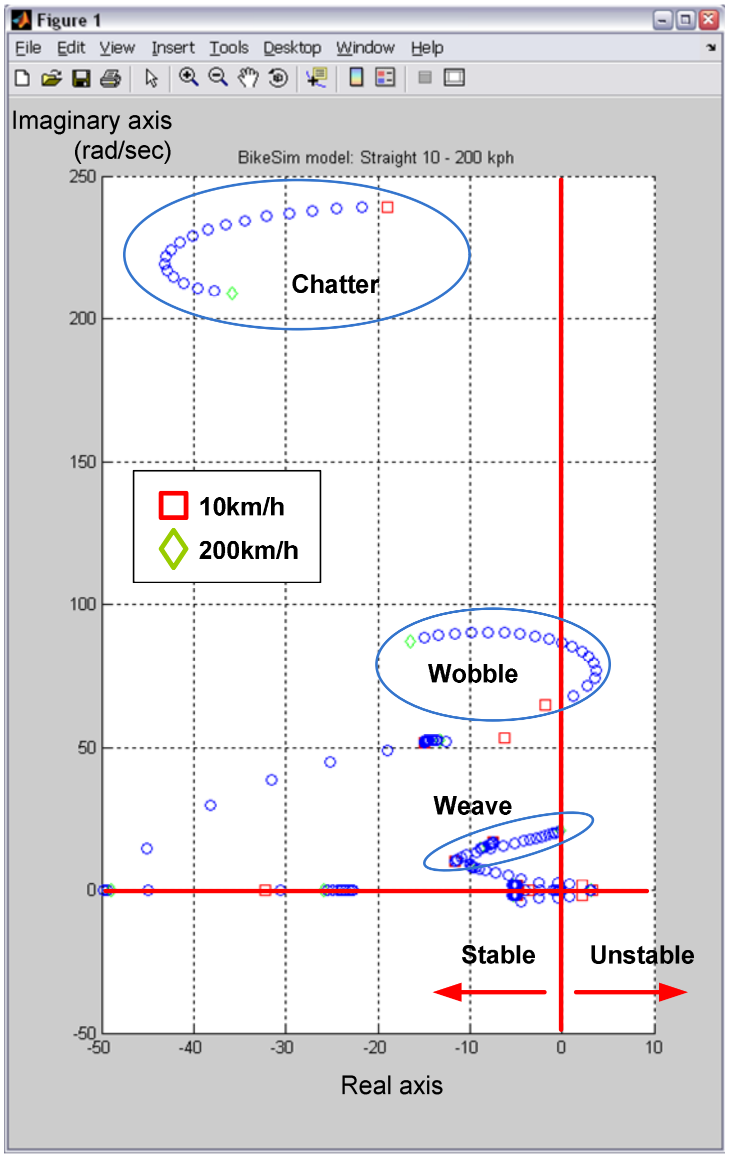


Figure 24. Observed weave, wobble and chatter with root locus on complex plane through the speed range 10 to 200 km/h.

The plot shows the weave mode in the frequency range between 10.4 and 20.7 rad/sec (1.7 to 3.3 Hz) where the highest frequency corresponds to the maximum speed of 200 km/h. The upper limit is also on the boundary of the unstable half of the plane.

The wobble mode lies in the frequency range between 64.7 and 87.1 rad/sec (10.3 to 13.9 Hz). As shown in this figure, the roots for wobble are unstable in the speed range from 10 km/h to 100 km/h. Around 60 to 70 km/h is the most unstable; 100 km/h is on the boundary between stable/unstable regions; and higher speeds become more stable. The frequency for 70 km/h (81.6 rad/sec ≈ 13 Hz) is close to the bump seen in the bode plot (Figure 16, page 14) and the time-domain simulation result (Figure 2, page 4).

The chatter mode is observed in the frequency range between 209 and 239 rad/sec. The frequency at 200 km/h (209 rad/sec ≈ 33 Hz) is close to the chatter frequency observed in the bode plot (Figure 16.)

Eigenvalues: Root Locus for High-Speed Cornering

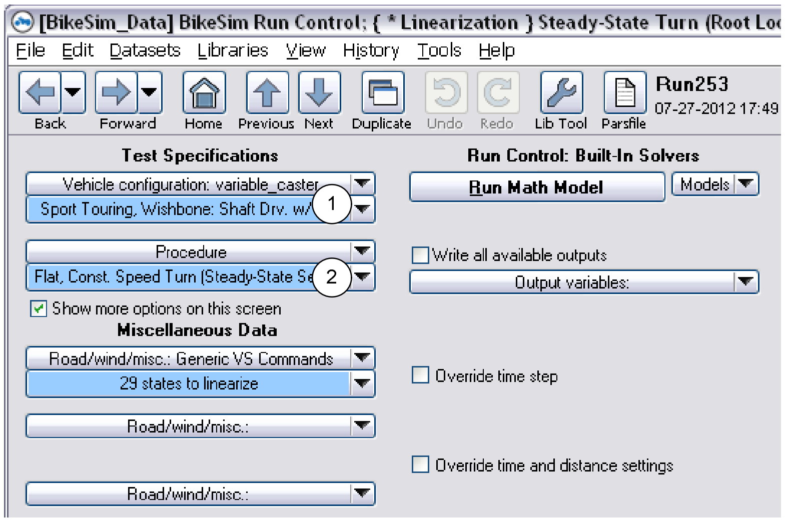
The previous section described how to get a root locus plot for straight-line conditions. This section shows how the method is modified to consider steady-state turning at a single speed.

Instead of defining a sequence of steady-state tests with increasing speed, in this case we look at a sequence of steady-state tests at a single speed near the stability limit (216 km/h), with increasing lateral acceleration.

As before, the analysis involves only eigenvalues calculated from the **A** matrix. Therefore, no input or output variables are identified.

A Sequence of Steady-State Tests with Increasing Cornering

This method is used for a dataset in the **Run Control** library named **Steady-State Turn (Root Locus)** in the category *** Linearization** (Figure 25). The vehicle configuration ① links the Sport Touring bike with a softened frame that causes chattering motion on the front fork as seen earlier (Figure 5, page 6). In this motorcycle, the torsional frame stiffness and corresponding damping at the steering head are reduced from the base parameters.


[BikeSim_Data] BikeSim Run Control: { * Linearization } Steady-State Turn (Root Lo

Run253
07-27-2012 17:49

Test Specifications
Vehicle configuration: variable_caster
Sport Touring, Wishbone: Shaft Drv. w/ ①
Procedure
Flat, Const. Speed Turn (Steady-State Se ②
☑ Show more options on this screen
Miscellaneous Data
Road/wind/misc.: Generic VS Commands
29 states to linearize
Road/wind/misc.:
Road/wind/misc.:

Run Control: Built-In Solvers
Run Math Model
Models
☐ Write all available outputs
Output variables:
☐ Override time step
☐ Override time and distance settings

Figure 25. Run Control screen for the steady-state turning test.

The **Procedures** dataset ② specifies a constant vehicle speed as 216 km/h and contains a link to a dataset from the **Events** library to initiate the running of the steady-state turn. The ground is flat and level. The series of events (Table 3) is similar to the series described in the previous section, with the difference being that each time through a loop a target lateral acceleration is increased.

Table 3. Series of ***Events*** *datasets used to run multiple steady-state turning tests.*

Dataset Title	**Actions**
A. Start Steady-State Turn	Define new parameters and variables that will be used later. Create a pending event to monitor the simulation time to wait long enough for the vehicle to be in steady state for straight (216 km/h.) When the simulation time reaches the specified limit (10 seconds), go to step B.
B. Check lateral acceleration	Compare the lateral acceleration to the target (`AY_TARG`). Either if they are not within a tolerance or exceeding maximum acceleration, end the run. Otherwise, go to step C.
C. Write M-File	Linearization is performed at this point; write the linear system matrices to M-File; clean up a pending event from step B; then go to step D.
D. Go to Next Lateral Acceleration	Increment the target lateral acceleration and derive the target lean angle based on the target lateral acceleration. Reset a local clock and monitor the local time to wait for the vehicle to be in steady state for the new lean angle. When the local time reaches the specified limit, go back to step B.

Figure 26 shows the time-history of the lateral acceleration, roll angle, and X-Y trajectory for this run. The plots show how the lean angle of the bike corresponds to the lateral acceleration. It starts at 0° and finishes at -53° for the final test. The linearization is performed for each target lateral acceleration just before switching to the next target.

After the simulation runs, the `Linearization` folder will contain 12 new M-Files, each with the **A** matrix for a different level of lateral acceleration.

Note In the events of A and D, the bike lean angle is calculated based on the target lateral acceleration such as:

```
BIKE_LEAN_CONSTANT = -ATAN(AY_TARG/G)
```

However, the above expression is not enough to achieve the target lateral acceleration. Therefore, in event A, the following expressions are added to compensate the lean angle:

```
define_variable SV_GAM_TARG 0;
eq_differential SV_GAM_TARG = -0.015 * (AY_TARG-AY_SM);
eq_in IMP_GAMMA_TARG = SV_GAM_TARG
IMP_GAMMA_TARG vs_add 0
```

In the above 4 lines, a new variable (`SV_GAM_TARG`) is defined and assigned with the error between the target and actual lateral acceleration multiplied by the control gain (`0.015`), which in turn is imported as an additional target lean angle.

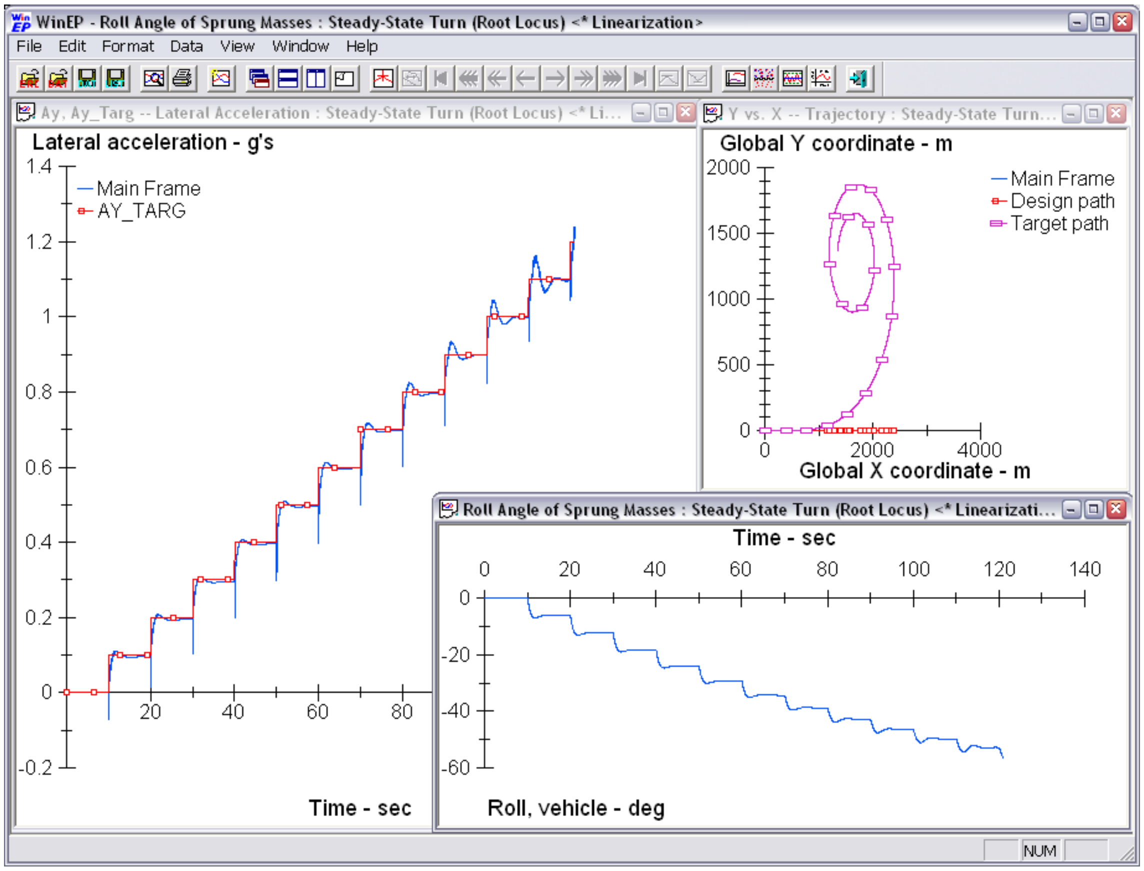

WinEP - Roll Angle of Sprung Masses : Steady-State Turn (Root Locus) <* Linearization>
File Edit Format Data View Window Help
Ay, Ay_Targ -- Lateral Acceleration : Steady-State Turn (Root Locus) <* Li...
Lateral acceleration - g's
Main Frame
AY_TARG
Time - sec
Y vs. X -- Trajectory : Steady-State Turn...
Global Y coordinate - m
Main Frame
Design path
Target path
Global X coordinate - m
Roll Angle of Sprung Masses : Steady-State Turn (Root Locus) <* Linearizati...
Time - sec
Roll, vehicle - deg
NUM

Figure 26. Time-history of the simulation during the series of event runs with various target lateral accelerations in steady-state turnings.

Root Locus Plot in MATLAB

As with the other examples, start MATLAB and change the current directory to the `Linearization` folder that contains the script files for the analyses in this memo and was the location specified for the M-files generated in this run.

In this case, type the script file name `root_latacc` as a command to generate the root locus plot. The script file loads the M-File for each lateral acceleration (`Linear_0_0G.m` – `Linear_1_1G.m`), calculates the eigenvalues for each lateral acceleration, and generates the root locus plot (Figure 27).

The plot shows that the weave mode is in the frequency range between 14.7 and 20.8 rad/sec (2.4 – 3.3 Hz) and is always unstable. The wobble mode is shown in the frequency range between 45.3 and 59.3 rad/sec (7.2 – 9.4 Hz) and is always stable.

The chatter mode is observed in the frequency range between 154 and 172 rad/sec (24.5 – 27.4 Hz) in the stable region. However, the root at 1.1g (27.4 Hz) is on the stability boundary. This part of the test is similar to the run shown earlier whose time-history result (chattering at 25 Hz) is shown in Figure 5.

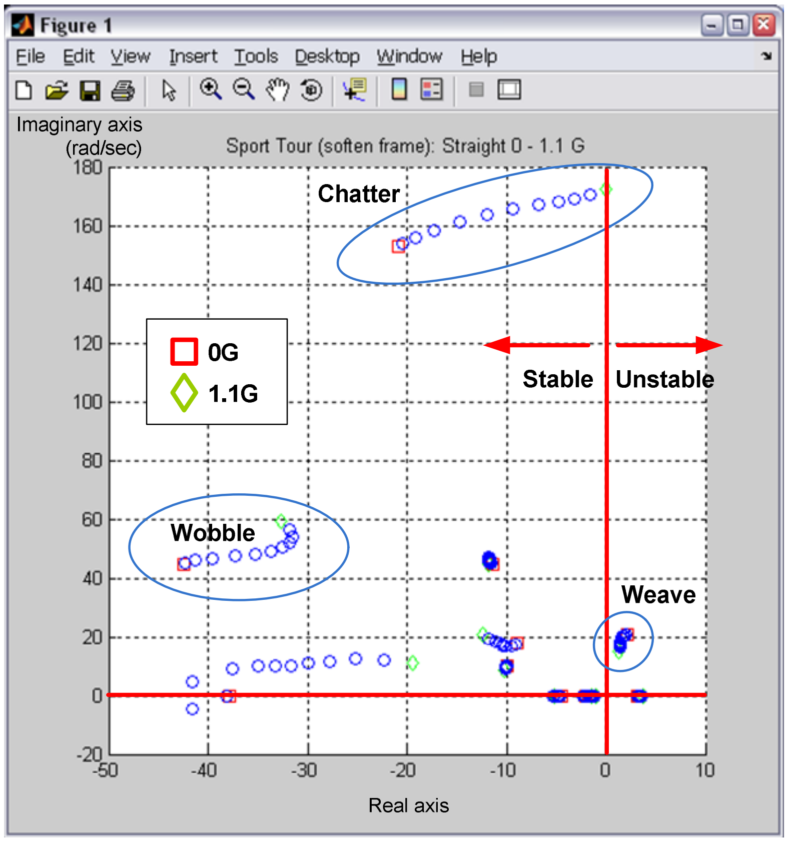


Figure 27. Observed weave, wobble and chatter with root locus on complex plane at 216 km/h through the lateral acceleration range 0 to 1.1 g.

The chatter observed in Figure 5 and Figure 27 involves a softened front fork frame and a corresponding lower frequency and less stability, compared to results shown in Figure 16 and Figure 24 with rigid frame setting. These results imply that the chatter can be caused by a highly aggressive cornering maneuver (high speed, high G, and high lean angle) such as racing motorcycle (e.g. MotoGP), especially with a front fork frame that is not rigid enough.

Note The chatter mode of vibration has not been clearly specified in the published technical literature. The examples in this memo complement a recent technical paper by Sharp and Watanabe that is published by the Journal of *Vehicle System Dynamics* [2].

References

1. Sharp, R. S., Evangelou, S. and Limebeer, D. J. N., "Advances in the modelling of motorcycle dynamics", Multibody System Dynamics, 12(3), 2004, 251-283.
2. Sharp, R. S. and Watanabe, Y., "Chatter vibrations of high-performance motorcycle", Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility, DOI:10.1080/00423114.2012.727440, October 2012.

Appendix: Calculation Details

As mentioned earlier, the purpose of linearization is to put the equations into the linear form:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)\end{aligned} \tag{1}$$

However, multibody models such as those used to represent vehicle dynamics in VS products have nonlinear equations of motion in the form:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{f}\left\{\mathbf{x}(t),\, \mathbf{u}(t)\right\} \\ \mathbf{y}(t) &= \mathbf{g}\left\{\mathbf{x}(t),\, \mathbf{u}(t)\right\}\end{aligned} \tag{2}$$

where **f** and **g** are nonlinear array functions of time and the model variables that are used to calculate the *n* derivatives of the state variables in the array **x** and the *m* output variables in the array of outputs **y**.

The assumption is that the linearized equations (Eq. 1) are valid for calculating variations of the state variables and output variables relative to the current state of the math model. For example, in the last section of this memo, the bike model was placed in a steady state condition of cornering at high speed; the linearized equations for that state are accurate for small perturbations from the steady state, but would not necessarily be accurate over large changes in speed or other variables.

Small perturbations in time derivatives of the state variables ($\Delta\dot{\mathbf{x}}$) and outputs ($\Delta\mathbf{y}$) can be derived by the partial derivatives of the nonlinear functions (**f** and **g**) with respect to the state variables and controls, such as:

$$\begin{aligned}\Delta\dot{\mathbf{x}}(t) &= \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Delta\mathbf{x}(t) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}\Delta\mathbf{u}(t) \\ \Delta\mathbf{y}(t) &= \frac{\partial \mathbf{g}}{\partial \mathbf{x}}\Delta\mathbf{x}(t) + \frac{\partial \mathbf{g}}{\partial \mathbf{u}}\Delta\mathbf{u}(t)\end{aligned} \tag{3}$$

Eq. 1 and 3 are similar. By considering the variables in the linearized system to be the perturbations of the variables of the full nonlinear system, we can equate the two descriptions to define the **A**, **B**, **C**, and **D** matrices for the linearized system as partial derivatives of the functions **f** and **g** with respect to the state variables **x** and control inputs **u**.

The partial derivatives are calculated numerically by perturbing each variable in **x** and **u**. Thus,

$$\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \approx \frac{\Delta\dot{\mathbf{x}}(t)}{\Delta\mathbf{x}(t)} \tag{4}$$

$$\mathbf{B} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \approx \frac{\Delta\dot{\mathbf{x}}(t)}{\Delta\mathbf{u}(t)} \tag{5}$$

$$\mathbf{C} = \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \approx \frac{\Delta\mathbf{y}(t)}{\Delta\mathbf{x}(t)} \tag{6}$$

$$\mathbf{D} = \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \approx \frac{\Delta \mathbf{y}(t)}{\Delta \mathbf{u}(t)} \tag{7}$$

For example, since the **x** array consists of *n* variables, **A** is an *n*×*n* matrix:

$$\mathbf{A} = \begin{bmatrix} \mathrm{A}_{11} & \cdots & \mathrm{A}_{1n} \\ \vdots & \ddots & \vdots \\ \mathrm{A}_{n1} & \cdots & \mathrm{A}_{nn} \end{bmatrix} \tag{8}$$

where, for example, each element in the first column is:

$$\begin{aligned} &\mathrm{A}_{11} = \Delta \dot{x}_1(t) / \Delta x_1(t) \\ &\mathrm{A}_{21} = \Delta \dot{x}_2(t) / \Delta x_1(t) \\ &\vdots \\ &\mathrm{A}_{n1} = \Delta \dot{x}_n(t) / \Delta x_1(t) \end{aligned} \tag{9}$$

Therefore, the elements in the first column of the **A** matrix are obtained by finding the deviation of each state variable time derivative that results from a small perturbation of the state variable Δx_1.

Similarly, **B** is an *n*×*r* matrix:

$$\mathbf{B} = \begin{bmatrix} \mathrm{B}_{11} & \cdots & \mathrm{B}_{1r} \\ \vdots & \ddots & \vdots \\ \mathrm{B}_{n1} & \cdots & \mathrm{B}_{nr} \end{bmatrix} \tag{10}$$

where, for example, each element in the first column is:

$$\begin{aligned} &\mathrm{B}_{11} = \Delta \dot{x}_1(t) / \Delta\, \mathrm{u}_1(t) \\ &\mathrm{B}_{21} = \Delta \dot{x}_2(t) / \Delta\, \mathrm{u}_1(t) \\ &\vdots \\ &\mathrm{B}_{n1} = \Delta \dot{x}_n(t) / \Delta\, \mathrm{u}_1(t) \end{aligned} \tag{11}$$

Therefore, the elements in the first column of the **B** matrix are obtained by the finding deviation of each state variable time derivative that results from a small perturbation of the input Δu_1.

Deviations are also calculated for the output variables in **y** in order to calculate the elements of the **C** and **D** matrices.

Figure 28 shows a flow chart of the operation of the VS solver concerning the `LINEARIZE` command.

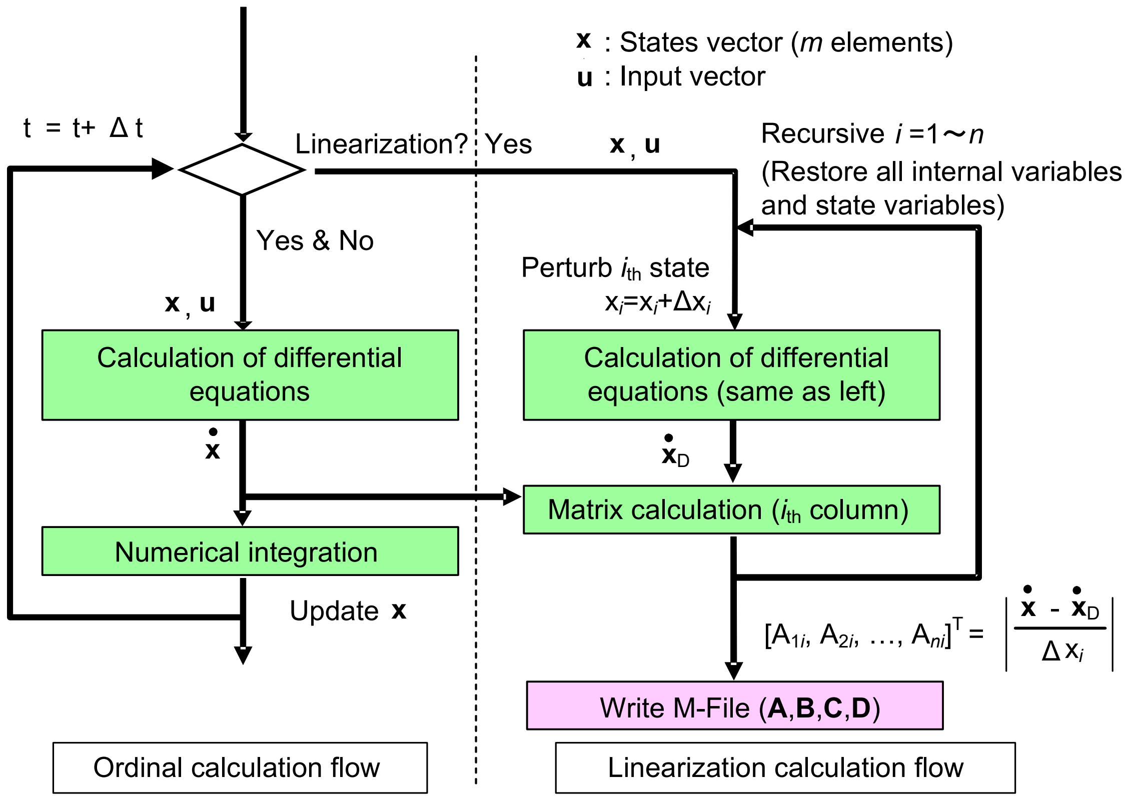

x : States vector (m elements)
u : Input vector
t = t+ Δt
Linearization?
Yes
x, u
Recursive i =1∼n
(Restore all internal variables and state variables)
Yes & No
Perturb i_th state
x_i=x_i+Δx_i
x, u
Calculation of differential equations
Calculation of differential equations (same as left)
ẋ
ẋ_D
Matrix calculation (i_th column)
Numerical integration
Update x
[A_1i, A_2i, …, A_ni]^T = |(ẋ - ẋ_D)/Δx_i|
Write M-File (A,B,C,D)
Ordinal calculation flow
Linearization calculation flow

Figure 28. VS solver calculation flow chart during the linearization.

When the solver detects the `LINEARIZE` command, it saves the current values of all the state variables and control inputs so the state of the model can be fully restored. Next, it goes through the array of state variables that were identified for use in the linearization via the `LINEAR_SV` command and perturbs that variable slightly. In the computer code of a VS solver, **f** is a function named `difeqn` that calculates the derivatives for all of the state variables in the full nonlinear math model, and **g** is a function named `output` that calculates all output variables for the model. These functions are applied using the values of the saved state variables with a single variable (either a state variable x_i or an input variable u_i) having been perturbed. Changes in the state variable derivatives and output variables are used to calculate the values of a single column of **A** and **C**.

The copy of the state variables and control inputs are used to restore the state, and the process is repeated for the next state variable.

When all of the state variables have been perturbed, the process is repeated for the control input variables, to calculate the elements of the **B** and **D** arrays.

After all elements of the four matrices have been calculated, the values are written into an M-File using the file name specified with the `LINEARIZE` command.

The state of the model is restored one more time, and the simulation then proceeds to the next time step via numerical integration.