Summary

This unit provides and introduction to modeling mechanical system kinetics using MATLAB scripts, Simulink models, and SimMechanics models. For an introduction to these modeling techniques, see Unit 10 of Volume I for applications in mechanical systems kinematics. The examples presented in this unit assume the reader is familiar with the programming concepts presented in Volume I.

As presented in Volume I, MATLAB scripts are text-based programs written in the MATLAB programming language. Simulink models are block-diagram-based programs that run in the MATLAB environment. SimMechanics models are block-diagram-based, multibody dynamics programs that run in the MATLAB/Simulink environment. MATLAB scripts can be used alone or in conjunction with Simulink and SimMechanics models.

Three of the six models presented herein simulate the free motion of an upright bicycle. The models are developed using the equations of motion developed in Unit 5 of this volume.

<table>
<thead>
<tr>
<th>Page Count</th>
<th>Examples</th>
<th>Models</th>
<th>Exercises</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

Trademarks: MATLAB and Simulink, and SimMechanics (now called Simscape Multibody) are all registered trademarks of The MathWorks, Inc. The MathWorks does not warrant the accuracy of the examples given in this volume.
Example 1: Rotating Simple Crank Shaft

The figure shows the simple crank shaft from Example 1 of Unit 2 of this volume. The crank shaft consists of seven segments, each considered to be a slender bar. Each segment of length \( \ell \) has mass \( m \). There are six segments of length \( \ell \) and one segment of length \( 2\ell \) (segment 4). The mass center of the system \( G \) is located on the axis of rotation.

In Unit 2, the elements of the third column of the inertia matrix of the crank shaft were calculated using the parallel axes theorems for moments and products of inertia and were found to be

\[
I_{XZ}^G = -2m\ell^2 \quad I_{YZ}^G = 0 \quad I_{ZZ}^G = \frac{10}{3}m\ell^2
\]

The remaining nonzero entries of the inertia matrix can be calculated similarly as follows. The segment numbers are shown to clarify the calculations.

\[
I_{X'X'}^G = 2\left[ \frac{1}{12} m\ell^2 + m\left( \frac{3}{2} \ell \right)^2 \right] + 2\left[ m\ell^2 \right] + 2\left[ \frac{1}{3} m\ell^2 \right] = 2\left( \frac{1}{12} + \frac{9}{4} + \frac{1}{3} \right) m\ell^2 = \frac{22}{3} m\ell^2
\]

\[
I_{YY'}^G = 2\left[ \frac{1}{12} m\ell^2 + m\left( \frac{3}{2} \ell \right)^2 \right] + 4\left[ \frac{1}{12} m\ell^2 + m\left( \ell^2 + \frac{1}{4} \ell^2 \right) \right] = 2\left( \frac{1}{12} + \frac{9}{4} + 4\left( \frac{1}{2} + 1 + \frac{1}{4} \right) \right) m\ell^2
\]

\[
= \left( \frac{2}{12} + \frac{54}{12} + \frac{4}{12} + \frac{48}{12} + \frac{12}{12} \right) m\ell^2
\]

\[
\Rightarrow I_{YY'}^G = 10 m\ell^2
\]

\[
I_{ZZ}^G = I_{XZ}^G = -2m\ell^2
\]

The remaining products of inertia are all zero due to symmetry of the system about the \( X'Z \) plane. Using these results, the inertia matrix of the crank shaft can be written as

\[
[I_G] = \begin{bmatrix}
I_{X'X'}^G & -I_{X'Y'}^G & -I_{X'Z}^G \\
-I_{Y'X'}^G & I_{Y'Y'}^G & -I_{Y'Z}^G \\
-I_{Z'X}^G & -I_{Z'Y}^G & I_{Z'Z}^G
\end{bmatrix} = m\ell^2 \begin{bmatrix}
\frac{22}{3} & 0 & 2 \\
0 & 10 & 0 \\
2 & 0 & \frac{10}{3}
\end{bmatrix}
\]

Given the free-body-diagram shown below, the following results were found using the Newton/Euler equations of motion. Note that the weight force was not included to focus the analysis on the dynamic loads only, that is, loads due solely to the motion and asymmetry of the crank shaft.
\[
\sum (F \cdot i') = A_{X'} + B_{X'} = 0 \\
\sum (F \cdot j') = A_{Y'} + B_{Y'} = 0 \\
\sum (M \cdot i') = -4 \ell B_{Y'} = 2m \ell^2 \dot{\omega} \\
\sum (M \cdot j') = 4 \ell B_{X'} = 2m \ell^2 \dot{\omega}^2 \\
\sum (M \cdot k) = T = \frac{10}{3} m \ell^2 \dot{\omega}
\]

(force equations)  (moment equations)

Solving these equations gives the following results for the support force components and the driving torque.

\[
B_{X'} = \frac{1}{2} m \ell \dot{\omega}^2 = -A_{X'} \\
B_{Y'} = -\frac{1}{2} m \ell \dot{\omega} = -A_{Y'}
\]

(support force components)  (driving torque)

In this example, a SimMechanics model is used to simulate the motion, support forces and driving torque on the simple crank shaft. A MATLAB script is used to initialize variables used in that model, to execute the model and plot its results, and to calculate and display analytical results for comparison.

The top layer of a SimMechanics model and two of its subsystems are shown in the figures below. The motion subsystem creates motion signals for the angle, angular velocity, and angular acceleration of the crank shaft that actuates the system through the custom joint at A. The measurement subsystem for the custom joint at A records the applied angular velocity and angular acceleration of the crank shaft as well as the components of the reaction force and torque associated with the joint. The measurement subsystem for the custom joint at B records the components of the reaction force associated with that joint.

The crank shaft is made of steel bar with a weight density of 0.284 (lb/in\(^3\)) and a diameter of 0.5 (in). Six segments have length \( \ell = 6 \) (in) and one segment has length \( 2\ell = 12 \) (in). The mass center \( G \) of the crank shaft is located at the origin of the world coordinate system, and the two custom joints are located at the coordinates (0,0, -12) and (0,0,12) inches, respectively. The Machine Environment block defines the analysis as forward dynamics, and it defines the gravity vector as [0,0,0] so the weight of the system is not included in the analysis.
To understand the reaction forces and torques being recorded, it is helpful to consider the details of the custom joints. The custom joint at A allows three axes of rotation and no translation. The first primitive (shown as R3) allows the crank shaft to rotate about the Z axis of the world coordinate system. The custom joint at B allows for three axes of rotation and one axis of translation along the world Z axis. As defined, there are no redundant constraints connecting the crank shaft to the ground.

Using this means of support, the reaction force components measured at each joint will be the same as those in the analysis above so long as they are measured in the coordinate system of the follower (i.e., the crank shaft). Note in the dialog box below for the joint sensor at A that the joint reactions are measured on the follower (the crank shaft) and with respect to the coordinate system rotating with it. The reaction torque measured at A should be nonzero only along the world Z direction (which is aligned with the Z axis of the crank shaft). This nonzero torque corresponds to the driving torque $T$ in the analysis above.
Note finally that values of the initial angular velocity, constant angular acceleration, mass, and inertia matrix of the crank shaft are all provided by workspace variables defined within the MATLAB script that controls the execution of the model. Also, the Visualization tab of the dialog box for the crank shaft (not shown here) indicates the body geometry will be provided by an external stereolithographic file.

The first two sections of the MATLAB script are shown in Panel 1 below. The first section provides a brief description of the script’s purpose, and the second defines the mass and inertia properties of the crank shaft. The mass and inertia properties are calculated based on the assumption the system is made of circular
steel bar with a weight density of 0.284 (lb/in$^3$) and diameter of 0.5 (in). The inertia values are calculated using the formula presented above. The characteristic length $\ell = 6$ (in).

Panel 1 – MATLAB Script:

```matlab
%% Simple Crank Shaft Simulation
% This file calculates input data for the SimMechanics model
% 'SimpleCrankShaftModel', executes the model and plots its results,
% and calculates analytical results and displays them for comparison
%
%% mass and inertia properties
accelerationofGravity = 32.2; % (ft/sec^2)
weightDensitySteel    = 0.284; % (lb/in^3)
diameter            = 0.5; % (in)
weightPerUnitLength = weightDensitySteel*pi*(diameter^2)/4.0; % (lb/in)

segmentLength = 6.0; % (length, L in inches)
segmentWeight = segmentLength*weightPerUnitLength; % (lb)
segmentMass   = segmentWeight/accelerationofGravity; % (slugs)

numberofSegments = 8; % (number of segments of length, L)
totalMass = numberofSegments*segmentMass; % (slugs)
inertiaXX = (22/3)*segmentMass*(segmentLength^2); % (slug-in^2)
inertiaYY = 10*segmentMass*(segmentLength^2); % (slug-in^2)
inertiaZZ = (10/3)*segmentMass*(segmentLength^2); % (slug-in^2)
inertiaXZ = -2*segmentMass*(segmentLength^2); % (slug-in^2)
inertiaZX = inertiaXZ; % (slug-in^2)
inertiaMatrix = [inertiaXX,0,-inertiaXZ; 0,inertiaYY,0; ...
                -inertiaXZ,0,inertiaZZ]; % (slug-in^2)

%% motion properties
initialAngularVelocity      = 50.0; % (r/s)
constantAngularAcceleration = 10.0; % (r/s^2)

%% execute the simulink model
sim('SimpleCrankShaftModel')
```

The next two sections provide values for the motion parameters, the initial angular velocity, and the constant angular acceleration. Given these values and those of the mass and inertia, the SimMechanics model is executed. See Panel 2 below.

Panel 2 – MATLAB Script:

```matlab
: % motion properties
initialAngularVelocity      = 50.0; % (r/s)
constantAngularAcceleration = 10.0; % (r/s^2)

%% execute the simulink model
sim('SimpleCrankShaftModel')
```

As part of the execution of the SimMechanics model, time histories of the reaction force and torque components are written to the MATLAB workspace. This is done using the scopes shown in the measurement subsystems. The next section of the MATLAB script plots these results. See Panel 3 below.
Panel 3 – MATLAB Script:

```matlab
%% Plot results from the simulation
% Plot the components of the force on the crankshaft at A
figure(1); clf;
subplot(3,1,1); plot(forceComponentsA.time,forceComponentsA.signals(1).values);
grid; title('Force Components at A'); xlabel('Time (sec)'); ylabel('X''-Component (lb)')

subplot(3,1,2); plot(forceComponentsA.time,forceComponentsA.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('Y''-Component (lb)')

subplot(3,1,3); plot(forceComponentsA.time,forceComponentsA.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('Z-Component (lb)')

% Plot the components of the torque on the crankshaft at A
figure(2); clf;
subplot(3,1,1); plot(torqueComponentsA.time,torqueComponentsA.signals(1).values);
grid; title('Torque Components at A'); xlabel('Time (sec)'); ylabel('X''-Component (ft-lb)')

subplot(3,1,2); plot(torqueComponentsA.time,torqueComponentsA.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('Y''-Component (ft-lb)')

subplot(3,1,3); plot(torqueComponentsA.time,torqueComponentsA.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('Z-Component (ft-lb)')

% Plot the components of the force on the crankshaft at B
figure(3); clf;
subplot(3,1,1); plot(forceComponentsB.time,forceComponentsB.signals(1).values);
grid; title('Force Components at B'); xlabel('Time (sec)'); ylabel('X''-Component (lb)')

subplot(3,1,2); plot(forceComponentsB.time,forceComponentsB.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('Y''-Component (lb)')

subplot(3,1,3); plot(forceComponentsB.time,forceComponentsB.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('Z-Component (lb)')
```

The last two sections of the script calculate analytical results (based on the equations presented above) and display those results in the MATLAB command window for comparison. See Panels 4 and 5 below.

Panel 4 – MATLAB Script:

```matlab
%% initial force and torque reactions on the bearings
forceBXPrime = (1/2)*segmentMass*(segmentLength/12)* ... 
(initialAngularVelocity^2);  % (lb)
forceBYPrime = -(1/2)*segmentMass*(segmentLength/12)* ... 
(constantAngularAcceleration);  % (lb)
forceAXPrime = -forceBXPrime; forceAYPrime = -forceBYPrime;  % (lb)

torqueSystemX = 0.0;
torqueSystemY = 0.0;
torqueSystemZ = (10/3)*segmentMass*((segmentLength^2)/144)* ... 
constantAngularAcceleration;  % (ft-lb)
```
Panel 5 – MATLAB Script:

```matlab
%; Display analytical results in Command Window
disp('Analytical Results (neglecting the weight force')
disp('-----------------------------------------------')
disp('-')
disp('Initial Forces Applied to Crank Shaft at A:')
disp('-------------------------------------------')
forceString = ['X': ',num2str(forceAXPrime), ...
', ' Y': ',num2str(forceAYPrime)];
disp(forceString);
disp('-')
disp('Initial Forces Applied to Crank Shaft at B:')
disp('-------------------------------------------')
forceString = ['X': ',num2str(forceBXPrime), ...
', ' Y': ',num2str(forceBYPrime)];
disp(forceString);
disp('-')
disp('Initial Torques Applied to Crank Shaft at A:')
disp('--------------------------------------------')
torqueString = ['X': ',num2str(torqueSystemX), ...
', ' Y': ',num2str(torqueSystemY), ...
', ' Z: ',num2str(torqueSystemZ)];
disp(torqueString);
```

Simulation Results:

When the script is executed, a geometric representation of the system is displayed. In this case, an external stereolithographic file is used to represent the three-dimensional geometry of the crank shaft. As the simulation progresses, the motion of the system is animated. The diagram shown here is the position of the system at \( t = 0.6 \) (sec). Note that, as shown in the diagram above, the crank shaft lies in the \( XZ \) plane of the coordinate system that rotates with the system. Rotation is about the \( Z \) axis.

The force and torque components measured by the measurement subsystem at \( A \) are shown in the diagrams below. The force components at \( B \) are not shown; they are simply the negatives of those found at \( A \). Although producing those results may seem redundant, observation of those expected results can provide the analyst with some additional level of confidence in the results.

Note that (as expected) only the \( X' \) and \( Y' \) components of force are nonzero, and only the \( Z \) component of the torque is nonzero. Note also that even though the \( Y' \) component of the force and the \( Z \) component of
the torque are \textit{constant}, the plots clearly shown some \textit{deviation} in these values. The small deviation is \textit{numerical error} (noise) associated with the level of accuracy of the calculations.

Analytical Results:

\textit{Analytical results} are calculated in the \textit{initial position} using the formulae presented above and \textit{displayed} in the MATLAB command window. The results for the $Y'$ components of the forces and the $Z$ component of the torque are dependent on the angular acceleration, so they are \textit{constant}. However, the $X'$ component of the force depends on the angular velocity of the system, the value given below applies only to the initial position. Beyond that time, the $X'$ components of the forces \textit{increase quadratically} with the angular velocity.
Note these results are identical to those found by the SimMechanics model at the initial position.

Example 2: Rotating Bar on a Rotating Frame

The system shown consists of two bodies, the frame \( F \) and bar \( B \). Frame \( F \) rotates about the fixed vertical direction annotated by the unit vector \( \hat{k} \). Bar \( B \) is pinned to and rotates about the horizontal, rotating arm of \( F \). \( F \) rotates relative to the ground at a rate \( \Omega \) (r/s) and \( B \) rotates relative to \( F \) at a rate of \( \omega \) (r/s).

In Unit 2 of this volume, it was shown that if the angular rate \( \Omega \) is known while the motion of \( B \) relative to the frame is free, the forces and torques on the pin at \( G \) required to maintain the motion and the differential equation governing the angle \( \theta \) may be written as

Unknown forces:

\[
\sum F \cdot \varepsilon_1 = F_1 = -md \dot{\Omega} C_\theta \\
\sum F \cdot n_2 = F_2 = -md \Omega^2 \\
\sum F \cdot \varepsilon_3 = F_3 = -md \dot{\Omega} S_\theta
\]

Unknown torques:

\[
\sum M \cdot \varepsilon_1 = T_1 = 0 \\
\sum M \cdot \varepsilon_3 = T_3 = \frac{m \ell^2}{12} (\dot{\Omega} C_\theta - 2 \omega \Omega S_\theta)
\]

Differential equation for angle \( \theta \):

\[
\ddot{\theta} + \Omega^2 S_\theta C_\theta = \frac{T_2}{\left(\frac{1}{12} m \ell^2\right)} \\
(\text{torque } T_2 \text{ is assumed to be a known torque})
\]

Note that to generate these equations, it was assumed that frame, \( F \) is light and that bar, \( B \) is slender. See Unit 2 of this volume for additional details.

In Unit 4 of this volume, using Lagrange’s equations, the following two differential equations of motion were derived for this system.
\[
\left( m d^2 + \frac{1}{12} m \ell^2 C_{\alpha}^2 \right) \ddot{\phi} - \left( \frac{1}{6} m \ell^2 S_{\alpha} C_{\alpha} \right) \dot{\phi} = M_{\phi}(t)
\]
\[
\left( \frac{1}{12} m \ell^2 \right) \ddot{\theta} + \left( \frac{1}{12} m \ell^2 S_{\alpha} C_{\alpha} \right) \dot{\phi}^2 = M_{\theta}(t)
\]
(generated in Unit 4)

In these equations, angle \( \phi \) represents the angular rotation of frame \( F \) about the \( k \) direction, and angle \( \theta \) is as defined in the figure above. The second equation is simply a restatement of the boxed equation above involving the torque \( T_2 \) (note that torque \( M_{\theta}(t) \) is the same as torque \( T_2 \)). If, as assumed above, angle \( \phi \) and its derivatives are known, then the first equation represents a means of calculating the driving torque acting on \( F \) about the \( k \) direction.

\[
M_{\phi}(t) = \left( m d^2 + \frac{1}{12} m \ell^2 C_{\alpha}^2 \right) \dot{\phi} - \left( \frac{1}{6} m \ell^2 S_{\alpha} C_{\alpha} \right) \ddot{\phi} \omega \Omega
\]

Note that this equation was derived assuming frame \( F \) is light.

In this example, two MATLAB models are presented of the rotating bar system. The first model is a SimMechanics model, and the second is a Simulink model that solves the basic equations found in Units 2 and 4. A description of each of the models is presented below.

Model 1 – Simulink/SimMechanics:

The SimMechanics model consists of three bodies, the vertical column and horizontal arm of frame, \( F \), and the bar, \( B \). The two segments of frame \( F \) are both assumed to be slender bars. The vertical column is 24 (in) long and has a mass of 0.01 (slugs), and the horizontal arm is 12 (in) long and has a mass of 0.005 (slugs). The two segments are welded together 18 (in) above the bottom support. Bar \( B \) is also assumed to be slender. It has length \( \ell = 1 \) (ft) and a mass of 0.2 (slugs). The top layer of the model is shown below.

The two custom joints that connect the vertical column of the frame to the ground are like those used in Example 1 for the simple crank shaft. Custom Joint A (at the bottom of the column) allows for three axes of rotation, but it restricts all translational motion. The first revolute axis of this joint allows rotation of the column about the world \( Z \) axis. Custom Joint B (at the top of the column) allows for three axes of rotation and one axis of translation along the world \( Z \) axis. The first revolute axis of this joint allows for rotation about the world \( Z \) axis. As noted in Example 1, there are no redundant constraints connecting the column to the ground. Using this method of support, the model can be used to calculate the reaction forces and torques at the two joints and the driving torque at the lower support.

The Machine Environment block defines the analysis as forward dynamics and the gravity vector as \([0,0,0] \), indicating that weight forces will not be included. Ground block A is located at \([0,0,0] \) and Ground block B is located at \([0,0,24] \) (in) of the world coordinate system. The end of the horizontal arm is welded to
the vertical column at \([0,0,18]\) (in). The *revolute joint* that connects the *horizontal arm to the bar* is defined to be about the long axis (the *y* axis) of the arm.

![Example System II - Rotating Bar on a Rotating Frame](image)

Three *sensor/actuator ports* are included on the revolute joint. The *two* shown on the left side of the joint block are *actuators*. The block labeled IC provides a means to apply *initial conditions* to the joint. The Block Parameters dialog box shown below indicates the workspace variable “initialTheta” defines the initial angle of the bar in degrees, and the workspace variable “initialThetaDot” defines the initial angular velocity of the bar relative to the arm in radians/second.

![Block Parameters: Joint Initial Condition](image)

The *second actuator port* is used to define a *joint damper* to reduce the motion between the bar and the arm. The Block Parameters dialog box (shown below) for the Joint Spring & Damper block indicates the workspace variable “dampingCoefficient” supplies the value of the *torsional damping coefficient* and that the spring constant (stiffness) has been set to *zero*. Because the joint spring and damper are connected to a *revolute joint*, the spring and damping elements *apply torques* through the joint. Given the units specified in the dialog box, the spring constant \(k\) has units of ft-lb/rad and the damping constant \(b\) has units of ft-lb-s/rad.
There are three measurement subsystems and one motion subsystem in this model. Two of the measurement subsystems are for the joints connecting the column to the ground, and the third is for the connection between the arm and the bar. The force and torque components measured at the lower and upper supports of the column are those acting on the column by the supports, and they are expressed in the column’s coordinate system. The force and torque components measured at the revolute joint are those acting on the bar, and they are expressed in the bar’s coordinate system. The motion subsystem is identical to the one presented in Example 1.

Note finally that values of the initial angular velocity, constant angular acceleration, and the masses and inertia matrices of each of the bodies are all provided by workspace variables defined within the MATLAB script that controls execution of the model. Also, the geometries of the bodies will be provided by external stereolithographic files.

The first five sections of the MATLAB script are shown in Panel 1 below. The first section contains a simple statement of purpose of the script and the second defines values for the masses and inertia matrices for each of the bodies in the system. The next two sections define the motion data for frame $F$, the initial conditions for the angle $\theta$ and its derivative $\dot{\theta}$, and the damping coefficient. In particular, the frame is given an initial angular velocity of $1$ (cycle/sec) (or $2\pi$ (rad/sec)) and a constant angular acceleration of $1$ (rad/sec$^2$). The long axis of the bar is initially aligned at an angle of $60$ (deg) off the horizontal which is increasing at a rate of $3$ (rad/sec). The fifth section executes the SimMechanics model.

Panel 2 shows the script to plot the force and torque components at bottom of the column, and Panel 3 shows the script to plot the force components at the top of the column. Panel 4 shows the script to plot the force and torque components and angle $\theta$ of the bar. Recall that structures in the MATLAB workspace that hold the data for plotting are filled by the scopes in the measurement subsystems.
Panel 1 – MATLAB Script: (define mass and inertia and motion data, damping coefficient, and execute)

```matlab
% This M-file defines the input data for a SimMechanics model of Example system II, executes the SimMechanics model, and plots results.

%% Mass and Inertia data
% Column, Arm, and Bar are all treated as slender bars
massColumn = 0.01;  % (slugs)
lengthColumn = 24;   % (inches)
inertiaColumn = (1/12)*massColumn*((lengthColumn/12)^2)*...
[1,0,0; 0,1,0; 0,0,0];  % (slug-ft^2)

massArm = 0.005;     % (slugs)
lengthArm = 12;      % (inches)
inertiaArm = (1/12)*massArm*((lengthArm/12)^2)*...
[0,0,0; 0,1,0; 0,0,1];  % (slug-ft^2)

massBar = 0.2;       % (slugs)
lengthBar = 12;      % (inches)
inertiaBar = (1/12)*massBar*((lengthBar/12)^2)*...
[0,0,0; 0,1,0; 0,0,1];  % (slug-ft^2)

%% Motion data
% for Frame, F (initial angle assumed to be zero)
constantAngularAcceleration = 1.0;  % (rad/s^2)
initialAngularVelocity = 2*pi;      % (rad/s)

%% Initial motion data for Bar, B, and the damping coefficient
initialTheta = 60.0;    % (deg)
initialThetaDot = 3.0;  % (r/s)
dampingCoefficient = 0.02;  % (ft-lb/s/rad)

%% Execute the SimMechanics model
sim('ExampleSystem2.slx')
```

Panel 2 – MATLAB Script: (plot the force and torque components at bottom of column)

```matlab
% Plot the components of the force on the bottom of the column (A)
figure(1); clf;
subplot(3,1,1); plot(forceComponentsA.time,forceComponentsA.signals(1).values);
grid; title('Force Components at Bottom of Column'); xlabel('Time (sec)'); ylabel('n_1 Component (lb)')

subplot(3,1,2); plot(forceComponentsA.time,forceComponentsA.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('n_2 Component (lb)')

subplot(3,1,3); plot(forceComponentsA.time,forceComponentsA.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('k Component (lb)')

% Plot the components of the torque on the bottom of the column (A)
figure(2); clf;
subplot(3,1,1); plot(torqueComponentsA.time,torqueComponentsA.signals(1).values);
grid; title('Torque Components at Bottom of the Column'); xlabel('Time (sec)'); ylabel('n_1 Component (ft-lb)')

subplot(3,1,2); plot(torqueComponentsA.time,torqueComponentsA.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('n_2 Component (ft-lb)')

subplot(3,1,3); plot(torqueComponentsA.time,torqueComponentsA.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('k Component (ft-lb)')
```
Panel 3 – MATLAB Script: (plot the force and torque components at top of column)

```matlab
% Plot the components of the force on the top of the column (B)
figure(3); clf;
subplot(3,1,1); plot(forceComponentsB.time,forceComponentsB.signals(1).values);
grid; title('Force Components at Top of Column');
xlabel('Time (sec)'); ylabel('n_1 Component (lb)')
subplot(3,1,2); plot(forceComponentsB.time,forceComponentsB.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('n_2 Component (lb)')
subplot(3,1,3); plot(forceComponentsB.time,forceComponentsB.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('k Component (lb)')
```

Panel 4 – MATLAB Script: (plot the force and torque components and \( \theta \) the angle of the bar)

```matlab
% Plot the components of the force on the bar
figure(4); clf;
subplot(3,1,1); plot(forceComponentsBar.time,forceComponentsBar.signals(1).values);
grid; title('Force Components on the Bar');
xlabel('Time (sec)'); ylabel('e_1 Component (lb)')
subplot(3,1,2); plot(forceComponentsBar.time,forceComponentsBar.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('e_2 (n_2) Component (lb)')
subplot(3,1,3); plot(forceComponentsBar.time,forceComponentsBar.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('e_3 Component (lb)')

% Plot the components of the torque on the bar
figure(5); clf;
subplot(3,1,1); plot(torqueComponentsBar.time,torqueComponentsBar.signals(1).values);
grid; title('Torque Components on Bar');
xlabel('Time (sec)'); ylabel('e_1 Component (ft-lb)')
subplot(3,1,2); plot(torqueComponentsBar.time,torqueComponentsBar.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('e_2 (n_2) Component (ft-lb)')
subplot(3,1,3); plot(torqueComponentsBar.time,torqueComponentsBar.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('e_3 Component (ft-lb)')

% Plot the angle of the bar, theta
figure(6); clf;
plot(angleBar.time,angleBar.signals.values);
grid; title('Angle of the Bar, Theta');
xlabel('Time (sec)'); ylabel('Angle (deg)')
```
Simulation Results:

When the script is executed, a geometric representation of the system is displayed. In this case, an external stereolithographic file is used to represent the three-dimensional geometry of the rotating arm system. As the simulation progresses, the motion of the system is animated. The diagram shown here is the position of the system at \( t = 0.9 \) (sec). Note that, as shown in the diagram above, the frame rotates about the world Z axis, and the bar rotates relative to the frame about its horizontal arm.

The first result shown to the right is \( \theta(t) \) the angle of the bar relative to the horizontal. Note that as specified in the M-file, the angle starts at 60 (deg) and begins to increase (recall the initial value of \( \dot{\theta} \) is 3 (rad/sec)). As the frame rotates, the simulation results show the bar is attracted to and finally settles in the steady-state position \( \theta = 0 \). The existence of this equilibrium position is easily verified by setting \( \dot{\theta} \) and \( \ddot{\theta} \) to zero in the equation of motion for the \( \theta \) and solving the resulting algebraic equation. That process yields two equilibrium positions, one at \( \theta = 0 \) and one at \( \theta = \frac{\pi}{2} \). It appears from these results that \( \theta = 0 \) is a stable equilibrium position.

The next set of results (shown below) are the force components acting on the column at the lower and upper supports. Recall the \( n_2 \) direction points outward along the horizontal arm, \( k \) points upward along the vertical column, and \( n_1 = n_2 \times k \). Note the \( n_1 \) components of the forces oscillate and settle into relatively small steady state values, whereas the \( n_2 \) components continue to increase (in absolute value) as the angular speed of the frame increases. The steady state \( n_1 \) components likely depend linearly on the angular acceleration of the frame, while the steady state \( n_2 \) components likely depend quadratically on its angular speed. The weight forces were not included in the analysis, so the \( k \) components of the forces are zero.
The torque components acting on the column at the bottom support are shown to the right. The $k$ component of this torque represents the driving torque on the frame. Note that it oscillates and settles into a steady state value associated with the constant angular acceleration of the frame. The custom joints only restrict translational motion of the frame, so the only non-zero torque generated at these locations is at the lower support due to the motion actuation at that point. The torque components at the upper support are all zero and are not shown here.
The force and torque components the frame exerts on the bar through the pin at $G$ are shown below. The components are resolved in the bar frame. Recall the $e_1$ direction is along the bar, and the $e_2$ and $e_3$ directions are perpendicular to the bar. The $e_2$ direction is directed along the horizontal arm as is $n_2$.

The force components in the $e_1$ and $e_3$ directions oscillate and settle into steady-state values. The steady state value in the $e_1$ direction is non-zero and is associated with the constant angular acceleration of the frame. The steady state value in the $e_3$ direction is zero. The $e_2$ (or $n_2$) component of the force continues to increase (in absolute value) as the angular speed of the frame increases. It likely increases quadratically with the angular speed.

The torque component in the $e_2$ direction is the damping torque associated with the damper located between the frame and the bar. This torque linearly opposes the angular velocity of the bar relative to the frame, so its value oscillates and settles to zero as the bar acquires its steady-state position. The $e_3$ component of the torque oscillates and settles to a small constant value associated with the angular acceleration of the frame. The torque component in the $e_1$ direction is zero, because the bar is assumed to be slender.
Model 2 – Simulink only:

The Simulink only model is based strictly on the analytical equations presented above. Specifically, given frame \( F \) has angular velocity \( \Omega(t) \), constant angular acceleration \( \dot{\Omega}_0 \), damping torque \( T_2 = -b \dot{\theta} \), and initial values for \( \theta \) and \( \dot{\theta} \), it solves the differential equation

\[
\ddot{\theta} + \Omega^2 S_\theta C_\theta = \frac{T_2}{\left(\frac{1}{12} m \ell^2\right)} = \frac{T_2}{I_{\text{bar}}}
\]

to find \( \theta(t) \). Then it uses the specified values of \( \Omega \), \( m \), \( \ell \), and \( d \) to find the unknown force and torque components acting on the bar and the driving torque on the frame. Recall these equations were derived assuming the mass and inertia of the frame are zero. The angular motion of the frame (\( \dot{\Omega} \) and \( \Omega \)), the length of the horizontal arm (\( d \)), the mass and length of the bar (\( m \) and \( \ell \)), and the damping coefficient \( b \) are all as defined in Model 1. A MATLAB script controls the execution of the model.

The overall structure (upper layer) of the model is shown below. Note in the lower left of the diagram there are two integrators (annotated by "1/s"). The first integrates the signal \( \dot{\theta}(t) \) to get \( \theta(t) \), and the second integrates the signal \( \dot{\theta}(t) \) to get \( \dot{\theta}(t) \). The initial values of \( \dot{\theta}(t) \) and \( \theta(t) \) are provided by the workspace variables “initialThetaDot” and “initialTheta”, respectively. Given current values for \( \dot{\theta} \) and \( \theta \), the current value of \( \ddot{\theta} \) is calculated using the differential equation. That is,

\[
\ddot{\theta} = -\Omega^2 S_\theta C_\theta - (b/I_{\text{bar}}) \dot{\theta}
\]

This calculation is done in the subsystem labelled “Calculate theta_double_dot”. The model has five other subsystems that use the signals \( \theta(t) \) and \( \dot{\theta}(t) \) to calculate and plot the angle \( \theta(t) \) in degrees, the damping torque \( T_2(t) \), the bar torque \( T_3(t) \), the bar force components, and the driving torque on the frame.
The subsystem that calculates $\dot{\theta}$ is shown below. In the upper left corner, an integrator is used to generate the signal $\Omega(t)$ given its initial value $\Omega_0$ and the constant value of $\dot{\Omega}$. The values of $\Omega_0$ and $\dot{\Omega}$ are given by the workspace variables "initialAngularVelocity" and "constantAngularAcceleration", respectively. The resulting signal is sent to the MATLAB function labelled “Differential Equation” that calculates the signal $\Omega^2 S_{\theta} C_{\theta}$. The lower portion of the block calculates the signal \((b / I_{\text{bar}}) \dot{\theta}\). The negatives of these two signals are added to give $\dot{\theta}(t)$. Note that the signal $\Omega(t)$ is also sent to a “Goto” block so the signal is available at other locations in the model.

MATLAB functions are simply scripts used to perform the necessary calculations. The “Differential Equation Function” script is shown below.

```
function y = fcn(capOmega,theta)
    y = (capOmega^2)*sin(theta)*cos(theta);
```

The subsystem that calculates the bar torque $T_3$ and its associated MATLAB function are shown below. Note the function requires values of $I_{\text{bar}}$ (the inertia of the bar), $\Omega$, $\dot{\Omega}$, $\theta$, and $\dot{\theta}$. Values of $I_{\text{bar}}$ and $\dot{\Omega}$ are provided by workspace constants, values of $\Omega$ are provided by the “From” block (associated with the “Goto” block discussed above), and values of $\theta$ and $\dot{\theta}$ are provided by the solution of the differential equation shown in the top layer.
The subsystem that calculates the driving torque on the frame and its associated MATLAB function are shown below. In addition to the variables required by the function “Bar Torque T_3”, this function (labelled “Driving Torque”) requires values of $m$ and $d^2$ which are provided by workspace constants.
The subsystem that calculates the damping torque \( T_2 = -b \dot{\theta} \) is shown below. The value of the damping coefficient \( b \) is provided by the workspace variable “dampingCoefficient”.

![Damping Subsystem Diagram](image)

The **subsystem** that calculates the **force components** acting on the **bar** and its **associated MATLAB function** are shown below. The body-fixed components of the bar force are stored in a \( 3 \times 1 \) vector. The first statement in the function is used to **size** and **zero** the array.

![Bar Force Component Function Diagram](image)

```matlab
function barForceComponent = fcn(capOmega, capOmegaDot, theta, massBar, lengthArm)

barForceComponent = zeros(3,1);

barForceComponent(1) = -massBar*lengthArm*capOmegaDot*cos(theta);
barForceComponent(2) = -massBar*lengthArm*(capOmega^2);
barForceComponent(3) = -massBar*lengthArm*capOmegaDot*sin(theta);
```

The **first five sections** of the MATLAB script are shown in the Panel 1 below. The first section has a brief description of the **purpose** of the script, the second, third, and fourth sections define the **workspace variables**, and the fifth section **executes** the Simulink model.
Panel 1 – MATLAB Script: (define workspace variables and execute the Simulink model)

```matlab
%% Example System II - Model 2
% This file defines the values of the input variables for a Simulink model
% of Example System II, executes the model, and plots results of the
% model.
% The model uses Simulink to solve the model equations of the system
% dynamics. It assumes that the angular acceleration of the frame,
% F is constant. It also assumes the motion of the bar, B relative to F
% is free. The torque T_2 is assumed to be due to damping, only.

%% Physical data for the bar, B and the frame, F
massBar   = 0.2;  % (slugs)
lengthBar = 1;    % (ft)
inertiaBar = (1/12)*massBar*(lengthBar^2);  % (slug-ft^2)
lengthArm = 1;    % (ft)
lengthArmSQ = lengthArm^2;  % (ft^2)

%% Motion data
constantAngularAcceleration = 1.0;  % (rad/s^2)
initialAngularVelocity      = 2*pi;  % (rad/s)

%% Initial motion data for Bar, B
initialTheta       = 60.0*(pi/180);   % (rad)
initialThetaDot    = 3.0;            % (r/s)
dampingCoefficient = 0.02;           % (ft-lb-s/rad)

%% Execute the Simulink model
sim('ExampleSystem2Model2.slx')
```

The sixth section of the MATLAB script is shown in Panels 2 and 3 below. Panel 2 shows the script that plots the bar angle $\theta$, the $e_2$ and $e_3$ components of torque acting on the bar, and the driving torque on the frame. Panel 3 shows the script that plots the $e_1$, $e_2$, and $e_3$ components of force acting on the bar.

Panel 2 – MATLAB Script: (plot the angle of the bar and the torque results)

```matlab
%% Plot the time-varying results
% Plot the angle of the bar, theta
figure(7); clf; plot(angleBar.time,angleBar.signals.values);
grid; title('Angle of the Bar, Theta'); xlabel('Time (sec)'); ylabel('Angle (deg)')

% Plot the e_3 component of the torque at center of bar, B
figure(8); clf; plot(torqueBar.time,torqueBar.signals.values);
grid; title('Torque Component T_3 Acting on Bar'); xlabel('Time (sec)'); ylabel('Torque (ft-lb)')

% Plot the damping torque in the e_2 direction at center of bar, B
figure(9); clf; plot(torqueDamping.time,torqueDamping.signals.values);
grid; title('Damping Torque in e_2 direction Acting on Bar'); xlabel('Time (sec)'); ylabel('Torque (ft-lb)')

% Plot the driving torque on frame, F
figure(10); clf; plot(torqueDrive.time,torqueDrive.signals.values);
grid; title('Driving Torque Acting on Frame, F'); xlabel('Time (sec)'); ylabel('Torque (ft-lb)')```

Copyright © James W. Kamman, 2017
Panel 3 – MATLAB Script: (plot the force results)

```matlab
% Plot the components of the force on the bar
figure(11); clf;
subplot(3,1,1); plot(forceComponentsBar.time,forceComponentsBar.signals(1).values);
grid; title('Force Components on the Bar');
xlabel('Time (sec)'); ylabel('e_1 Component (lb)')

subplot(3,1,2); plot(forceComponentsBar.time,forceComponentsBar.signals(2).values);
grid; xlabel('Time (sec)'); ylabel('e_2 (n_2) Component (lb)')

subplot(3,1,3); plot(forceComponentsBar.time,forceComponentsBar.signals(3).values);
grid; xlabel('Time (sec)'); ylabel('e_3 Component (lb)')
```

Simulation Results:

The forces and torques acting on the bar and the angular response of the bar produced by Model 2 are all identical to those produced by Model 1. The driving torque on frame $F$ produced by Model 2 follows the same pattern as that produced by Model 1, but the peak values are slightly lower. See the Model 2 result in the figure below. The reason for this slight difference is that the two models assume different inertias for frame $F$. Model 2 assumes that the inertia of $F$ is zero, whereas Model 1 assumes it is small but non-zero. So, the slightly larger inertia of Model 1 requires a slightly higher driving torque. For comparison, see the $k$ component of the torque on the bottom of the column shown above for Model 1.
Example 3: Free Motion of an Upright, Two-Wheeled Bicycle

(See the bicycle references for more details)

This example includes three models of the free motion of an upright bicycle. The first model is a MATLAB script that uses MATLAB’s ODE45 to integrate the equations of motion (EOM). The second and third models consist of MATLAB scripts coupled with Simulink models to integrate the EOM. The EOM for the bicycle were developed using Kane’s equations (or D’Alembert’s Principle) in Unit 5 of this volume. The following sections provide information related to the analytical model of the bicycle including definitions of variables, initial conditions, and geometric, mass, and inertia data. Finally, details of the three numerical models are provided.

Configuration: Aligned Position

The analysis that follows is for a two-wheeled, upright bicycle as shown. The bicycle is modeled as four interconnected bodies – the rear wheel A, the rear frame and rider B, the steering column and fork C, and the front wheel D. The bicycle is assumed to be moving freely on a horizontal surface with the rear wheel contacting the surface at point P and the front wheel contacting the surface at point Q. The distance between the contact points is the wheelbase w.

The steering column is assumed to be tilted relative to the vertical at an angle \( \lambda \). This angle is a right-hand rotation about the \( \hat{N}_2 = \hat{N}_3 \times \hat{N}_1 \) direction. The projection of the steering axis onto the horizontal plane is assumed to be a distance c in front of the contact point Q. Positive rotations of the front and rear wheels are also measured about the \( \hat{N}_2 \) direction, so for forward motion of the bicycle, both angles \( \theta_F \) and \( \theta_R \) are negative.

In the configuration shown above, the \( x \) and \( z \) axes fixed in the rear frame B are aligned with the global (inertial) directions defined by the unit vector set \( R : (\hat{N}_1, \hat{N}_2, \hat{N}_3) \). The unit vectors \( \hat{N}_1 \) and \( \hat{N}_2 \) are parallel to the horizontal plane, and the unit vector \( \hat{N}_3 \) points vertically downward.

The points \( G_A, G_B, G_C, \) and \( G_D \) are the mass centers of the bodies and are located using the following geometric data for the bicycle. Recall that body B includes both the rear frame and the rider. All the data given in the table are referenced to the global directions.
It can be shown using a geometric analysis that a point $S$ located on the steering axis (common to both the rear frame and the steering column) can be located using the following equations.

Horizontal distance from $G_A$ to $S$: $$x_S = (w+c)C_x^2 - r_{rw}S_xC_x$$

Vertical distance from $G_A$ to $S$: $$z_S = x_S \tan \lambda$$

Two additional sets of unit vectors are defined. The set $B: \left( b_1, b_2, b_3 \right)$ is fixed in the rear frame $B$, with $b_1$ pointing in the $x$-direction (forward), $b_2$ pointing in the $y$-direction (to the rider’s right), and $b_3 = b_1 \times b_2$.

In the figure above, the unit vectors of $B$ are aligned with the unit vectors of $R$. A second set $C:\left( \xi_1, \xi_2, \xi_3 \right)$ is fixed in the steering column $C$. In the position where frame $B$ is aligned with frame $R$, the unit vectors in frame $C$ are directed as follows. Unit vectors $\xi_1$ and $\xi_3$ are at an angle $\lambda$ with their counterparts in the $B$ frame, and $\xi_2$ is aligned with $b_2$.

For the analysis that follows, the position vectors of $G_B$ and $S$ relative to $G_A$ are expressed in the $B$ frame, and the position vectors of $G_C$ and $G_D$ relative to $S$ are expressed in the $C$ frame as follows.

$$\mathbf{r}_{G_B/G_A} = x_B b_1 - z_B b_3$$

$$\mathbf{r}_{S/G_A} = x_S b_1 - z_S b_3$$

$$\mathbf{r}_{G_C/S} = x_C \xi_1 + z_C \xi_3$$

$$\mathbf{r}_{G_D/S} = x_D \xi_1 + z_D \xi_3$$

The fixed values of the components of these vectors are provided in the adjacent table. The values are based on the data and formulae provided above.

**Configuration:** General Position

The figure to the right shows the bicycle in a more general configuration. The figure illustrates five of the six angles used in the analysis. The rear frame $B$ is oriented relative to the inertial frame $R$ using a 3-1-2 orientation angle sequence. The steering column $C$ and rear wheel are each oriented relative to $B$ by single angles, and the front wheel is oriented relative to $C$ by a single angle.
The 3-1-2 orientation angles \((\psi, \phi, \theta)\) orient the **rear frame** relative to \(R\). They represent the **yaw**, **roll** (or lean), and **pitch** angles of \(B\). The diagram shows **positive** yaw and roll (or lean) angles. The pitch angle is not shown. The value of the pitch angle is small and is determined to ensure the front wheel remains in contact with the horizontal surface. The diagram also shows a **positive steering angle** \(\delta\). So, for the angles shown, the bicycle is **rolling** and turning to the **right**. As noted earlier, the angles of the wheels relative to the frame are both **negative** for **forward motion**.

**Equations of Motion**

The **equations of motion** describing the **free motion** of the bicycle on a horizontal plane are **derived** and **presented** in Unit 5 of the volume. The equations are developed based on the **implicit assumption** that the point of the rear wheel that contacts the ground has **zero velocity**. This allows the equations to be written in terms of **six generalized coordinates**, the three orientation angles \((\psi, \phi, \theta)\) of the rear frame, the steering angle \(\delta\), and the angles of spin \((\theta_R, \theta_F)\) of the rear and front wheels relative to the frame.

As the bicycle has only **three degrees of freedom**, the application of Kane’s equations (or D’Alembert’s principle) provides only **three equations of motion**. These equations are supplemented with **three constraint equations** that require the **front wheel** remain in **contact** with, and **not slip**, on the horizontal plane. Finally, to track the **path** of the rear wheel contact point, the equations are supplemented with three equations that track the coordinates of that point. The \(x\) and \(y\) coordinates show the **path** of the contact point on the horizontal plane, and the \(z\) coordinate is provided as a numerical check, as its value should be **zero**.

For **full development** of the equations of motion, see Unit 5. Due to their length, the equations are **not** repeated here.

**Initial Conditions**

It is assumed that the roll angle \(\phi\), the rear wheel angle \(\theta_R\), and the steering angle \(\delta\) form a set of three independent generalized coordinates. As independent coordinates, their **initial values** can be **specified independently**. The values of the rest of the coordinates must be chosen to be **consistent** with the **rolling constraints**.

The **distance** of the front wheel contact point \(Q\) from the horizontal surface is **independent** of the values of the \(x\) and \(y\) **coordinates** of the **rear contact point** \(P\), the **yaw angle** of the bicycle \(\psi\), and the **angle of the front wheel** relative to the front frame \(\theta_F\). Hence, their **initial values** can be **arbitrarily specified**. For **simplicity**, their **initial values** are all taken to be **zero**. That leaves the **initial value** of the **rear frame pitch angle** \(\theta\) to be determined.
It is shown in Unit 5 that the following non-linear, algebraic equation can be solved to find the pitch angle $\theta$, given arbitrary values of roll angle $\phi$ and steering angle $\delta$.

\[
-\left(r_{RW} + x_s S_\phi + z_s C_\phi\right) C_\phi + x_D S_\phi S_\delta - x_D C_\phi C_\delta S_{\lambda+\theta} + z_D C_\phi C_{\lambda+\theta} \\
+ r_{FW} \left(\left(S_\phi S_\delta - C_\phi C_\delta S_{\lambda+\theta}\right)^2 + C_\phi^2 C_{\lambda+\theta}^2\right)^{\frac{1}{2}} = 0
\]

Note that this equation does not include $\theta_R$ the angle of the rear tire relative to the bicycle frame, and (it can be shown) that the pitch angle is zero if the roll and steering angles are zero.

Mass and Inertia Data

The masses and inertia matrices of the four bodies are taken to be

- $m_A = 2 \text{ (kg)}$
- $m_B = 85 \text{ (kg)}$
- $m_C = 4 \text{ (kg)}$
- $m_D = 3 \text{ (kg)}$

The inertia matrices are all referenced to the inertial directions when the bicycle is in its aligned position. That is, they are referenced to the directions defined by the $B$-frame. Consequently, the inertia matrix $[I_{G_B}]^B$ must be transformed into the $C$-frame using a transformation matrix associated with a single rotation through the angle $\lambda$. Due to the symmetries of the front and rear wheels, no transformations are required for their inertia matrices.

Using the results from Unit 1 of this volume, the inertia matrix for the steering column and fork ($B$) can be transformed as follows.

\[
[I_{G_C}]^C = [C_\lambda 0 -S_\lambda] [I_{G_B}]^B [C_\lambda 0 S_\lambda]
\]

Model 1 – MATLAB Script Using ODE45 to Integrate the Equations of Motion

The MATLAB script consists of two modules. The main module defines a set of global variables, defines the geometric and physical characteristics, sets the initial conditions, initializes the state vector, calls the MATLAB function ODE45 to integrate the equations of motion over a specified time interval, and plots the
results. As part of this process, the main module provides the name of the function that calculates the derivative of the state vector. This function is the second module of the script.

Main Script

Panel 1 shows the first section of the main script that contains a brief functional description and clears the variables from the workspace. It notes that the state vector of the system is defined to be

\[
y = \begin{bmatrix}
  \phi \\
  \theta_R \\
  \delta \\
  \psi \\
  \theta \\
  \theta_F \\
  \dot{\phi} \\
  \dot{\theta}_R \\
  \dot{\delta} \\
  \dot{\psi} \\
  \dot{\theta} \\
  \dot{\theta}_F \\
  x_P \\
  y_P \\
  z_P
\end{bmatrix}^T
\]

Note that the first twelve variables are the angles and their first derivatives, and the last three are the global coordinates of point P. The first three angles are assumed to be independent variables, and the last three are assumed to be dependent variables. The dependent variables are related to the independent ones using the constraint equations described above.

Panel 1: Main Script – Functional Description and Clearing Workspace

```matlab
%% This script solves the equations of motion (EOM) of a bicycle using ODE 45
%% The state vector "y" for the bicycle is a 15x1 vector defined as
%% y = [phi;thetaR;delta;psi;theta;thetaF;phidot;thetaRDdot;deltaDot;psiDot;thetaDot;thetaFDot;xP;yP;zP]
phi = roll angle of rear frame
thetaR = rolling angle of the rear wheel
delta = steering angle
psi = yaw angle of rear frame
theta = pitch angle of the rear frame
thetaF = rolling angle of the front wheel
xP,yP,zP = global coordinates of P (contact point of the rear wheel)

The angles (phi,thetaR,delta) and their derivatives are taken as independent. The angles (psi,theta,thetaF) and their derivatives are taken as dependent. The EOM do not depend on the coordinates of P.

Phase I of the script:
1. Defines the bicycle’s geometric parameters
2. Defines the bicycle’s mass and inertia properties
3. Defines the initial yaw angle (psi), the initial rear wheel rolling angle (thetaR), and the initial front wheel rolling angle to all be zero.
4. Given an initial rear-frame roll angle (phi) and steering angle (delta), and initial rear-frame pitch angle is calculated to place the front wheel in contact with the surface.
5. Initial values are specified for {u_I} the vector of initial derivatives of (phi,thetaR,delta)
6. The bicycle transformation matrices, the C-frame components of N_3, and the constraint equation matrices (C_1 and C_2) relating {u_D} the derivatives of the dependent angles to {u_I} the derivatives of the independent angles at the initial state are calculated.
7. Using the constraint equation matrices, {u_D} the derivatives of the dependent angles are calculated.
8. The above results are used to define the initial state vector for the bicycle.

Phase II of the script:
1. Define the time space [tMin:tDelta:tMax] on which to present the solution. Values of the elements of the state vector are provided at each of the times within the time space.
2. Define a function handle for the function that computes dy/dt, the derivative of the state vector.
3. Call MATLAB function "ode45" to integrate the equations of motion on the defined time interval.

Phase III of the script:
Plot the results of the integration process.

clear variables;
```

Copyright © James W. Kamman, 2017

Volume II – Unit 7: page 28/48
Panels 2 and 3 below show the next seven sections of the script. The first two define a set of global variables and three constants used later in the script. The global variables are used in the “calculateLoopClosureEquation” and “bicycleEOM” functions. The remaining sections define the geometric parameters of the bicycle and the initial values for the angles, coordinates of P, and the independent speeds. The initial values of five of the six angles are chosen arbitrarily. The initial value of the sixth angle (the pitch angle) is determined by solving the non-linear algebraic equation given above. This finds the pitch angle that forces the z coordinate of point Q to be zero. The solution is done using the MATLAB function “fzero”. The function “calculateLoopClosureEquation” calculates the z coordinate of Q as a function of the pitch angle. Trigonometric functions of the angles are also calculated for later use.

Panel 2: Main Script – Define Global Variables and Constants

```matlab
%% Define global variables required for the analysis
global rRW rFW lambda XB zB XS zS XC XD zD
global sinLambda cosLambda
global sinPhi cosPhi sinDelta cosDelta
global mA mB mC mD aG
global inGA inGB inGC inGD
%% Define acceleration of gravity and radian and degree conversions
aG      = 9.81; % g (N/kg) or (m/s^2)
deg2Rad = pi/180; % conversion from degrees to radians
rad2Deg = 180/pi; % conversion from radians to degrees
%% Define orientation angles for the bicycle and initial coordinates for P
phiDeg    = 0; % roll angle (deg)
deltaDeg  = 0; % steering angle (deg)
phi       = phiDeg*deg2Rad; % roll angle(rad)
delta     = deltaDeg*deg2Rad; % steering angle(rad)
psi       = 0.0; % yaw angle (rad)
thetaR    = 0.0; % define initial thetaR to be zero (rad)
thetaF    = 0.0; % define initial thetaF to be zero (rad)
xP = 0.0; yP = 0.0; zP = 0.0;
%% Define u_I the vector of independent speeds
phiDot    = 0.50; % roll rate (rad/s)
thetaRDot = -15.3333333; % rear wheel spin rate(rad/s)
deltaDot  = 0.00; % steering rate(rad/s)
u_I = [phiDot; thetaRDot; deltaDot];
%% Define the geometric parameters of the bicycle
[rRW, rFW, w, c, lambda, XB, zB, XS, zS, XC, XD, zD] = defineBicycleGeometricParameters(deg2Rad); % defi ned functions
sinLambda = sin(lambda); cosLambda = cos(lambda);
%% Find the initial pitch angle, theta
fun = @calculateLoopClosureEquation;
theta0 = [-pi/10 pi/10];
theta = fzero(fun,theta0);
sinTheta = sin(theta); cosTheta = cos(theta);
end
```

Panel 3: Main Script – Define Initial Conditions, Geometric Parameters, and Some Trigonometric Functions

```matlab
%% Define orientation angles for the bicycle and initial coordinates for P
phiDeg    = 0; % roll angle (deg)
deltaDeg  = 0; % steering angle (deg)
phi       = phiDeg*deg2Rad; % roll angle(rad)
delta     = deltaDeg*deg2Rad; % steering angle(rad)
psi       = 0.0; % yaw angle (rad)
thetaR    = 0.0; % define initial thetaR to be zero (rad)
thetaF    = 0.0; % define initial thetaF to be zero (rad)
xP = 0.0; yP = 0.0; zP = 0.0;
%% Define u_I the vector of independent speeds
phiDot    = 0.50; % roll rate (rad/s)
thetaRDot = -15.3333333; % rear wheel spin rate(rad/s)
deltaDot  = 0.00; % steering rate(rad/s)
u_I = [phiDot; thetaRDot; deltaDot];
%% Calculate sine and cosine of all angles, except theta
sinPsi    = sin(psi); cosPsi    = cos(psi);
sinPhi    = sin(phi); cosPhi    = cos(phi);
sinDelta  = sin(delta); cosDelta  = cos(delta);
%% Define the geometric parameters of the bicycle
[rRW, rFW, w, c, lambda, XB, zB, XS, zS, XC, XD, zD] = defineBicycleGeometricParameters(deg2Rad); % defined functions
sinLambda = sin(lambda); cosLambda = cos(lambda);
%% Find the initial pitch angle, theta
fun = @calculateLoopClosureEquation;
theta0 = [-pi/10 pi/10];
theta = fzero(fun,theta0);
sinTheta = sin(theta); cosTheta = cos(theta);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);
```
Panels 4 and 5 show *details* of the functions that define the *bicycle’s geometric parameters* and calculate the *loop closure equation*.

### Panel 4: Main Script – Define the Bicycle’s Geometric Parameters

```matlab
function [rRW,rFW,w,c,lambda,xB,zB,xS,zS,xC,zC,xD,zD] = defineBicycleGeometricParameters(deg2Rad)
% Define geometric parameters of the bicycle
rRW = 0.30; % (m)
rFW = 0.35; % (m)
w = 1.02; % (m)
c = 0.08; % (m)
lambdaDeg = 18; % fixed steering column tilt angle (deg)
lambda = lambdaDeg*deg2Rad; % fixed steering column tilt angle (rad)
sinLambda = sin(lambda); cosLambda = cos(lambda);
xB = 0.3; % (m)
zB = (0.9 - rRW); % (m)
xS = (((w + c)*cosLambda) - rRW*sinLambda)*cosLambda; % (m)
zS = xS*sinLambda/cosLambda; % (m)
tempMatrix = [cosLambda, 0, -sinLambda; 0, 1, 0; sinLambda, 0, cosLambda];
xHat_C = (0.9 - xS); zHat_C = (zS - (0.7 - rRW));
tempVector = tempMatrix*[xHat_C; 0; zHat_C];
xC = tempVector(1); zC = tempVector(3);
xHat_D = (w - xS); zHat_D = (rRW + zS - rFW);
tempVector = tempMatrix*[xHat_D; 0; zHat_D];
xD = tempVector(1); zD = tempVector(3);
```

### Panel 5: Main Script – Calculate Loop Closure Equation Function

```matlab
function zQ = calculateLoopClosureEquation(theta)
global rRW rFW lambda xS zS xD zD
global sinPhi cosPhi sinDelta cosDelta

sinTheta = sin(theta); cosTheta = cos(theta);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);
temp = rFW*sqrt()(((sinPhi*sinDelta)-(cosPhi*cosDelta*sinLambdaPTheta))^2) + ...
     (cosPhi*cosLambdaPTheta)^2);

zQ = -(rRW*cosPhi) - (xS*cosPhi*sinTheta) - (zS*cosPhi*cosTheta) + ...
     (xD*((sinPhi*sinDelta)-(cosPhi*cosDelta*sinLambdaPTheta))) + ...
     (zD*cosPhi*cosLambdaPTheta) + temp;
```

Panel 6 shows the *next five sections* of the script. *Functions* are called to define the bicycle’s *mass* and *inertia properties*, the bicycle’s *transformation matrices*, the *C-frame components of the unit vector* $\mathbf{N}_3$, and the *constraint matrices* $\left[ C_1 \right]$, $\left[ C_2 \right]$, and $\left[ J \right]$. The *initial values* of the dependent speeds $\psi$, $\dot{\theta}$, and $\dot{\theta}_p$ are then calculated using the *initial values* of the independent speeds and the *constraint matrix* $\left[ J \right]$.

Panel 7(a) shows details of the function that defines the *mass* and *inertia properties* of the bicycle, and Panel 7(b) shows details of the functions that calculate the *transformation matrices* and extract the *C-frame components of the unit vector* $\mathbf{N}_3$. Panel 8 shows details of the function that calculates the *constraint matrices* $\left[ C_1 \right]$ and $\left[ C_2 \right]$. 

```matlab
%% Define the mass and inertia properties of the bicycle
[mA,mB,mC,mD,inGA,inGB,inGC,inGD] = defineBicycleMassInertiaProperties(sinLambda,cosLambda);

%% Calculate the bicycle transformation matrices
[R_R2B,R_B2C,R_R2C] = calculateBicycleTransformationMatrices...
    (sinPsi,cosPsi,sinPhi,cosPhi,sinTheta,cosTheta,sinLambda,cosLambda,sinDelta,cosDelta);

%% Define the components of N_3 in the C frame
[N_3] = findCFrameComponentsofN3(R_R2C);

%% Calculate the constraint matrices C_1, C_2, and J
[C_1,C_2] = calculateConstraintMatrices...
    (rRW,rFW,xS,zS,xD,zD,sinPhi,cosPhi,sinTheta,cosTheta,N_3,R_B2C);

%% Calculate the matrix J
J = -inv(C_2)*C_1;

%% Calculate u_D the vector of dependent speeds
u_D = J*u_I; psiDot = u_D(1); thetaDot = u_D(2); thetaFDot = u_D(3);
```

Panel 7(a): Main Script – Function to Define Bicycle Mass/Inertia Properties

```matlab
function [mA,mB,mC,mD,inGA,inGB,inGC,inGD] = defineBicycleMassInertiaProperties...
    (sinLambda,cosLambda)

% All masses in (kg); All inertias in (kg-m^2). All inertia values given relative to global xyz
% Inertia matrix for C is transformed into the C frame using the steering column angle lambda.
mA = 2.0; % mass
inGAxx = 0.0603; inGAyy = 0.12; inGAzz = inGAxx; % inertia values
inGA = [inGAxx, 0, 0; 0, inGAyy, 0; 0, 0, inGAzz]; % inertia matrix in B frame
mB = 85.0; % mass
inGBxx = 9.2; inGBxz = -2.4; inGByy = 11.0; inGBzz = 2.8; % inertia values
inGB = [inGBxx, 0, -inGBxz; 0, inGByy, 0; -inGBxz, 0, inGBzz]; % inertia matrix in B frame
R_B2Cprime = [cosLambda, 0, -sinLambda; 0, 1, 0; sinLambda, 0, cosLambda];
mC = 4.0; % mass
inGCxx = 0.05892; inGCxz = 0.00756; inGCyy = 0.06; inGCzz = 0.00708; % inertia values
inGC = [inGCxx, 0, -inGCxz; 0, inGCyy, 0; -inGCxz, 0, inGCzz]; % in global frame
inGC = R_B2Cprime*(inGC*R_B2Cprime'); % inertia matrix in C frame
mD = 3.0; % mass
inGDxx = 0.1405; inGDyy = 0.28; inGDzz = inGDxx; % inertia values
inGD = [inGDxx, 0, 0; 0, inGDyy, 0; 0, 0, inGDzz]; % inertia matrix in D frame
```
Panel 7 (b): Main Script – Functions to Calculate Transformation Matrices and Extract the C-frame Components of $N_3$

```matlab
% R_R2B = calculateTransformationMatrixFromAngles312; Compute the transformation matrix
R_R2B = zeros(3);
% first row
R_R2B(1,1) = (cosPsi*cosTheta) - (sinPsi*sinPhi*sinTheta);
R_R2B(1,2) = (sinPsi*cosTheta) + (cosPsi*sinPhi*sinTheta); R_R2B(1,3) = -cosPhi*sinTheta;
% second row
R_R2B(2,1) = -sinPsi*cosPhi;
R_R2B(2,2) = cosPsi*cosPhi;
R_R2B(2,3) = sinPhi;
% third row
R_R2B(3,1) = (cosPsi*sinTheta) + (sinPsi*sinPhi*cosTheta);
R_R2B(3,2) = (sinPsi*sinTheta) - (cosPsi*sinPhi*cosTheta);
R_R2B(3,3) = cosPhi*cosTheta;

function [R_B2C] = calculateTransformationMatrixFromAngles23; Compute the transformation matrix
R_B2C = zeros(3);
% first row
R_B2C(1,1) = cosLambda*cosDelta;
R_B2C(1,2) = sinDelta;
R_B2C(1,3) = -sinLambda*cosDelta;
% second row
R_B2C(2,1) = -cosLambda*sinDelta;
R_B2C(2,2) = cosDelta;
R_B2C(2,3) = sinLambda*sinDelta;
% third row
R_B2C(3,1) = sinLambda;
R_B2C(3,2) = 0.0;
R_B2C(3,3) = cosLambda;
R_R2C = R_B2C*R_R2B;

function [N_3] = findCFrameComponentsofN3(R_R2C)
N_3 = zeros(3,1);
N_3(1) = R_R2C(1,3);
N_3(2) = R_R2C(2,3);
N_3(3) = R_R2C(3,3);
```

Panel 8: Main Script – Function to Calculate the Constraint Matrices

```matlab
function [C_1,C_2] = calculateConstraintMatrices...(rRW,rFW,xS,zS,xD,zD,sinPhi,cosPhi,sinTheta,cosTheta,N_3,R_B2C)
%% Calculate temporary variables for matrices C_1 and C_2
tempScalar1 = rRW + (xS*sinTheta) + (zS*cosTheta);
tempScalar2 = cosPhi*((xS*cosTheta) - (zS*sinTheta));
tempScalar3 = sqrt((N_3(1)^2) + (N_3(3)^2));
tempScalar4 = (rRW*cosTheta) + zS;
tempScalar5 = (rRW*sinTheta) + xS;
tempMatrix1 = [0, zD, 0; -zD, 0, xD; 0, -xD, 0];
tempMatrix2 = [cosTheta, 0, 0; 0, 0, 0; sinTheta, 0, 0];
tempMatrix3 = [0, N_3(3), 0; -N_3(3), 0, N_3(1); 0, -N_3(1), 0];
tempMatrix4 = [(-cosPhi*sinTheta), 0, 0; sinPhi, 1, 0; (cosPhi*cosTheta), 0, 0];
%% Calculate the matrix C_1
tempMatrix = [0, -rRW*cosTheta, 0; tempScalar1, 0, 0; 0, -rRW*sinTheta, 0];
C1A = R_B2C*tempMatrix;
tempMatrix = [0, 0, 0; 0, 0, 0; 0, 0, 1];
C1B = tempMatrix1*(R_B2C*tempMatrix2) + tempMatrix;
tempMatrix = [0, 0, 0; 0, 0, N_3(1); 0, 0, 0];
C1C = tempMatrix3*R_B2C*tempMatrix4 + tempMatrix;
C1C = rFW*C1C/tempScalar3;
C_1 = C1A + C1B + C1C;
%% Calculate the matrix C_2
tempMatrix = [(-sinPhi*tempScalar4), -tempScalar4, 0; tempScalar2, 0, 0; -sinPhi*tempScalar5, -tempScalar5, 0];
C2A = R_B2C*tempMatrix;
C2B = tempMatrix1*R_B2C*tempMatrix4;
tempMatrix = [0, 0, N_3(3); 0, 0, 0; 0, 0, -N_3(1)];
C2C = (tempMatrix3*R_B2C*tempMatrix4) + tempMatrix;
C2C = rFW*C2C/tempScalar3;
c_2 = C2A + C2B + C2C;
```
Panel 9 shows the **next two sections** of the main script. The first of these sections **defines** the bicycle’s **initial state vector** and **displays** the values in the MATLAB command window. The second section defines the parameters necessary to run the **differential equation solver** ODE45. First, it defines the **time grid** (or space) on which the solution will be provided. The time grid has a starting value of “\( t_{\text{Min}} \)”, a final value of “\( t_{\text{Max}} \)”, and an increment of “\( t_{\Delta} \)”.

ODE45 **numerically integrates** the equations of motion to find values of the state vector on this set of selected times, irrespective of the increment used in the numerical integration process. ODE45 may use smaller increments in this process to meet the **selected error criteria**. Next it defines the name of the MATLAB **function** that calculates the **derivative** of the **state vector** at each time step. In this example, the function name is “\( \text{bicycleEOM} \)”. Finally, before making the call to ODE45, it defines options for the numerical integration process using the function “\( \text{odeset} \)”. In this example, the **relative error tolerance** (RelTol) is set to \( 1 \times 10^{-5} \) and the **absolute error tolerance** (AbsTol) is set to \( 1 \times 10^{-7} \). See the MATLAB help feature for a description of these and other available options.

**Panel 9: Main Script – Define Initial State Vector, Set ODE45 Options, and Perform Numerical Integration**

```matlab
%% Define the initial state vector, statey0
statey0 = [phi; thetaR; delta; psi; theta; thetaF;
         phiDot; thetaRDot; deltaDot; psiDot; thetaDot; thetaFDot;
         xP; yP; zP];
disp('Initial state vector, statey0')
disp('-----------------------------------')
disp(statey0)
```

```matlab
%% Integration loop using ode45
TMin = 0.0; % starting time (sec)
TMax = 10.0; % stopping time (sec)
TDelta = 0.025; % time increment for output vector
TSpace = TMin:TDelta:TMax; % solution space

functionName = @bicycleEOM;
options = odeset('RelTol',1e-5,'AbsTol',1e-7);
[t,y] = ode45(functionName,TSpace,statey0,options);
```

The call to ODE45 fills vector “\( t \)” and matrix “\( y \)”. In this case, “\( t \)” is a \( 401 \times 1 \) vector of **time values** (the same values as in the vector “\( tSpace \)”), and “\( y \)” is a \( 401 \times 15 \) matrix of **state values**. Each **column** of “\( y \)” is the **time history** of one of the state variables. The first column is \( \phi(t) \), the second is \( \theta_R(t) \), the third is \( \delta(t) \), etc.

Panel 10 shows a sample of the details required to **plot** the **time histories** of the **single elements** of the state vector. Panel 11 shows a sample of the details required to **plot** the **time histories** of the **multiple elements** of the state vector on the same graph. Note that angles are converted from radians to degrees before plotting. Also, note that the “colon” is used in MATLAB to refer to the **entire range** of a subscript. In this case, the “colon” is used to refer to an entire column of the “\( y \)” matrix.
Panel 10: Main Script – Plot Time Histories of State Variables

```matlab
Panel 10: Main Script – Plot Time Histories of State Variables

% Plot the yaw, roll, and pitch angles of the rear frame
figure(1); clf; temp = rad2Deg*y(:,4); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Yaw angle (deg)');
title('Bicycle Yaw Angle (psi) vs. Time');
figure(2); clf; temp = rad2Deg*y(:,1); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Roll angle (deg)');
title('Bicycle Roll Angle (phi) vs. Time');
figure(3); clf; temp = rad2Deg*y(:,5); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Pitch angle (deg)');
title('Bicycle Pitch Angle (theta) vs. Time');

% Plot the steering angle
figure(4); clf; temp = rad2Deg*y(:,3); plot(t,temp); grid on;
xlabel('Time (s)'); ylabel('Steering angle (deg)');
title('Bicycle Steering Angle (delta) vs. Time');

% Plot the yaw rate, roll rate, and pitch rate of the rear frame
figure(5); clf; plot(t,y(:,10)); grid on;
xlabel('Time (s)'); ylabel('Yaw rate (rad/s)');
title('Bicycle Yaw Rate (psidot) vs. Time');
figure(6); clf; plot(t,y(:,7)); grid on;
xlabel('Time (s)'); ylabel('Roll rate (rad/s)');
title('Bicycle Roll Rate (phidot) vs. Time');
figure(7); clf; plot(t,y(:,9)); grid on;
xlabel('Time (s)'); ylabel('Pitch rate (rad/s)');
title('Bicycle Pitch Rate (theta dot) vs. Time');

%% Combined Plots
figure(17); clf; yawDeg = rad2Deg*y(:,4); rollDeg = rad2Deg*y(:,1);
steerDeg = rad2Deg*y(:,3); plot(t,yawDeg,'r',t,rollDeg,'g',t,steerDeg,'b'); grid on;
xlabel('Time (s)'); ylabel('Angle (deg)');
legend('Yaw Angle', 'Roll Angle', 'Steering Angle', 'Location', 'east')
title('Bicycle Angles vs. Time');
figure(18); clf; yawRate = y(:,10); rollRate = y(:,7); steerRate = y(:,9);
plot(t,yawRate,'r',t,rollRate,'g',t,steerRate,'b'); grid on;
xlabel('Time (s)'); ylabel('Angle Rate (r/s)');
legend('Yaw Angle Rate', 'Roll Angle Rate', 'Steering Angle Rate', 'Location', 'northeast')
title('Bicycle Angle Rates vs. Time');
figure(19); clf; rearWheelRate = y(:,8); frontWheelRate = y(:,12);
plot(t,rearWheelRate,'r',t,frontWheelRate,'g'); grid on;
xlabel('Time (s)'); ylabel('Angle Rate (r/s)');
legend('Rear Wheel Angle Rate', 'Front Wheel Angle Rate', 'Location', 'east')
title('Bicycle Wheel Angle Rates vs. Time');
```

Panel 11: Main Script – Plot Time Histories of Multiple State Variables on the Same Graph

```matlab
Panel 11: Main Script – Plot Time Histories of Multiple State Variables on the Same Graph

%% Combined Plots
figure(17); clf; yawDeg = rad2Deg*y(:,4); rollDeg = rad2Deg*y(:,1);
steerDeg = rad2Deg*y(:,3); plot(t,yawDeg,'r',t,rollDeg,'g',t,steerDeg,'b'); grid on;
xlabel('Time (s)'); ylabel('Angle (deg)');
legend('Yaw Angle', 'Roll Angle', 'Steering Angle', 'Location', 'east')
title('Bicycle Angles vs. Time');
figure(18); clf; yawRate = y(:,10); rollRate = y(:,7); steerRate = y(:,9);
plot(t,yawRate,'r',t,rollRate,'g',t,steerRate,'b'); grid on;
xlabel('Time (s)'); ylabel('Angle Rate (r/s)');
legend('Yaw Angle Rate', 'Roll Angle Rate', 'Steering Angle Rate', 'Location', 'northeast')
title('Bicycle Angle Rates vs. Time');
figure(19); clf; rearWheelRate = y(:,8); frontWheelRate = y(:,12);
plot(t,rearWheelRate,'r',t,frontWheelRate,'g'); grid on;
xlabel('Time (s)'); ylabel('Angle Rate (r/s)');
legend('Rear Wheel Angle Rate', 'Front Wheel Angle Rate', 'Location', 'east')
title('Bicycle Wheel Angle Rates vs. Time');
```

Function: `bicycleEOM`

Details of the function “`bicycleEOM`” that calculates the derivative of the state vector are presented in the panels that follow. This function calls a series of other functions to complete the calculations.

Panel 1 shows the function statement and its first two sections. In the first section, global variables are defined. In the second section, values of the elements of the state vector are unpacked, the independent and dependent speed vectors are defined, and trigonometric data is calculated, all for later use. For free motion of the bicycle, the derivative of the state vector is a function of the state vector itself, but not explicitly time. Hence, the state vector is the only required argument of the function.

Panels 2(a) and 2(b) show all the function calls required to build the equations of motion. The first three function calls to calculate the transformation matrices `[R_{RB}]`, `[R_{B2C}]`, and `[R_{R2C}]`, find the C-frame
components of the unit vector $N_3$, and calculate the constraint matrices $C_1$ and $C_2$ are the same as those required to find the initial values of the state vector discussed earlier. The next four function calls calculate the partial angular velocity matrices and angular velocity vectors of the bodies, and the partial velocity matrices and the velocity vectors of the mass centers of the bodies. The eighth section calculates the generalized forces associated with the weight forces of the bodies.

The next four sections calculate the time derivatives of the following – the $C$-frame components of $N_3$, the constraint matrices $C_1$, $C_2$, and $J$, the transformation matrix $R_{B2C}$, the partial angular velocity matrices of the bodies $[WA]$, $[WB]$, $[WC]$, and $[WD]$, and the partial velocity matrices of the mass centers of the body $[VA]$, $[VB]$, $[VC]$, and $[VD]$. The last two function calls calculate the generalized mass matrix and the right-side vector of the differential equations of motion.

Panel 1: “bicycleEOM” Function – Global Variables, State Vector Unload, Trigonometric Values

```matlab
function dydt = bicycleEOM(~, y)
    global rRW rFW lambda xB zB xS zS xD zD
    global sinLambda cosLambda
    global mA mB mC mD aG
    global inGA inGB inG
    global C_1 C_2

    %% Define the variables from the state vector, y
    phi = y(1); delta = y(3); thetaR = y(2);
    psi = y(4); theta = y(5); thetaF = y(6);
    phiDot = y(7); thetaRDot = y(8); deltaDot = y(9);
    psiDot = y(10); thetaDot = y(11); thetaFDot = y(12);
    xP = y(13); yP = y(14); zP = y(15);
    independent and dependent speed vectors
    u_I = [phiDot; thetaRDot; deltaDot]; u_D = [psiDot; thetaDot; thetaFDot];
    trigonometric data
    sinPsi = sin(psi); cosPsi = cos(psi);
    sinPhi = sin(phi); cosPhi = cos(phi);
    sinTheta = sin(theta); cosTheta = cos(theta);
    sinDelta = sin(delta); cosDelta = cos(delta);
    sinLambdaPTheta = sin(lambda + theta); cosLambdaPTheta = cos(lambda + theta);

    %% Calculate the bicycle transformation matrices
        sinPsi, cosPsi, sinPhi, cosPhi, sinTheta, cosTheta, sinLambda, cosLambda, sinDelta, cosDelta);

    %% Define the components of N_3 in the C frame
    [N_3] = findCFrameComponentsofN3(R_R2C);

    %% Calculate the constraint matrices C_1, C_2, and J
    [C_1, C_2] = calculateConstraintMatrices(rRW, rFW, xS, zS, xD, zD, sinPhi, cosPhi, sinTheta, cosTheta, sinLambda, cosLambda, sinDelta, cosDelta, N_3, R_B2C);
    J = -inv(C_2)*C_1;
```

Panel 2(a): “bicycleEOM” Function – Function Calls
Panel 2(b): “bicycleEOM” Function – Function Calls (continued)

Panel 3 shows the *last two sections* of the function. The first of these *calculates* the *time derivatives* of the $R$-frame components of $r_P$, the *position vector* of the *rear wheel contact point* $P$. As point $P$ is to remain *in contact* with the surface at all times, the $N_3$ component is expected to be *approximately zero*, and the $N_1$ and $N_2$ components map out the *path* of $P$ along the surface. The *last section* of the function *calculates* the *derivatives of the independent generalized speeds* from the generalized mass matrix “$g\text{MassMatrix}$” and the right-side vector “$f\text{RHS}$”. The *derivatives* of the *dependent generalized speeds* are *calculated* from the independent speeds, their time derivatives, and the constraint matrix $[J]$ and its time derivative. Finally, the independent speeds, the dependent speeds, the derivatives of the independent speeds, the derivatives of the
dependent speeds, and the derivatives of the \( R \)-frame components of \( r_P \) are **concatenated** to form the derivative of the state vector.

### Panel 3 – Calculate Derivative of the State Vector

```matlab
%% Calculate the derivatives of the coordinates of the point P
tempVector = rRW*[-sinTheta; 0; cosTheta];
xPDot = (R_R2B')*(vGA + (omegaB*tempVector));

%% Calculate the derivatives of the independent and dependent speeds
u_IDot = gMassMatrix\fRHS;  u_DDdot = (J*u_IDot) + (JDot*u_I);

dydt = [u_I; u_D; u_IDot; u_DDdot; xPDot];
```

### Simulation Results

#### Initial Conditions

To compare with previously published results of free motion of the bicycle, the initial conditions are specified as in the following tables. The rear wheel is assumed to contact the surface at the origin, so the initial coordinates of \( P \) are \((0,0,0)\).

<table>
<thead>
<tr>
<th>Angle</th>
<th>( \phi )</th>
<th>( \theta_R )</th>
<th>( \delta )</th>
<th>( \psi )</th>
<th>( \theta )</th>
<th>( \theta_F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value (deg)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Angle Rate</th>
<th>( \dot{\phi} )</th>
<th>( \dot{\theta_R} )</th>
<th>( \dot{\delta} )</th>
<th>( \dot{\psi} )</th>
<th>( \dot{\theta} )</th>
<th>( \dot{\theta_F} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value (r/s)</td>
<td>0.5</td>
<td>-15.3333333</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-13.1428571</td>
</tr>
</tbody>
</table>

The initial pitch angle \( \theta \) is calculated from the initial rear-frame roll angle, the initial steering angle, and the bicycle’s geometric parameters. Given zero roll and steering angles, it is easy to show that the initial pitch angle is also zero. The angle rates \( \dot{\phi} \), \( \dot{\theta_R} \), and \( \dot{\delta} \) are independently specified, and the rates \( \dot{\psi} \), \( \dot{\theta} \), and \( \dot{\theta_F} \) are calculated from the independent rates using the constraint equations. The rate \( \dot{\theta_R} = -15.3333333 \) (r/s) corresponds to a **forward bicycle speed** of 4.6 (m/s).

#### Results

Physically, the bicycle is initially **upright** and **moving forward** at 4.6 (m/s) when its motion is **perturbed** by a **roll rate** of 0.5 (r/s). The simulation results show how will the bicycle **responds** with no controlling input from the rider. Note (from the results) that the motion is **stable** at this forward speed.

The first set of plots show time(histories of the angles \( \psi \), \( \phi \), and \( \delta \) and their **time derivatives**. The angles all exhibit **under-damped** responses and nearly reach **steady values** over a period of 10 seconds. The **yaw angle** is settling to a **non-zero steady-state value**, whereas the **roll** and **steering angles** are settling to **zero**. So,
the bicycle returns to its upright position traveling in a slightly different direction, approximately 13.6 degrees to the right of its original direction. Note that all angle rates are settling to zero.

![Bicycle Angles vs. Time](image1)

![Bicycle Angle Rates vs. Time](image2)

The second set of plots show the path of the rear wheel contact point \( P \) and the angular rates of the front and rear wheels. The plot on the left shows the true shape of the path of \( P \) and verifies the bicycle acquires straight-line motion roughly 14 degrees to the right of its original path. The plot on the right shows the angular rates of the both wheels also exhibit under-damped behavior that settles to a new steady-state value. The rear wheel starts at a rate of \(-15.3333333 \text{ (r/s)}\) and settles to a final value of about \(-15.41 \text{ (r/s)}\), and the front wheel starts at a rate of \(-13.1428571 \text{ (r/s)}\) and settles to a final value of about \(-13.21 \text{ (r/s)}\). This indicates the bicycle speeds up from 4.6 (m/s) to about 4.623 (m/s).

![Global Path of Rear Wheel Contact Point P](image3)

![Bicycle Wheel Angles vs. Time](image4)

As modeled, there are no mechanisms for energy loss, so the motion of the system must conserve energy. In this case, the energy of the initial roll rate is converted to a higher forward speed.
The last plot (to the right) shows the pitch angle of the rear frame. After 10 (s), the bicycle is nearly in an upright position, so the roll, pitch, and steering angles are all nearly zero. In intermediate positions, the pitch angle attains small values for non-zero roll and steering angles.

Validation

The results presented above are identical to those presented in the bicycle references listed below.

Model 2 – MATLAB Script Using Simulink to Integrate the Equations of Motion

Model 2 uses a MATLAB script very much like the script described above for Model 1. There are only minor differences between the two scripts. First, Model 2 calls the function “bicycleEOM” from its Simulink model, and not directly through MATLAB script. Hence, the Model 2 script requires global variables to support only function “calculateLoopClosureEquation”. The resulting global statements for the Model 2 script are shown here.

```
%% Define the geometric parameters of the bicycle
[rRW,rFW,w,c,lambda,xB,zB,xS,zS,xC,zC,xD,zD] = defineBicycleGeometricParameters(deg2Rad);
sinLambda = sin(lambda); cosLambda = cos(lambda);
geoParameters = [rRW,rFW,lambda,xB,zB,xS,zS,xC,zC,xD,zD];

%% Find the initial pitch angle, theta
fun = @calculateLoopClosureEquation; theta0 = [-pi/10 pi/10];
theta = fzero(fun,theta0); sinTheta = sin(theta); cosTheta = cos(theta);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);

%% Define the mass and inertia properties of the bicycle
[mA,mB,mC,mD,inGA,inGB,inGC,inGD] = defineBicycleMassInertiaProperties(sinLambda,cosLambda);
massParameters = [mA,mB,mC,mD];
inertiaParameters = [inGA,inGB,inGC,inGD];
```

Also, to provide the bicycle’s geometric, mass, and inertia data to the Simulink model, the Model 2 script defines the 1x11 array “geoParameters”, the 1x4 array “massParameters”, and the 3x12 array “inertiaParameters”. These additions are highlighted in red in Panel 1 below.

Panel 1: Load the Geometric Parameters, Masses, and Inertia Matrices into the Workspace

```
%% Load the geometric parameters into the workspace
global rRW rFW lambda xS zS xD zD
global sinPhi cosPhi sinDelta cosDelta

%% Load the geometric parameters of the bicycle
[rRW,rFW,w,c,lambda,xB,zB,xS,zS,xC,zC,xD,zD] = defineBicycleGeometricParameters(deg2Rad);
sinLambda = sin(lambda); cosLambda = cos(lambda);
geoParameters = [rRW,rFW,lambda,xB,zB,xS,zS,xC,zC,xD,zD];

%% Load the initial pitch angle
fun = @calculateLoopClosureEquation; theta0 = [-pi/10 pi/10];
theta = fzero(fun,theta0); sinTheta = sin(theta); cosTheta = cos(theta);
sinLambdaPTheta = sin(lambda+theta); cosLambdaPTheta = cos(lambda+theta);

%% Load the mass and inertia properties of the bicycle
[mA,mB,mC,mD,inGA,inGB,inGC,inGD] = defineBicycleMassInertiaProperties(sinLambda,cosLambda);
massParameters = [mA,mB,mC,mD];
inertiaParameters = [inGA,inGB,inGC,inGD];
```

Finally, after the starting time, stopping time, and time increment are defined, the Model 2 script executes the Simulink model “BicycleEOMSimulink01”. Note that a time vector to define the solution space is not required.

```
%% Integration loop using Simulink
tMin = 0.0; % starting time (sec)
tMax = 10.0; % stopping time (sec)
tDelta = 0.025; % time increment for output vector
sim('BicycleEOMSimulink01')
```
Simulink Model: “BicycleEOMSimulink01”

The top layer of the Simulink model is shown in the figure to the right. The model structure is very simple. The lower loop integrates the fifteen bicycle equations of motion. The initial state vector is used to calculate its derivative, and this derivative is used to estimate the state vector at the next instant of time. This integration process continues from the initial time to the final time specified by the MATLAB script. Each new state vector is also sent to the measurement subsystem to save the results to the MATLAB workspace.

The figure below on the left shows the model’s configuration parameters. Specifically, it shows that the variables “tMin” and “tMax” provide values for the simulation start time and stop time, respectively. It also shows that the numerical integration is performed using a fixed-step, Dormand-Prince, eighth-order method with a fixed step size provided by the variable “tDelta”. The figure on the right shows the dialog box for the integrator block. The “Initial condition” section of the box indicates that the initial conditions of the fifteen state variables are provided by the 1×15 vector (or, array) “statey0”.

Note under the “Solver options” section of the model’s configuration parameters dialog box that the solver “Type” and “Solver” are chosen using drop-down boxes. The solver type can be fixed-step or variable-step, and there are many solvers from which to choose. This is one of the advantages provided by Simulink.
The figures below show details of the two subsystems. The figure on the left shows details of the subsystem that calculates \( \frac{dy}{dt} \) the derivative of the state vector. The input signals to this process are the current state vector, the geometric parameters, masses, and inertia matrices of the bicycle’s components, and the acceleration of gravity. The derivative of the state vector is calculated in a single MATLAB function “bicycleEOM”. The figure on the right shows the measurement subsystem that saves the current values of the state vector components to workspace arrays. The names of the workspace arrays are specified in the dialog boxes of the plotting scopes.

The initial lines of the “bicycleEOM” function are shown in Panel 2 below. Note that elements of the state vector, geometric parameters, masses, inertia matrices, independent and dependent generalized speed vectors, and some trigonometric function values are loaded into the necessary variables for the calculations that follow. The rest of this function performs the same calculations as those described for the function of the same name in Model 1. The only difference is that the “bicycleEOM” function of Model 2 does make any additional function calls – all calculations are done explicitly in this function.

To illustrate this point, the calculations done in the “bicycleEOM” function of Model 2 associated with the functions “calculateBicycleTransformationMatrices” and “findCFrameComponentsofN3” are shown in Panel 2. Note the calculations are not done by calling the two functions, but rather by doing the calculations explicitly within “bicycleEOM”. The rest of the contents of the function are not shown here.
As the Simulink model executes, state vector data is saved to arrays in the MATLAB workspace. The first column of each array contains the time values, and the second column contains the values of the state variable. Panel 3 below shows the plot commands associated with the first three figures. Note that the angles are converted from radians to degrees before plotting.
Panel 3: Plot Commands for the First Three Figures

```matlab
%% Plot the results

% Plot the yaw, roll, and pitch angles of the rear frame
figure(1); clf;
plot(psi(:,1),rad2Deg*psi(:,2)); grid on;
xlabel('Time (s)'); ylabel('Yaw angle (deg)');
title('Bicycle Yaw Angle (psi) vs. Time');

figure(2); clf;
plot(phi(:,1),rad2Deg*phi(:,2)); grid on;
xlabel('Time (s)'); ylabel('Roll angle (deg)');
title('Bicycle Roll Angle (phi) vs. Time');

figure(3); clf;
plot(theta(:,1),rad2Deg*theta(:,2)); grid on;
xlabel('Time (s)'); ylabel('Pitch angle (deg)');
title('Bicycle Pitch Angle (theta) vs. Time');
```

Simulation Results:

The simulation results for this model are identical to those presented above for Model 1.

Model 3 – MATLAB Script Using Simulink to Integrate the Equations of Motion

Model 3 is identical to Model 2, except in the way it calculates “dy/dt”. The single function of Model 2 is replaced by a detailed Simulink model with multiple function calls. This makes the structure of the algorithm required to compute “dy/dt” more explicit and observable within the model.

The MATLAB script and the top layer of the Simulink portion of Model 3 are identical to those of Model 2. The differences are in the subsystem required to calculate “dy/dt”. Panel 1 below shows the details of this subsystem. In this model, the work of the subsystem is divided into ten additional subsystems, labeled one through ten in the diagram. Details of each of these subsystems are given below.

Panel 1: Model 3 Subsystem to Calculate the State Vector Derivative
The contents of the **first two subsystems** are shown below in Panel 2. The first subsystem (diagram on the left) **loads variables** from the current state vector, then it defines the $3 \times 1$ vectors of **independent** and **dependent generalized speeds**. Note that some elements of the state vector are **not needed** to compute the equations of motion, so their signals are **terminated**. The second subsystem **calculates** and **stores** the $3 \times 3$ bicycle transformation matrices and the **C-frame components** of the unit vector $N_3$. Details of the MATLAB function are not shown as these calculations have been discussed earlier.

**Panel 2: Subsystems 1 and 2**

![Diagram showing Load the Variables of the Current State](image)

The contents of the **third through tenth subsystems** are shown below in Panels 3 and 4. These subsystems calculate the **constraint matrices**, **partial angular velocity matrices**, **mass-center partial velocity matrices**, **angular velocity and mass center velocity vectors**, **derivatives of the constraint matrices**, **derivatives of the partial angular velocity matrices**, and **derivatives of the mass-center partial velocity matrices**. The tenth subsystem **completes** the **calculation** of the $15 \times 1$ vector “$\frac{dy}{dt}$”, the derivative of the current state vector. Again, details of the MATLAB functions are not shown as they were previously discussed.

**Simulation Results:**

The simulation results for this model are **identical** to those presented above for Model 1.
Panel 3: Subsystems 3 through 8
Panel 4: Subsystems 9 and 10
Exercises:

7.1 A three-body mechanical system is shown in the diagram. Disk $D$ is connected to the ground with a revolute joint allowing rotation only in the fixed $\varepsilon_3$ direction. Bar $B_1$ is connected to the disk and bar $B_2$ is connected to bar $B_1$ with revolute joints allowing relative rotations only in the $\varepsilon_1 = \varepsilon_2 \times \varepsilon_3$ direction. The disk rotates at a constant rate of $\Omega = 2\pi$ (rad/s), while the motions of the bars are free. Torsional springs and dampers restrain the motion of the bar. Develop a SimMechanics model of the system using the following data.

- Disk mass: $m = 0.2$ (slug)
- Disk physical dimensions: $R = 1$ (ft), $b = 0.5$ (ft), $d = 0.2$ (ft)
- Disk inertia: disk is assumed to be a thin circular disk
- Bars’ masses: $m = 0.1$ (slug)
- Bars’ physical dimensions: lengths $\ell = 2$ (ft) and radii $r = 0.1$ (ft)
- Bars’ inertias: bars are assumed to have cylindrical shape with the given length and radius
- Bar $B_1$ spring and dampers: $k_1 = 50$ (ft-lb/rad) and $c_1 = 4$ (ft-lb-s/rad)
- Bar $B_2$ spring and dampers: $k_2 = 5$ (ft-lb/rad) and $c_2 = 0.4$ (ft-lb-s/rad)

The bars initially both hang vertically downward, that is $\theta_1 = \theta_2 = 0$. Using your model identify the final values of the angles of the two bars. Without a set of explicit equations of motion, how can you validate the model (at least qualitatively)?

7.2 The system shown is a three-dimensional double pendulum or arm. The first link is connected to ground and the second link is connected to the first with ball and socket joints at $O$ and $A$. The orientation of each link is defined relative to the ground using a 3-1-3 body-fixed rotation sequence. The links are identical with mass $m$ and length $\ell$. The links are assumed to be slender bars with circular cross sections (diameter, $d$) and mass centers at their midpoints. Using the physical data listed below and the equations of motion developed in Unit 5 of this volume, develop a MATLAB script to simulate the motion of the system under the action of gravity. Use the MATLAB function ODE45.

Physical data:

$\ell = 0.5$ (m), $m = 2.8$ (kg), $d = 3$ (cm)
7.3 Model the system of Exercise 7.2 using a MATLAB script and a Simulink model. The MATLAB script should initialize all necessary variables to integrate the equations of motion using Simulink. Compare the results with the model of Exercise 7.2.

7.4 Model the system of Exercise 7.2 using a MATLAB script and a SimMechanics model. The MATLAB script should initialize all necessary variables to support the SimMechanics model. Compare the results with the previous two models.

References:
12. Bicycle references: