California State Polytechnic University
San Luis Obispo

Handbook and Reference for the

ME326 Activity

Spring 2022

Charlie Refvem
Department of Mechanical Engineering
Cal Poly San Luis Obispo

Last Revised: March 21, 2022

Special thanks to Tristan Perry, Toan Le, Zachary Richter, Dominic Riccoboni, and Dr. William R. Murray for helping with the formulation of this handout.

Contents

This page is intentionally left blank.

Exercise 1

Introduction to Numerical Analysis

1.0 Prelab

It is suggested you read the following appendices to help you with this lab.

1. Read Appendix A regarding state determined systems.
2. Read Appendix D regarding report formatting.

There is no pre-lab due at the beginning of this activity period. From now on however, there will be. Pre-labs are to be turned in at the beginning of each activity period and will not be accepted late.

1.1 Objectives

The objectives of this exercise are to:

1. Survey the solution methods that will be used in this course.
2. Solve a simple linear ODE using multiple methods.
3. Consider the differences between numerical and analytical solutions to differential equations.

1.2 Background

A differential equation is any equation involving derivatives of one or more functions. Many different forms of differential equations exist, but only a limited selection of them will be considered in this course: ordinary differential equations, henceforth referred to as ODEs. Ordinary differential equations are differential equations that only include derivatives with respect to one variable, such as time. Many methods exist for solving ODEs, both analytically and numerically. This first assignment intends to compare and contrast several solution methods for a simple ODE.

High-order ODEs

Mechanical engineers often deal with systems of coupled second order ODEs resulting from $F = ma$, since acceleration is the second derivative of position. The order of an ordinary differential equation, N, is the number of differential operators in the equation, often equal to the highest order of differentiation in the equation.

Linear Time Invariant ODEs

A linear ODE is a differential equation that can be represented as a polynomial with respect to an unknown parameter and its derivatives. One specific type of linear ODE is an equation with constant coefficients of the following form:

$$a_0 x(t) + a_1 \frac{dx}{dt}(t) + a_2 \frac{d^2x}{dt^2}(t) + \cdots + a_{(N-1)} \frac{d^{(N-1)}x}{dt^{(N-1)}}(t) + a_N \frac{d^N x}{dt^N}(t) = f(t) \tag{1.1}$$

The solutions to ODEs of this nature can be computed analytically using many techniques such as the method of undetermined coefficients or the Laplace transform. Additionally, the equations can be solved numerically to find a very accurate approximation of the solution without needing a closed form solution. For most numerical methods to work properly, it is usually necessary to split up a high-order ODE into many first-order ODEs.

Systems of First Order ODEs

Any N$^{\text{th}}$ order ordinary differential equation[1] can be split into a set of N first order equations in the form shown below.

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ \vdots \\ x_n \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} f_1\left(t, x_1, \ldots, x_N\right) \\ \vdots \\ f_n\left(t, x_1, \ldots, x_N\right) \\ \vdots \\ f_N\left(t, x_1, \ldots, x_N\right) \end{bmatrix} \tag{1.2}$$

Where $x_1 \ldots x_N$ are the N state variables for the system, as described in Appendix C. The n$^{\text{th}}$ row of this vector equation describes the first order ODE corresponding to the n$^{\text{th}}$ state variable. This can be described more succinctly as a vector equation as shown below.

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}\left(t, \boldsymbol{x}\right) \tag{1.3}$$

This is sometimes written in an equivalent form, which better illustrates that the input to the system is a function of time, $\boldsymbol{u}(t)$.

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}\left(\boldsymbol{x}, \boldsymbol{u}(t)\right) \tag{1.4}$$

Notice that in Equation 1.4, the vector $\boldsymbol{x}$ is in bold typeface[2] and represents the set of N state variables $\{x_1 \ldots x_N\}$, and the vector $\boldsymbol{u}$ represents the set of M time-dependent inputs to the system. Equation 1.2 and the equivalent forms, Equation 1.3 and Equation 1.4, are all different representations of the same system of first order ODEs.

[1] For this assignment we are only concerned with linear time-invariant systems, but even nonlinear time-varying ODEs can be split up into a set of first order ODEs.

[2] In this document, vectors will always be denoted using a bold typeface.

1.3 Assignment

Solve a linear time-invariant ordinary differential equation using two different numerical methods. The numerical methods that you will be investigating this week are:

1. Using an analytical method.
2. Using a function file and a solver in MATLAB®.
3. Using a block diagram in Simulink®.

Differential Equation

The ODE you will be solving is indicated below in Equation 1.6. This second order equation could represent many different systems. For the sake of brevity, the physical meaning of the equation will be left to the imagination of the student.

$$2\frac{d^2q}{dt^2} + 10\frac{dq}{dt} + 12q = e^{-3t} \tag{1.5}$$

This ODE can be expressed using the more familiar "dot" notation because the derivatives are with respect to time.

$$2\ddot{q} + 10\dot{q} + 12q = e^{-3t} \tag{1.6}$$

This ODE is to be solved with the following initial conditions. At $t = 0$:

$$q(0) = 2 \tag{1.7}$$
$$\dot{q}(0) = 1 \tag{1.8}$$

Using an analytical method

Solve the second-order ODE presented above for both $q(t)$ and $\dot{q}(t)$ using the analytical method of your choice on paper; the analytical solution will be part of your hand-calculations for this assignment. During your analysis, work in terms of symbolic variables only. That is, do not plug in numbers during your analysis; instead, use MATLAB® to evaluate any coefficients you might need when it comes time to plot the analytical solution against the numerical solutions.

Using a solver in MATLAB®

Convert the second order equation to a system of first order equations using the state vector $\boldsymbol{x} = \begin{bmatrix} q & \dot{q} \end{bmatrix}^\top$. Represent this system of equations as a single vector equation in the form $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x})$. To do this, first construct an equation representing the time-derivative of each state variable; that is, express the time-derivative of each variable in the state vector as a function of only the state variables and time.

$$\frac{d}{dt}(q) = f_1\left(t, q, \dot{q}\right) \tag{1.9}$$

$$\frac{d}{dt}(\dot{q}) = f_2\left(t, q, \dot{q}\right) \tag{1.10}$$

Then, place these equations together into a single vector equation.

$$\frac{d}{dt}\begin{bmatrix} q \\ \dot{q} \end{bmatrix} = \begin{bmatrix} f_1\left(t, q, \dot{q}\right) \\ f_2\left(t, q, \dot{q}\right) \end{bmatrix} \tag{1.11}$$

In MATLAB®,

1. Write a function file, `odefun.m` that takes in the state vector, `x` and and time, `t` and returns the derivative of the state vector, `xdot`. The following lines of code may be used as a template for the function file.

```
function [ xdot ] = odefun( t, x )
    xdot = [ 0
             0 ];    % replace these zeros with expressions for xdot
end
```

   Pay attention to the order of the inputs to your function. Time, `t`, must be the first input followed by the state vector, `x` as the second argument. One of the challenges in this step is indexing the variable `x` properly to represent the values of q and $\dot{q}$.

2. Now, back in your main script, define the initial conditions for the simulation using a variable `x_0` which contains the value of the state vector at time $t = 0$.

3. Define the time span for the simulation as a variable `tspan`. This should be a [1x2] vector containing the start time and stop time for the simulation.

4. Solve the ode over the course of 10 seconds using the built-in solver `ode45`[3].

```
[tout, xout] = ode45(@odefun, tspan, x_0)
```

[3] Numerical solvers will be discussed in further detail in upcoming assignments. `ode45` is a good general purpose solver that is provided by MATLAB®.

Using a block diagram in Simulink®

Construct a block diagram that can be used to simulate the response of the system to the specified input.

In MATLAB®,

1. Define the initial conditions for the simulation using a variable `x_0` which contains the value of the state vector at time $t = 0$.

In Simulink®,

2. String together blocks representing the differential equation presented above. The diagram will consist of many blocks including a source block, a math function block, several gain blocks, two integrator blocks, a summing junction, and a to-workspace block.

3. Run the model from within Simulink® and observe the output using a "Scope" block.

After confirming the results from within Simulink®, return to MATLAB®,

4. Put a line of code into your script that will allow you to run the Simulink® model from your script. This can be done using the `sim()` function.

1.4 Discussion and Deliverables

Produce a properly formatted report according to the guidelines specified in Appendix D. Be sure to include an image or screen shot of your block diagram in the report.

In addition to your formatted code and screen-captured block diagram, please provide the following plots:

1. Using your results from either MATLAB® or Simulink®, create a single figure with two subplots. The first subplot should show the input to the system, $u(t) = e^{-3t}$, and the second subplot should show the response of the system, $q(t)$, each plotted against time. Change the size or aspect ratio of this figure so that in your report each subplot has reasonable proportions.

   Make sure that the horizontal scale is the same for both subplots so that the reader may compare like points in time between the two plots.

2. Using results from the analytical method and the two numerical methods, plot on a single set of axes the value $\dot{q}(t)$ against time. Change the line style of each numerical result so that they show up as symbols on the plot; be sure to include a legend.

Answer the following discussion questions:

1. What are the pros and cons of each method used? Compare and contrast the visual block diagram approach and the text based function approach.

2. In what circumstances does the numerical approach make more sense than the analytical approach?

This page is intentionally left blank.

Exercise 2

Euler Solver: Simple Pendulum

2.0 Prelab

This pre-lab is due at the beginning of the laboratory period.

1. Read Appendices A through C to understand the fundamentals of numerical solvers.

2. Write a script that generates a vector `t` representing time from 0 to 10 seconds at evenly spaced increments of 0.1 seconds. This can be done using the `linspace` command or by using the `:` operator. Your resulting time vector should be:

$$t = [0.0, 0.1, \ldots, 9.9, 10]$$

3. Add to your script code that will evaluate the following expression on the vector `t`:

$$f(t) = C_1 + C_2 t + C_3 e^{-\lambda_1 t} + C_4 e^{-\lambda_2 t}$$

where: $C_1 = -3.0580$, $C_2 = 2.0000$, $C_3 = 3.3665$, $C_4 = -0.3085$, $\lambda_1 = 0.8520$, and $\lambda_2 = 2.8147$. You should not use a loop for this.

4. Create a plot showing the value of $f(t)$ for the period of time described.

5. Next, create the following vectors and matrices in your script

$$A = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \; B = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} \; C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

6. Using the previously created variables, compute:

$$D = \begin{bmatrix} A - B \\ CB \end{bmatrix}$$

Please output the contents of variable D; this can be done by omitting the semicolon at the end of the line defining `D`.

2.1 Objectives

The objectives of this exercise are to:

1. Solve a nonlinear ODE using a simple Euler solver algorithm.

2.2 Background

Unlike ME212 Dynamics, which involves snapshot analysis focused on providing solutions valid only at a single point in time, ME326 Intermediate Dynamics includes techniques to provide time-dependent solutions valid over a range of time. These solutions will reflect the kinematic and kinetic behaviors of various systems in time and space.

Producing time-dependent solutions requires solving the equations of motion of the system, which are generally expressed as ordinary differential equations (ODEs). Solving these equations analytically has been studied in prerequisite coursework; however, as systems become more complicated, the equations of motion often become coupled nonlinear equations that are impractical or impossible to solve analytically. In practical engineering applications, these equations are normally solved numerically, and the process of determining the solution to such equations in time and space is called numerical simulation.

Consider the simple pendulum shown in Figure 2.1. The pendulum consists of a bob attached to a light rod which pivots freely about a fixed point. The bob is 0.75 $[m]$ from the pivot, has a mass of 0.5 $[kg]$, and can be modeled as a point mass. Unless otherwise specified, assume the pendulum is displaced counter clockwise from the vertical and released from rest.

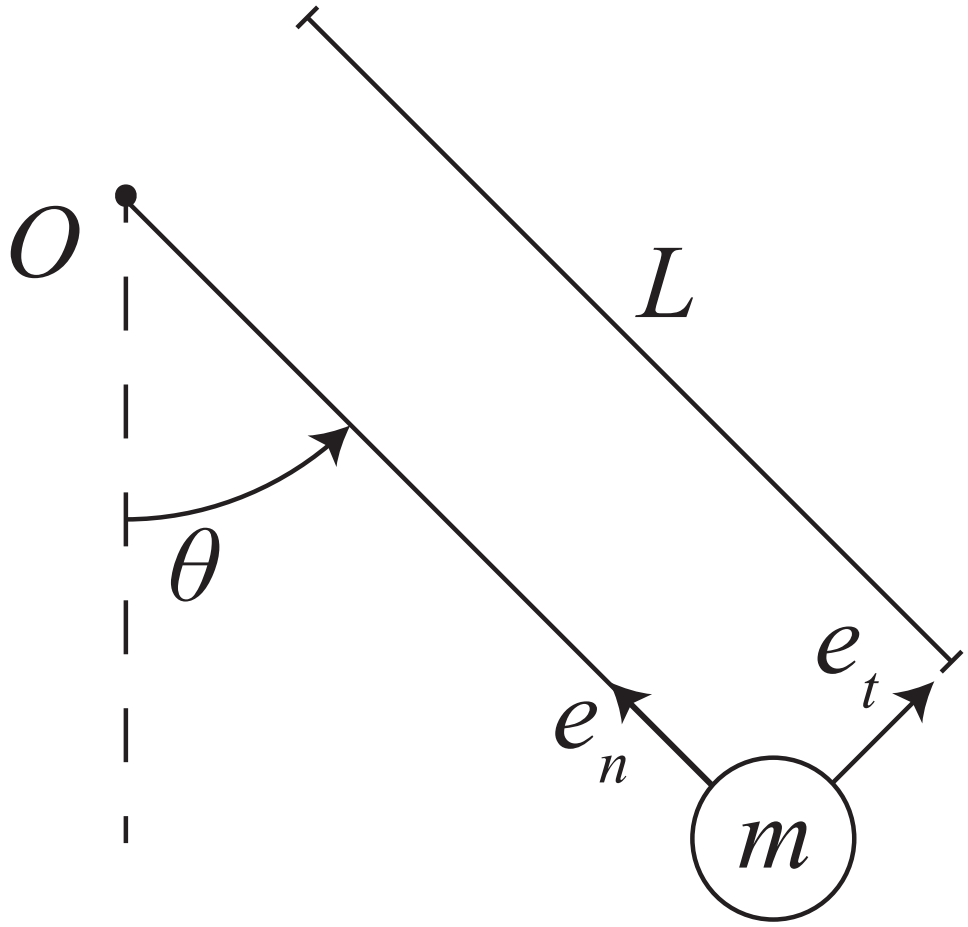


Figure 2.1: Simple Pendulum schematic. Mass 'm' is hung from point O by a rigid, lightweight rod of length 'L'.

2.3 Assignment

(1) Your instructor will derive the equations of motion with you using free-body and kinetic diagrams.You will split the resulting second-order ODE into a set of first-order ODEs; that is, you will put your model in state variable form and write a function file, 'pendulumEOM.m', that contains these split first order ODEs.

(2) Now create a second function file, 'EulerSolver.m' to act as a numerical integrator by utilizing Euler's method. Use the following template to start your Euler solver:

```
%% EulerSolver
% Update this file with your own code and comments!
function [tout, xout] = EulerSolver(func_hand, tf, x0, tstep)
% Step  1: Create a column vector, tout, containing the value of time
%          for each iteration of the solution, starting with 0,
%          counting by tstep, and ending at tf.
% Step  2: Create an array containing zeros, xout, that is the same
%          number of rows as the time vector, tout, and the same number
%          of columns as there are rows in initial state vector, x0.
% Step  3: Replace the first row of xout with the initial
%          conditions, x0, so that each following row can be computed.
% Step  4: Using a for loop, iterate through each row in xout
% Step 4a: Compute the time rate of change of the vector x by
%          executing the function handle.
% Step 4b: Using the current time rate of change, the time step, and
%          the current value of xout, compute the next value of
%          xout.
end
```

Where `func_hand` is a function handle[1] for an arbitrary ODE we are trying to integrate. Later on you will pass in a handle for `pendulumEOM` as the function handle. This will give `EulerSolver` something to integrate. The final time, `tf`, specifies when to end the simulation. All ODEs require initial conditions to solve; these will be passed in as `x_0`, a vector containing the values of the state variables at time zero. Finally, `tstep` determines the step size `EulerSolver` will use when computing the solution to the ODE.

Refer to Appendix C for more details on how to implement Euler's method. The function `EulerSolver` should include creation of a time array and a loop implementing Euler's method to successively integrate the ODE(s) passed in.

[1] A function handle is a variable that contains access to a function. The `@` operator is used to create a function handle in MATLAB®; for example, `pend_handle = @pendulumEOM` will create a function handle for your equations of motion. This function handle can then be passed into `EulerSolver` to be integrated.

(3) Create a new script that will act as the main file for running the simulation. This file should declare the constants used, run the simulation, and generate your plots.

(4) Run your simulation using `EulerSolver` using a function call within your main script. Make sure to use a small enough time step to keep your solution from diverging over a 10 $[s]$ interval[2].

[2]Divergence can be recognized by looking at the amplitude of the periodic results from the simulation. The amplitude should remain constant as the pendulum swings back and forth.

2.4 Discussion and Deliverables

Produce the following plots:

1. Pendulum angular position, θ, versus time for both $\theta_0 = 15°$ and $\theta_0 = 120°$ overlaid on the same axes.
2. Pendulum angular position, θ, and pendulum angular velocity, $\dot{\theta}$, versus time as separate subplots on one figure. Use the results for $\theta_0 = 120°$.
3. Pendulum angular velocity versus angular position. This is sometimes called a "phase portrait" and shows how the state of the system changes with time.

Answer the following discussion questions in your report:

1. Considering plot #1, comment on the differences between the simulation results with $\theta_0 = 15°$ and $\theta_0 = 120°$. Consider differences beyond the change in amplitude. What may be the reason for the difference in shape between the two series on the plot?
2. Considering plot #2, would you say that the results are sinusoidal? Explain your reasoning.
3. Considering plot #3, what conclusion can be made about the energy in the system? How would the "phase portrait" look if there was friction included in the system?

Turn in a published form of your script fully annotated according to Appendix D.

Exercise 3

Massless Bumper

3.0 Prelab

This pre-lab is due at the beginning of the laboratory period.

1. Read the entire assignment thoroughly, and become familiar with the system described in the background section. This problem may have already been assigned as a homework problem.
2. From free-body and kinetic diagrams, determine the differential equation describing $x(t)$ for the time during which the mass is in contact with the bumper plate.
3. From new free-body and kinetic diagrams, determine the differential equation describing $x(t)$ for the time during which the mass is *not* in contact with the bumper plate.
4. Solve the equation from part 2 analytically to find $x(t)$, the displacement of the bumper plate, after the collision occurs but while the mass remains in contact with the bumper plate. Note: during this time, $x(t)$ is also the displacement of the mass. Assume that after the collision, the bumper and mass move together with the same initial velocity of the mass.
5. Use your solution from part 4 to analytically determine the time after collision, t_r, at which the mass releases contact with the bumper plate. The free-body diagram of the bumper plate may be helpful in determining the condition(s) under which the mass loses contact with the bumper plate.
6. Use your solution from parts 3 and 5 to analytically find $x(t)$, the displacement of the bumper plate, after the mass loses contact with the bumper plate.
7. Use MATLAB® to plot the displacement $x(t)$ as a function of time. Indicate on the plot the time at which the mass loses contact with the bumper.

3.1 Objectives

The objectives of this exercise are to:

1. Solve a hybrid system[1].
2. Compare numerical and analytical solutions to an ODE solution.

3.2 Background

Consider Figure 3.1, which depicts a mass M with the velocity $V_0 = 1$ $[^m/_s]$ on its way to collide with a bumper. The bumper consists of plate of negligible mass, a spring, and a damper. Use $M = 1000$ $[kg]$, $K = 9850$ $[^N/_m]$ and $B = 12500$ $[^{N \cdot s}/_m]$.

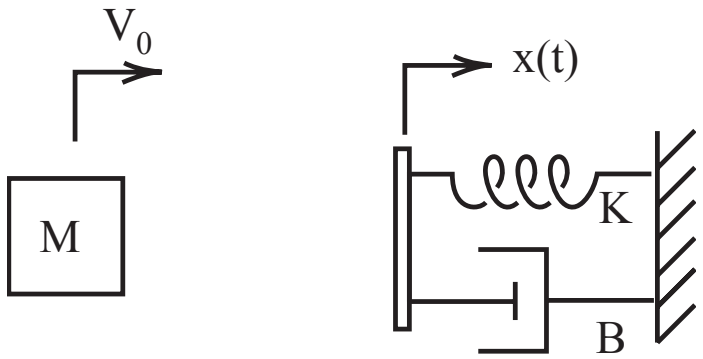


Figure 3.1: Mass, M, traveling toward a bumper, B & K, with a velocity V_0.

[1] A hybrid system is any system that has multiple regimes of motion. The model representing the hybrid system changes when the regime of motion changes.

3.3 Assignment

Solve the same problem as you did in the prelab, but use `ode45` to numerically compute your results.

1. Solve the equation from part 2 of the prelab numerically to find $x(t)$, $v(t)$, and $a(t)$, the displacement, velocity, and acceleration of the bumper plate after the collision occurs. Assume that the mass remains in contact with the bumper plate indefinitely. Do this by simulating the system with `ode45`, utilizing the equation you developed in the prelab step 2.

2. Use your numerical solution to determine the time after collision at which the mass loses contact with the bumper plate. Do this by analyzing the results from the numerical solution and finding the value of time, t_r, where the conditions are met for the mass to release from the bumper.

3. Use the equation from part 3 of the prelab and the value of t_r computed from your `ode45` solution to numerically find $x(t)$, the displacement of the bumper plate, after the mass loses contact with the bumper plate. That is, use the state of the system at t_r as the initial conditions for a second simulation in which the mass is not in contact with the bumper.

4. Combine the results from the two simulations to find the entire solution $x(t)$ including the time when the mass is in contact with the bumper and the time after the mass releases from the bumper.

3.4 Discussion and Deliverables

Produce the following plots:

1. Provide evidence that your numerical solution is in agreement with your analytical solution by overlaying the numerical results on top of the analytically calculated results.
2. A single figure with three subplots showing the position of the bumper, $x(t)$, and its first and second derivatives, $\dot{x}(t)$ and $\ddot{x}(t)$, versus time. Indicate on the plot the time at which the mass loses contact with the bumper.
3. A plot showing the position of the bumper, $x(t)$, for the case when the bumper releases and in the case when the mass permanently attaches to the bumper. Adjust the scale of this plot so that the two lines may be easily distinguished from one another.

Answer the following discussion questions:

1. What is the difference, or error, in value of t_r between the analytical and numerical methods? Discuss the significance of this error in terms of the solution method and how the error could be reduced.
2. Discuss the benefits of the numerical and analytical methods used here. Which method was simpler for this specific problem? In what situations might the other method be easier?
3. Considering plot #2, discuss why the position, velocity, and acceleration of the bumper are or are not continuous with respect to time.

Exercise 4

Quarter-Car Suspension Simulation

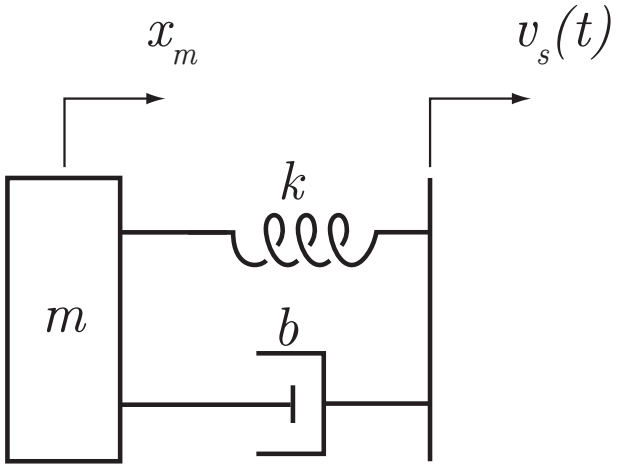
4.0 Prelab

This pre-lab is due at the beginning of the lab period.

1. Draw a free-body diagram (FBD) and a kinetic diagram[1] (KD) for the mass-spring-damper system shown below, which can be described using the following parameters: $k = 10\ {}^{lbf}/_{in}$, $m = 2\ [{}^{lbf \cdot s^2}/_{in}]$, and $b = 1\ [{}^{lbf \cdot s}/_{in}]$.

2. Develop a second-order differential equation that can be solved to find the time-dependent motion of the mass. The ODE can be found by summing forces in the horizontal direction using your FBD and KD.

   Recall that the constitutive relationship for a spring is $F_k = k\,\Delta x$ and the constitutive relationship for a damper is $F_b = b\,\Delta \dot{x}$. That is, the force in a spring is proportional to the change in length of the spring and the force in a damper is proportional to the relative velocity between the two ends of the damper.



[1] Your past instructors may have referred to kinetic diagrams as mass acceleration diagrams (MADs).

4.1 Objectives

The objectives of this exercise are to:

1. Work with higher-order ODEs.
2. Compare the effectiveness and speed of different ODE solvers.
3. Learn about anonymous functions and function handles.

4.2 Background

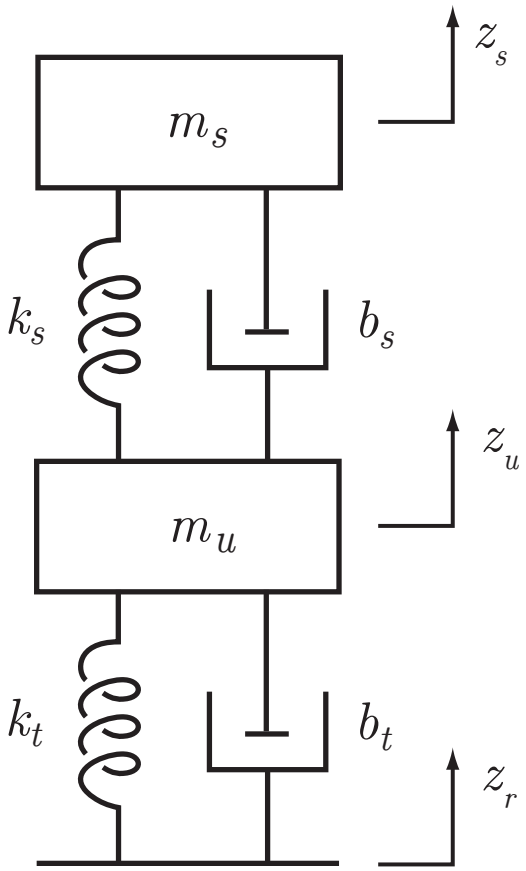


Figure 4.1: Schematic representation of a quarter-car suspension.

Consider the quarter-car suspension model for one wheel on a moving vehicle. A schematic representation of the quarter-car is shown in Figure 4.1. The mass of one quarter of the

vehicle is lumped together as m_s and is called the sprung mass because it is above the suspension. Similarly, all of the mass moving with one wheel is lumped together as m_u and is called the unsprung mass because it is below the suspension. The sprung and unsprung masses are separated by the suspension which is modeled as a spring, k_s, and a damper, b_s in parallel. Finally, the unsprung mass is separated from the road by the tire, modeled as a spring, k_t, and a damper, b_t, in parallel. The parameters associated with these elements are shown in table 4.1.

m_s	200	$[kg]$
m_u	30	$[kg]$
k_s	18000	$[N/m]$
b_s	1300	$[N \cdot s/m]$
k_t	125000	$[N/m]$
b_t	10	$[N \cdot s/m]$

Table 4.1: Lumped parameters for the quarter-car suspension model.

The analysis in this lab will consider only deviations from the static equilibrium position of the quarter-car suspension. The validity of neglecting the effects of gravity can be seen by considering the static conditions for the quarter-car suspension. When there is no motion in the suspension there is a static force in each spring directly opposing the weight of the vehicle. In the following dynamic analysis it is then possible to consider only deviations in spring force from those static spring forces, resulting in completely dynamic analysis.

Three coordinate systems have been overlaid on the schematic in Figure 4.1; the z_s axis represents the vertical deviation from static equilibrium for the sprung mass, m_s, the z_u axis represents the vertical deviation from static equilibrium for the unsprung mass, and the z_r axis represents the vertical deviation in road elevation relative to a datum at static equilibrium.

Once the system of ODEs is formulated for the quarter-car suspension, both the road elevation, z_r, and the rate of change of the road elevation, $\dot{z}_r$, will appear as forcing functions. It is impossible to specify both z_r and $\dot{z}_r$ independently, so an auxiliary ODE will be used to incorporate z_r as a state variable:

$$\frac{d}{dt} z_r = \dot{z}_r \tag{4.1}$$

With this auxiliary ODE merged into the system of ODEs for the quarter-car, the only required input is the rate of change of elevation, $\dot{z}_r$.

A simple road profile is described by the elevation of the road as a function of displacement along the road. That is, $z_r = z_r(x_c)$; where, x_c is a variable representing the horizontal displacement of the vehicle, and $z_r(x_c)$ is a function relating the distance traveled along the road to the elevation at that point on the road. The time rate-of-change of the elevation can be found by applying the chain-rule while differentiating $z_r(x_c)$.

$$\dot{z}_r = \frac{d}{dt} z_r \tag{4.2}$$

$$\dot{z}_r = \frac{d}{dt} \left(z_r \left(x_c \right) \right) \tag{4.3}$$

$$\dot{z}_r = \frac{d}{dx_c} \left(z_r (x_c) \right) \frac{d}{dt} x_c \tag{4.4}$$

$$\dot{z}_r = \frac{dz_r}{dx_c} \dot{x}_c \tag{4.5}$$

The derivative $\frac{dz_r}{dx_c}$ is simply the slope of the road; therefore, the rate of change of the elevation of the road is simply the horizontal speed of the vehicle multiplied by the slope of the road at the location of the vehicle.

4.3 Assignment

Perform the following analysis in preparation for simulation in MATLAB®.

1. Draw a FBD and KD for each mass in the quarter-car suspension model.
2. From your FBDs and KDs, develop a pair of second-order ODEs that will represent the dynamic behavior of the quarter-car suspension.
3. Combine the two second-order ODEs representing the quarter-car suspension model with the first-order ODE in Equation 4.1 to compose a system of five first-order ODEs in state variable form. That is, represent the system of odes in the vector form $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}(t))$ where

$$\boldsymbol{x} = \begin{bmatrix} z_s \\ z_u \\ z_r \\ \dot{z}_s \\ \dot{z}_u \end{bmatrix} \quad \text{and} \quad \boldsymbol{u}(t) = \begin{bmatrix} \dot{z}_r \end{bmatrix}.$$

   You do not need to represent the system in the matrix form $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}(t)$ but it may make implementation in MATLAB® simpler.

4. For a road profile defined by $z_r(x_c) = \frac{1}{25}\sin\left(\frac{2\pi}{3}x_c\right)$, determine an expression for $\dot{z}_r$ in terms of only the horizontal speed of the car, $\dot{x}_c$, and time, t. Assume that the vehicle is traveling at constant velocity.

In MATLAB®, simulate the response of the quarter car for varying horizontal velocities.

1. Create a function file to represent the system of five first-order ODEs that you determined in your analysis above. Use the code in Listing 4.1 as a template for your ODE.

   Listing 4.1: Template for MATLAB® function `qcarEOM`.

   ```
   function [x_dot] = qcarEOM(t, x, xc_dot)
   u = 0;
   x_dot = [ 0
             0
             0
             0
             0 ];
   end
   ```

   You will of course need to add your own documentation to this file. Also, notice that `qcarEOM` is *not* of the standard form expected by your `EulerSolver` or the built-in solver `ode45`; the function is non-standard because of the extra parameter `xc_dot` that represents the constant horizontal velocity of the car.

2. Open and save a backup copy of your main script for Exercise 2A just in case you want to revert back after making modifications for Exercise 2B.

   In your main script, define a function handle that will adapt the non-standard form of `qcarEOM` into the standard form required by the solvers. The function handle will be defined using an anonymous function[1].

   Listing 4.2: Example code snippet for creating a function handle.

   ```
   % Select a horizontal speed
   xc_dot = 15;    % [m/s]
   % Define a function handle to use with the ODE solvers
   qcarODE = @(t, x) qcarEOM(t, x, xc_dot)
   ```

   In this example, a function handle called `qcarODE` is created using an anonymous function. The only thing that the anonymous function does is call the function file `qcarEOM`; but, in doing so, it passes along the value of `xc_dot` already present in the MATLAB® workspace instead of requiring it as an input parameter.

   The function handle `qcarEOM` can then be used with solvers like `ode45` to simulate the response of the quarter-car suspension when the car is traveling at a given horizontal velocity.

3. Use both `EulerSolver` and `ode45` to simulate the response of your quarter-car suspension when the car is traveling horizontally at 30 [m/s]. Adjust the time-step value for `EulerSolver` until the results are consistent with those from `ode45`.

[1] An anonymous function is a MATLAB® function defined in a single line of code within a script or within another function. See the MATLAB® documentation for Anonymous Functions for additional details.

4. Set up a `for` loop to run your quarter-car suspension simulation for a wide range of speeds.

   (a) Define a vector representing each speed you wish to use when running the simulation iteratively. This vector should at least span the range from 1 $[m/s]$ to 50 $[m/s]$.

   (b) Use a `for` loop and an index variable to iterate through each element in the vector of horizontal speeds.

   (c) For each iteration, redefine the function handle `qcarODE` to use the correct horizontal speed for that iteration by indexing the vector of horizontal speeds within the anonymous function.

   (d) For each iteration, run your simulation using `ode45`; in your function call to `ode45`, set `Refine`[2] to `4` and `RelTol`[3] to `1e-9` using the function `odeset`.

   (e) For each iteration, use the simulation results to compute the ratio $\frac{\max(z_s)}{\max(z_r)}$ and record the ratio in a separate vector. This non-dimensional value is often called the "Transmissibility" and represents the amount of road motion that is transmitted to the car.

[2]The parameter `Refine` specifies a multiplier on the number of output points from the solver and can be used to improve the resolution of your data.

[3]The parameter `RelTol` is short for relative tolerance and specifies the required accuracy for the simulation. Choosing smaller numbers for `RelTol` will improve the accuracy of your results up to a certain point around `1e-9` after which there are diminishing returns.

4.4 Discussion and Deliverables

Produce the following figures:

1. Plot your simulation results from the initial run at 30 $[m/s]$ in such a way to verify that `EulerSolver` and `ode45` agree on the solution to the system of ODEs.
2. Plot the transmissibility ratio as a function of horizontal speed using the results from your `for` loop. On the plot, use the MATLAB® function called `text` to indicate on your plot the peak value of the transmissibility as well as the horizontal speed corresponding to peak transmissibility.

Answer the following discussion questions in your report:

1. Compare the results between `ode45` and `EulerSolver`. The higher-order solver `ode45` will require fewer steps than `EulerSolver` to compute the same quality of solution. Compare this difference in step count for the fifth-order quarter-car system to the difference in step count for a simple second-order system like `odefun`.
2. Closely examine the second requested figure and attempt to answer the following questions using your best engineering intuition.
   (a) The peak transmissibility occurs when there is resonance in the system. Describe in your own words what resonance means in the context of the quarter-car. Consider using an example or analogy to illustrate your point intuitively.
   (b) The horizontal speed that causes resonance is an important factor when designing vehicle suspensions. For a regular passenger vehicle, what speed might be ideal for causing resonance? Are their practical limitations to specifying the resonant speed? Which elements in the suspension model might have the greatest affect on the resonant speed?
   (c) Does the model of the road affect the speed for which resonance occurs?

Turn in a published form of your script fully annotated according to Appendix D.

Exercise 5A

The Vector-Loop Method

5A.0 Prelab

This pre-lab is due at the beginning of the laboratory period.

1. Read the background section of the assignment thoroughly and familiarize yourself with each mechanism shown.
2. For each mechanism shown in the background section draw on paper a schematic representation of the mechanism.
3. For each mechanism determine which links, if any, can move continuously. That is, determine which links can rotate to an arbitrary angle without causing the mechanism to bind up.

5A.1 Objectives

The objectives of this exercise are to:

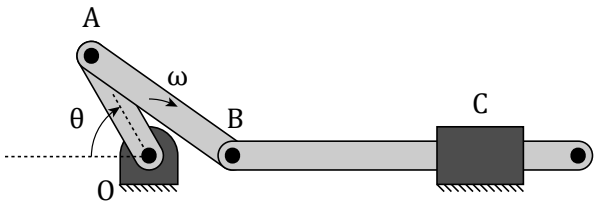
1. Become familiar with various planar mechanisms.
2. Understand the fundamentals of the vector-loop approach to analyzing first and second order kinematics for mechanisms.
3. Use a minimization tool to find valid initial conditions for a constrained differential equation.

5A.2 Background

The vector-loop process is covered in detail in Appendix E. The appendix covers kinetic analysis and simulation methods which will be covered in a future assignment. For now only consider the kinematic analysis discussed in Appendix E.

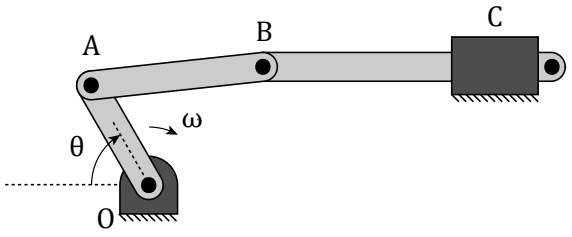
Several mechanisms are shown below. Each mechanism can be analyzed using the vector-loop approach. For each mechanism assume that the input link is rotating with constant angular velocity ω and has an initial angle θ. The origin for each mechanism is shown as point O on each diagram.

Mechanism #1: Crank-Slider



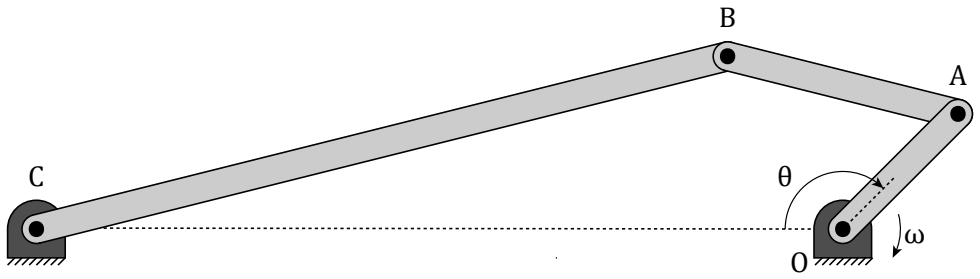
Initial Crank Angle:	$\theta = 60°$
Crank Velocity:	$\omega = 10$ [RPM]
Crank Length:	$l_{OA} = 80$ [mm]
Connecting Rod Length:	$l_{AB} = 120$ [mm]

Mechanism #2: Offset Crank-Slider



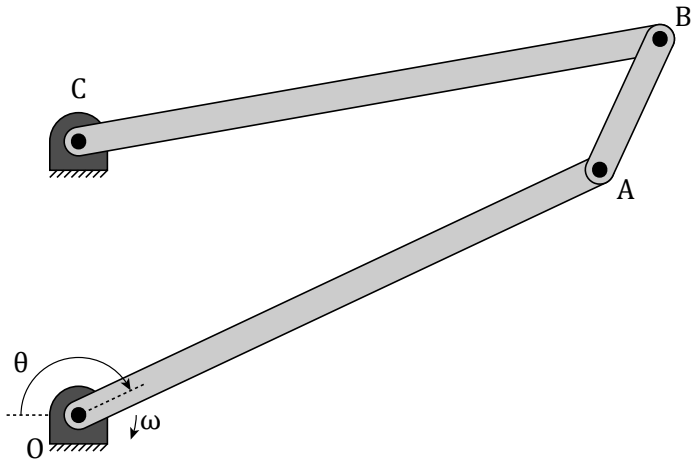
Initial Crank Angle:	$\theta = 60°$
Crank Velocity:	$\omega = 15$ RPM
Crank Length:	$l_{OA} = 80$ [mm]
Connecting Rod Length:	$l_{AB} = 100$ [mm]
Collar Height:	$y_{OC} = 80$ [mm]

Mechanism #3: Crank-Rocker



Initial Crank Angle:	$\theta = 135°$
Crank Velocity:	$\omega = 5$ RPM
Crank Length:	$l_{OA} = 100$ [mm]
Connecting Rod Length:	$l_{AB} = 180$ [mm]
Rocker Length:	$l_{BC} = 500$ [mm]
Ground Length:	$l_{CO} = 560$ [mm]

Mechanism #4: Double-Rocker



Initial Rocker Angle:	$\theta = 150^\circ$
Rocker Velocity:	$\omega = 8$ RPM
Rocker Length:	$l_{OA} = 400$ [mm]
Connecting Rod Length:	$l_{AB} = 100$ [mm]
Rocker Length:	$l_{BC} = 410$ [mm]
Ground Length:	$l_{CO} = 190$ [mm]

5A.3 Assignment

Select three (3) of the mechanisms shown in the background section. For each selected mechanism you will set up a vector loop and write a short script that performs preliminary analysis. In a future assignment you will do a time-dependent simulation using these same methods.

On paper, perform the following:

1. Form a valid vector loop according to the procedure in Section E.1 of Appendix E:
   (a) Show the vector loop graphically by adding to the schematic representations you drew in the prelab.
   (b) Show the symbolic vector loop equation, similar to Equation E.2, and indicate which angles and lengths are known (✓), unknown (?), or inputs (I). Use these results to determine which variables will go in the position vector $\boldsymbol{x}$.
2. Split the vector-loop equations into components and then differentiate each component equation to find the velocity and acceleration equations. Express these equations in matrix form according to the procedure described in Section E.2 of Appendix E.

In MATLAB®, write a short script to perform the following analysis:

1. Compute the initial conditions for the mechanism given the input angle and velocity:
   (a) Compute the initial position vector, $\boldsymbol{x}_i$, based on the minimization method described in Section E.3.1 of Appendix E.
   (b) Compute the initial velocity vector, $\dot{\boldsymbol{x}}_i$, by solving the velocity equations you have already represented in matrix form.
2. Find the value of acceleration, $\ddot{\boldsymbol{x}}_i$, at these initial conditions by solving the acceleration equations, also represented in matrix form.
3. Make a properly scaled plot showing the mechanism in its starting orientation. Use the template provided in Section E.6 of Appendix E.

5A.4 Discussion and Deliverables

Properly format each script and combine the published output into a single report.

This page is intentionally left blank.

Exercise 5B

Planar Mechanism Simulation

5B.1 Objectives

1. Extend the snapshot analysis performed in Exercise 5A to time-dependent analysis.
2. Create an animation reflecting accurate kinematics throughout the complete range of motion.

5B.2 Background

In the previous lab you considered the kinematics for three different mechanisms using the vector-loop method, but only at a single instant in time. In this lab you will be extending the analysis to perform a time-dependent simulation of one mechanism.

The final step of the vector-loop method performed in Exercise 5A was to determine a matrix equation $A\ddot{\boldsymbol{x}} = \boldsymbol{b}$ where $A = A(\boldsymbol{x})$ and $b = b(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u})$ are matrices and vectors dependent on lower-order terms and the input to the system.

The preceding matrix equation can be solved for $\ddot{\boldsymbol{x}}$, which can then be doubly integrated to determine $\dot{\boldsymbol{x}}$ and $\boldsymbol{x}$. Consider the block diagram shown in Figure 5B.1 depicting a method for simulating the kinematics of a crank-slider mechanism. The MATLAB® function block is the core of the Simulink® diagram and computes $\ddot{\boldsymbol{x}}$ in terms of $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$, and $\boldsymbol{u}$.

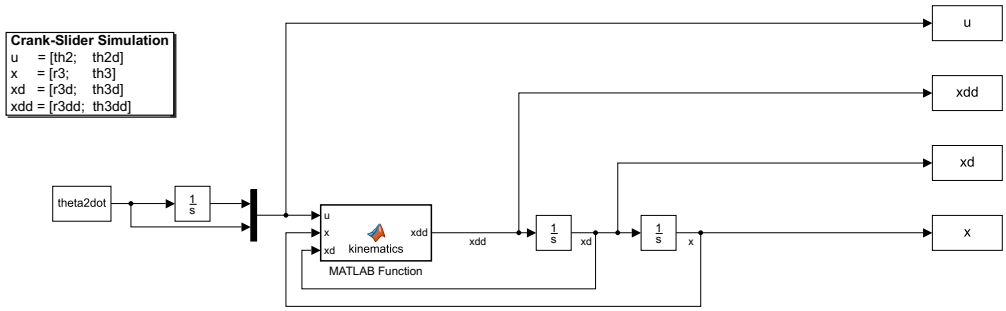


Figure 5B.1: Suggested layout for simulating planar mechanism kinematics in Simulink®.

5B.3 Assignment

Select either Mechanism #1 or Mechanism #3 from your analysis in Exercise 5A. Adapt your work to perform time-dependent kinematic analysis.

1. Begin by copying the pertinent files from the previous assignment to be modified for use in this assignment.

2. Create a Simulink® model similar to the one shown in Figure 5B.1. Double-click on the "MATLAB Function" block to open up the editor window in MATLAB®. Cut-and-paste the code from your previous assignment associated with solving $\ddot{\boldsymbol{x}} = A^{-1}\boldsymbol{b}$.

   You may need to adjust your code slightly to work with the inputs and outputs from the rest of the Simulink® model. *Reminder:* don't forget to enter the initial conditions for $\dot{\boldsymbol{x}}$ and $\boldsymbol{x}$ in the two "Integrator" blocks in your Simulink® model.

3. Run your simulation long enough for the crank to complete two full revolutions. Adjust the solver tolerance to ensure that your simulation converges appropriately over the full simulation duration.

4. Analyze your results by plotting the x- and y-components of velocity for points A and B.

5. Locate your code from Exercise 5A that you used to create the scaled plot of the mechanism in the initial configuration. Adapt this code to create an animation by placing your plot commands within a `for()` loop. Refer to the example provided in Section E.6 of Appendix E for further assistance.

   Be sure your animation includes the paths traced out by points A and B.

5B.4 Discussion and Deliverables

Produce the following plots:

1. Plot the horizontal and vertical velocities for points A and B, each against time, as separate subplots on a single figure.
2. Plot the mechanism in its final orientation showing clearly the path traced out by points A and B. The final frame of your animation may automatically show in your report handling this deliverable for you automatically.

Include the following in the body of your properly formatted report:

1. Include a screen-capture or image of your block diagram with sufficient signal labels and organization. Inclusion of the block diagram can be done through a MATLAB® command in your script, or by attaching a screenshot in your report.

Create an animated GIF in MATLAB® showing the time-dependent motion of your selected mechanism.

1. From your animation, create an animated GIF that you will submit on Canvas along with your report submission. Refer to the documentation for `imwrite` for additional support on creating the animated GIF.

Answer the following discussion questions:

1. Consider how the the simulation method would change if the input to the system were a torque applied to the crank instead of the angular velocity of the crank. Comment on your proposed changes to the analytical modeling as well as any differences you foresee in the structure of your Simulink® diagram. Would the kinetic model be significantly more complicated than the kinematic model covered in this lab?

   Note: this question is meant to be conceptual. You do not need to write any code or create any Simulink® models but a sketch of a block diagram and an equation or two may help support your reasoning.

2. What is the order of the system we are modeling? Provide explanation or validation for your answer.

3. Suppose all we wanted to know was the velocity of each point in the mechanism during its full range of motion. In this case, would it still be required to perform acceleration analysis in order to set up the Simulink® model? Explain why or why not in your response.

This page is intentionally left blank.

Exercise 6

Walking Arm Trebuchet

6.0 Prelab

This pre-lab is due at the beginning of the laboratory period.

1. Consider the walking arm trebuchet discussed at the following link.
2. Read the background section of this assignment thoroughly. Also review Exercise 5A, Exercise 5B, and Appendix E, which all cover the vector-loop approach to mechanism analysis.
3. Consider specifically the impact portion of the launch sequence, referred to as Regime 2 in the assignment. Determine a method for computing the angular velocities of Links 1, 2 and 3 just after impact, based on their angular velocities just before impact. Consider using momentum methods for this analysis. Set up diagrams and equations to support your proposed method but you don't need to work through or evaluate any of the equations.
4. Part B of this assignment will be assigned later in the Quarter and will adapt the simulation from part A into Stateflow®. Nothing is due for Part B on the prelab but it is highly recommended that you prepare for part B by taking the Stateflow® Onramp course. It is located in the Simulink® Start Page under "Learn".

6.1 Objectives

The objectives of this exercise are to:

1. Solve a system of coupled nonlinear ODEs numerically.
2. Use the vector-loop approach to simulate the kinematics associated with a planar mechanism.
3. Combine multiple simulations into one hybrid simulation.
4. Create a rudimentary animation of a mechanism.

6.2 Background

The mechanism considered in this assignment is called a walking arm trebuchet. The trebuchet is composed of two rigid bodies, AC and BD, as well as a projectile at E connected to point D by a taut cord. Figures 6.1 through 6.4 show four distinct configurations that occur during the launch sequence.

The following conventions will be used in the coming analysis: **Link 1** is defined as the rigid body AC; **Link 2** is defined as the rigid body BD, and **Link 3** is defined as the rigid body DE[1]. All links are of fixed length, and have angles defined with respect to the positive extension of the x-axis as shown in Figure 6.5. The configuration shown in Figure 6.5 does not occur during launch but is a convenient configuration to use for analysis because all of the angles are within the first quadrant.

Initial The configuration shown in Figure 6.1 is the initial configuration that occurs just as the launch is started. From this starting configuration, the trebuchet falls as a single rigid body, pivoting about fixed point D.

Just Before Impact Figure 6.2 shows the configuration after the trebuchet falls, but just before it impacts with the ground. In this configuration, point D is still the center of rotation.

Just After Impact Figure 6.3 shows the configuration just after point A on the trebuchet impacts with the ground. It should be assumed that, after impact, point A becomes a fixed point of rotation for AC.

Release The final configuration shown in Figure 6.4 is the configuration which results in the projectile at E releasing from the mechanism. The release occurs when the angle between **Link 2** and **Link 3** exceeds a specified release angle.

[1]Link 3 (DE) is not truly a rigid body, but because the cord remains taut it can be modeled as a rigid body, albeit one with no moment of inertia.

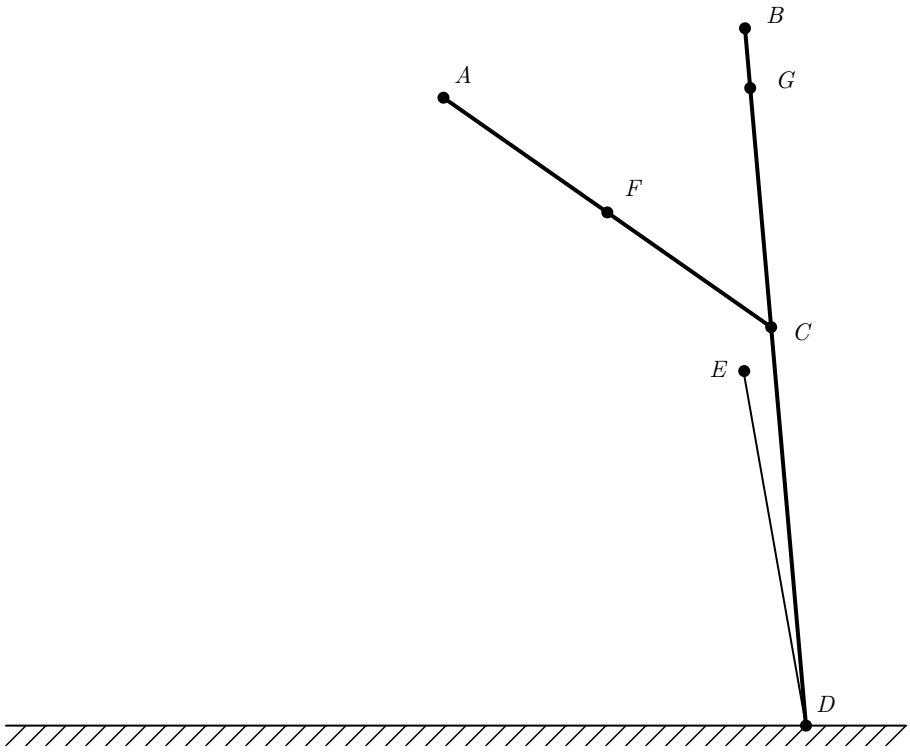


Figure 6.1: Initial configuration of walking arm trebuchet before launch.

Reference The reference configuration shown in Figure 6.5 is not part of the motion of the trebuchet during normal operation. This configuration is meant only to act as a reference in which all angles are in the first quadrant. Refer to this configuration for any kinematic or kinetic analysis.

The center of mass of **Link 1** is located at point F, the center of mass of **Link 2** is located at point G[2], and the center of mass of **Link 3** is located at point E. The cord connecting points D and E can be considered massless. Rigid bodies AC and BD can both be considered as uniform slender rods. The mass and geometric parameters you will use in your analysis of the walking arm trebuchet are listed in Table 6.1.

This system is a good example of a hybrid system, because there are several clearly distinct regimes of motion: Falling, Impact, Swinging, and Flying. One method to handle the hybrid nature of the system is to simulate the complete behavior of the system through several separate simulations combined together into one composite result. Consider the following regimes:

[2]The location of point G can be found by finding the composite center of mass for the rigid body BD and the point mass located at B.

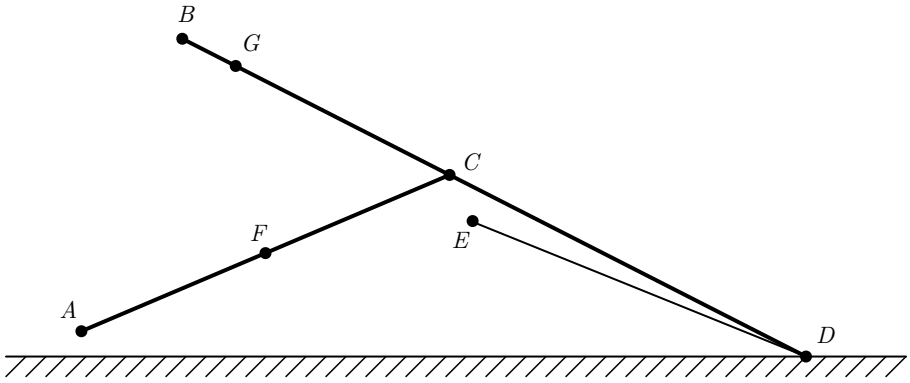


Figure 6.2: Configuration of walking arm trebuchet just before impact.

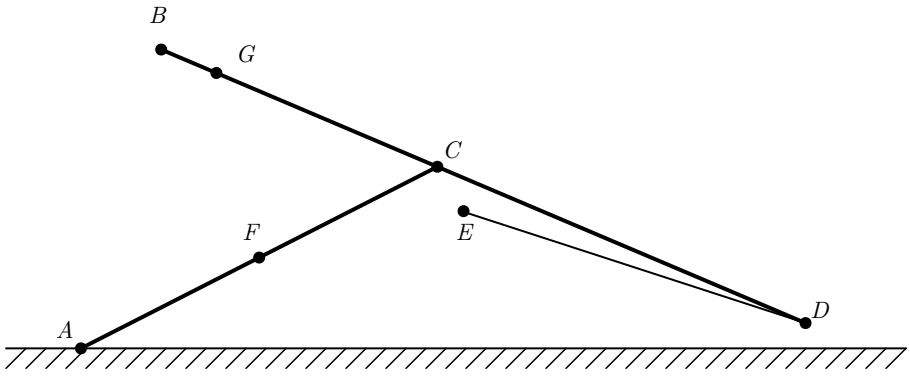


Figure 6.3: Configuration of walking arm trebuchet just after impact.

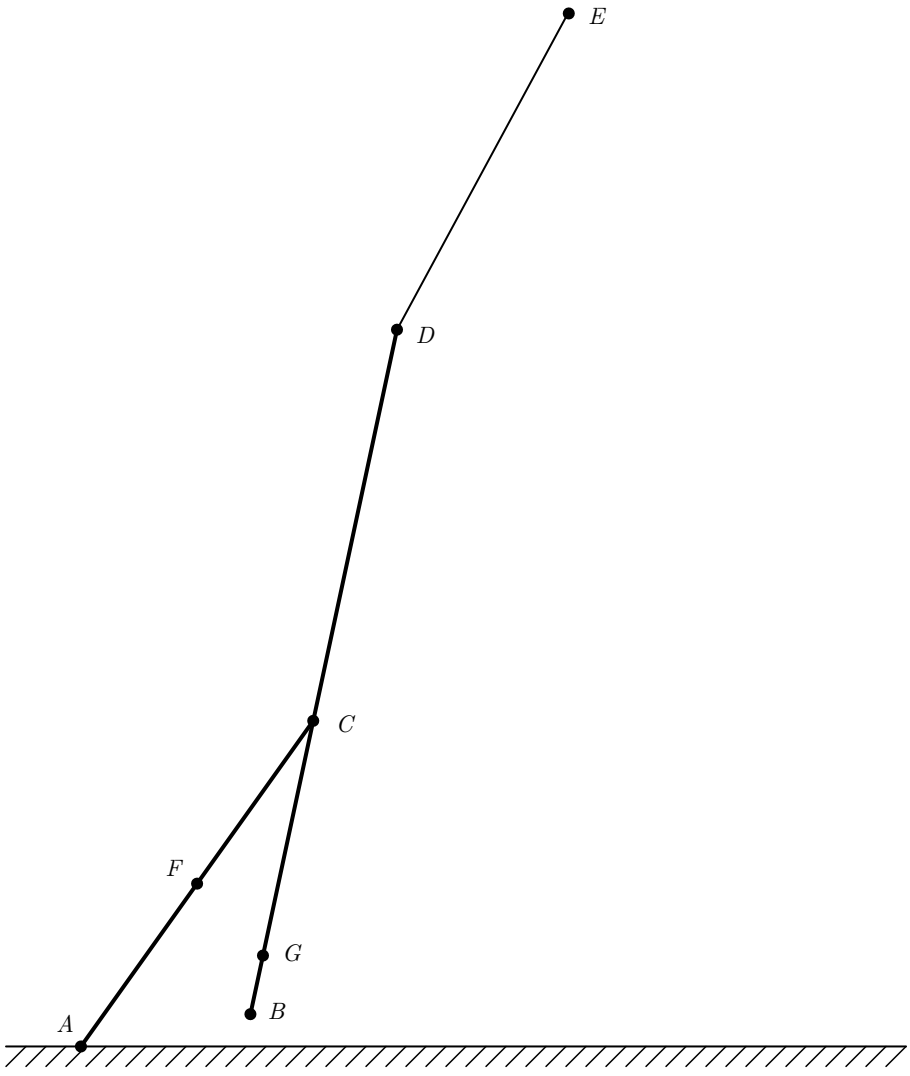


Figure 6.4: Release configuration of walking arm trebuchet.

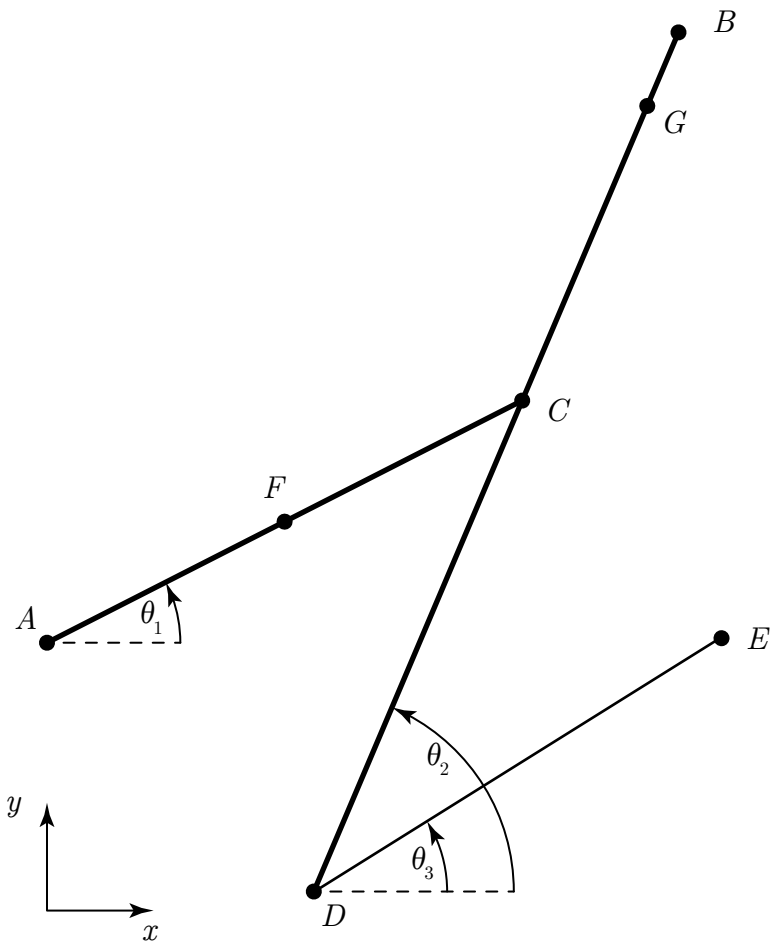


Figure 6.5: Reference configuration of the walking arm trebuchet with angle definitions.

Regime 1 - Falling In the first regime, the trebuchet falls from the initial configuration, shown in Figure 6.1, to the configuration just before impact, shown in Figure 6.2. In this regime, all parts of the mechanism fall together and can be considered as a single rigid body pivoting about point D.

Regime 2 - Impact When the trebuchet hits the ground, it begins to pivot about point A, instead of point D, as illustrated by the changes between Figures 6.2 and 6.3. This regime occurs instantaneously and is best modeled using momentum based methods.

Regime 3 - Swinging After impact, the trebuchet swings back upwards as the counterweight continues to descend toward the ground. In this regime, AC, BD, and DE can all be considered as separate rigid bodies rotating with independent angular velocities. This regime continues until the release condition is met when the angle between **Link 2** and **Link 3** exceeds the specified release angle.

Regime 4 - Flying Once the release condition is satisfied, the projectile at E departs from the mechanism and flies away toward the target.

Table 6.1: Geometric and mass properties for walking arm trebuchet.

Length of AC	r_{AC}	20 $[in]$
Length of BC	r_{BC}	15 $[in]$
Length of CD	r_{CD}	20 $[in]$
Length of DE	r_{DE}	18 $[in]$
Linear weight density of links	ρ	0.7 $[^{lbf}/_{ft}]$
Weight of counterweight	w_B	20 $[lbf]$
Weight of projectile	w_3	4 $[ozf]$

6.3 Assignment

On paper, perform analysis in preparation for your simulations. Complete each of the following symbolically in terms of variables; that is, not in terms of numbers. Some of the following analysis may be provided by your instructor as reference material.

1. Solve the system kinematics before impact using the vector loop approach. That is, form three separate vector loops, split each loop into components, and then differentiate the resulting expressions to find the horizontal and vertical components of velocity and acceleration at the center of mass for each link when point D is fixed.

2. Solve the system kinematics after impact using the vector loop approach. That is, form three separate vector loops, split each loop into components, and then differentiate the resulting expressions to find the horizontal and vertical components of velocity and acceleration at the center of mass for each link when point A is fixed.

3. Solve the system kinetics for the Falling regime of motion by defining equations of motion using a free-body diagram and a kinetic diagram. Remember that all parts of the mechanism fall together so all links have the same angular velocity and angular acceleration.

4. Solve the system kinetics for the Impact regime of motion: define expressions relating the angular velocity after impact to the angular velocity before impact using an impulse-momentum diagram. Determine these expressions symbolically so that you can find the angular velocities just after impact for any arbitrary velocities and orientation just before impact.

5. Solve the system kinetics for the Swinging regime of motion by defining equations of motion using a free-body diagram and a kinetic diagram. Remember that all parts of the mechanism swing independently.

6. Solve the system kinetics for the Flying regime of motion by defining equations of motion for the projectile[3] during the Flying regime of motion using a free-body diagram and a kinetic diagram.

[3]None of the analysis considers motion of the trebuchet after the Swinging regime ends. The Flying regime exclusively considers the projectile.

Using the starter project provided by your instructor, complete the development of a set of MATLAB® and Simulink® files that simulate the complete[4] behavior of the walking arm trebuchet. Some of the following may already be implemented in the starter project; if not, adjust the project to achieve the following:

1. Define all of the constant parameters that will be used across all parts of your project. Define all of the parameters together in only one place, so that if you need to adjust any parameters you can be sure the values remain consistent throughout your entire project.

2. Define the initial conditions associated with your desired starting configuration for the trebuchet. Also define the release condition that dictates when the projectile leaves the sling. These four parameters allow you to optimize the distance that your trebuchet throws. As you are changing values to increase your launch distance, make sure that the counterweight at point B doesn't hit the ground before the projectile departs; in practice, such a condition would cause a misfire.

3. Create a Simulink® model for Regime 1 and simulate the trebuchet falling using the Simulink® model. Use a "Stop" block to automatically end the simulation when point A impacts with the ground.

4. Create a function for Regime 2 that relates the angular velocities just after impact to the angular velocities just before impact. Use the final system state from Regime 1 as the conditions just before impact. The conditions just after impact will be used as the initial conditions for Regime 3.

5. Create a second Simulink® model for Regime 3 and simulate the trebuchet swinging. Use another "Stop" block to end the simulation when the release condition is satisfied.

6. From the final system state of Regime 3 determine the initial conditions for Regime 4.

7. Create a third and final simulation for Regime 4 that simulates the motion of the projectile starting from the instant it departs from the mechanism.

8. Make an animation showing the mechanism moving through all ranges of motion.

   (a) Include in the animation the trajectory of the counterweight during Regime 1 through Regime 3 and the trajectory of the projectile during all four regimes. The trajectories should show in the animation as "trails" following behind each point of interest as the mechanism moves through the full range of motion.

   (b) Ensure that the animation is properly scaled; properly scaled means that the links do not change length as the animation proceeds *and* that the relative size of the mechanism compared to your figure window is large enough to easily

[4] Your simulation does not need to consider motion of the trebuchet after the projectile departs.

examine the trajectory of each point of interest. It may be useful to have the figure window follow the trajectory of the projectile after it departs from the mechanism so that the whole trajectory can be animated without causing scaling issues in your figure.

(c) Save the animation as GIF; see the Discussion and Deliverables section below for additional details.

You may use the template provided in Section E.6 of Appendix E as a starting point for your animation code.

9. Produce any additional plots requested in the Discussion and Deliverables section.

6.4 Discussion and Deliverables

Produce the following figures:

1. Plot the speed of the projectile as a function of time during all four regimes of motion.
2. Create a figure with three subplots showing the starting configuration, the impact configuration, and the release configuration. It may be useful to reuse some of the code from your animation.

Create and save an animation using MATLAB®:

1. The animation you have generated provides evidence that your kinematics and kinetics are reasonable but requires you to run your script in order to display the animation. From this animation, produce an animated GIF[5] that you will submit on Canvas along with your finished report.

   See the documentation for the `imwrite` function for additional support on creating an animated GIF.

Include the following in the body of your properly formatted report:

1. Include a screen-capture or image of each block diagram with detailed signal labels and proper organization. Inclusion of the block diagram can be done programmatically through a MATLAB® command in your script, or by attaching a screenshot at the end of your report.

Answer the following discussion questions:

[5]There is some debate on the pronunciation of GIF. In the author's opinion it is pronounced with a hard G as in 'Graphics Interchange Format', and *not* with a soft G as in 'Giraffics Interchange Format'.

1. In this assignment you are not required to simulate the motion of the mechanism once the projectile departs. Describe how you could analyze the residual motion of the mechanism by simulating an additional regime of motion. In your description provide details about your proposed method, including, but not limited to, the number and selection of state variables.

2. Determine the theoretical maximum distance that the projectile could travel under ideal conditions, assuming that the ground is flat and level. Compare this theoretical maximum distance to the actual distance that your simulated projectile travels. Why does your simulation throw the projectile a shorter distance than the theoretical maximum? What adjustments or tuning could be performed to maximize the travel distance?

3. In regards to your simulation results, what percentage of the initial potential energy (of the whole system) is converted to kinetic energy of the projectile at the time of release? In other words, how efficient is the trebuchet at converting potential energy to kinetic energy? Considering your answer to the previous question, does maximum efficiency translate to maximum distance traveled?

This page is intentionally left blank.

Appendix A

State Determined Systems

Various methods exist for analyzing and solving ODEs. Generally, these methods are applied to a set of first-order ODEs describing the system. By introducing N intermediate variables, called state variables, an ODE of order N can be expressed as a system of N first-order ODEs. The first-order equations can be written in the matrix form shown below, where x_i is a state variable and $f_i\left(x_1, \ldots, x_N, t\right)$ is a function of only x_1 through x_N and time.

The set of state variables can be arranged in a column vector, $\boldsymbol{x}$, called the state vector. The system of N first-order ODEs is commonly referred to as the state space representation of the system. The word 'state' refers to the fact that the set of state variables is sufficient to fully characterize the state of the system. For any system, the choice of state variables is not unique, however, regardless of the choice of state variables, the order of the system, N, is unique.

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} f_1\left(x_1, \ldots, x_N, t\right) \\ \vdots \\ f_i\left(x_1, \ldots, x_N, t\right) \\ \vdots \\ f_N\left(x_1, \ldots, x_N, t\right) \end{bmatrix} \text{ or } \frac{d}{dt}\boldsymbol{x} = \boldsymbol{f}\left(\boldsymbol{x}, t\right) \tag{A.1}$$

A.1 Forming a System of First Order ODEs

Any ODE of arbitrary order can be split into a collection of first-order ODEs. Here we will consider an example of a dynamic system: a parallel mass spring damper with a force input. Refer to the diagram in Figure A.1, showing a schematic of the system.

At time zero, when the forcing function starts acting on the mass, the mass is at an initial displacement x_0 and an initial velocity $\dot{x}_0$. First begin with a free body diagram and a kinetic diagram to develop the equation of motion, often called an EOM. The free body diagram and kinetic diagram are shown in Figure A.2. The EOM is then rearranged to solve for the highest-order ODE, as seen in equation A.2.

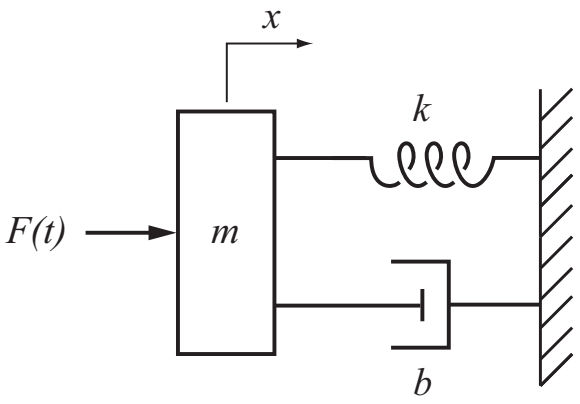


Figure A.1: Mass spring damper with forcing function

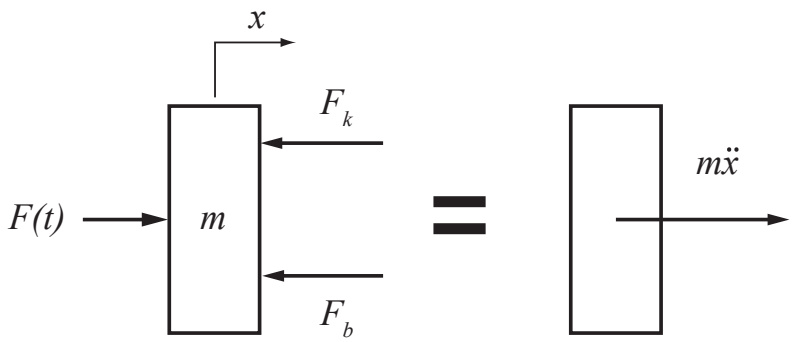


Figure A.2: Free Body Diagram and Kinetic Diagram

$$\begin{aligned} \Sigma F &= m\ddot{x} \\ -F_b - F_k + F(t) &= m\ddot{x} \\ -b\dot{x} - kx + F(t) &= m\ddot{x} \\ \ddot{x} &= -\frac{b}{m}\dot{x} - \frac{k}{m}x + \frac{1}{m}F(t) \end{aligned} \tag{A.2}$$

This second-order ODE can be split up into two first order ODEs with a change of variable.

$$\frac{d}{dt}\dot{x} = -\frac{b}{m}\dot{x} - \frac{k}{m}x + \frac{1}{m}F(t) \tag{A.3}$$

and

$$\frac{d}{dt}x = \dot{x} \tag{A.4}$$

These two equations can be put into matrix form:

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ -\frac{b}{m}\dot{x} - \frac{k}{m}x + \frac{1}{m}F(t) \end{bmatrix} \tag{A.5}$$

The vector $\boldsymbol{x} = \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]$ is denoted as the state vector. Equation A.5 is the system state equation that describes the time rate of change of the system's states. An important property of a state space model is that the time rate of change of the system's states are functions of $\boldsymbol{x}$ and $\boldsymbol{u}$, where $\boldsymbol{u}$ is a function of time. A state-determined system only requires information of the external input to the system and the current state of the system to predict any future state of the system.

This page is intentionally left blank.

Appendix B

State Space Representation

While the model presented in Appendix A is a state space model, that term is typically reserved for a certain representation of state determined systems. It is often desired to have a set of equations representing the state space model in the form of $\dot{\boldsymbol{x}} = A(\boldsymbol{x}, t)\boldsymbol{x} + B(\boldsymbol{x}, t)\boldsymbol{u}(t)$. In this form the ODE is split into the part resulting from the state of the system, $A(\boldsymbol{x}, t)\boldsymbol{x}$ and the part resulting from any external inputs $B(\boldsymbol{x}, t)\boldsymbol{u}(t)$.

B.1 Linear State Space Form

If A and B are both constant-valued matrices containing the coefficients representing the system of ODEs, then the system is linear. This allows techniques from linear algebra to be used when solving the system of ODEs.

Reconsider the mass-spring-damper examined in Appendix A. The form $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u}$ can be found by manipulating the right hand side of the vector equation found previously.

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ -\frac{b}{m}\dot{x} - \frac{k}{m}x + \frac{1}{m}F(t) \end{bmatrix} \tag{B.1}$$

$$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}\begin{bmatrix} F(t) \end{bmatrix} \tag{B.2}$$

Where

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix}, \quad \text{and } \boldsymbol{u} = \begin{bmatrix} F(t) \end{bmatrix}$$

Since MATLAB® works well with matrices, a convenient way to post-process a simulation's results is through a linear set of equations of the form $\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u}$. The outputs, $\boldsymbol{y}$, are a set of system variables of interest. An important property of the linear state equations are that all outputs can be expressed as a linear combination of the system's states, $\boldsymbol{x}$, and external inputs to the system, $\boldsymbol{u}$.

Consider the mass spring damper system above where $x, \dot{x}, \ddot{x}$, and F_{total} are the system's outputs of interest.

$$\begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ F_{total} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \\ k & b \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m} \\ 0 \end{bmatrix}\begin{bmatrix} F(t) \end{bmatrix}$$

Where

$$\boldsymbol{y} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ F_{total} \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -\frac{k}{m} & -\frac{b}{m} \\ k & b \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m} \\ 0 \end{bmatrix}$$

In the context of MATLAB®, this allows us to build a state-space system object using the `ss` command or to use the state-space block in Simulink®.

Appendix C

Solving ODEs

Many methods exist for integrating ODEs including Euler's method and many more sophisticated higher-order methods. This document will only discuss one method in detail.

C.1 Euler's method

Once a set of first-order state equations have been developed, Euler's method can be applied. Euler's method begins by making the assumption that the rate of change of each state of the system can be approximated as constant if considered over a sufficiently small period of time. This assumption allows the calculation of a small change in the state of the system over the small period of time.

$$\frac{\Delta x_i}{\Delta t} \approx \frac{dx_i}{dt} \tag{C.1}$$

$$\Delta x_i \approx \frac{dx_i}{dt}\Delta t \tag{C.2}$$

Recall that the rate of change of each state x_i can be expressed as a function of the all the states and time.

$$\frac{dx_i}{dt} = f_i\left(x_1, \ldots, x_N, t\right) \tag{C.3}$$

The "constant" $\frac{dx_i}{dt}$ leads quickly to the standard forward Euler method. The change in state is accumulated according to $x_{i,k+1} = x_{i,k} + \Delta x_i$. This allows Euler's method to predict the next state based on the current state, and the rate of change of the state.

$$\Delta x_i \approx f_i\left(x_1, \ldots, x_N, t\right)\Delta t \tag{C.4}$$

$$x_{i,k+1} \approx x_{i,k} + f_i\left(x_{1,k}, \ldots, x_{N,k}, t_k\right)\Delta t \tag{C.5}$$

The value of x_i at step $k+1$ is computed as the current value of x_i at step k plus an increment calculated by multiplying the derivative at step k by the small increment in time Δt. The numerical solution marches forward through time: based on the current solution values and the time step, estimate the current derivative, apply that derivative over the small interval Δt to find the new solution value, and repeat the process.

This method can easily be extended to a vector of states.

$$\boldsymbol{x}_{k+1} \approx \boldsymbol{x}_k + \boldsymbol{f}\left(\boldsymbol{x}_k, t_k\right) \Delta t \tag{C.6}$$

C.2 Higher-Order Solvers

As shown above, a method of solving differential equations can be programmed into MATLAB® manually. By adjusting the time step size, the solution can be found to converge over a desired range. However, this process is rather tedious and the solution requires thousands of calculations to solve a single ODE.

Somewhat counter-intuitively, the speed and efficiency of numerical integration depends upon efficient and accurate calculation of the instantaneous derivative with respect to time. These derivatives are calculated (numerically estimated) and used to predict the value of each state variable at the next time step. The derivatives are recalculated, and the process continues, marching forward in time. Luckily, there are more efficient algorithms than the Euler solution developed above, including many available in the MATLAB® environment.

Throughout this lab manual we will suggest using `ode45` to compute the solutions to most simulations. The built-in function `ode45`, uses a variant of the Runge-Kutta method, which itself is an extension of Euler's method. The `ode45` solver is capable of adjusting the step size throughout the solution process to maintain a specified error unlike fixed time step methods.

All of the built-in solvers function the same way: they integrate a function, of a specified format, provided by the user. Typically, a function file is used, which receives the current instant in time `t` and the current state vector `x` and returns its derivative, `xdot`. That is, the built-in solvers can only solve functions of the form

```
xdot = odefun(t,x)
```

Depending on the order of the system, the length of input/output vector will change, but the same process applies. For reference, a standard ODE45 call has the following form:

```
[tout, xout] =  ode45(@odefun, [tspan], [x0], options);
```

Where `@odefun` is a handle for the function to be solved. The time interval can be specified as a row vector `tspan` and the initial conditions can be specified as a vector of equal length to the output vector, `x0`.

Appendix D

Report Format

The reports for ME326 can be generated automatically by MATLAB® but you need to make sure that you format your code properly for the report to turn out right. All reports must be written in a technical format beginning with a detailed introduction including a description of the problem statement and an overview of your solution method. Describe any new functions and concepts that you implemented and why they were appropriate to use.

Additional detailed comments should be present, explaining the usage of particular lines of code. All new variables should have descriptive names and comments indicating their physical representation, including units. Each non-trivial operation should have a brief explanation of how the operation works and why it is being used. Any new built-in functions should include a description of operation.

All plots must have a figure caption as well as engineering analysis. The level of documentation you should strive for is such that if you were not present, someone of similar skill to yourself could pick up your code and easily understand its functionality.

User defined functions should be documented in their own file according to these same principles.

Submitted lab reports should include neat hand-written derivations and diagrams where appropriate. All hand calculations should be written on engineering paper and include a single equation per line. If online submission is required, properly scan your hand calculations and append them to your report PDF.

D.1 Live Scripts

Live scripts were introduced in MATLAB® R2016a. A live script allows you to write your code and your report at the same time with the content and formatting seamlessly integrated. Live scripts can greatly reduce the level of frustration associated with the publishing techniques mentioned in the previous section. If you use a live script, formatting your code is a simple as any other WYSIWYG[1] editor like Microsoft Word.

[1] WYSIWYG stands for "what you see is what you get".

D.2 Example Report

A version of the following example report is available on Canvas as a MATLAB® script.

ME326 Ex. #0 - Example Report Format (Main Script)

Author: Charlie Refvem

Date Created: 04/06/2020

Date Modified: 04/06/2020

Description:

This script serves as an example report for a simple dynamic simulation. It utilizes the Live Editor in MATLAB® to generate a formatted report that can be exported from MATLAB® to a PDF, a word document, or a webpage. This example will analyze the motion of a wheel rolling down an incline due to gravity. Additional details can be read below in the problem statement section.

Required Files:

- **`sample_data.mat`** (Data File) - This file contains a collection of data that must be loaded directly into the workspace. There are two variables contained with the file: **`theta`** and **`time`**. These variables provide the angle of the rolling wheel sampled over a certain interval of time.
- **`my_diff.mlx`** (Function File) - This function computes the difference between adjacent rows in a vector or array. Note: this function is already built into MATLAB®, but is included for the sake of demonstration.

Table of Contents

Problem Statement

Consider a wheel of diameter 40 cm rolling down an incline with a constant grade of approximately 5%. Data have been collected over ten seconds that describe the angle of rotation of the wheel with respect to the ground. It is desired to find the velocity of the center of the wheel as well as the average grade of the incline. Assume that there is no slip between the wheel and the ground and that it can be modeled as a uniform cylinder.

Analysis

The hand calculations below provide foundation for the MATLAB® solution. It is important to have evidence or verification for the mathematical solution methods being employed in MATLAB®.

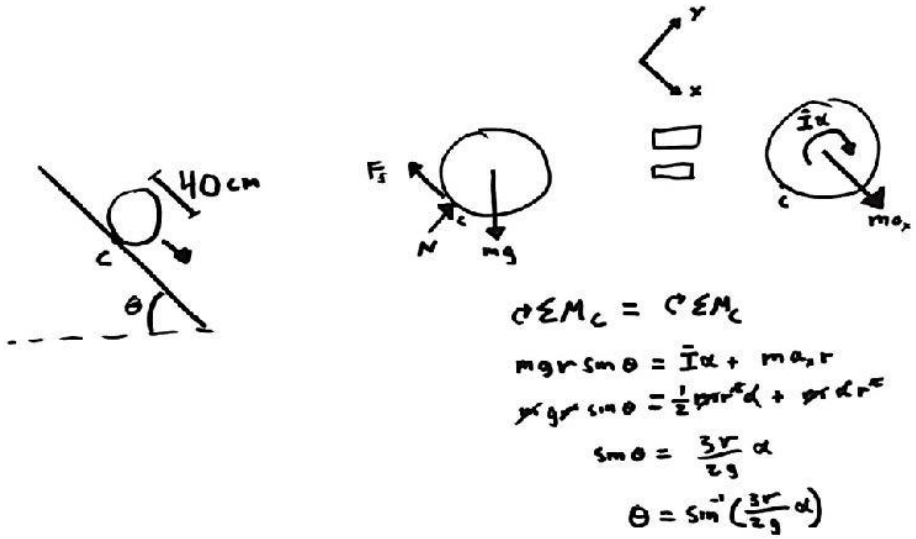


Figure 1: Rolling Cylinder Kinetic Analysis

Constant Parameters

```
r = .20; % wheel radius [m]
g = 9.81; % acceleration due to gravity [m/s^2]
```

The following code loads the wheel rotation angle data and time arrays.

- Angle data are in radians.
- Time data are in seconds.

```
data = load('sample_data');
```

Here the number of datapoints is computed.

```
N=length(data.time);
```

Velocity Calculation

This section will calculate the velocity of the center of the rolling wheel by first differentiating the provided angle data, and then relating that angular velocity to the linear velocity using the wheel's radius. Here the angular velocity is approximated by performing an element-wise division of the difference in theta values by the difference in time values.

```
omega = my_diff(data.theta)./my_diff(data.time);
velocity = r*omega;
```

Acceleration Calculation

Here the velocity is approximated by performing an element-wise division of the difference in theta values by the difference in time values.

```
alpha = my_diff(omega)./my_diff(data.time(1:end-1));
```

Grade (Slope) Calculation

Here the angular acceleration is calculated by differentiating the velocity results from the previous section. Then the grade can be back-calculated from the angular acceleration using the results from the hand calculations. The angle of inclination is found by summing moments at the contact point and solving for the unknown angle. See the hand calculations above in Figure 1.

```
incline=asin(3*r*alpha/(2*g));
```

FInally, the angle in radians is converted to a percent grade.

```
grade=100*tan(incline);
```

Velocity Results

Here, a plot is generated to show the velocity of the wheel as a function of time. To make a nicely captioned figure in MATLAB® simply use the Live Editor to place text below the block of plotting code; the results look quite nice. In addition to captions, every figure needs a reference in the body of the report. You should reference the figure by its number and comment on your thorough analysis of the plot.

First, a new figure is created and the figure handle is stored as a variable. It is useful to include a figure number as an argument to the **`figure`** command because this argument instructs MATLAB® to reuse an existing figure if it already exists.

```
h2 = figure(2);
```

Here, a plot of velocity versus time is created on the recently generated figure.

```
plot(data.time(1:end-1),velocity);
```

Finally, the label and units are defined for the vertical and horizontal axis.

```
xlabel({'Time [s]'});
ylabel('Velocity Along Incline [m/s]');
```

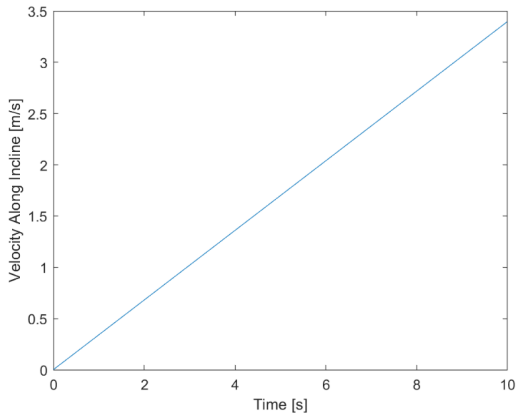


Figure 2: Linear velocity versus time for the center of the wheel.

If multiple pieces of data were shown on this plot, then a legend would also be necessary.

The linear velocity of the wheel is plotted against time in Figure 2. From the plot, it would appear that the velocity is a linear function with respect to time. This makes great sense since the linear acceleration should be constant for an incline of constant slope, as shown in the hand calculations back in Figure 1. So long as the slope remains constant, we would expect the straight trend to continue if more data points were provided.

Grade Results

The results of the grade calculation are presented using the **`fprintf`** command. The grade results are very close to the approximate value of 5% given in the problem statement; this is a good indicator that the results are correct. Further investigation into the variance of the grade may give more or less confidence to the quality of the solution.

The following line of code allows the script to output formatted text to the command window or the published report. Here **`%3.2G`** is the format specifier indicating how the mean value of the grade is printed, in this case as a truncated decimal number. The **`3`** indicates the field width (three characters). The **`.2`** indicates the precision (two decimal places). And the **`G`** indicates fixed point notation without any trailing zeros.

```
fprintf('The average grade is %3.2G%%.\n',mean(grade));
```

```
The average grade is 5.2%.
```

Discussion Questions

Question #1

Consider the first requested plot which displays the velocity at the center of the wheel as a function of time. Does the wheel start at rest? Would your code need to change if the wheel started with initial upward velocity, initial downward velocity, or no initial velocity?

In reference to Figure #2 above, it is apparent that the velocity starts at zero, or very near zero. This could be confirmed by simply comparing the first two data-points to see if any motion has occured between these two data points.

```
disp(data.theta(1:2))
```

```
    1.0e-04 *

         0
    0.8491
```

```
disp(data.time(2)-data.time(1))
```

```
    0.0100
```

Based on these numerical results, it appears that the angle of the wheel only changes on the order of 10^{-4} [rad] between the first two datapoints over an interval of 0.01 seconds. Therefore, within the resolution of the sample data it appears that the initial velocity is in fact zero.

The process performed to find the grade of inclination only depends on the angular acceleration, not the angular velocity. Therefore, the initial value of the velocity is irrelevant to the grade calculation.

ME326 Ex. #0 - `my_diff()` Function

Author: Charlie Refvem

Date Created: 04/06/2020

Date Modified: 04/06/2020

Description:

Computes the difference between rows in a vector or an array.

`difference = my_diff(data)` computes the numerical difference between rows in an array or vector **`data`**. This function can be used to calculate a rudimentary derivative of sorts by doing an elementwise division of the numerical difference from data set by the numerical difference from another data set. The return value **`difference`** will be one row shorter than the input array or vector **`data`**.

```
function difference = my_diff(data)
% Compute the difference by subtracting the first (N-1) rows from the
% last (N-1) rows.
difference = data(2:end,:) - data(1:end-1);
end
```

Appendix E

The Vector-Loop

Vector-loop analysis can be applied to any planar mechanism, however the following example will focus on the crank-slider with a rotating collar shown below in Figure E.1. As the crank rotates at constant velocity ω, the slider reciprocates within the collar.

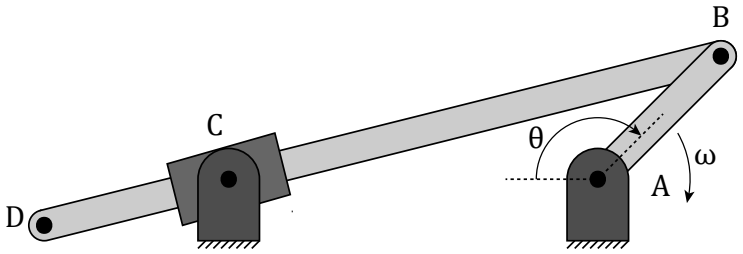


Figure E.1: Link $\overline{\text{BD}}$ slides through the pivoted collar at C, driven by crank $\overline{\text{AB}}$ rotating clockwise at constant angular velocity ω.

E.1 Forming the vector-loop

The vector-loop is formed by representing each link in the mechanism with a vector $\boldsymbol{r}_n$. A valid loop is formed if a set of vectors $\boldsymbol{r}_1 ... \boldsymbol{r}_n$, each representing a link in the mechanism, forms a *closed* loop.

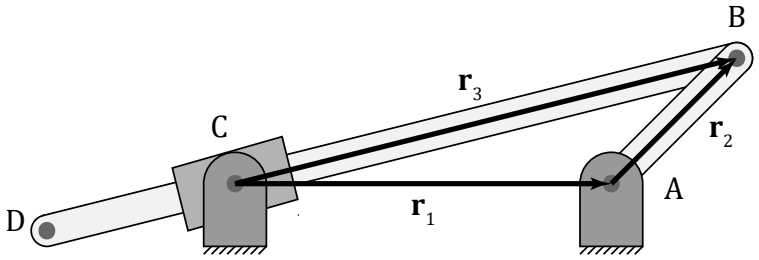


Figure E.2: Vectors $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, and $\boldsymbol{r}_3$ are overlaid on the crank-slider mechanism to form a closed vector-loop.

For the crank-slider mechanism shown in Figure E.2:

$$\boldsymbol{r}_1 + \boldsymbol{r}_2 = \boldsymbol{r}_3 \tag{E.1}$$

This equation is the fundamental relationship that will be used to derive the kinematic relationships for the crank-slider mechanism. Note that different signs can be assumed for the vectors representing each link, and that the resulting equation can be written in several equivalent forms[1]; however, the kinematic relationships derived will be the same.

A vector can be thought of as a directed line segment, implying the notion of length and direction. Figure E.3 shows the vector loop freed from the mechanism in order to clearly illustrate the length and direction of each vector $\boldsymbol{r}_1$ through $\boldsymbol{r}_3$.

Note: Even though the crank angle is defined clockwise in the original problem as θ, a new angle θ_2 must be defined. This is to enforce the convention that all angles are defined with respect to the positive extension of the x-axis.

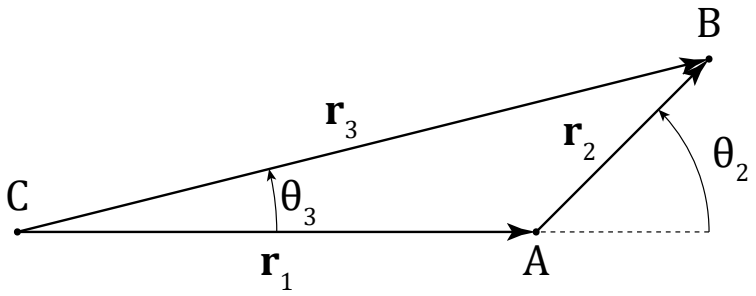


Figure E.3: Vectors $\boldsymbol{r}_2$ and $\boldsymbol{r}_3$ both change angle and vector $\boldsymbol{r}_3$ changes length.

Equation E.1 can be rewritten using the idea of length and direction to help bring further insight into the vector loop. Analysis of each vector in the loop shows which variables will be unknown, which will be known, and which will be inputs to the system.

$$\begin{array}{c} \checkmark\ \checkmark \qquad \checkmark\ I \qquad ?\ ? \\ \boldsymbol{r}_1(r_1,\theta_1) + \boldsymbol{r}_2(r_2,\theta_2) = \boldsymbol{r}_3(r_3,\theta_3) \end{array} \tag{E.2}$$

Here, a check mark ($\checkmark$) indicates a known variable, a question mark (?) indicates an unknown variable, and the input is indicated by an (I). Both the length and angle of $\boldsymbol{r}_1$ are constant. The length of $\boldsymbol{r}_2$ is constant but its angle is changing in a known manner so it is indicated as an input. Neither the length nor the angle of $\boldsymbol{r}_3$ is known, so both the length, r_3, and angle, θ_3, will be considered unknown.

[1]For the vector signs assumed in Figure , the following equations are several of the equivalent ways of writing the corresponding vector loop equation: $\boldsymbol{r}_1 + \boldsymbol{r}_2 = \boldsymbol{r}_3$; $\boldsymbol{r}_1 + \boldsymbol{r}_2 - \boldsymbol{r}_3 = 0$; $\boldsymbol{r}_1 = \boldsymbol{r}_3 - \boldsymbol{r}_2$; $\boldsymbol{r}_2 = \boldsymbol{r}_3 - \boldsymbol{r}_1$.

The two lower order terms will be grouped together to compose the state vector $\boldsymbol{x}$. For properly constrained mechanisms[2] the number of lower order unknowns should always be two, satisfied by the two vector loop component equations.

$$\boldsymbol{x} = \begin{bmatrix} r_3 \\ \theta_3 \end{bmatrix}$$

Similarly, the input to the system along with its derivatives will be grouped together to compose the input vector $\boldsymbol{u}$.

$$\boldsymbol{u} = \begin{bmatrix} \theta_2 \\ \dot{\theta}_2 \\ \ddot{\theta}_2 \end{bmatrix}$$

Note: It is critical to properly identify which variables are part of the state vector, which are constants, and which are inputs to the system in order to properly simulate the behavior of the mechanism.

E.2 Formulating differential equations

In order to find a differential equation, the system first needs to be split into horizontal and vertical components. If a Cartesian coordinate system is overlaid at the origin of each link, then the length and direction of each associated vector give

$$x_n = r_n \cos\theta_n \text{ and } y_n = r_n \sin\theta_n$$

where r_n is the length of the vector and θ_n is the angle measured CCW between the positive extension of the x-axis and the vector. For simplicity and compactness of notation, a shorthand can be defined as

$$c_n = \cos\theta_n \text{ and } s_n = \sin\theta_n$$

so that

$$x_n = r_n c_n \text{ and } y_n = r_n s_n$$

Again consider Figure E.3, which depicts the vectors $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, and $\boldsymbol{r}_3$ forming a valid vector loop. To derive the kinematic equations from the vector loop, the x-components and y-components of the vector-loop equation must be considered separately.

[2] That is, for properly constrained *single degree of freedom* mechanisms, the number of unknowns should be two.

Note: Getting the signs correct in this process is facilitated by distorting the vector loop so that each vector is in the first quadrant[3]. Of course any distortion of the mechanism must preserve the relationship $\boldsymbol{r}_1 + \boldsymbol{r}_2 = \boldsymbol{r}_3$.

Crank-slider Position Equations

Horizontal Components:

$$r_1 + r_2 c_2 = r_3 c_3 \tag{E.3}$$

Vertical Components:

$$r_2 s_2 = r_3 s_3 \tag{E.4}$$

Differentiation of equations E.3 and E.4 with respect to time[4] yields two coupled first order differential equations. The two coupled equations can be put into a single matrix equation of the form $A\dot{\boldsymbol{x}} = \boldsymbol{b}$, where $A = A(\boldsymbol{x})$ is a matrix that is a function of lower order terms and $\boldsymbol{b} = \boldsymbol{b}(\boldsymbol{x}, \boldsymbol{u})$ is a vector that is a function of the lower order terms and the input to the system.

[3]The first quadrant is helpful in determining signedness because both the x- and y-components are positive.

[4]Note on differentiation of implicit functions: c_n and s_n are functions of θ_n, which is a function of time. This functional relationship makes c_n and s_n implicit functions of time, and the chain rule must be utilized, e.g., $\frac{dc_n}{dt} = \frac{dc_n}{d\theta_n}\frac{d\theta_n}{dt}$. Therefore,

$$\dot{c}_2 = -\dot{\theta}_2 s_2 \qquad \dot{s}_2 = \dot{\theta}_2 c_2$$
$$\dot{c}_3 = -\dot{\theta}_3 s_3 \qquad \dot{s}_3 = \dot{\theta}_3 c_3$$

Crank-slider Velocity Equations

Horizontal Components:

$$\begin{aligned} \frac{d}{dt}\left(r_1 + r_2c_2 = r_3c_3\right) \\ \dot{r}_1 + \dot{r}_2c_2 + r_2\dot{c}_2 = \dot{r}_3c_3 + r_3\dot{c}_3 \\ 0 + 0 - r_2s_2\dot{\theta}_2 = c_3\dot{r}_3 - r_3s_3\dot{\theta}_3 \\ c_3\dot{r}_3 - r_3s_3\dot{\theta}_3 = -r_2s_2\dot{\theta}_2 \end{aligned} \tag{E.5}$$

Vertical Components:

$$\begin{aligned} \frac{d}{dt}\left(r_2s_2 = r_3s_3\right) \\ \dot{r}_2s_2 + r_2\dot{s}_2 = \dot{r}_3s_3 + r_3\dot{s}_3 \\ 0 + r_2c_2\dot{\theta}_2 = s_3\dot{r}_3 + r_3c_3\dot{\theta}_3 \\ s_3\dot{r}_3 + r_3c_3\dot{\theta}_3 = r_2c_2\dot{\theta}_2 \end{aligned} \tag{E.6}$$

Matrix Form:

$$\begin{matrix} & A & & \dot{\boldsymbol{x}} & & \boldsymbol{b} & \\ \begin{bmatrix} c_3 & -r_3s_3 \\ s_3 & r_3c_3 \end{bmatrix} & & \begin{bmatrix} \dot{r}_3 \\ \dot{\theta}_3 \end{bmatrix} & = & \begin{bmatrix} -r_2s_2\dot{\theta}_2 \\ r_2c_2\dot{\theta}_2 \end{bmatrix} & \begin{matrix} (From\ Eq.\ E.5) \\ (From\ Eq.\ E.6) \end{matrix} \end{matrix} \tag{E.7}$$

Now Equation E.7 can be solved to find the velocities from the lower order terms and the input.

$$\dot{\boldsymbol{x}} = A(\boldsymbol{x})^{-1}\boldsymbol{b}(\boldsymbol{x}, \boldsymbol{u}) \tag{E.8}$$

Or more simply,

$$\dot{\boldsymbol{x}} = A^{-1}\boldsymbol{b} \tag{E.9}$$

Further differentiation of equations E.5 and E.6 with respect to time results in acceleration equations. The acceleration equations can also be put into a matrix equation of the form $A\ddot{\boldsymbol{x}} = \boldsymbol{b}$ where $A = A(\boldsymbol{x})$ and $\boldsymbol{b} = \boldsymbol{b}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u})$.

Crank-slider Acceleration Equations

Horizontal Components:

$$\begin{aligned}
\frac{d}{dt}\left(c_3\dot{r}_3 - r_3 s_3\dot{\theta}_3 = -r_2 s_2\dot{\theta}_2\right) \\
\dot{c}_3\dot{r}_3 + c_3\ddot{r}_3 - \dot{r}_3 s_3\dot{\theta}_3 - r_3\dot{s}_3\dot{\theta}_3 - r_3 s_3\ddot{\theta}_3 = -\dot{r}_2 s_2\dot{\theta}_2 - r_2\dot{s}_2\dot{\theta}_2 - r_2 s_2\ddot{\theta}_2 \\
-s_3\dot{r}_3\dot{\theta}_3 + c_3\ddot{r}_3 - s_3\dot{r}_3\dot{\theta}_3 - r_3 c_3\dot{\theta}_3^2 - r_3 s_3\ddot{\theta}_3 = -0 - r_2 c_2\dot{\theta}_2^2 - 0 \\
c_3\ddot{r}_3 - r_3 s_3\ddot{\theta}_3 = -r_2 c_2\dot{\theta}_2^2 + 2s_3\dot{r}_3\dot{\theta}_3 + r_3 c_3\dot{\theta}_3^2
\end{aligned} \tag{E.10}$$

Vertical Components:

$$\begin{aligned}
\frac{d}{dt}\left(s_3\dot{r}_3 + r_3 c_3\dot{\theta}_3 = r_2 c_2\dot{\theta}_2\right) \\
\dot{s}_3\dot{r}_3 + s_3\ddot{r}_3 + \dot{r}_3 c_3\dot{\theta}_3 + r_3\dot{c}_3\dot{\theta}_3 + r_3 c_3\ddot{\theta}_3 = \dot{r}_2 c_2\dot{\theta}_2 + r_2\dot{c}_2\dot{\theta}_2 + r_2 c_2\ddot{\theta}_2 \\
c_3\dot{r}_3\dot{\theta}_3 + s_3\ddot{r}_3 + c_3\dot{r}_3\dot{\theta}_3 - r_3 s_3\dot{\theta}_3^2 + r_3 c_3\ddot{\theta}_3 = 0 - r_2 s_2\dot{\theta}_2^2 + 0 \\
s_3\ddot{r}_3 + r_3 c_3\ddot{\theta}_3 = -r_2 s_2\dot{\theta}_2^2 - 2c_3\dot{r}_3\dot{\theta}_3 + r_3 s_3\dot{\theta}_3^2
\end{aligned} \tag{E.11}$$

Matrix Form:

$$\begin{matrix} & A & & \ddot{\boldsymbol{x}} & & \boldsymbol{b} & \\ \begin{bmatrix} c_3 & -r_3 s_3 \\ s_3 & r_3 c_3 \end{bmatrix} & & \begin{bmatrix} \ddot{r}_3 \\ \ddot{\theta}_3 \end{bmatrix} & = & \begin{bmatrix} -r_2 c_2\dot{\theta}_2^2 + 2s_3\dot{r}_3\dot{\theta}_3 + r_3 c_3\dot{\theta}_3^2 \\ -r_2 s_2\dot{\theta}_2^2 - 2c_3\dot{r}_3\dot{\theta}_3 + r_3 s_3\dot{\theta}_3^2 \end{bmatrix} & \begin{matrix} (From\ Eq.\ E.10) \\ (From\ Eq.\ E.11) \end{matrix} \end{matrix} \tag{E.12}$$

Note: While the velocity equation, $A\dot{\boldsymbol{x}} = \boldsymbol{b}$, and the acceleration equation, $A\ddot{\boldsymbol{x}} = \boldsymbol{b}$, have the same form, the A and $\boldsymbol{b}$ matrices *are not* generally the same for both the velocity and acceleration equations.

Now that the acceleration equations have been represented as a single matrix equation, it is possible to find the acceleration as a function of the lower order terms, $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$, and the input $\boldsymbol{u}$.

$$\ddot{\boldsymbol{x}} = A^{-1}\boldsymbol{b} \tag{E.13}$$

The accelerations in $\ddot{\boldsymbol{x}}$ can be integrated to find the lower order terms, $\dot{\boldsymbol{x}}$ and $\boldsymbol{x}$, in a simulation or can be used to extend the analysis to include mechanism kinetics in addition to kinematics.

E.3 Initial Conditions

It can be especially challenging to select initial conditions for a mechanism like the crank-slider because the state variables are constrained; that is, the variables need to be consistent with each other for the equation to solve properly.

E.3.1 Initial Positions

For the crank-slider example, the initial values of r_3 and θ_3 must be consistent with the input θ_2 such that both the horizontal and vertical vector loop equations are valid. In other words, Equations E.3 and E.4 must both be satisfied by the selected initial conditions. If the initial values for r_3 and θ_3 are not consistent with the selected θ_2 then there will be a residual "gap" in the mechanism.

A simple procedure can be used to select valid initial positions. Consider Figure E.4 which depicts the crank-slider mechanism with improper initial conditions. Point B and B* should be coincident, but because of the incorrect length r_3 and angle θ_3 there is a gap $\boldsymbol{e}$.

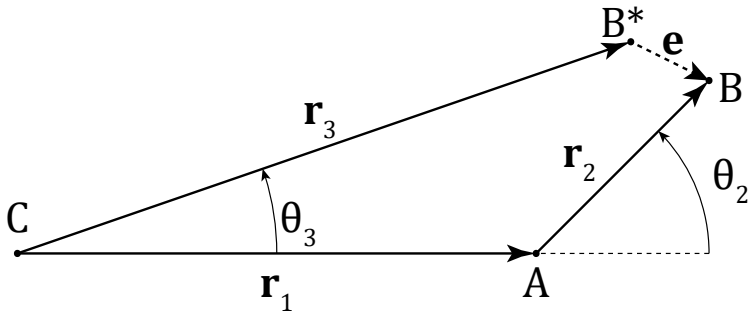


Figure E.4: When the initial conditions are not consistent a gap will be present in the vector-loop.

Equations E.3 and E.4 are valid only if the vector-loop is a valid closed loop, which will only be the case if the angle and length of $\boldsymbol{r}_3$ are consistent with the crank angle. The equations can be adjusted to include extra terms, e_x and e_y, representing the gap in the mechanism. Valid initial conditions are found when the vector magnitude of the gap is zero.

Crank-slider Position Equations With Gap

Horizontal Components:

$$r_1 + r_2 c_2 = r_3 c_3 + e_x$$
$$e_x = r_1 + r_2 c_2 - r_3 c_3 \tag{E.14}$$

Vertical Components:

$$r_2 s_2 = r_3 s_3 + e_y$$
$$e_y = r_2 s_2 - r_3 s_3 \tag{E.15}$$

Magnitude:

$$e = \sqrt{e_x^2 + e_y^2}$$
$$e = \sqrt{(r_1 + r_2 c_2 - r_3 c_3)^2 + (r_2 s_2 - r_3 s_3)^2} \tag{E.16}$$

The following procedure can be used to select valid initial conditions by minimizing the magnitude of the gap, e.

1. Represent the horizontal and vertical gap in terms of the variables in the state vector and the input vector. For the crank slider, that is r_3, θ_3, and θ_2.
2. Represent the residual gap as a scalar by combining the horizontal and vertical components of the gap.
3. Select an initial value for the input. For the crank slider, select the crank angle[5], $\theta_{2,i}$.
4. Estimate initial conditions that result in a small starting gap given the selected initial input. For the crank-slider, guess $r_{3,g}$ and $\theta_{3,g}$ based on the selected value for $\theta_{2,i}$.
5. Attempt to reduce the residual gap to zero by minimization. For the crank-slider, minimize e starting with $r_3 = r_{3,g}$ and $\theta_3 = \theta_{3,g}$. The values of $r_{3,i}$ and $\theta_{3,i}$ that minimize e will be valid initial conditions.

Many different algorithms can be used to minimize the residual function described in the steps above but most work by minimizing a scalar output, like e, based off of a vector input, like $\boldsymbol{x}$. One such method is called the simplex method and is implemented in MATLAB® by the function `fminsearch`. Consider the contour plot shown in Figure E.5 which shows the value e as a function of r_3 and θ_3.

[5]It is necessary to ensure that valid initial conditions exist for a selected input. In the case of the crank-slider it is not an issue because every angle of θ_2 results in a valid configuration. Some mechanisms do not have complete range of motion so the initial input must be selected carefully.

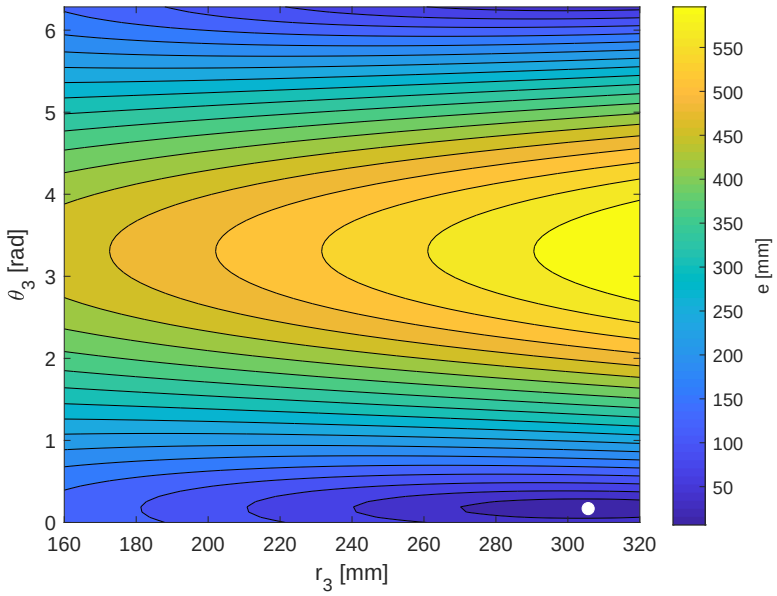


Figure E.5: The gap size e is minimum when r_3 and θ_3 form valid initial conditions shown by the white dot on the plot.

E.3.2 Initial Velocities

The initial velocities can be calculated explicitly once the initial positions are known. The initial velocities can be found by solving Equation E.7 using the initial positions computed by minimization.

E.4 Adding kinetics

Once the initial vector loop has been used to develop acceleration equations in matrix form, the system can be augmented to also compute reaction forces and moments. Consider Figures E.6 and E.7 which depict free-body and kinetic diagrams for links 2 and 3 respectively. Summation of forces and moments for each link result in a new set of equations coupled to the acceleration equations found in the kinematic anlaysis.

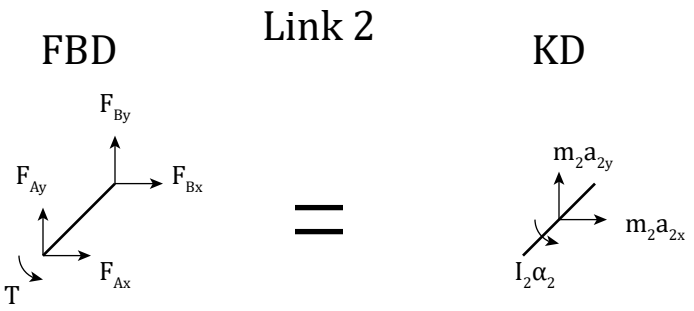


Figure E.6: Free-body and kinetic diagrams for link 2. The crank is driven by an input torque T.

Link 2 Kinetics:

$\Sigma F(\rightarrow)$:

$$F_{Ax} + F_{Bx} = m_2 a_{x2}$$
$$F_{Ax} + F_{Bx} = -\frac{1}{2} m_2 r_2 c_2 \dot{\theta_2}^2 \tag{E.17}$$

$\Sigma F(\uparrow)$:

$$F_{Ay} + F_{By} = m_2 a_{y2}$$
$$F_{Ay} + F_{By} = -\frac{1}{2} m_2 r_2 s_2 \dot{\theta_2}^2 \tag{E.18}$$

$\Sigma M_g(\circlearrowleft)$:

$$T + \frac{1}{2} r_2 s_2 F_{Ax} - \frac{1}{2} r_2 s_2 F_{Bx} - \frac{1}{2} r_2 c_2 F_{Ay} + \frac{1}{2} r_2 c_2 F_{By} = I_2 \alpha_2$$
$$T + \frac{1}{2} r_2 s_2 F_{Ax} - \frac{1}{2} r_2 s_2 F_{Bx} - \frac{1}{2} r_2 c_2 F_{Ay} + \frac{1}{2} r_2 c_2 F_{By} = 0 \tag{E.19}$$

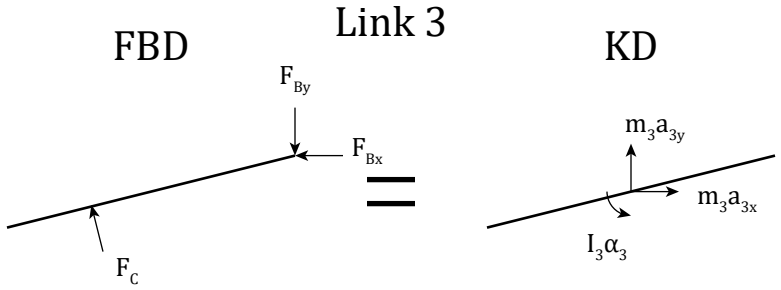


Figure E.7: Free-body and kinetic diagrams for link 3.

Link 3 Kinetics:

$\Sigma F(\rightarrow)$:

$$\begin{aligned} -s_3F_c - F_{Bx} &= m_3a_{x3} \\ -s_3F_c - F_{Bx} &= m_3\Big(c_3\ddot{r}_3 - 2s_3\dot{r}_3\dot{\theta}_3 \\ &\quad - (r_3 - \frac{1}{2}L_3)c_3\dot{\theta}_3^2 - (r_3 - \frac{1}{2}L_3)s_3\ddot{\theta}_3\Big) \end{aligned}$$

$$-s_3F_c - F_{Bx} - m_3c_3\ddot{r}_3 + m_3(r_3 - \frac{1}{2}L_3)s_3\ddot{\theta}_3 = -2m_3s_3\dot{r}_3\dot{\theta}_3 - m_3(r_3 - \frac{1}{2}L_3)c_3\dot{\theta}_3^2 \quad \text{(E.20)}$$

$\Sigma F(\uparrow)$:

$$\begin{aligned} c_3F_c - F_{By} &= m_3a_{y3} \\ c_3F_c - F_{By} &= m_3\Big(s_3\ddot{r}_3 + 2c_3\dot{r}_3\dot{\theta}_3 \\ &\quad - (r_3 - \frac{1}{2}L_3)s_3\dot{\theta}_3^2 + (r_3 - \frac{1}{2}L_3)c_3\ddot{\theta}_3\Big) \end{aligned}$$

$$c_3F_c - F_{By} - m_3s_3\ddot{r}_3 - m_3(r_3 - \frac{1}{2}L_3)c_3\ddot{\theta}_3 = 2m_3c_3\dot{r}_3\dot{\theta}_3 - m_3(r_3 - \frac{1}{2}L_3)s_3\dot{\theta}_3^2 \quad \text{(E.21)}$$

$\Sigma M_g(\circlearrowleft)$:

$$\begin{aligned} -F_c(r_3 - \frac{1}{2}L_3) - \frac{1}{2}L_3c_3F_{By} + \frac{1}{2}L_3s_3F_{Bx} &= I_3\alpha_3 \\ -F_c(r_3 - \frac{1}{2}L_3) - \frac{1}{2}L_3c_3F_{By} + \frac{1}{2}L_3s_3F_{Bx} - I_3\ddot{\theta}_3 &= 0 \end{aligned} \quad \text{(E.22)}$$

Equations E.17 through E.22 can also be packed into a matrix equation. First a new output vector, $\boldsymbol{y}$, can be composed from all of the unknown forces as

$$\boldsymbol{y} = \begin{bmatrix} F_{Ax} \\ F_{Ay} \\ F_{Bx} \\ F_{By} \\ F_C \\ T \end{bmatrix}$$

A new, quite large, matrix can be composed that will solve the kinetics and kinematics simultaneously.

$$\begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{x}} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix} \tag{E.23}$$

Where A_{11} and $\boldsymbol{b}_1$ represent the A matrix and $\boldsymbol{b}$ vector for the kinematics only and A_{21}, A_{22}, and $\boldsymbol{b}_2$ solve the newly added kinetics. A_{21}, A_{22}, and $\boldsymbol{b}_2$ can be determined from Equations E.17 through E.22.

$$A_{21} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -m_3 c_3 & m_3(r_3 - \frac{1}{2}L_3)s_3 \\ -m_3 s_3 & -m_3(r_3 - \frac{1}{2}L_3)c_3 \\ 0 & -I_3 \end{bmatrix}$$

$$A_{22} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ \frac{1}{2}r_2 s_2 & -\frac{1}{2}r_2 c_2 & -\frac{1}{2}r_2 s_2 & \frac{1}{2}r_2 c_2 & 0 & 1 \\ 0 & 0 & -1 & 0 & -s_3 & 0 \\ 0 & 0 & 0 & -1 & c_3 & 0 \\ 0 & 0 & \frac{1}{2}L_3 s_3 & -\frac{1}{2}L_3 c_3 & -(r_3 - \frac{1}{2}L_3) & 0 \end{bmatrix}$$

$$\boldsymbol{b}_2 = \begin{bmatrix} -\frac{1}{2}m_2 r_2 c_2 \dot{\theta}^2 \\ -\frac{1}{2}m_2 r_2 s_2 \dot{\theta}^2 \\ 0 \\ -2m_3 s_3 \dot{r}_3 \dot{\theta}_3 - m_3(r_3 - \frac{1}{2}L_3)c_3 \dot{\theta}_3^2 \\ 2m_3 c_3 \dot{r}_3 \dot{\theta}_3 - m_3(r_3 - \frac{1}{2}L_3)s_3 \dot{\theta}_3^2 \\ 0 \end{bmatrix}$$

The equation $A_{21}\ddot{\boldsymbol{x}} + A_{22}\boldsymbol{y} = \boldsymbol{b}_2$ is too large to display in its fully expanded form. The output forces, $\boldsymbol{y}$, can be solved for by inverting the square matrix A_{22}.

$$\boldsymbol{y} = A_{22}^{-1}\left(\boldsymbol{b}_2 - A_{21}\ddot{\boldsymbol{x}}\right) \tag{E.24}$$

E.5 Simulating the mechanism

The kinematics and kinetics of the mechanism can be simulated in a variety of ways. One effective method is to use a Simulink® model to route the information and perform the numerical integration but to use a MATLAB® function to implement the matrix operations necessary for the equations derived above. Figures E.8 and E.9 show to possible implementations in Simulink®, one for kinematics only and one for both kinematics and kinetics.

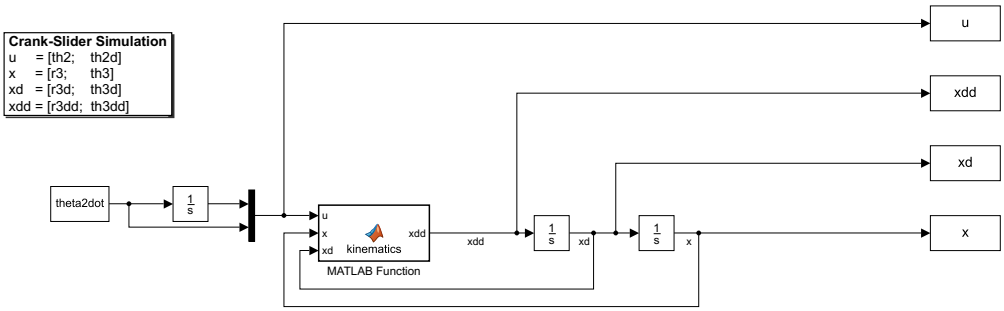


Figure E.8: This model shows a method for simulating only the kinematics of the crank-slider mechanism. The MATLAB® function block is the core of the Simulink® diagram.

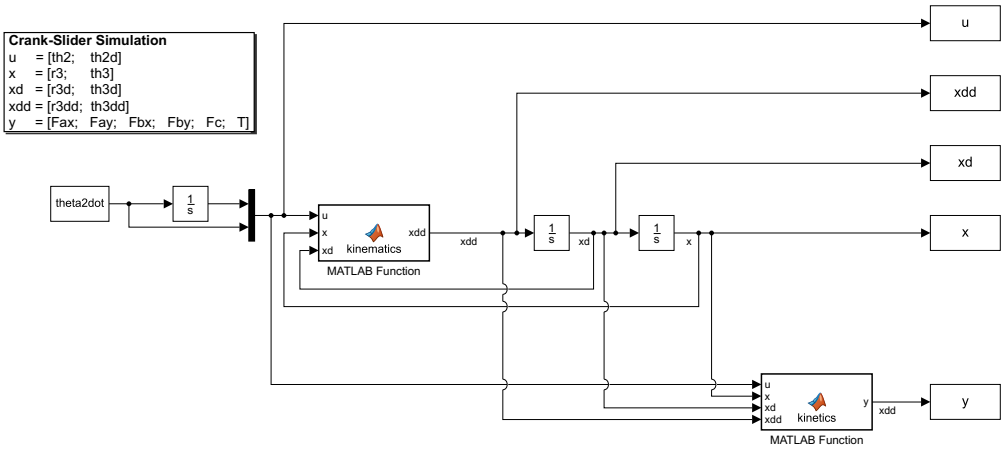


Figure E.9: This model shows a method for simulating both the kinematics and kinetics of the crank-slider mechanism. Here two MATLAB® function blocks make up the core of the Simulink® diagram.

E.6 Visualizing the mechanism

The code below can be used to generate an animation of the crank-slider mechanism using simulation output data.

AnimationTemplate.m

```
%% Crank Slider Geometry
r1 = 240;                                % Length of vector r1 (ground)
r2 = 80;                                 % Length of vector r2 (crank)
L  = 350;                                % Length of slider

%% Input
th2_i  = 0;                              % Initial crank angle
th2d_i = 0;                              % Initial crank angle

%% Simulation
% The code in this section may eventually be replaced by simulation data.
u = [th2_i th2d_i];                      % input vector
x = [r1+r2 0];                           % state vector

%% Animation
N = size(x,1);                           % Number of rows of data

c2 = cos(u(:,1));                        % cos(theta2)
s2 = sin(u(:,1));                        % sin(theta2)

c3 = cos(x(:,2));                        % cos(theta3)
s3 = sin(x(:,2));                        % sin(theta3)

% X and Y components for each point on the crank-slider mechanism.
%           Pt. C       Pt. A           Pt. B       Pt. D
x_coord = [ zeros(N,1)  r1*ones(N,1)    r1+r2*c2    r1+r2*c2-L*c3 ];
y_coord = [ zeros(N,1)  zeros(N,1)         r2*s2       r2*s2-L*s3 ];

% Loop through the frames and plot them all.
for n = 1:N
    figure(1);                           % Make sure the animation stays put
    plot(x_coord(n,:),y_coord(n,:));     % Plot the whole mechanism at once
    axis('equal');                       % Makes x- and y- scalle equal
    drawnow('limitrate');                % Forces MATLAB to update plot
end
```

If desired, the animation can be saved as a GIF using the `imwrite` command.