ME C231 Assignment: Modeling and Simulation

1. Car Engine Model (the reference for this problem is Cho and Hedrick, "Automotive Powertrain Modelling for Control," *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 111, No. 4, December 1989): In this problem we consider a 1-state model for an automotive engine, with the transmission engaged in 4th gear. The engine state is m_a, the mass of air (in kilograms) in the intake manifold. The state of the drivetrain is the angular velocity, ω_e, of the engine. The input is the throttle angle, α, (in radians). The equations for the engine is

$$\begin{aligned} \dot{m}_a(t) &= c_1 T(\alpha(t)) - c_2 \omega_e(t) m_a(t) \\ T_e &= c_3 m_a(t) \end{aligned}$$

where we treat ω_e and α as inputs, and T_e as an output. The drivetrain is modeled

$$\dot{\omega}_e(t) = \frac{1}{J_e}\left[T_e(t) - T_f(t) - T_d(t) - T_r(t)\right]$$

The meanings of the terms are as follows:

- $T(\alpha)$ is a throttle flow characteristic depending on throttle angle,

$$T(\alpha) = \begin{cases} 0.00032 & \text{for} \quad \alpha < 0 \\ 1 - \cos(1.14\alpha - 0.0252) & \text{for} \quad 0 \le \alpha \le 1.4 \\ 1 & \text{for} \quad \alpha > 1.4 \end{cases}$$

- T_e is the torque from engine on driveshaft, $c_3 = 47500$ $\texttt{Nm/kg}$.
- T_f is engine friction torque ($\texttt{Nm}$), $T_f = 0.106\omega_e + 15.1$
- T_d is torque due to wind drag ($\texttt{Nm}$), $T_d = c_4\omega_e^2$, with $c_4 = 0.0026$ $\texttt{Nms}^2$
- T_r is torque due to rolling resistance at wheels ($\texttt{Nm}$), $T_r = 21.5$.
- J_e is the effective moment of inertia of the engine, transmission, wheels, car, $J_e = 36.4\texttt{kgm}^2$
- $c_1 = 0.6\texttt{kg/s}$, $c_2 = 0.095$
- In 4th gear, the speed of car, v ($\texttt{m/s}$), is related to ω_e as $v = 0.129\omega_e$.

(a) Plot the throttle flow characteristic $T(\alpha)$ for $0 \le \alpha \le 1.4$

(b) Combine the governing equations into state variable form.

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)) \\ y(t) &= h(x(t)) \end{aligned}$$

where $x_1 = m_a$, $x_2 = \omega_e$, $u = \alpha$ and $y = v$.

(c) For the later purpose of calculating the Jacobian linearization, explicitly write out the function $f(x, u)$. Note that f maps 3 real numbers (x_1, x_2, u) into 2 real numbers. There is **no** explicit time dependence in this particular f. Write a function called `hcModel` with function declaration line begin code

```
function xDot = hcModel(x, u, c1, c2 ,c3, c4)
```

end code

The input arguments are

- x, 2-by-1 array, state x
- u, scalar, input u
- c1, scalar-valued constant c_1
- c2, scalar-valued constant c_2
- c3, scalar-valued constant c_3
- c4, scalar-valued constant c_4
- c1, scalar-valued constant c_1

The output argument is a 2-by-1 array which equals the derivative $\dot{x}$, as given by the governing equations.

(d) Let $\bar{v} > 0$ denote an equilibium car speed. Find expressions for the corresponding equilibrium values $\bar{m}_a$, $\bar{\omega}_e$ and $\bar{\alpha}$. The expressions should all be functions of $\bar{v}$. Write a function `hcEqPt`, with function declaration line

begin code

```
function [maBar, wBar, alphaBar] = hcEqPt(vBar, c1, c2, c3, c4)
```

end code

which computes the equilibrium values of m_a, ω, α associated with a given equilibrium car speed, with values specified for the 4 parameters.

(e) Based on the throttle flow characteristic, T, what is the maximum equilibrium speed of the car?

(f) Use the `hcEqPt.m` function to compute the equilibrium values of $\bar{m}_a$, $\bar{\omega}_e$ and $\bar{\alpha}$ so that the car travels at a constant speed of $\bar{v} = 22$ m/s. Repeat calculation for an equilibrium speed of $\bar{v} =$ 32 m/s. Make sure the results make sense, and the relationship among the variables is as you expect.

(g) Write an mfile, called `constantThrottleSim.m` which uses ode45 to simulate the response of the car/engine model **starting from the initial condition** $\omega_e(0) = \bar{\omega}_e, m_a(0) = \bar{m}_a$, with a constant input of

$$\alpha(t) = \bar{\alpha} + \beta$$

obtaining the response for v and m_a, where $\bar{v}$ is given, and defines the other values $(\bar{m}_a, \bar{\omega}_e, \bar{\alpha})$. Run the code for 6 values of β, $\beta = \pm 0.01, \pm 0.04, \pm 0.1$. Do this for two cases, $\bar{v} = 22$ and $\bar{v} = 32$. Plot the results in a concise fashion, and use the `legend` command to clearly label the various curves.

The layout for the function is

begin code

```
function [tSol, maSol, vSol] = constantThrottleSim(vBar, Beta, ...
    c1, c2, c3, c4, TFinal)
% Compute maBar, wBar and alphaBar from vBar
% Define uVal = alphaBar + Beta
% Define 2-by-1 initial condition vector, xInit
% Define timespan of simulation, tSpan, from 0 to TFinal
% Create function handle, suitable for ODE45
```

```
odeSimModel = @(t, x) hcModel(x, uVal, c1, c2 ,c3, c4)
% Call ODE45
[tSol, xSol] = ode45(odeSimModel, xInit, tSpan)
% Extract maSol from xSol
% Define vSol from xSol
```
end code

(h) Write an mfile, called `sinusoidThrottleSim.m` which uses ode45 to simulate the response of the car/engine model **starting from the initial condition** $\omega_e(0) = \bar{\omega}_e, m_a(0) = \bar{m}_a$, with a constant input of

$$\alpha(t) = \bar{\alpha} + \beta \sin(\omega t)$$

obtaining the response for v and m_a, where $\bar{v}$ is given, and defines the other values $(\bar{m}_a, \bar{\omega}_e, \bar{\alpha})$. Run the code for $\omega = 0.5$ and 3 values of β, $\beta = 0.01, 0.04, 0.1$. Do this for two cases, $\bar{v} = 22$ and $\bar{v} = 32$. Plot the results in a concise fashion, and use the `legend` command to clearly label the various curves.

The layout for the function is

begin code
```
function [tSol, maSol, vSol] = sinusoidThrottleSim(vBar, Beta, w, ...
    c1, c2, c3, c4, TFinal)
% Compute maBar, wBar and alphaBar from vBar
% Define uHan = @(t) alphaBar + Beta*sin(w*t)
% Define 2-by-1 initial condition vector, xInit
% Define timespan of simulation, tSpan, from 0 to TFinal
% Create function handle, suitable for ODE45, using uHan
odeSimModel = @(t, x) hcModel(x, uHan(t), c1, c2 ,c3, c4)
% Call ODE45
[tSol, xSol] = ode45(odeSimModel, xInit, tSpan)
% Extract maSol from xSol
% Define vSol from xSol
```
end code

(i) Consider deviations from these equilibrium values,

$$\begin{aligned} \alpha(t) &= \bar{\alpha} + \delta_\alpha(t) \\ \omega_e(t) &= \bar{\omega}_e + \delta_{\omega_e}(t) \\ m_a(t) &= \bar{m}_a + \delta_{m_a}(t) \\ y(t) &= \bar{y} + \delta_y(t) \end{aligned}$$

Find the (linear) differential equations that approximately govern these deviation variables (Jacobian Linearization discussed in class),

$$\begin{aligned} \dot{\delta}_x(t) &= A\delta_x(t) + B\delta_u(t) \\ \delta_y(t) &= C\delta_x(t) + D\delta_u(t) \end{aligned}$$

Your answer should consist of 4 matrices which, in general, depend on $\bar{v}$. Use the notation

$$A(\bar{v}), B(\bar{v}), C(\bar{v}), D(\bar{v})$$

for the 4 matrices. Based on these calculations, write a function `hcLinearModel`, with function declaration line

```
function [A, B, C, D, maBar, wBar, alphaBar] = hcLinearModel(vBar)
```

which computes the matrices of the linearized model (and other equilibrium values) associated with a given equilibrium car speed.

(j) Compute the numerical values of the Jacobian Linearization at two specific cases, namely

(k) Using ode45, compute the response of the Jacobian linearization starting from initial condition $\delta_{\omega_e}(0) = 0$ and $\delta_{m_a}(0) = 0$, with the input $\delta_\alpha(t) = \beta$ for for 6 values of β, $\beta = \pm 0.01, \pm 0.04, \pm 0.1$. Name the function constantThrottleLinearSim, with layout

```
[tSol, d_maSol, d_vSol] = constantThrottleLinearSim(vBar, Beta, ...
       c1, c2, c3, c4, TFinal)
% Compute linearization from vBar
% Define d_uVal = Beta
% Define 2-by-1 initial condition vector, d_xInit
% Define timespan of simulation, tSpan, from 0 to TFinal
% Create function handle, suitable for ODE45, using A,B matrices
% Call ODE45
% Extract d_maSol from state response
% Define d_vSol from state response
```

Add the responses $\delta_v(t)$ and $\delta_{m_a}(t)$ to the equilibrium values $\bar{v}, \bar{m}_a$ and plot these sums, which are approximations of the actual response. Compare the linearized response with the actual response from earlier problem. Comment on the difference between the results of the nonlinear simulation and the results from the linearized analysis.

(l) Using ode45, compute the response of the Jacobian linearization starting from initial condition $\delta_{\omega_e}(0) = 0$ and $\delta_{m_a}(0) = 0$, with the input $\delta_\alpha(t) = \beta \sin \omega t$ for for 3 values of β, $\beta = 0.01, 0.04, 0.1$, with $\omega = 0.5$. Name the function sinusoidalThrottleLinearSim, with layout

```
[tSol, d_maSol, d_vSol] = sinusoidalThrottleLinearSim(vBar, Beta, w, ...
       c1, c2, c3, c4, TFinal)
% Compute linearization from vBar
% Define d_uVal = Beta
% Define 2-by-1 initial condition vector, d_xInit
% Define timespan of simulation, tSpan, from 0 to TFinal
% Create function handle, suitable for ODE45, using A,B matrices
% Call ODE45
% Extract d_maSol from state response
% Define d_vSol from state response
```

Add the responses $\delta_v(t)$ and $\delta_{m_a}(t)$ to the equilibrium values $\bar{v}, \bar{m}_a$ and plot these sums, which are approximations of the actual response. Compare the linearized response with the actual response from earlier problem. Comment on the difference between the results of the nonlinear simulation and the results from the linearized analysis.

2. Consider the discrete-time dynamic system with the following state space representation:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & \alpha & 0 \\ 0 & \frac{1}{2} & -\frac{5}{4} & 0 \\ -\frac{1}{2} & 0 & 0 & \frac{1}{3} \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} + \begin{bmatrix} 0 \\ -2 \\ 4 \\ 0 \end{bmatrix} u(k)$$

$$y(k) = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{3} & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix}$$

Write a Matlab function with function declaration line

begin code
```
function [TF] = isSystemStable(alpha)}
```
end code

to analyze the system's stability as a function of the variable α. The input argument `alpha` is a scalar. The output argument `TF` is a `logical` value (1 represents `true` and 0 represents `false`). If you are unfamilar with `logical` values and operators, examine/run the code below to begin learning about them.

```
T = true
class(T)
F = false
class(F)

TF = 1.7==1.7;
class(TF)
size(TF)

TF = [1.6==1.5  3>4  5<=5    6<9.2]
size(TF)
class(TF)
any(TF)
all(TF)

V = randn(1,5);
V>2
any(V>2)
all(abs(V)<6)
```

3. **Euler Discretization of a Building Heat Transfer Model**
Consider the nonlinear heat transfer dynamics building room temperature with forced air ventilation:

$$m_z c_z \dot{T}(t) = q(t) + c_p u_1(t)(u_2(t) - T(t))$$

where

$$
\begin{aligned}
T(t) &= \text{room temperature} \\
u_1(t) &= \text{air mass flow rate} \\
u_2(t) &= \text{supply air temperature.} \\
q(t) &= \text{ heat load} \\
m_z &= \text{effective room mass (constant)} \\
c_z &= \text{effective room heat capacity (constant)} \\
c_p &= \text{heat capacity of air (constant)}
\end{aligned}
\tag{1}
$$

(a) Write a function `bldgHTM`, with function declaration line

begin code

```
function Tdot = bldgHTM(T, u1, u2, q, mz, cz, cp)
```

end code

that implements this model, returning $\dot{T}(t)$, given the values $T(t), u_1(t), u_2(t), q(t)$ and the values for the 3 fixed parameters. **Hint:** This can be a one-line function.

(b) Use `ode45` and `bldgHTM` to simulate the room temperature under the following conditions

$$T(0) = 25, \quad u_1(t) = 1000{*}(2{+}\sin(0.1t)), \quad u_2(t) = 25{+}\sin(0.2t), \quad q(t) = \begin{matrix} 0, \text{for } t < 50 \\ 1000, \text{for } t \geq 50 \end{matrix}$$

over the interval $t = 0$ to $t = 200$. The room parameters are $c_p = 1$, $m_z = 100$ and $c_z = 20$.

Use function handles to represent these input functions, namely

begin code

```
u1H = @(t) 2 + sin(0.1*t);
u2H = @(t) 25 + sin(0.2*t);
qH = @(t) 1000*(t>50);
```

end code

(c) Next, repeat the simulation using first-order, forward integration. Recall that the first-order, forward Euler solution for an ode $\dot{x}(t) = f(t, x(t), u(t))$ is

$$x_E((k+1)T_S) = x_E(kT_S) + T_S f(kT_S, x_E(kT_S), u(kT_S))$$

for $k = 0, 1, \ldots$. The Matlab implementation requires a little care, since Matlab indexing uses 1 for the first element. If `xEsol` represents the Matlab vector containing the Euler solution, the indices into that need to start with 1 (representing $t = 0$). Conversely, the function handle, u_H for the input function is usually written in continuous-time (t), and accepts 0 as a valid argument for the continuous time, and the same is true for the first argument of f. Taken together, it means that the Euler iteration, with Matlab indexing, looks like

```
xEsol(k+2) = xEsol(k+1) + TS*f( k*TS, xEsol(k+1), uH(k*TS) ),   k=0,1,...
```

Try this for several sample times, $T_S = 0.1, 0.5, 1, 2$, plotting the Euler solution and "exact" solution (from `ode45`) on the same graph. Note the accuracy, or lack thereof, of the discrete approximation.

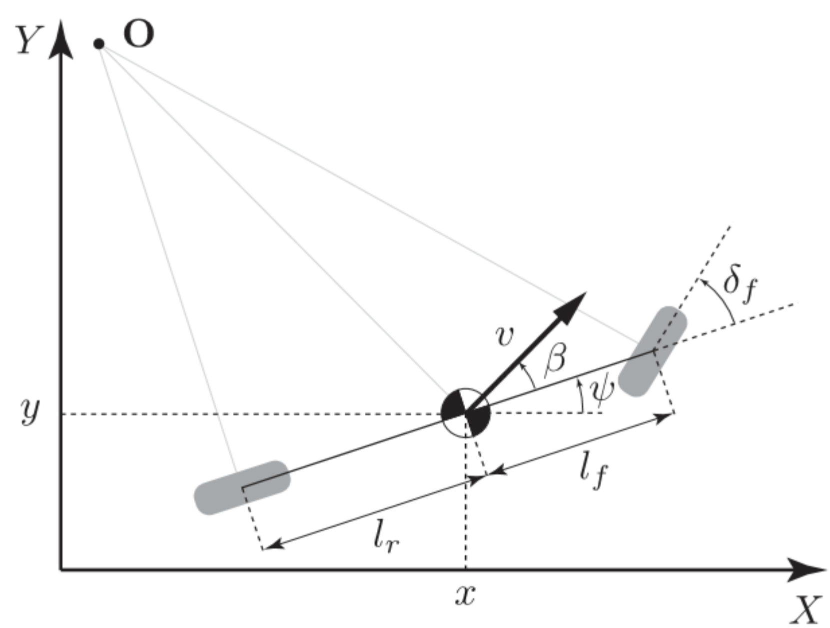
4. Consider a simplified kinematic bicycle model:

$$
\begin{aligned}
\dot{x} &= v\cos(\psi+\beta) \\
\dot{y} &= v\sin(\psi+\beta) \\
\dot{v} &= a \\
\dot{\psi} &= \frac{v}{l_r}\sin(\beta) \\
\beta &= \tan^{-1}\left(\frac{l_r}{l_f+l_r}\tan(\delta_f)\right)
\end{aligned}
$$

where

$x =$ global x coordinate, mass center location
$y =$ global y coordinate, mass center location
$v =$ speed of the vehicle (direction shown)
$\psi =$ global heading angle
$a =$ tangential acceleration of the center of mass (along direction of velocity) (input)
$\delta_f =$ steering angle of the front wheels with respect to the longitudinal axis of the car (input)
$l_r =$ distance from the center of mass of the vehicle to the rear axle
$l_f =$ distance from the center of mass of the vehicle to the front axle
$\beta =$ angle of the current velocity with respect to the longitudinal axis of the car

(3)



For this problem, you will simulate the car's movement given a predetermined input sequence. The states are x, y, v, ψ, and the inputs are a, δ_f. Use $l_f = l_r = 1.738$.

(a) Find the one-step Forward Euler discretization of the kinematic model using a sampling time T_S. Specifically, write a Matlab function:

```
function [xp, yp, vp, psip] = bikeFE(x, y, v, psi, a, deltaF, TS)
```

that takes as its inputs the current state, the current control action, the sample time, and outputs the state at the next time step using the first-order, forward Euler, discrete model you derive.

(b) Write a Matlab function:

```
function [xE, yE, vE, psiE] = bikeFEsim(aSeq, deltaFSeq, initState, TS)
```

which simulates the system states starting at the initial state `initState`, given a sequence of acceleration (a) and steering angle (`deltaF`) inputs, with sample time `TS`. The inputs `aSeq` and `deltaFSeq` will be vectors of length `N`, and the outputs should be vectors on length `N+1`. The ordering in `initState` should be (x, y, v, ψ).

(c) Write a Matlab script `simAndAnimate.m` to test your code by downloading the file `bikeInputData.mat`. This file contains three vectors: `time`, and the corresponding inputs at each time instant: `a` (acceleration) and `deltaF` (δ_f). Verify that the time instants are uniformly spaced, and convert that into a fixed sampling time T_S. Simulate the bike using the forward, first-order Euler. Plot the results, and compare to the ode solution (from Lab). Make appropriate modifications to `movingLineMovieDemo.m` to create an animation of the bicycle's trajectory in space. It would be nice to have the bicycle body, as well as the steering angle (ie., two lines) being shown.