AE353: Design Problem 02

Nico Alba

March 10, 2017

1 Goal

The code from `DesignProblem02` simulates an under-actuated robot. In this case, it means that there are less actuators than there are degrees of freedom. There are 2 joints (2 degrees of freedom) and only 1 actuator. The motor will apply a torque about the first motor, but not the second, which spins freely. My goal will be to design a controller that can make the tip of the robot arm move to a sequence of points in a 3D space. As described in the example goal, I have to make the second joint angle track a piecewise-constant reference trajectory.

`Requirements:` The second joint angle will reference track to within ± 0.0873 Radians of 0.3 Radians from a the equilibrium point $\vec{x} = 0$ I will also include the ability to reject disturbance.

`Verification` The MATLAB function `DesignProblem02('Controller','datafile','data.mat')` will be be used to simulate the system. The data generated will be imported as `'data.mat'` and analyzed. The error at each time step will be found by taking the difference between the second joint angle and the reference value 0.3 Radians. If the maximum error after an 20 seconds is continuously less than 0.0873 Radians then we will know if we have succeeded.

2 Model

The motion of the robot is governed by an ODE of the form:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + N(q, \dot{q}) = \tau, \tag{1}$$

Matrix of joint angles:

$$q = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

Velocity Matrix:

$$\dot{q} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Matrix of applied torques:

$$\tau = \begin{bmatrix} \tau_1 \\ 0 \end{bmatrix}$$

$$M(q) \qquad C(q, \dot{q}) \qquad N(q, \dot{q})$$

are matrix-valued functions of q and/or $\dot{q}$. These functions depend on parameters like mass and moment of inertia. To load these parameters:

```
% Load the equations of motion.
load('DesignProblem02_EOMs.mat');
% Parse the equations of motion.
M = symEOM.M;
C = symEOM.C;
N = symEOM.N;
tau = symEOM.tau;
% Define symbolic variables that appear in the equations of motion.
syms q1 q2 v1 v2 tau1 real
% Next, we must linearize the system
```

Figure 1: Example code to get a symbolic description of the equations of motion in MATLAB.

In order to linearize the system, we must first solve for $\ddot{q}$

$$\ddot{q} = \frac{\tau - C(q, \dot{q})\dot{q} - N(q, \dot{q})}{M(q)} \tag{2}$$

Take the Jacobian of $\ddot{q}$ and q and this is your$A$ matrix for angular accelerations: $\ddot{q}$ Then use the MATLAB command $simplify$ to find A. For B, you just divide τ by $M(q)$.

```
z=[q1;q2;v1;v2];
qdot=[v1;v2];
T=[tau1;0];
qddot=M\(-C*qdot-N+T);
A=[0 0 1 0; 0 0 0 1;simplify(jacobian(qddot,x))];
B=[0;0;jacobian(qddot,tau1)];
```

Figure 2: Linearize commands in MATLAB

If we initialize $\vec{x} = 0$ for our reference values, we get

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 4.97 & -54.77 & -3.28 & -0.63 \\ -2.18 & -10.50 & -0.63 & -0.31 \end{bmatrix}$$

and

$$B = \begin{bmatrix} 0 \\ 0 \\ 3.28 \\ 0.63 \end{bmatrix}$$

Next, in order to reference track, we must choose a C and D matrix. Because we only care about tracking one value (q_2), with:

$$\vec{x} = \begin{bmatrix} q_1 \\ q_2 \\ v_1 \\ v_2 \end{bmatrix}$$

we set,

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}, D = \begin{bmatrix} 0 \end{bmatrix}$$

From this, we can find that

$$y(t) = q_2$$

This gives us an open-looped system with $\dot{x} = Ax + B(0)$ and $y = Cx$.

Check for stability and controllability: since, the eigenvalues of A are do not all have negative real parts, then the system is unstable.

$$eig(A) = \begin{bmatrix} 2.76 \\ -4.24 \\ -1.06 + 3.7i \\ -1.06 - 3.7i \end{bmatrix}$$

and using $ctrb(A, B)$, we find that the controllability matrix, W, ans is full rank (4), meaning it is controllable.

$$W = \begin{bmatrix} 0 & 3.28 & -11.18 & 20.0 \\ 0 & 0.63 & -2.26 & -6.04 \\ 3.28 & -11.18 & 20.0 & 6.49 \\ 0.63 & -2.26 & -6.04 & 37.42 \end{bmatrix}$$

Find K and kRef: This will allow us to not only make our open loop system stable, but also allow us to track a reference. Using lqr and setting $Q = eye(size(A))$,

$$K = lqr(A, B, Q, 0.01) = \begin{bmatrix} 47.62 & -36.12 & 14.93 & -23.80 \end{bmatrix}$$

and

$$kRef = 1/(-C(A - BK)^{-1}B) = -19.45$$

And to check, this value of K now gives us a stable loop!

$$eig(A - BK) = \begin{bmatrix} -33.42 \\ -1.31 + 1.43i \\ -1.31 - 1.43i \\ -1.60 \end{bmatrix}$$

3 Design a Controller

In order to get q_2 to stabalize at 0, we must include an input so that

$$\dot{x} = Ax + Bu \tag{3}$$

such that

$$u = -Kx \tag{4}$$

So what if we want to include reference tracking? then

$$u = -Kx + (kRef)r \tag{5}$$

such that r is the value we wish q_2 to reach. And we just found $kRef$!

What about disturbance rejection?

$$u = -Kx + (kRef)r + (kInt)v(t) \tag{6}$$

where $kInt$ is any number we choose and

$$v = \int_0^t (y(\tau) - r(\tau))d\tau \tag{7}$$

Because $v(t)$ is an integral, $v_{t+1} = v_t + (y_t - r_t)h$ where h is the time-step.

We initialize the controller with:

```
function [actuators,data] = initControlSystem(sensors,references,parameters,data)

data.K=[47.6161  −36.1224   14.9294  −23.8025];
data.kRef= −19.4454;
data.r= 0.3;
data.kInt=5;
data.v= 0;

[actuators,data] = runControlSystem(sensors,references,parameters,data);
end
```

Figure 3: Initialization

Now we specify what we want our input to be. In this case it's τ_1, the only motor we can control.

```
function [actuators,data] = runControlSystem(sensors,references,parameters,data)

data.x=[sensors.q1; sensors.q2; sensors.v1; sensors.v2];
data.v= data.v+(sensors.q2 − data.r)∗.02;

%Input: u= −K∗x +kRef∗r +kInt∗v(t)
actuators.tau1 = −data.K∗data.x +data.kRef∗data.r +data.kInt∗data.v;

end
```

Figure 4: Loop and Input

4 Implement and Test your Controller in Simulation

To implement the controller, we can run `DesignProblem02('Controller','diagnostics',true,'datafile','data.mat','disturbance',true,'reference',@(t)0.3)` After running the controller, we get:

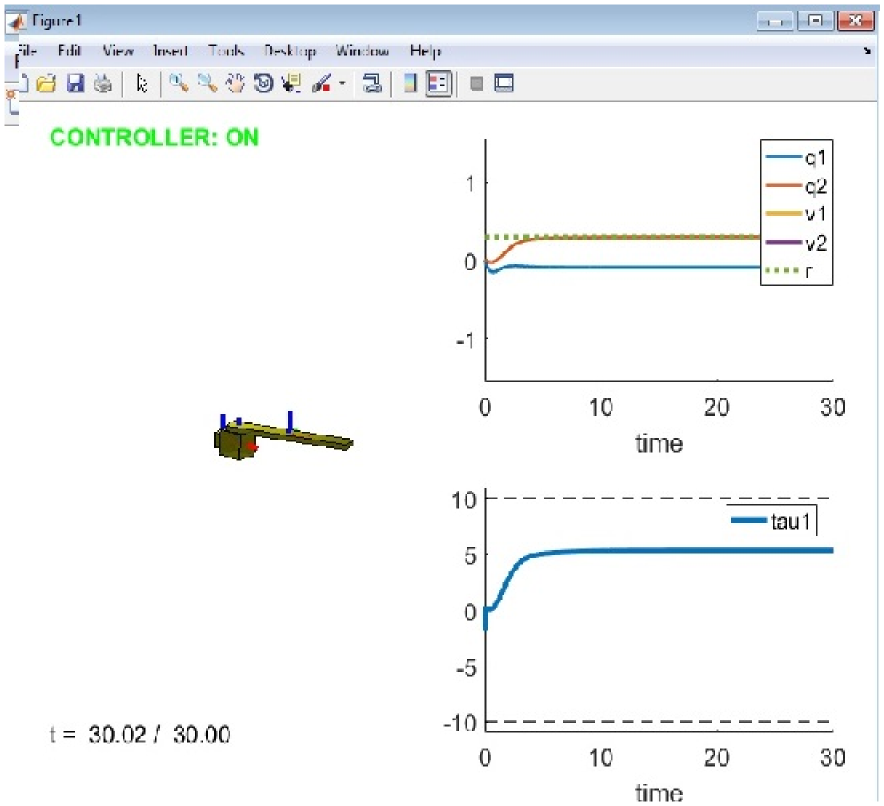

Figure 5: DesignReport02

We can then import the data and analyze q_2 with respect to time with:

```
load('data.mat')
t=processdata.t;
q2=processdata.q2

figure(2)
plot(t,q2,'linewidth',2)
axis([0 30 -.4 .4])
legend('q2')
figure(3)
error=abs(.3-q2)
plot(t,error)
```

Figure 6: Import and Plot

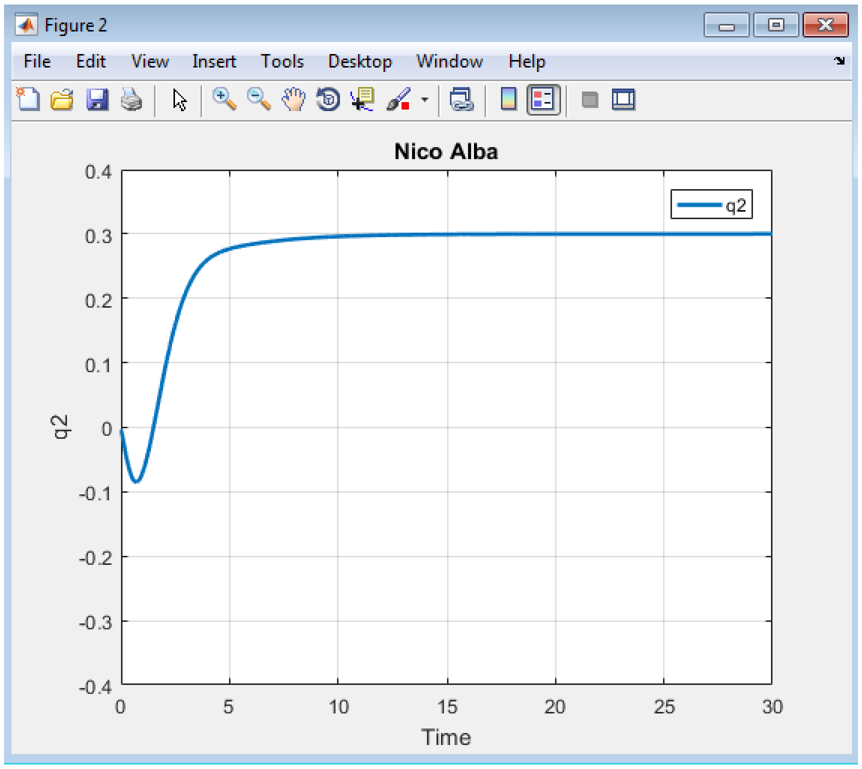

Figure 7: q_2 v.s. time

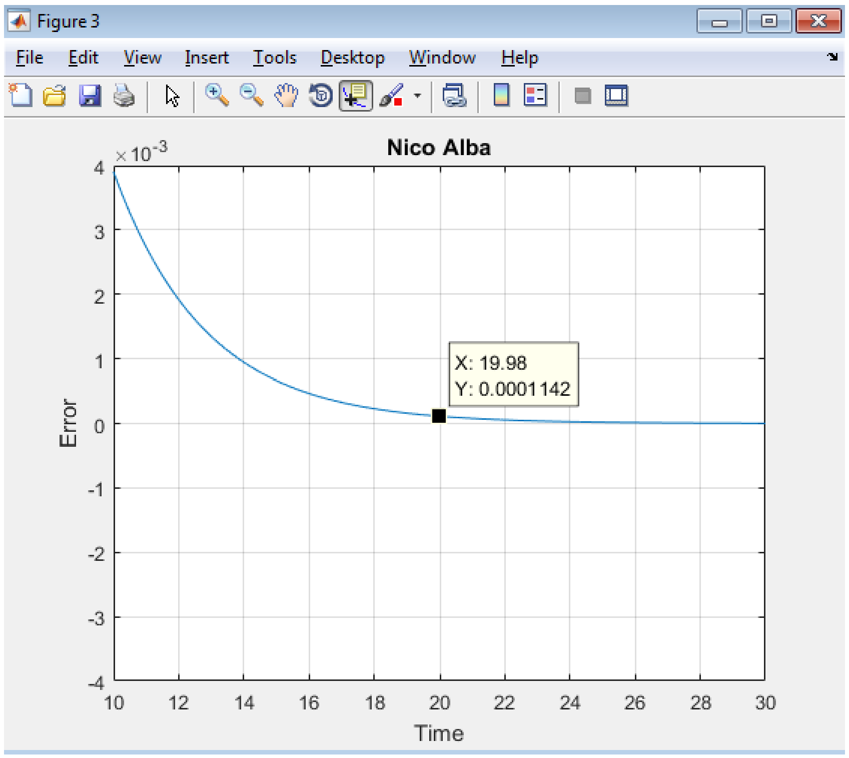

Figure 8: Error v.s.time

SUCCESS!

Now, let's attempt the same thing but implementing a piece-wise function and return to the reference value we set before. Then our controller is now:

```
function [actuators,data] = runControlSystem(sensors,references,parameters,data)

if sensors.t<=15
    data.r=.3;
else
    data.r=0;
end

data.x=[sensors.q1; sensors.q2; sensors.v1; sensors.v2];
data.v= data.v+(sensors.q2 − data.r)∗.02;

%Input: u= −K∗x +kRef∗r +kInt∗v(t)
actuators.tau1 = −data.K∗data.x +data.kRef∗data.r +data.kInt∗data.v;

end
```

Figure 9: Piecewise Input

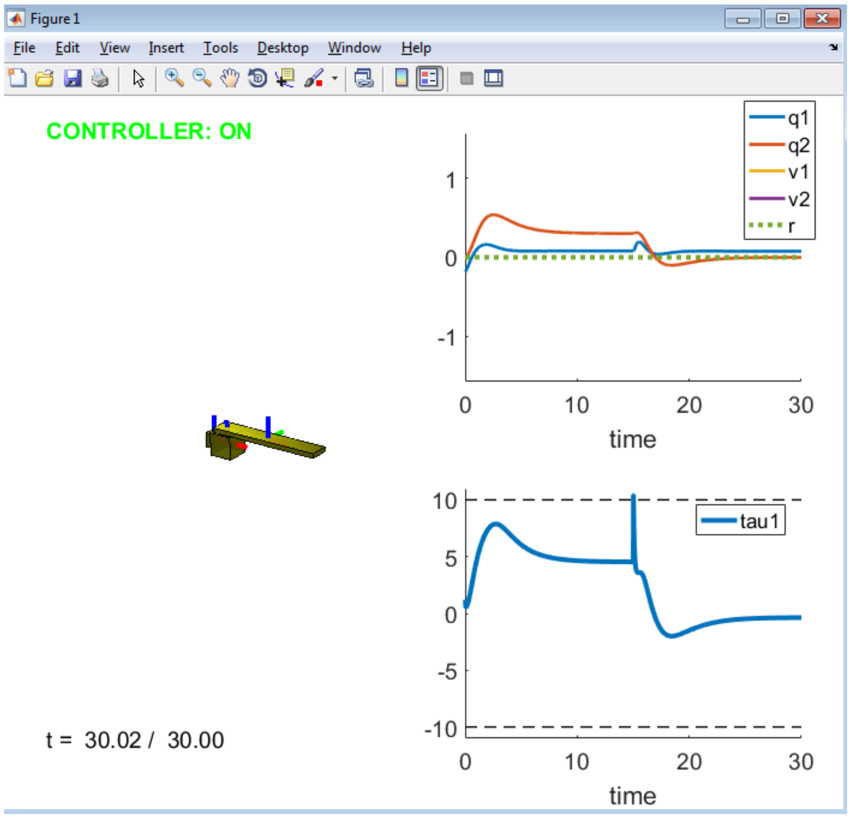

Figure 10: Piecewise DesignReport02

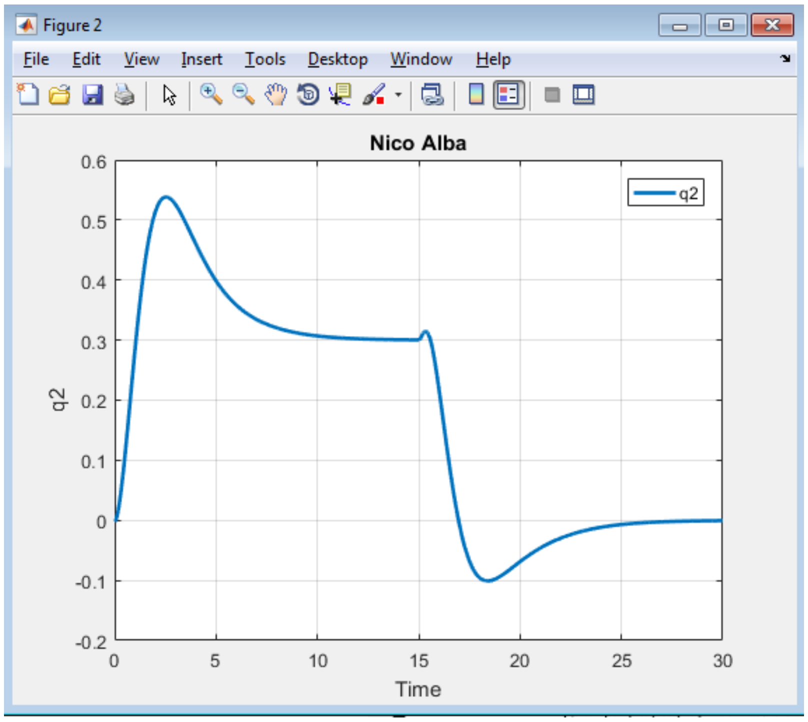

Figure 11: q_2 v.s. time

Similarly to before, we can plot the errors from $0 \leq t \leq 15$ and $15 < t \leq 30$ seconds.

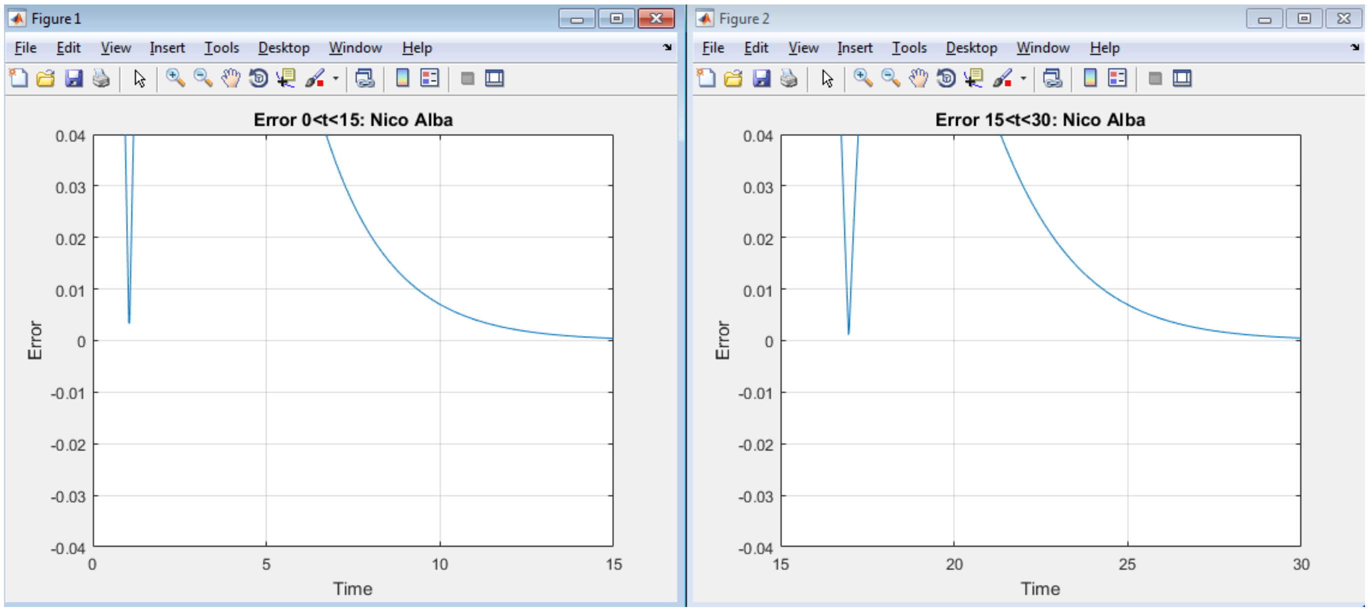

Figure 12: Piecewise Errors

As you can see, the q_2 never has enough time to get to as close of an error as before, but does get fairly close! We can adjust this by choosing a different value for $lqr(A, B, Q, 0.01)$ instead of 0.01.

Can we get q_2 to behave sinusoidally and track it?

```
function [actuators,data] = runControlSystem(sensors,references,parameters,data)

data.r=.2*sin(sensors.t)
data.x=[sensors.q1; sensors.q2; sensors.v1; sensors.v2];
data.v= data.v+(sensors.q2 - data.r)*.02;

%Input: u= -K*x +kRef*r +kInt*v(t)
actuators.tau1 = -data.K*data.x +data.kRef*data.r +data.kInt*data.v;

end
DesignProblem02('Controller','diagnostics',true,'datafile','data.mat','disturbance',
    true,'reference',@(t) .2*sin(t))
```

Figure 13: Sinusoidal input

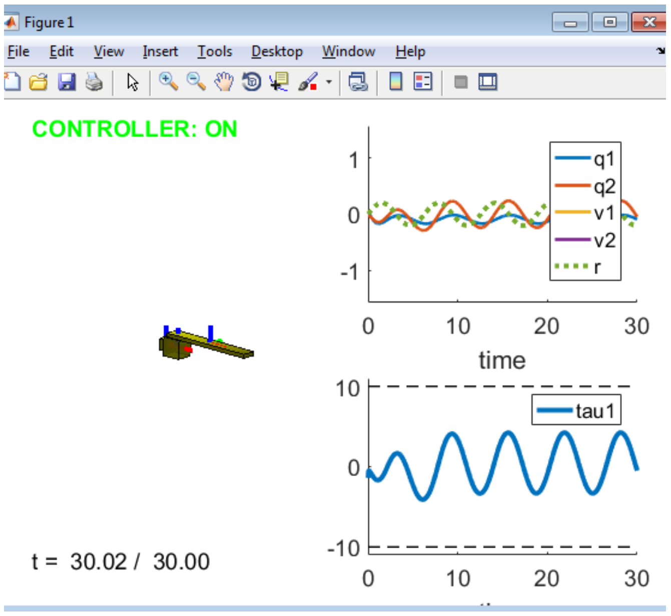

Figure 14: Sinusoidal DesignReport02

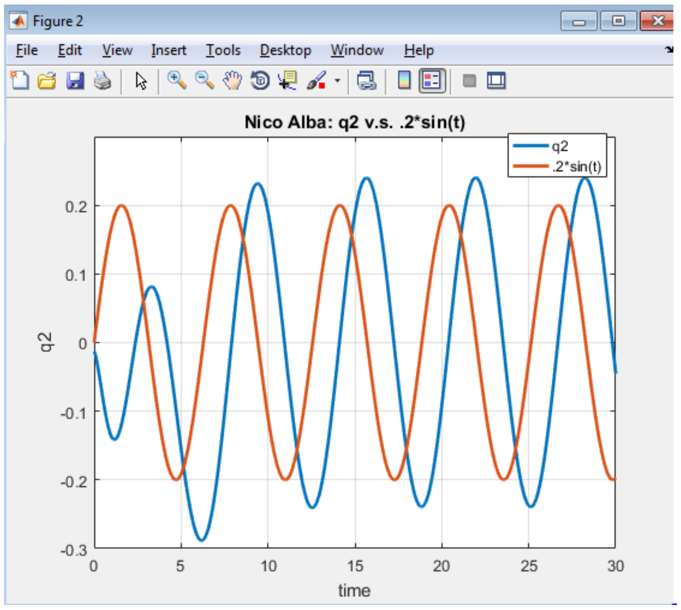

Figure 15: q_2 v.s 0.2*sin(t)

Welp... Maybe next time!