Path Planning

Lecture #7

ECE 761: Robotics

Class taught at North Dakota State University
Department of Electrical and Computer Engineering

Please visit www.BisonAcademy.com for corresponding lecture notes, homework sets, and solutions.

Path Planning

- How to go from one point to another in a fixed amount of time.
- There are several ways to do this.

Objective

- Show you some of the methods
- Showcase some problems with each

Example:

- Puma robot
- Trace a triangle in 6 seconds

	y	time
P0	-40	0
P1	40	2
P2	0	4
P3 = P0	-40	6

Linear Interpolation (1st-order polynomial)

- Go from point 1 to point 2 in T seconds
- Linear interpolation

$$y(t) = at + b$$

To solve for a and b,

- Assume t = 0 at point Pi
- Assume t = T at point Pi+1

Then, plugging in the endpoints

$$y(0) = P_i = b$$

$$y(T) = P_{i+1} = aT + b$$

Solving

$$b = P_i$$

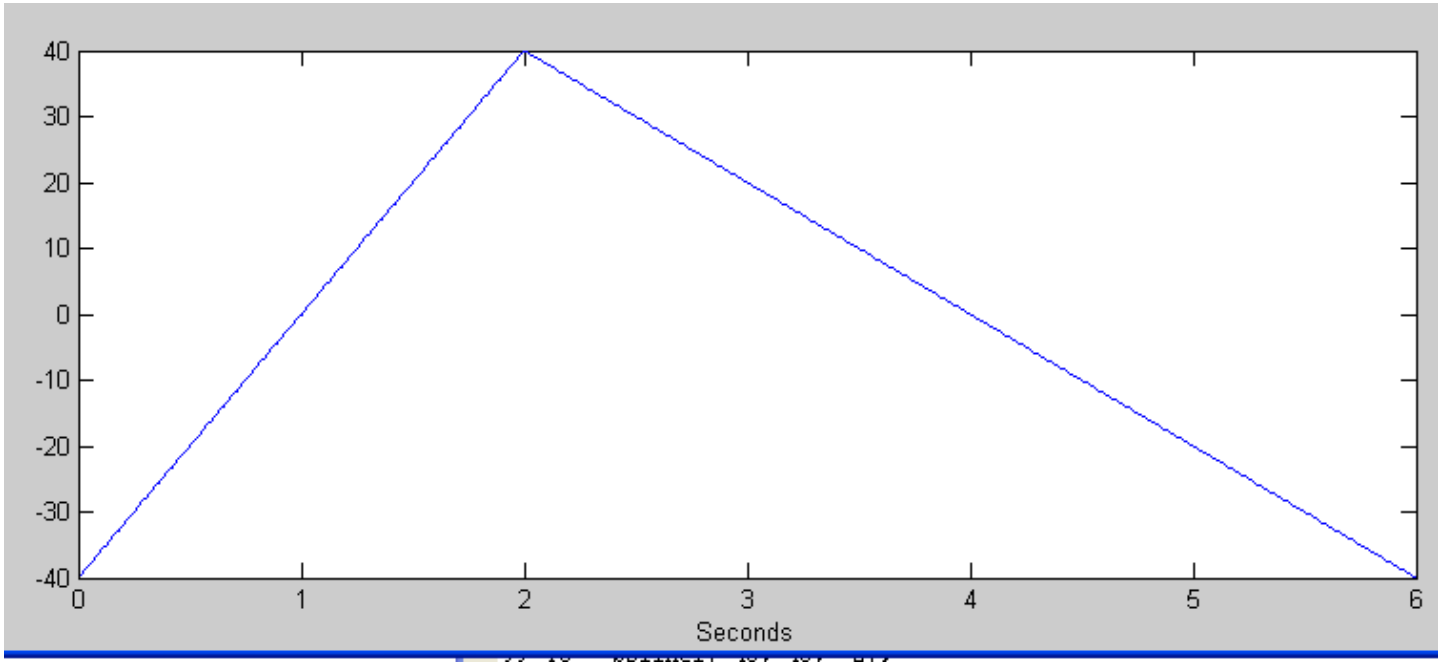
$$a = \left(\frac{P_{i+1} - P_i}{T}\right)$$

Matlab Code: Assume time increments every 10ms

```
function [Y] = Spline1(P0, P1, T)

   t = [0.01:0.01:T];

   a = (P1 - P0) / T;
   b = P0;

   Y = a*t + b;

   end
```

Tracing out a triangle:

```
Y0 = Spline1(-40, 40,  2);
Y1 = Spline1( 40,  0,  2);
Y2 = Spline1(  0, -40, 2);

Y = [Y0, Y1, Y2];
t = [1:length(Y)] * 0.01;

plot(t,Y)
```

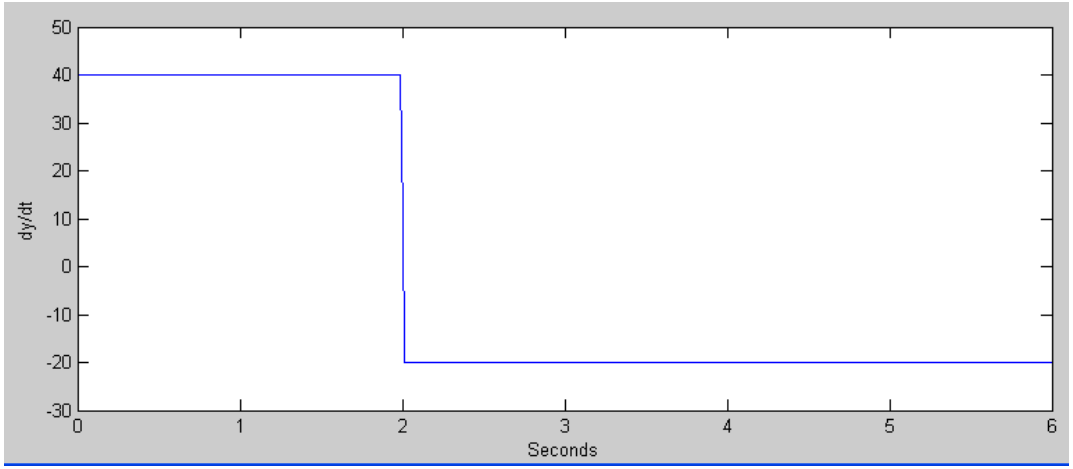


Tip Position (Y) vs. Time for Linear Interpolation between points P0, P1, and P2

Problem:

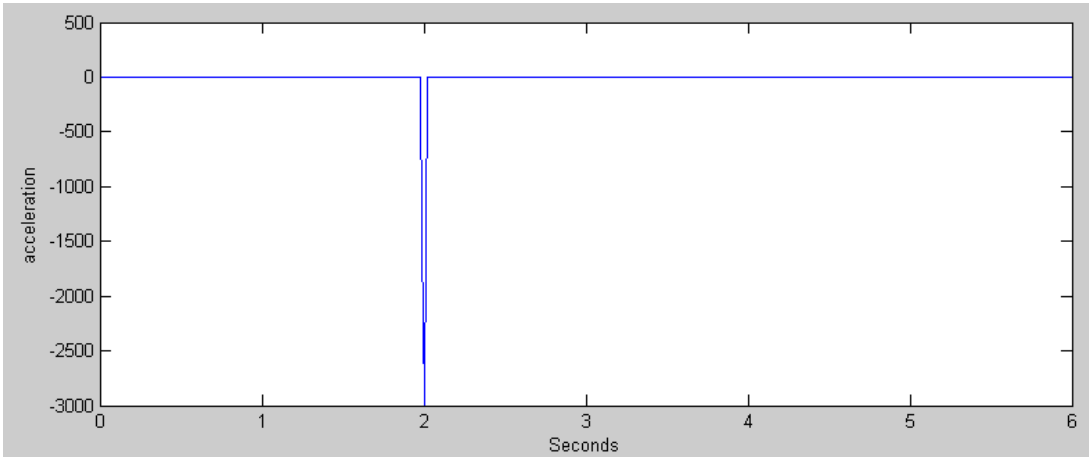
- First (voltage) has jump discontinuities

```
dY = derivative(Y);
plot(t,dY)
xlabel('Seconds');
```



- Second derivative has a delta function (infinite current)

```
ddY = derivative(dY);
plot(t,ddY)
xlabel('Seconds');
```



Cosine Interpolation

- Bring the velocity to zero at the endpoints

$$\tau = \left(\frac{1 - \cos\left(\frac{\pi t}{T}\right)}{2} \right)$$

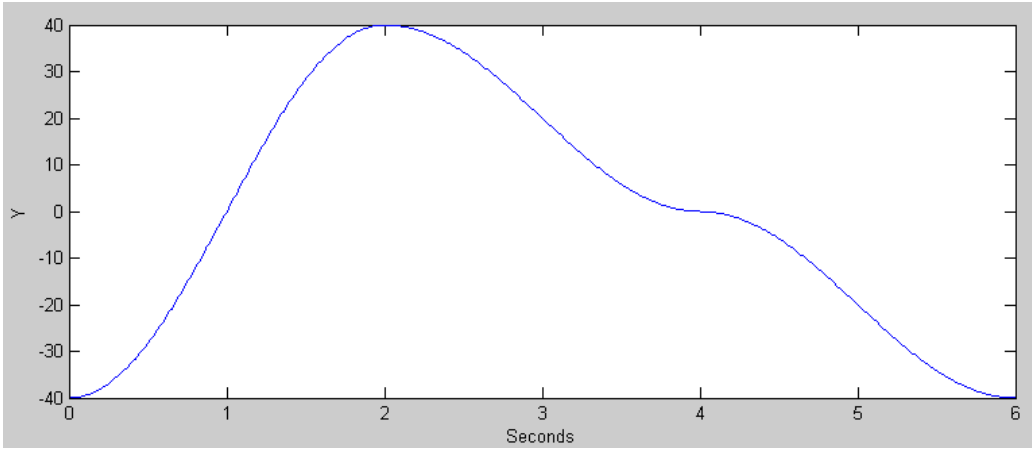
$$y(t) = a\tau + b$$

Matlab Code:

```
function [Y] = Spline1(P0, P1, T)

   t = [0.01:0.01:T];
   tau = (1 - cos(pi*t/T)) / 2;

   a = (P1 - P0) / T;
   b = P0;

   Y = a*tau + b;

   end
```

Repeating with this new spline curve fit:

```
Y0 = Spline1(-40, 40,  2);
Y1 = Spline1( 40,  0,  2);
Y2 = Spline1(  0, -40, 2);

Y = [Y0, Y1, Y2];
t = [1:length(Y)] * 0.01;

plot(t,Y)
```

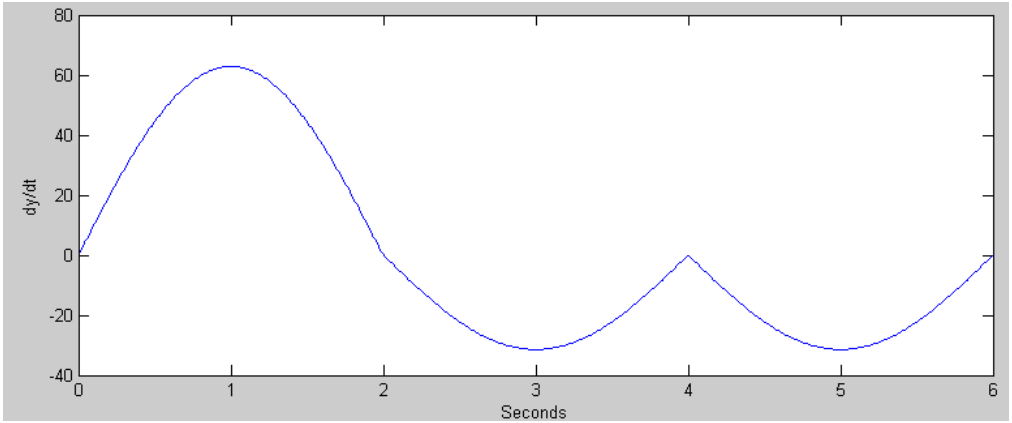


Tip Position)

- Comes to a stop at each point

Tip Velocity

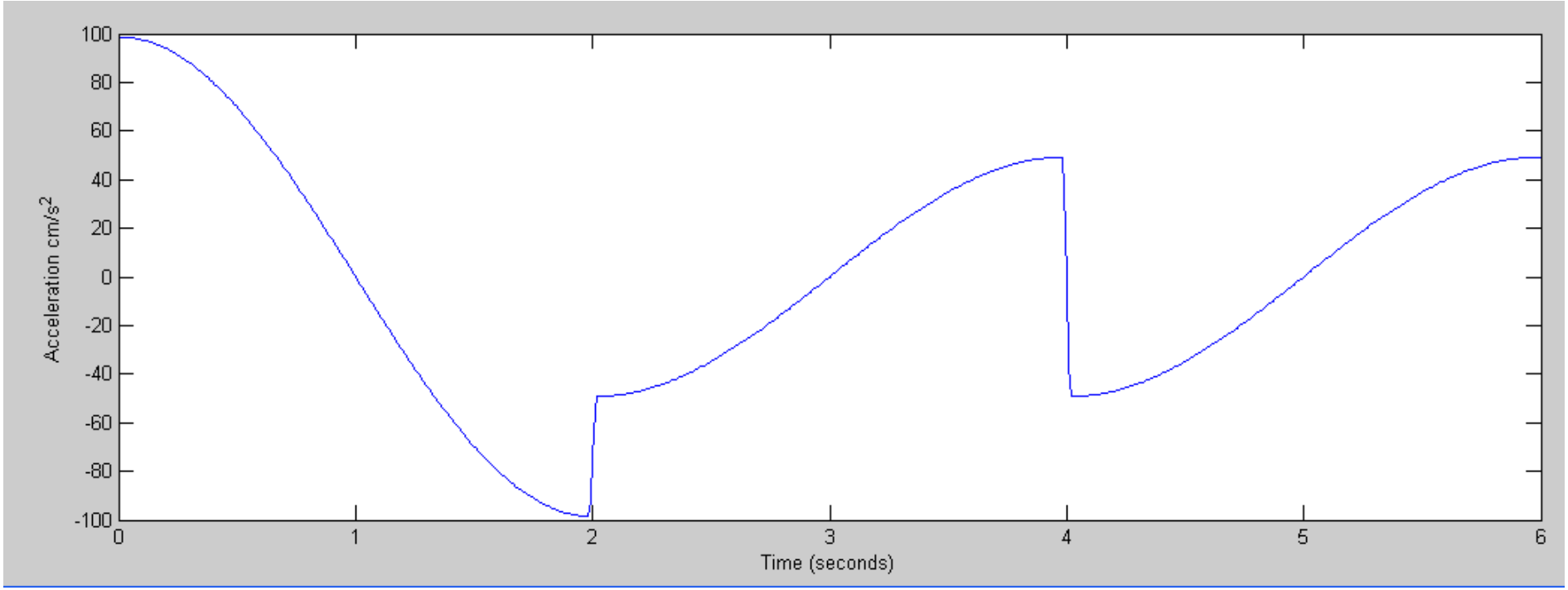
- Continuous



Tip Acceleration

- Finite

```
ddY = derivative(dY);
plot(t,ddY)
xlabel('Seconds');
ylabel('acceleration')
```



Tip Acceleration vs. Time for Cosine Interpolation from P0 to P1 ot P2

Cosine Interpolation

- The acceleration is finite (good) - it doesn't contain any delta functions. This means you can implement this with a DC motor (where current is proportional to acceleration)
- For each path (0-2, 2-4, 4-6 seconds), the acceleration is a cosine function. The second derivative of cosine is minus cosine.
- There are jump discontinuities in the acceleration. This doesn't bother mechanical systems: people don't like sudden changes in acceleration, mechanics don't care.

Cubic Interpolation (spline curve fit)

Specify

- The position and velocity at point 0 (at t = 0)
- The position and velocity at point 1 (at t = T)

Four equations for four unknowns:

$$x(t) = at^3 + bt^2 + ct + d$$

Endpoints:

$$x(0) = d$$

$$x(T) = aT^3 + bT^2 + cT + d$$

$$\frac{dx}{dt}(0) = c$$

$$\frac{dx}{dt}(T) = 3aT^2 + 2bT + c$$

Put in matrix form

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ T^3 & T^2 & T & 1 \\ 0 & 0 & 1 & 0 \\ 3T^2 & 2T & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ \frac{dx_0}{dt} \\ \frac{dx_1}{dt} \end{bmatrix}$$

Matlab Code:

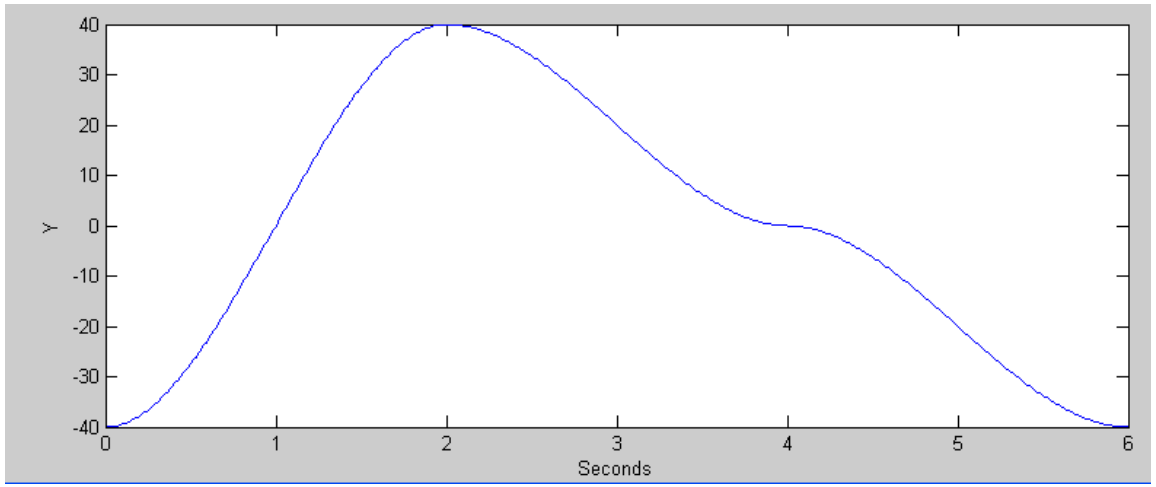
```
function [X] = Spline3(x0, x1, dx0, dx1, T)

A = [0,0,0,1 ; T^3, T^2, T, 1 ; 0,0,1,0 ; 3*T^2, 2*T, 1, 0];
B = [x0 ; x1 ; dx0 ; dx1];
ABCD = inv(A) * B;

a = ABCD(1);
b = ABCD(2);
c = ABCD(3);
d = ABCD(4);

t = [0.01:0.01:T];
X = a*(t.^3) + b*(t.^2) + c*t + d;

end
```

Example: Go from

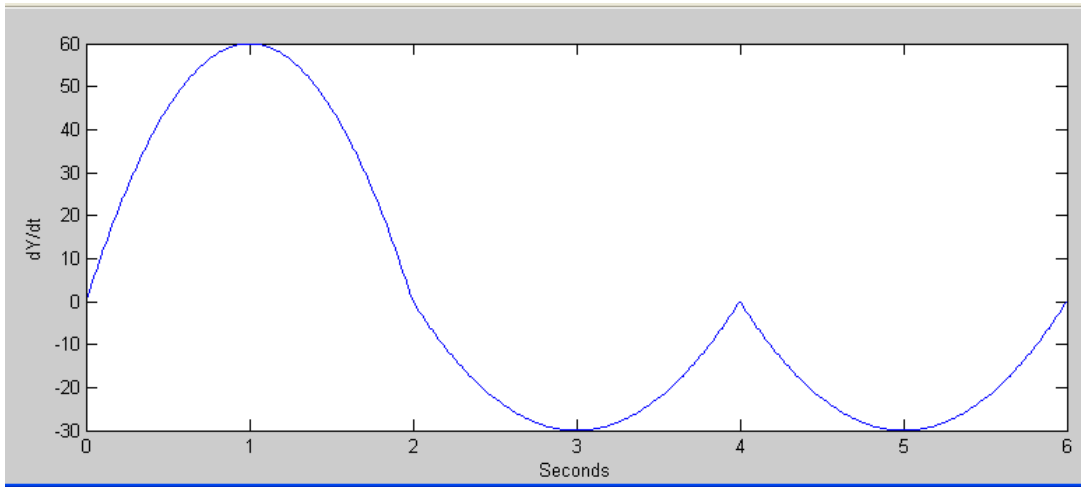
- P0 = -40 at t=0
- P1 = +40 at t=2
- P2 = 0 at t=4
- P3 = P0 = 0 at t=6

10 to 20 in 2 seconds with the initial and final velocities being zero

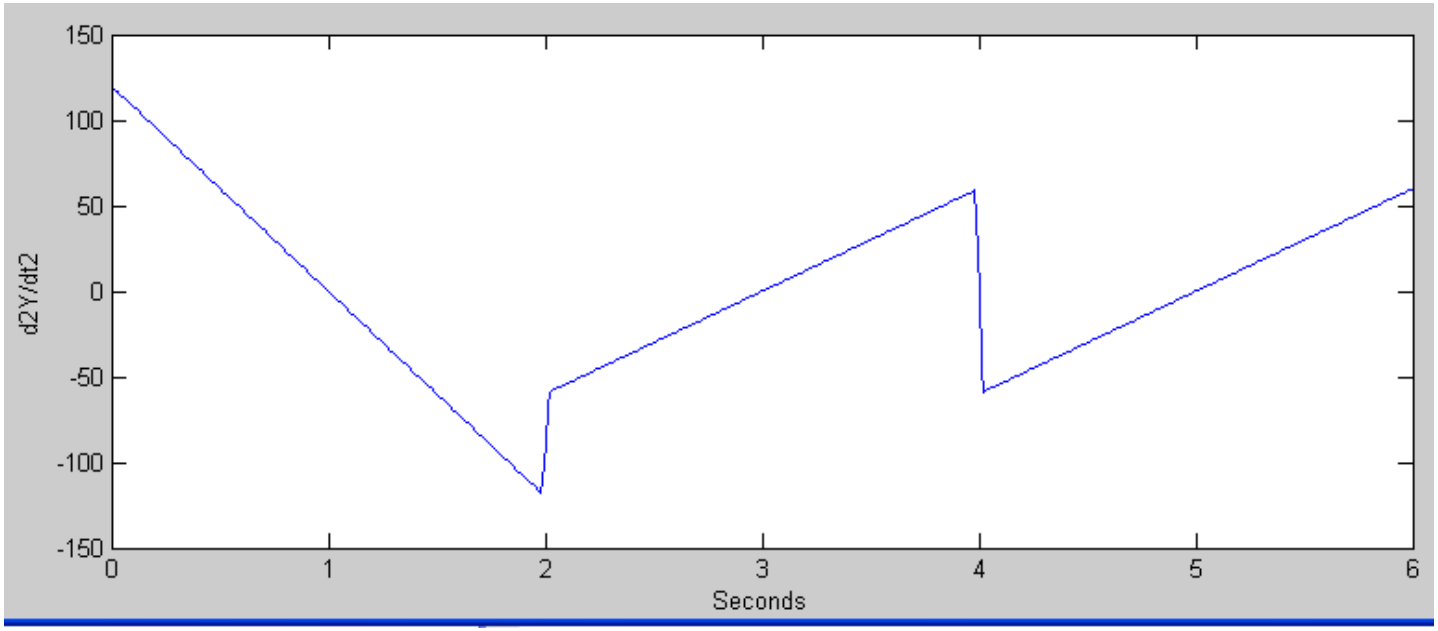
```
Y1 = Spline3( 40,  0,  0, 0, 2);
Y2 = Spline3(  0, -40, 0, 0, 2);
Y = [Y0, Y1, Y2];
t = [1:length(Y)] * 0.01;
plot(t,Y)
xlabel('Seconds');
ylabel('Y')

X = spline3(10, 20, 0, 0, 2);
t = [1:length(X)] * 0.01;
plot(t,X);
```



Tip Position vs. Time for Cubic Spline Interpolation between points P0, P1, and P2



Tip Velocity vs. Time for Cubic Spline Interpolation between points P0, P1, and P2



Tip Acceleration vs. Time for Cubic Spline Interpolation between points P0, P1, and P2

Note that

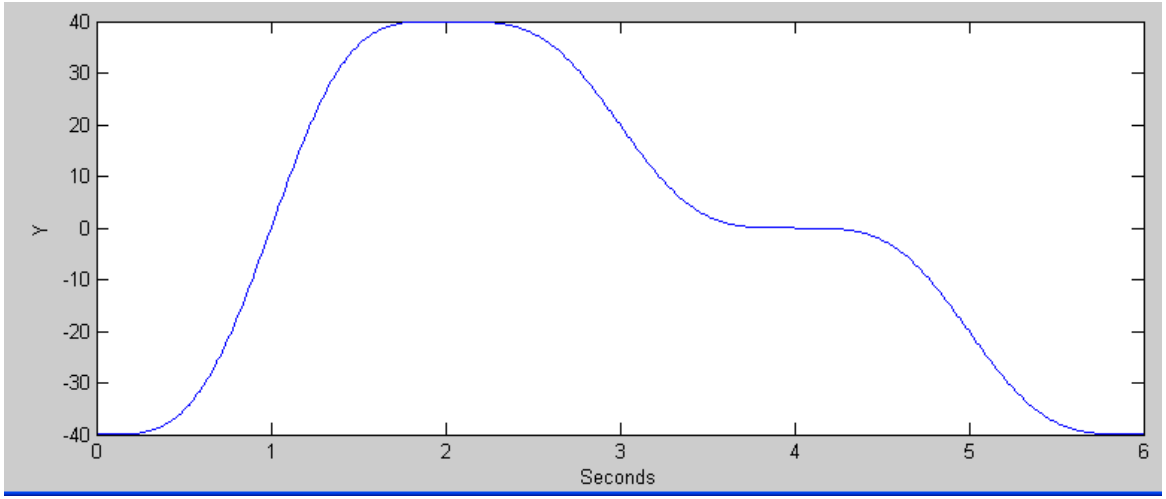
- The acceleration is finite (good) - it doesn't contain any delta functions. This means you can implement this with a DC motor (where current is proportional to acceleration)
- For each path (0-2, 2-4, 4-6 seconds), the acceleration is a line. The second derivative of cubic is a straight line.
- There are jump discontinuities in the acceleration.

Cubic Curve Fitting with Cosine Interpolation

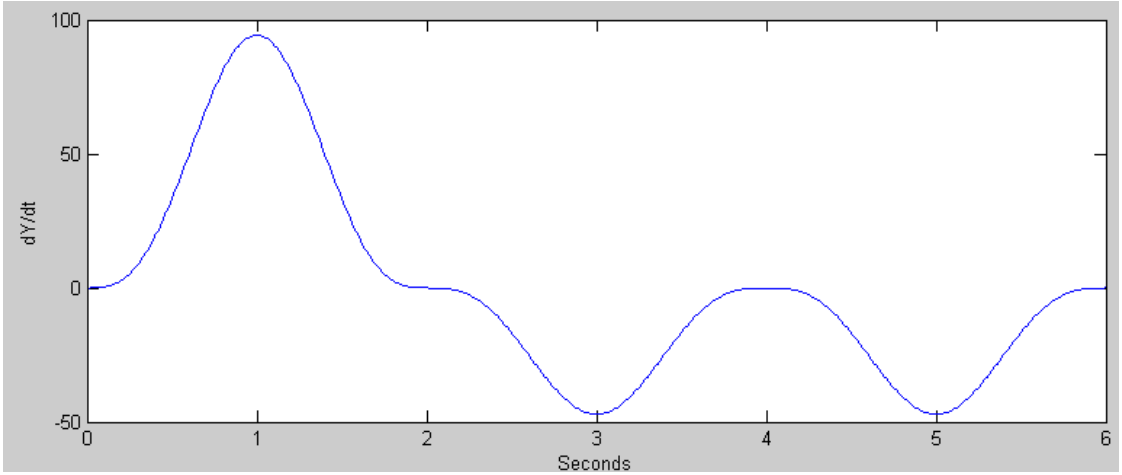
If you want to eliminate the jump discontinuities in acceleration at the endpoints, you could use cosine interpolation along with a cubic spline curve fit:

```
function [X] = Spline3(x0, x1, dx0, dx1, T)

A = [0,0,0,1 ; T^3, T^2, T, 1 ; 0,0,1,0 ; 3*T^2, 2*T, 1, 0];
B = [x0 ; x1 ; dx0 ; dx1];
ABCD = inv(A) * B;

a = ABCD(1);
b = ABCD(2);
c = ABCD(3);
d = ABCD(4);

t = [0.01:0.01:T];
tau = (T/2) * (1 - cos(pi*t/T));

X = a*(tau.^3) + b*(tau.^2) + c*tau + d;

end
```

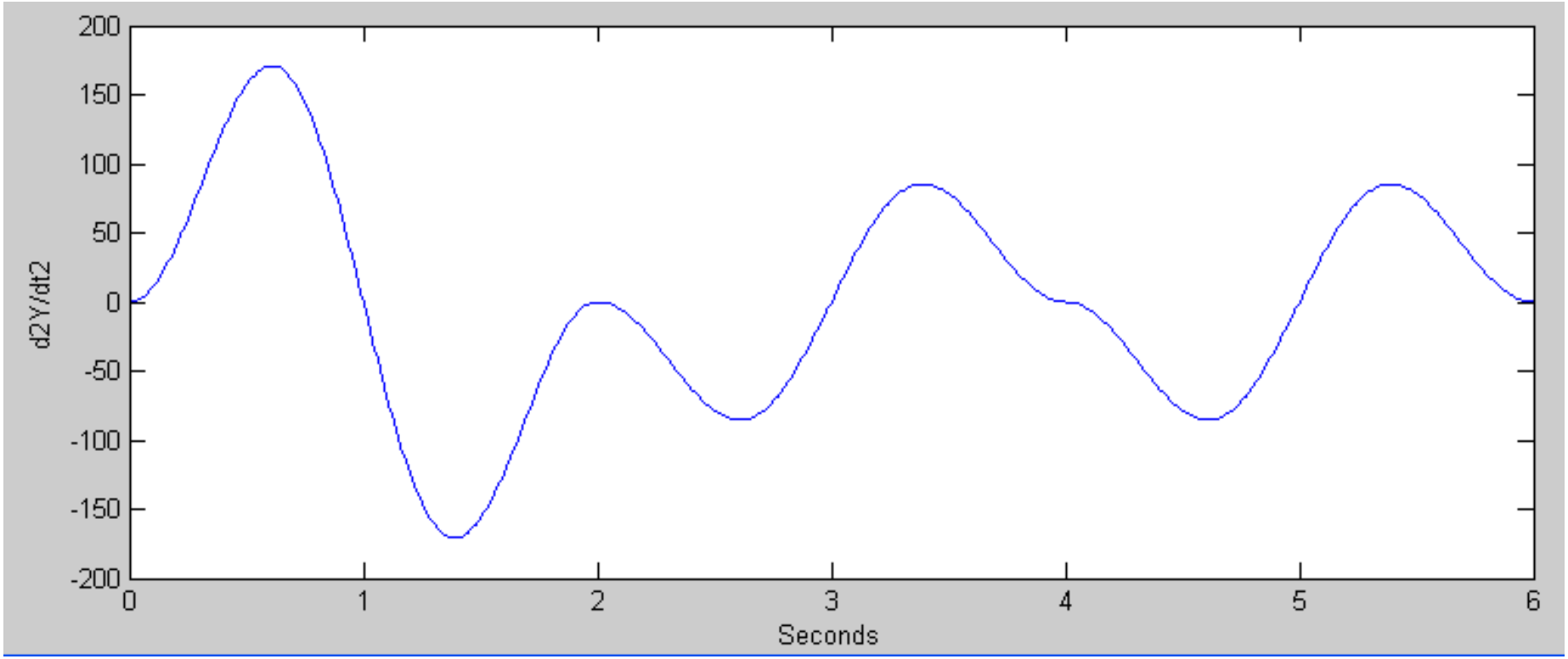
Repeating the previous solution results in the following:



Tip Position vs. Time for Cubic Spline & Cosine Interpolation along between points P0, P1, and P2.



Tip Velocity vs. Time for Cubic Spline & Cosine Interpolation along between points P0, P1, and P2.

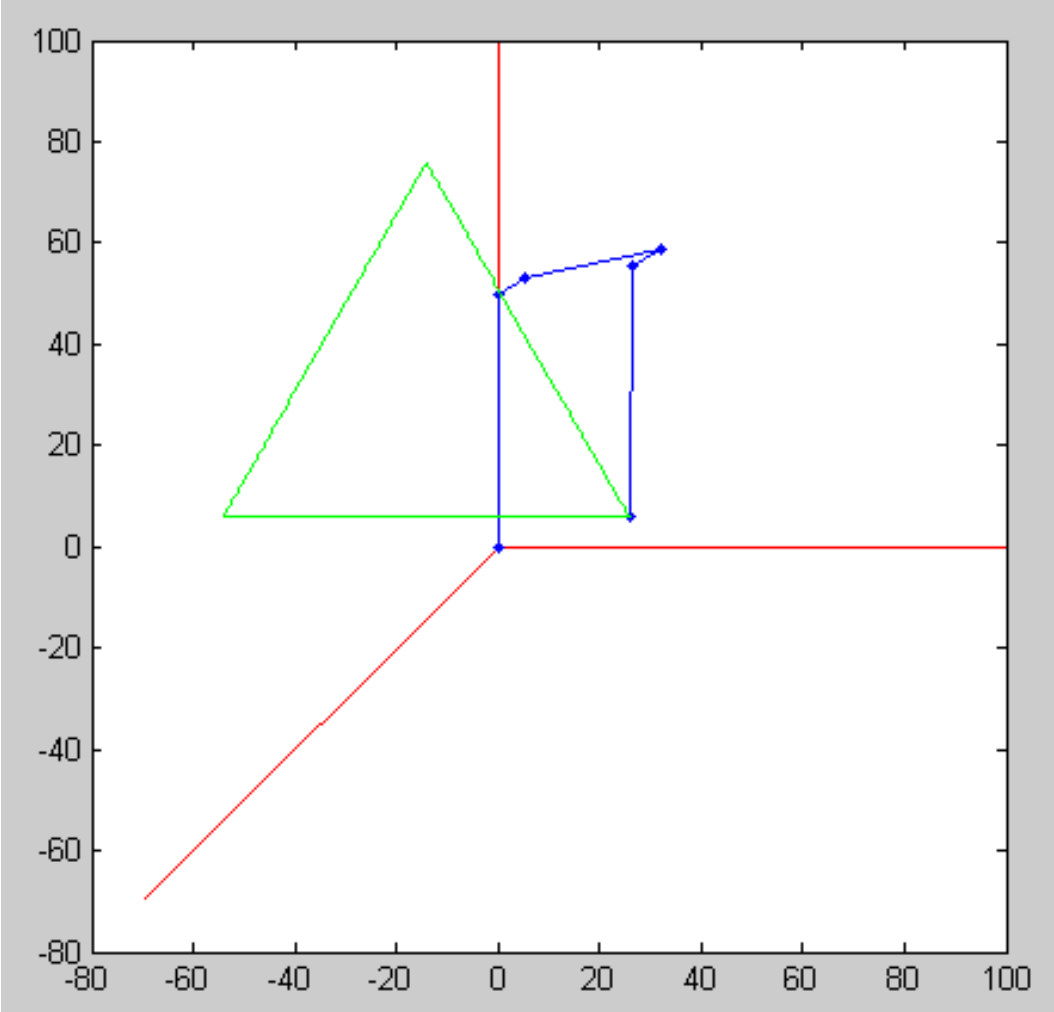


Tip Acceleration vs. Time for Cubic Spline & Cosine Interpolation along between points P0, P1, and P2.

Example: Puma Robot Tracing out a Triangle:

To illustrate how spline curve fitting can be applied to a robot manipulator, consider the following example:

Define the tip position vs. time so that a Puma robot traces out a triangle in 6 seconds:



The tip positions are

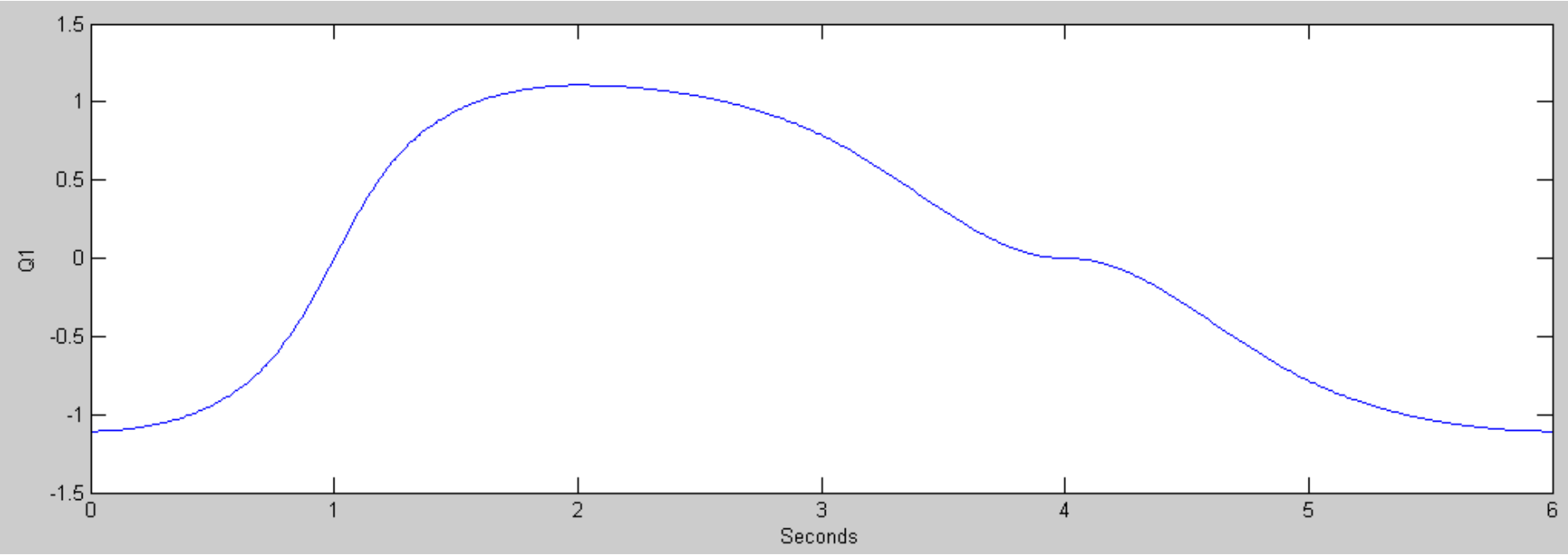
Point	P0	P1	P2	P3 = P0
X	20	20	20	20
Y	-40	40	0	-40
Z	20	20	90	20
Time	0	2	4	6

To do this,

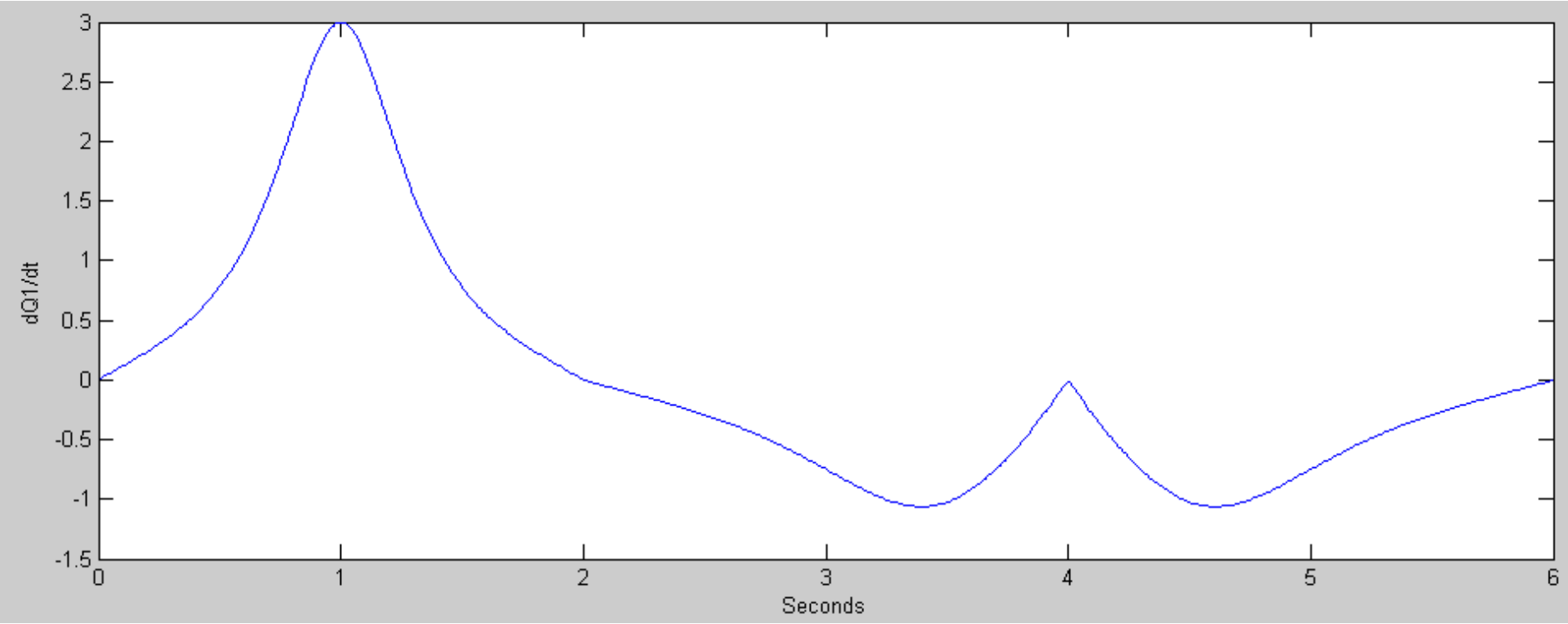
- Define where the robot should be every 10ms
- Use a spline curve fit for X, Y, and Z from point to point
- Use the same spline curve fit so that the resulting path is a straight line

Matlab Code:

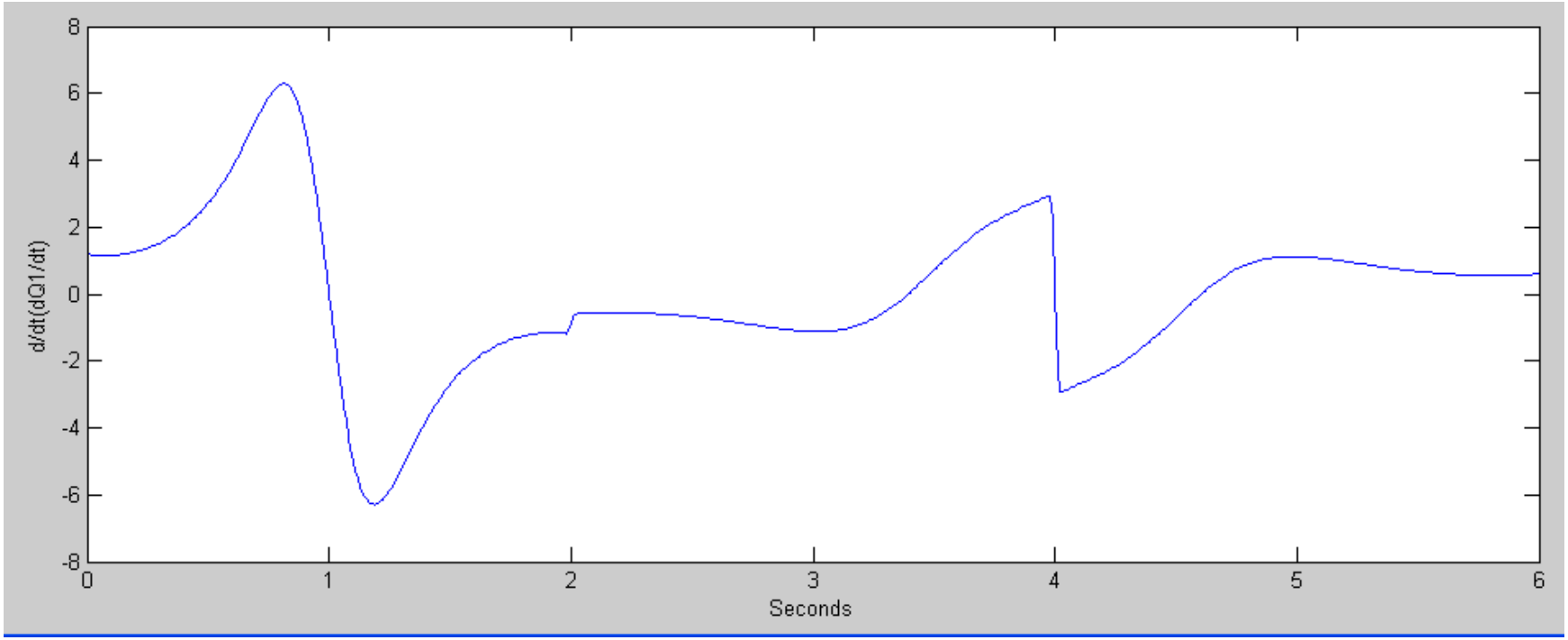
```
X0 = Spline3(20, 20, 0, 0, 2);
X1 = Spline3(20, 20, 0, 0, 2);
X2 = Spline3(20, 20, 0, 0, 2);
X = [X0, X1, X2];

Y0 = Spline3(-40, 40,  0, 0, 2);
Y1 = Spline3( 40,  0,  0, 0, 2);
Y2 = Spline3(  0, -40, 0, 0, 2);
Y = [Y0, Y1, Y2];

Z0 = Spline3(20, 20, 0, 0, 2);
Z1 = Spline3(20, 90, 0, 0, 2);
Z2 = Spline3(90, 20, 0, 0, 2);
Z = [Z0, Z1, Z2];

TIP = [X ; Y ; Z];
Q = [];


for i=1:length(TIP)
   q = InversePuma(TIP(:,i));
   T = Puma(q, TIP);
   Q = [Q, q];
   pause(0.01);
   end
```



Joint Angle Q1 vs. Time for tracing out a triangle:



Joint Velocity of Q1 vs. Time when tracing out a triangle



Joint Acceleration in Q1 vs. Time when tracing out a triangle.

Homework #7

- Use a Puma robot
- Trace out a shape (your pick)
- Keep acceleration finite
- For one of the angles, plot
  - The resulting joint position
  - Angle, and
  - Acceleration