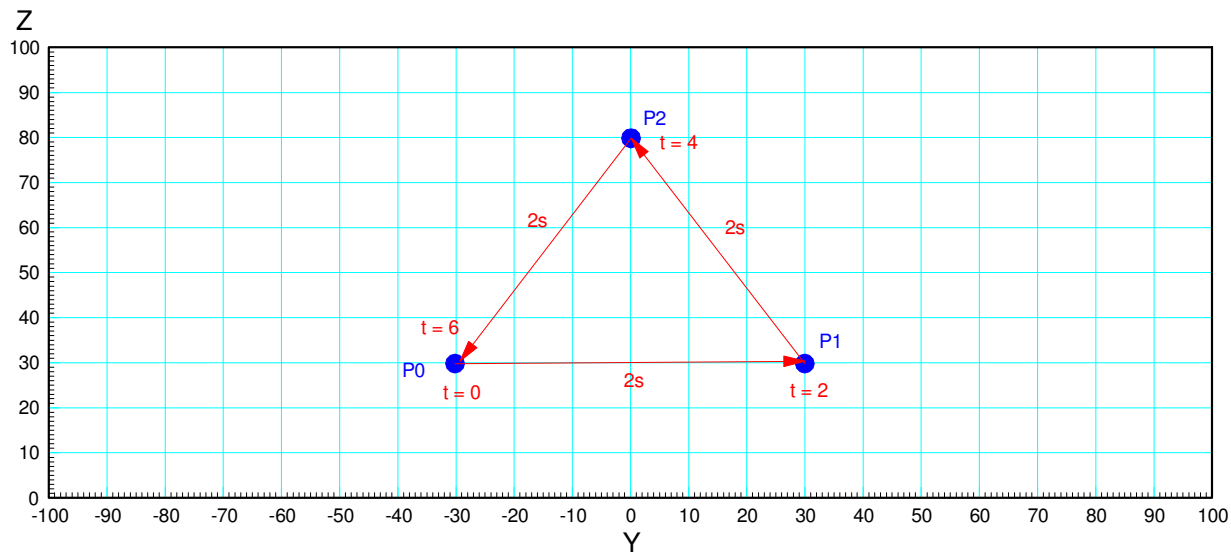
Path Planning

Path Planning is how to go from one point to another in a fixed amount of time. There are several ways to do this. The objective here is to expose you to several of these options and showcase the problems with each.

As an example, suppose you have a single mass which is to go from points P0 to P3 with 2 seconds elapsing from point to point



How you go from one point to another (i.e. the position vs. time) is what path planning is all about. This lecture covers several approaches and points out some of the advantages and disadvantages of each.

One way to go from one point to the next (such as P0 to P1) is to use linear interpolation:

$$P(k) = (1-k)P_0 + kP_1$$

where

- k = 0 at the start of the movement, and
- k = 1 at the end of the movement

The challenge is how to define **k** as a funciton of time.

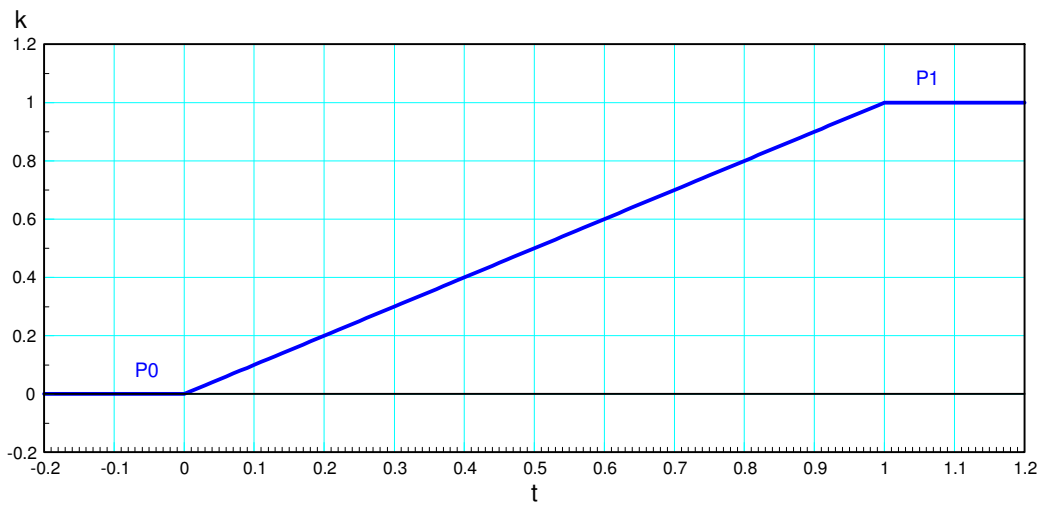
For the sake of simplicity, assume the duration of the move is 1 second. Scaling time by T will still give an arbitary solution.

Option 1: Linear Interpolation

The simplest definition is straight line.

$$k = t$$

The advantage is simplicity. The disadvantage is the velocity has a jump discontinuity:



Normalized position, velocity, and acceleration with linear interpolation

If you apply this to the problem of tracing out a triancle, you get the following:

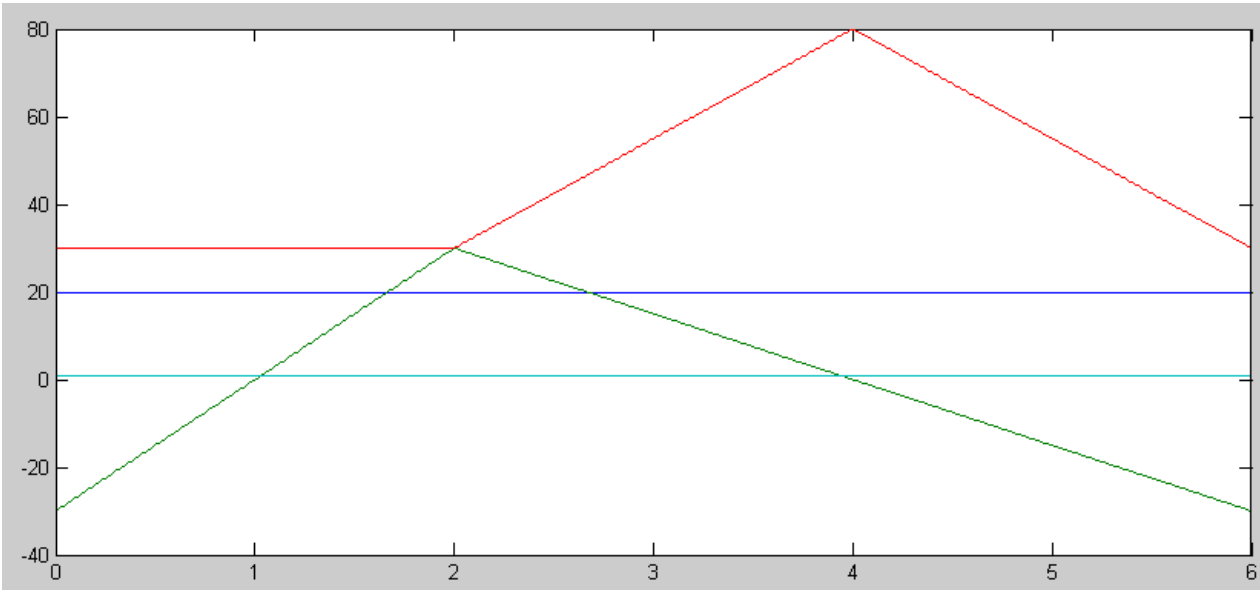
Code:

```
P0 = [20 ; -30 ; 30 ; 1];
P1 = [20 ;  30 ; 30 ; 1];
P2 = [20 ;   0 ; 80 ; 1];

P01 = Spline(P0, P1, 2);
P12 = Spline(P1, P2, 2);
P20 = Spline(P2, P0, 2);

TIP = [P01, P12, P20];

t = [1:length(TIP)]' * 0.01;

plot(t,TIP')
pause(5)

Vel = derivative(TIP);
Acc = derivative(Vel);
plot(t,Acc')
```

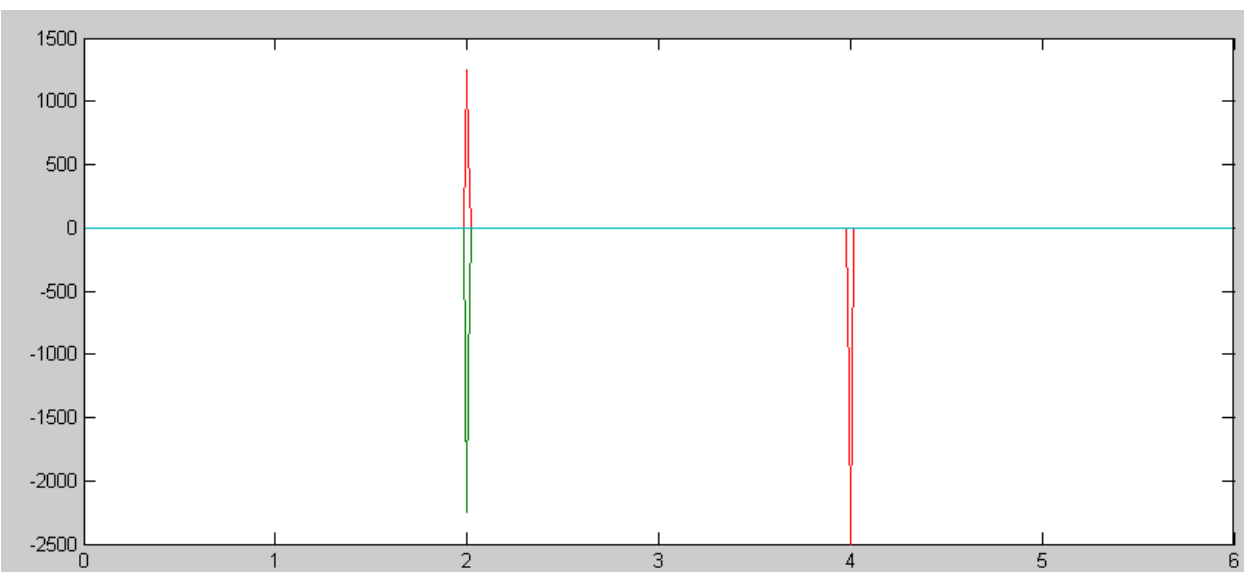
Spline is a routine which defines the tip position every 10ms as you go from point P0 to point P1:

```
function [Y] = Spline(P0, P1, T)
   t = [0.01:0.01:T] / T;
   a - t;
   Y = [];
   for i=1:length(a)
       Y = [Y, (1-a(i))*P0 + a(i)*P1];
       end
   end
```

The resulting tip position and acceleration with linear interpolation is as follows:



Tip position vs. time with Linear Interpolation



Acceleration vs. Time with linear interpoaltion

Since acceleration is proportional to current, this form of path plannign requires infinte currents: it's not the best choice.

Option 2: Cosine Interpolation

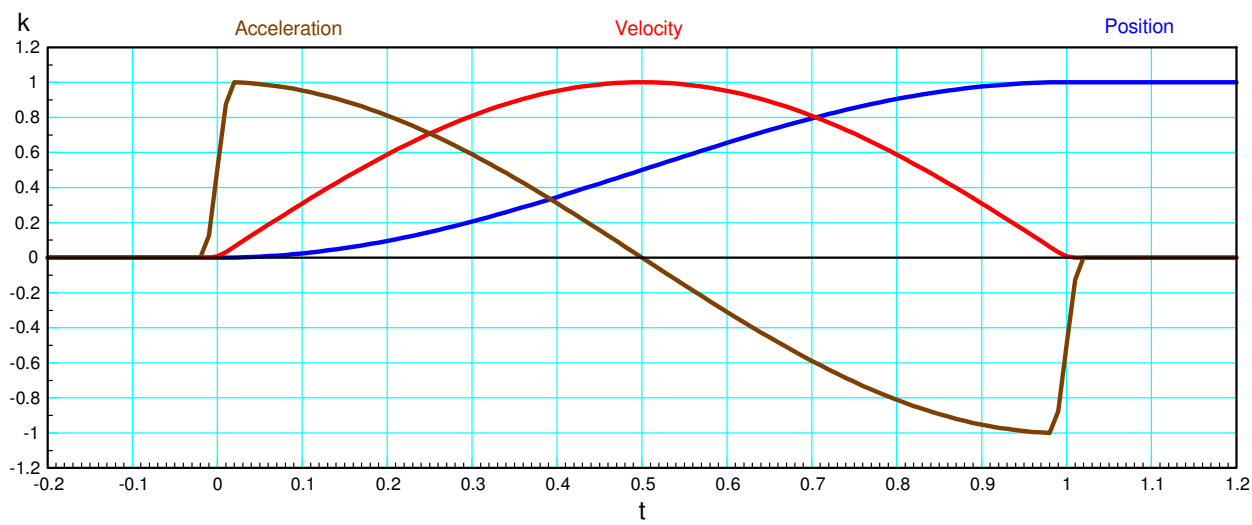
To avoid the jump discontinuities in velocity, let

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

This then results in finite velocities and finite accelerations:

$$k' = \frac{\pi}{2}\sin(\pi t)$$

$$k'' = \frac{\pi^2}{2}\cos(\pi t)$$

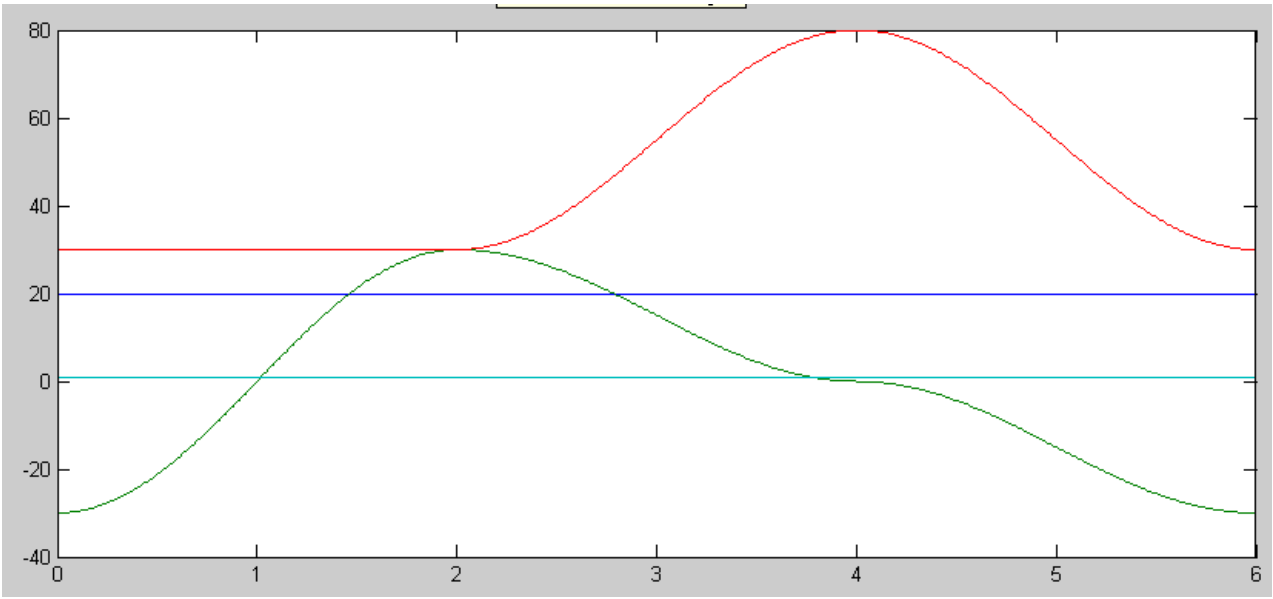


Normalized position, velocity, and acceleration with cosine interpolation

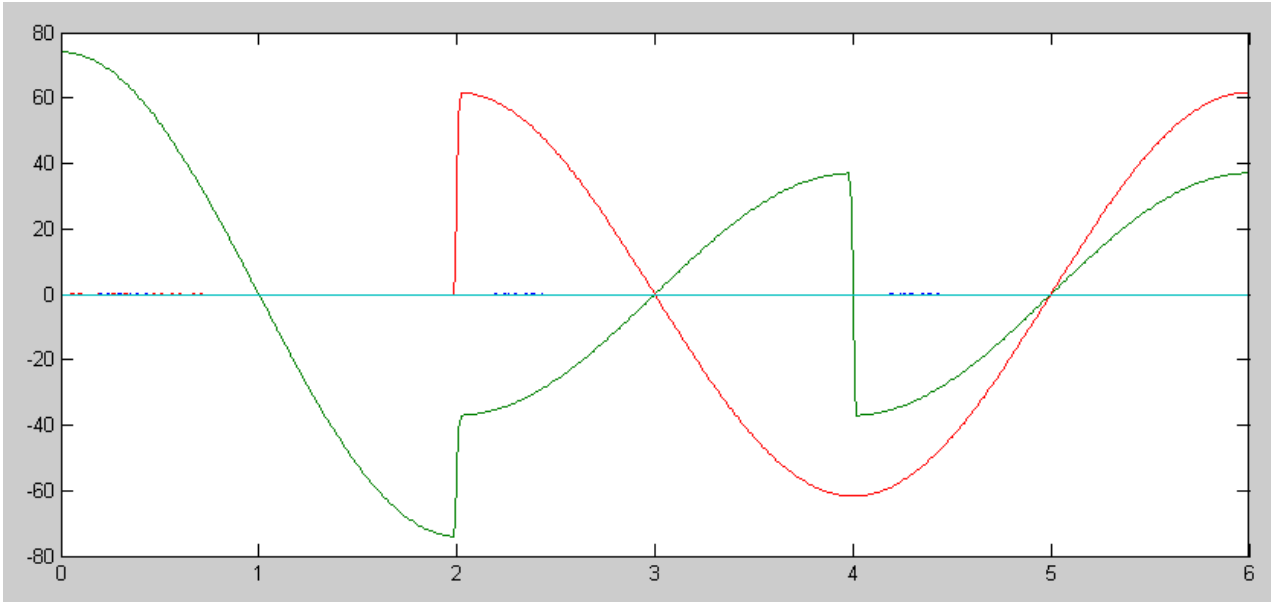
The spline function then becomes:

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

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

   Y = [];

   for i=1:length(a)
       Y = [Y, (1-a(i))*P0 + a(i)*P1];
   end

   end
```

Tracing out a triangle using a cosine interpolation results in the following tip position and angles:



Angles vs. Time with cosine interpolation



Tip acceleration vs. time with cosine interpolation

Option 3: Cubic Interpolation

If you constrain the velocity at the endpoints to be zero, you need four degrees of freedom (two position and two velocity constraints). A cubic polynomial can satisfy all of these

$$k = at^3 + bt^2 + ct + d$$

Plugging in the constraints:

$$k(0) = 0 = d$$

$$k'(0) = 0 = c$$

$$k(t) = 1 = a + b$$

$$k'(1) = 0 = 3a + 2b$$

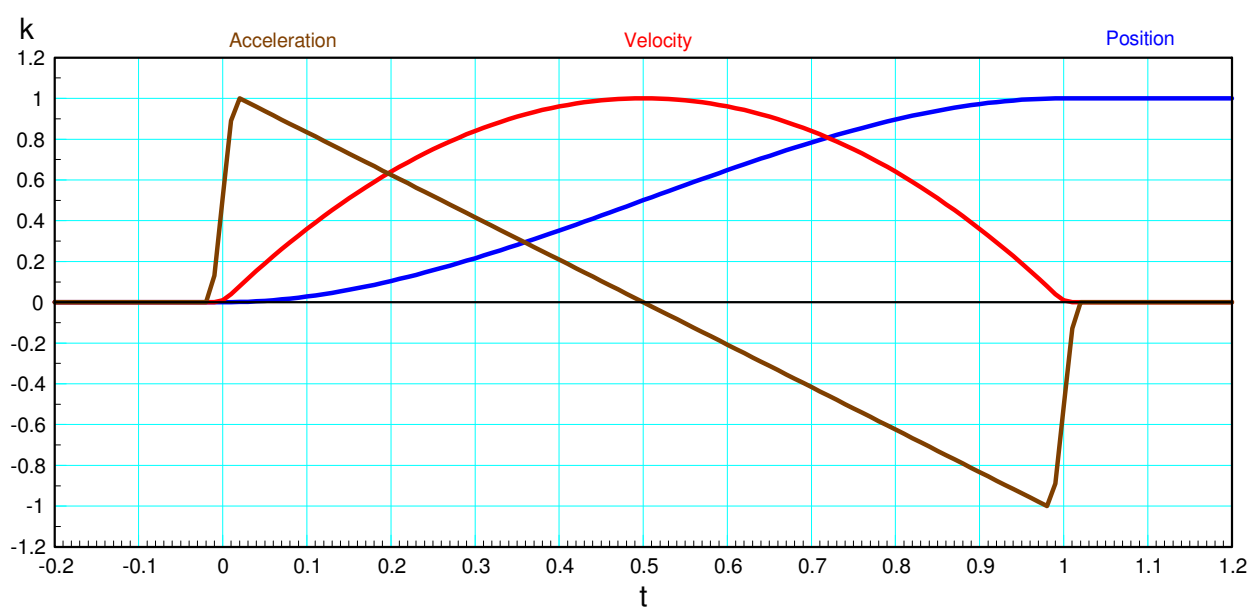
Solving gives

$$k = -2t^3 + 3t^2$$

Cubic interpolation is almost identical to cosine interpolation. There is a very small difference:

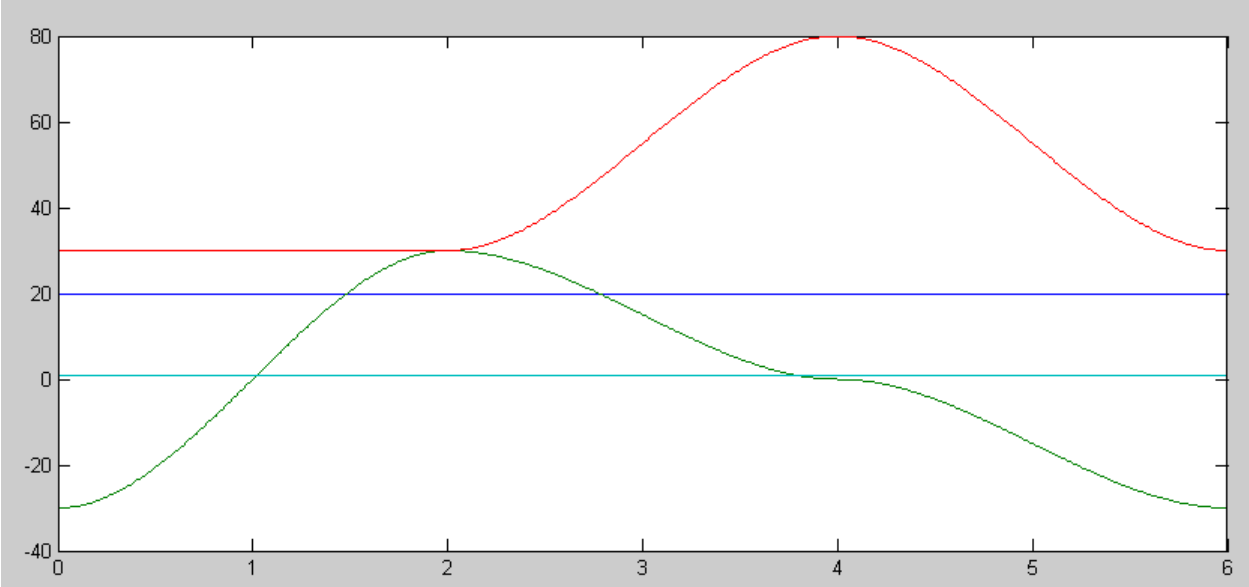
- Velocity is a parabola rather than a sine function
- Acceleration is a line rather than a cosine function

From a practical standpoint, it's about the same.

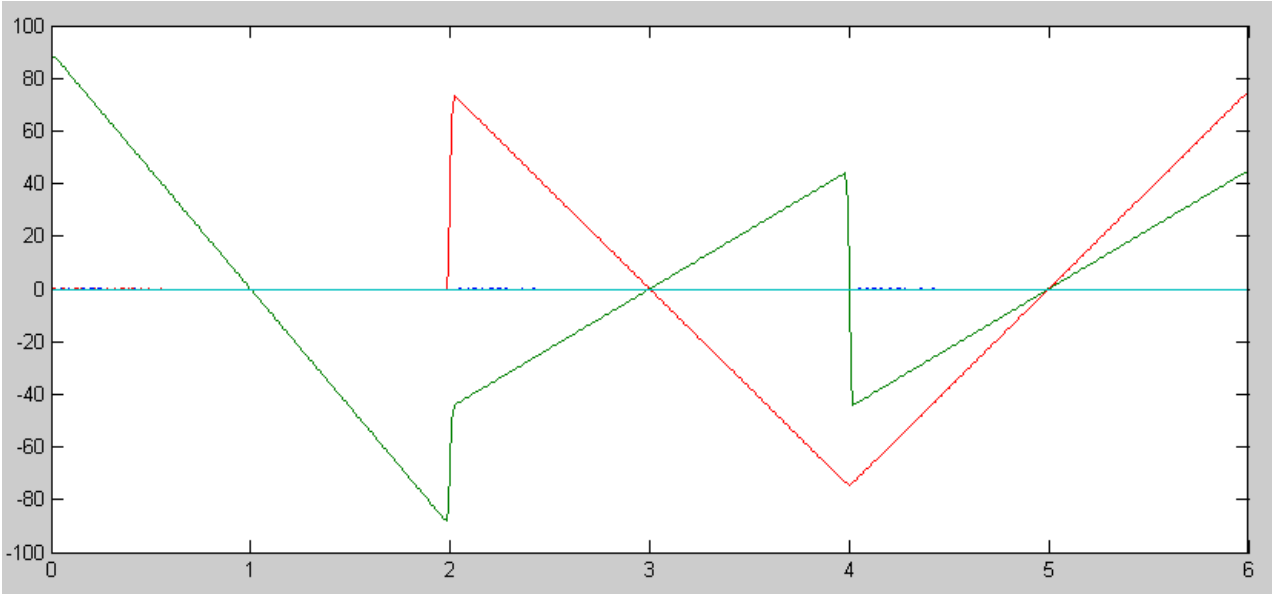


Normalized position, velocity, and acceleration with cubic interpolation

Using cubit interpolation for tracing out a triangle gives the following tip positions and accelerations



Tip position vs. time for cubit interpolation



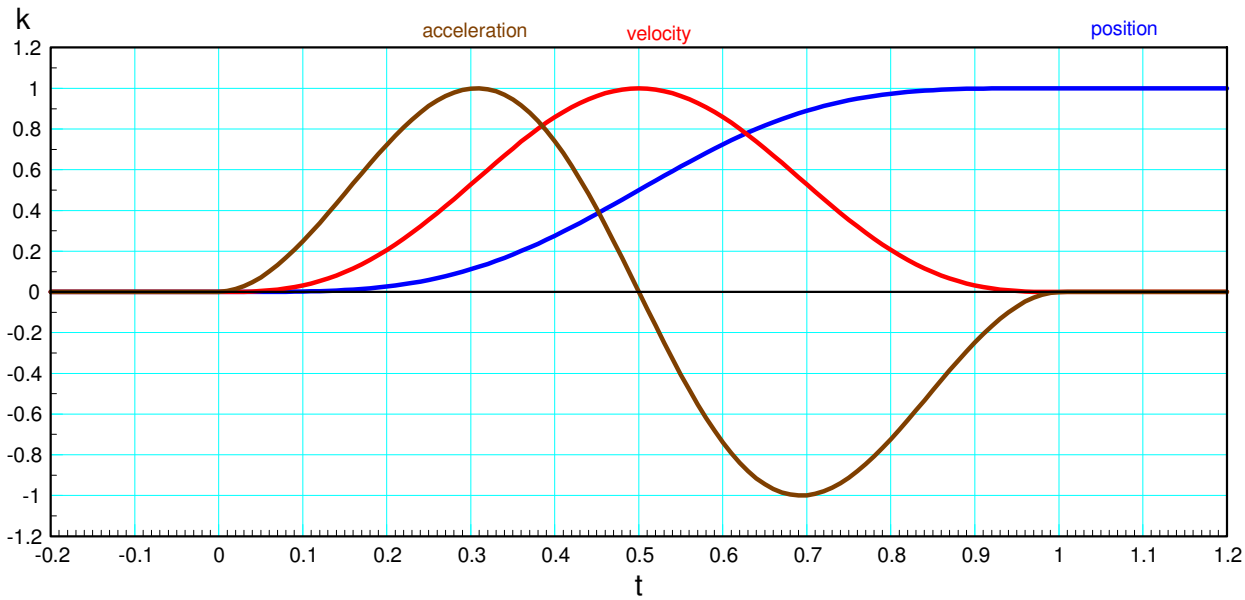
Tip acceleration with cubic interpolation

Option 4: Cubic + Cosine Interpolation

If you combine cosine interpolation with the cubic funciton, you get zero acceleration at the endpoints

$$\tau = -2t^3 + 3t^2$$

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

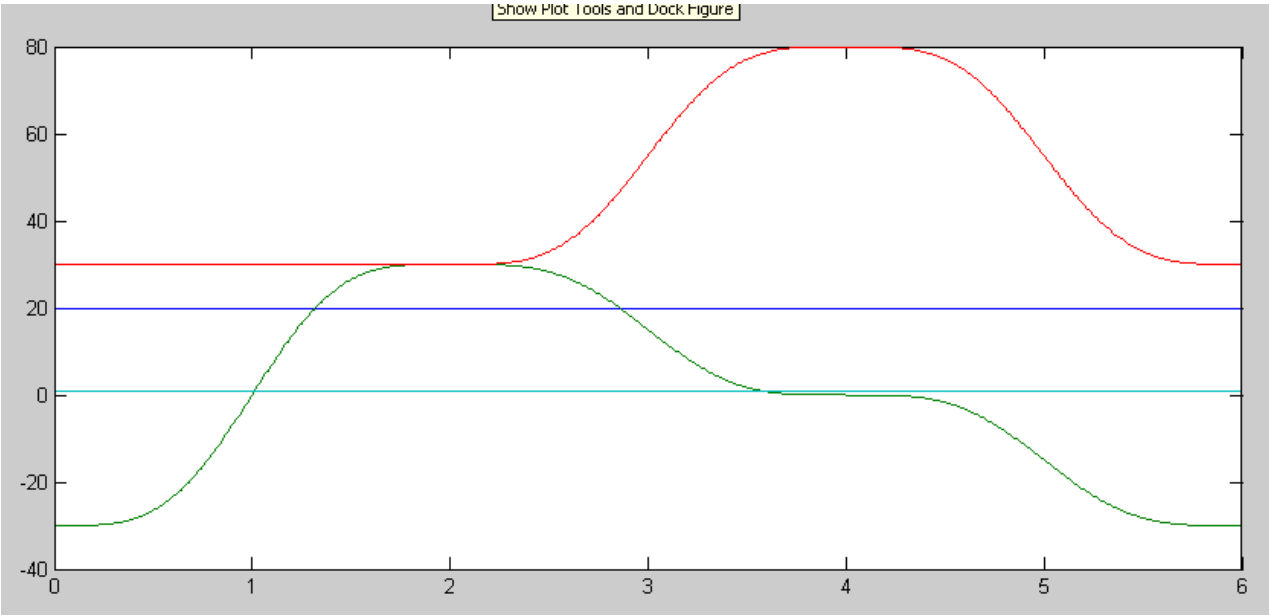


Normalized position, velocity, and acceleration with cubic + cosine interpolation

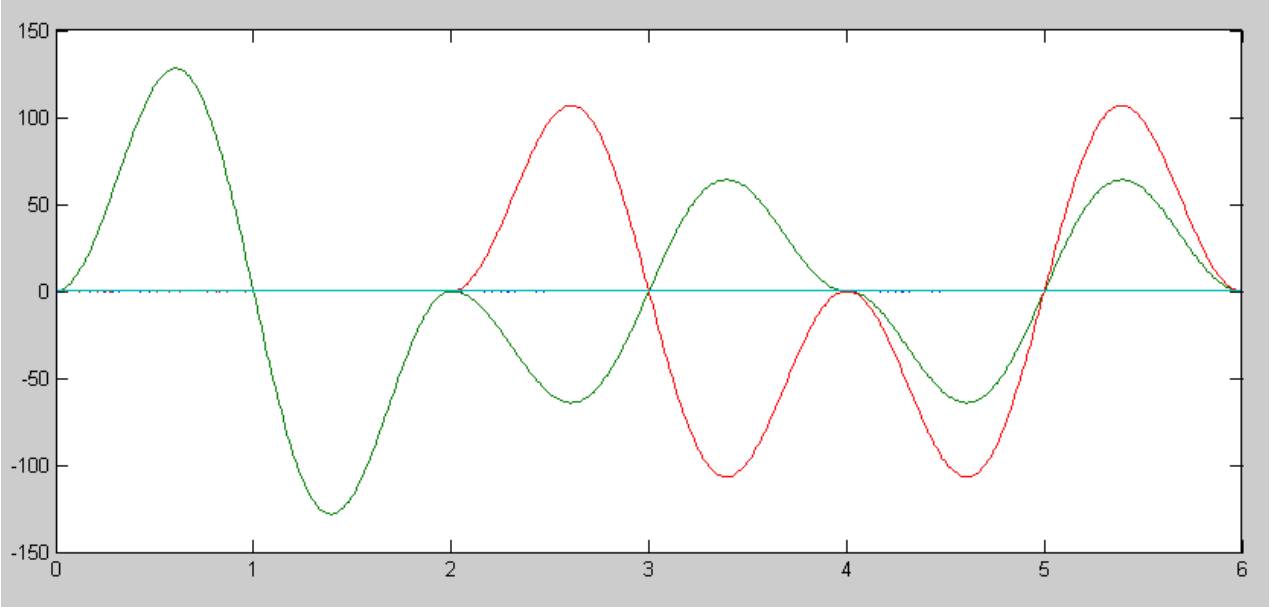
The spline function then becomes:

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

   t = [0.01:0.01:T] / T;
   t1 = (1 - cos(pi*t)) / 2;
   a = -2*(t1.^3) + 3*(t1.^2);

   Y = [];

   for i=1:length(a)
       Y = [Y, (1-a(i))*P0 + a(i)*P1];
       end

   end
```

The tip position and acceleration is then:



Tip position vs. time for cosine & cubic interpolation

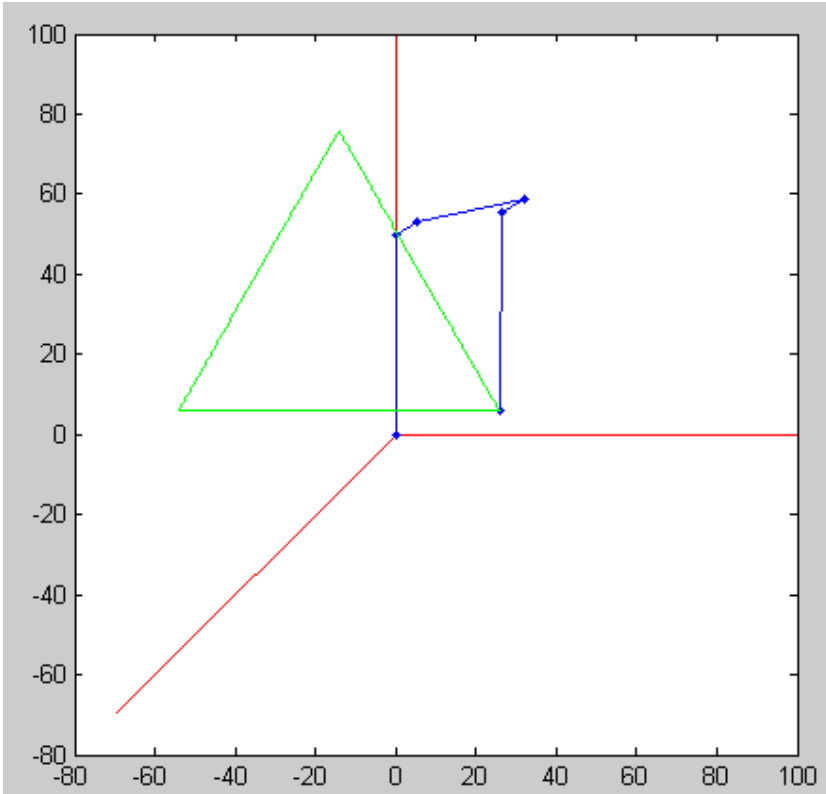


Tip acceleration vs. time for cosine & cubic interpolation

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:



Desired Path for the PUMA robot

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,

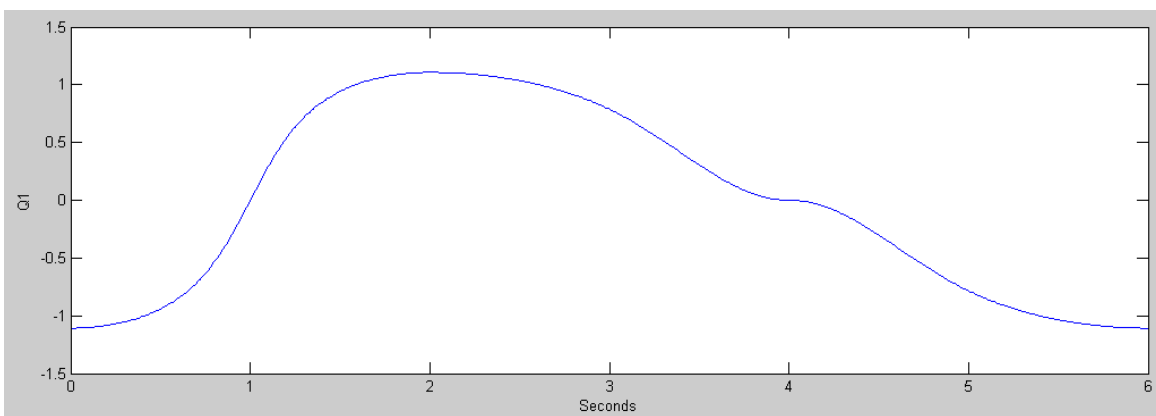
- Define where the robot should be every 10ms
- Use a spline curve fit for X, Y, and Z from point to point
- Use the same spline curve fit so that the resulting path is a straight line

Matlab Code:

```
P0 = [20 ; -30 ; 30 ; 1];
P1 = [20 ;  30 ; 30 ; 1];
P2 = [20 ;   0 ; 80 ; 1];

P01 = Spline(P0, P1, 2);
P12 = Spline(P1, P2, 2);
P20 = Spline(P2, P0, 2);

TIP = [P01, P12, P20];

Q = [];

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

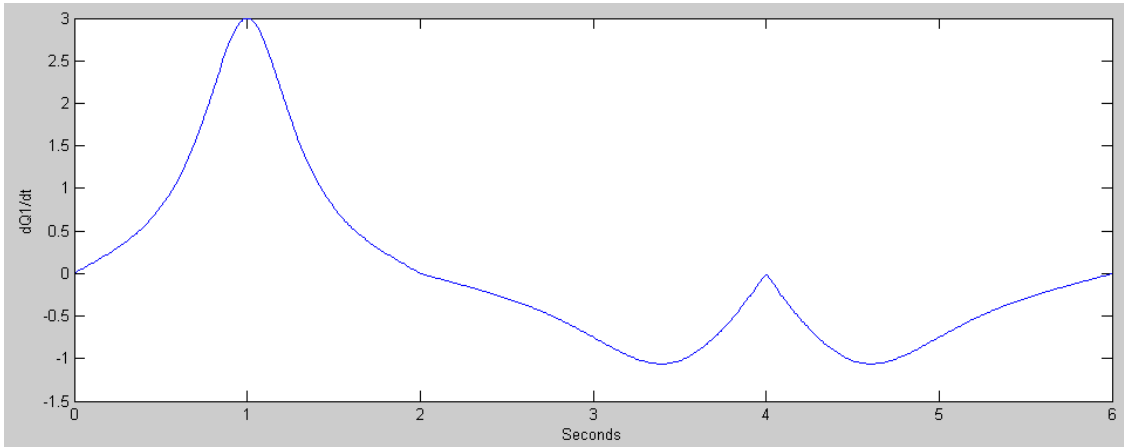
Once done, plot the resulting joint angles, velocities, and accelerations. Just plot angle Q1 for convience here:

```
Q1 = Q(1,:);
Q2 = Q(2,:);
Q3 = Q(3,:);
t = [1:length(Q1)] * 0.01;
plot(t,Q1);
xlabel('Seconds');
ylabel('Q1')
```



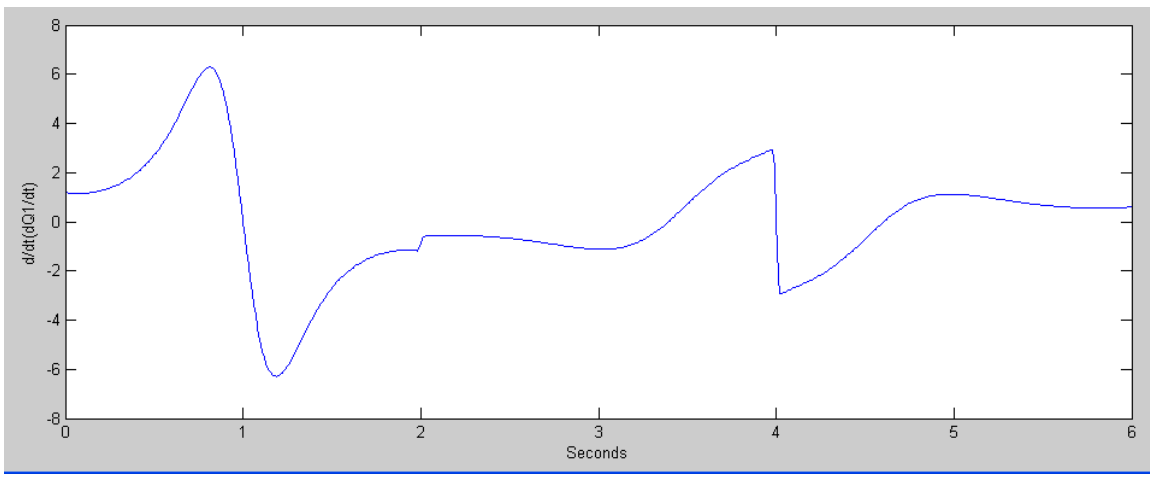
Joint Angle Q1 vs. Time for tracing out a triangle:

```
dQ1 = derivative(Q1);
plot(dQ1,t);
xlabel('Seconds');
ylabel('dQ1/dt')
```



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

```
ddQ1 = derivative(dQ1);
plot(t,ddQ1)
xlabel('Seconds');
ylabel('d/dt(dQ1/dt)')
```



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

Note that by making the tip position twice differentiable, the resulting joint angles are also twice differentiable.