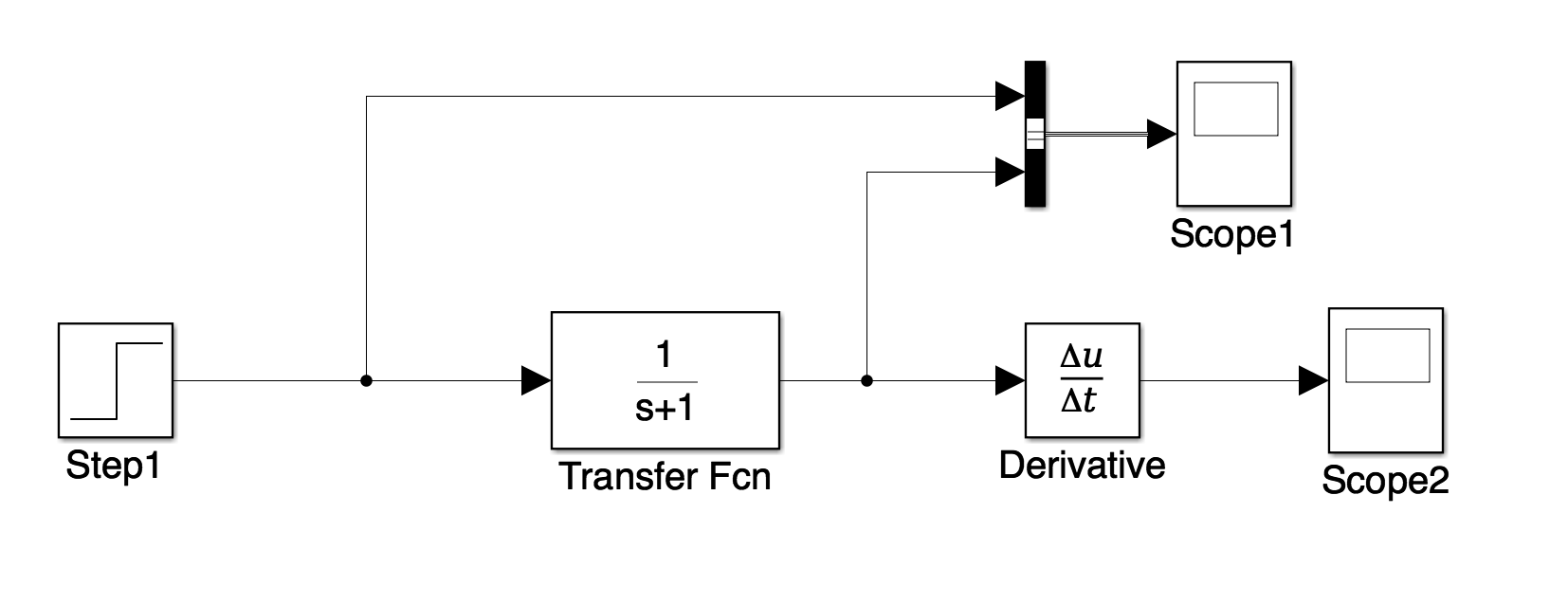
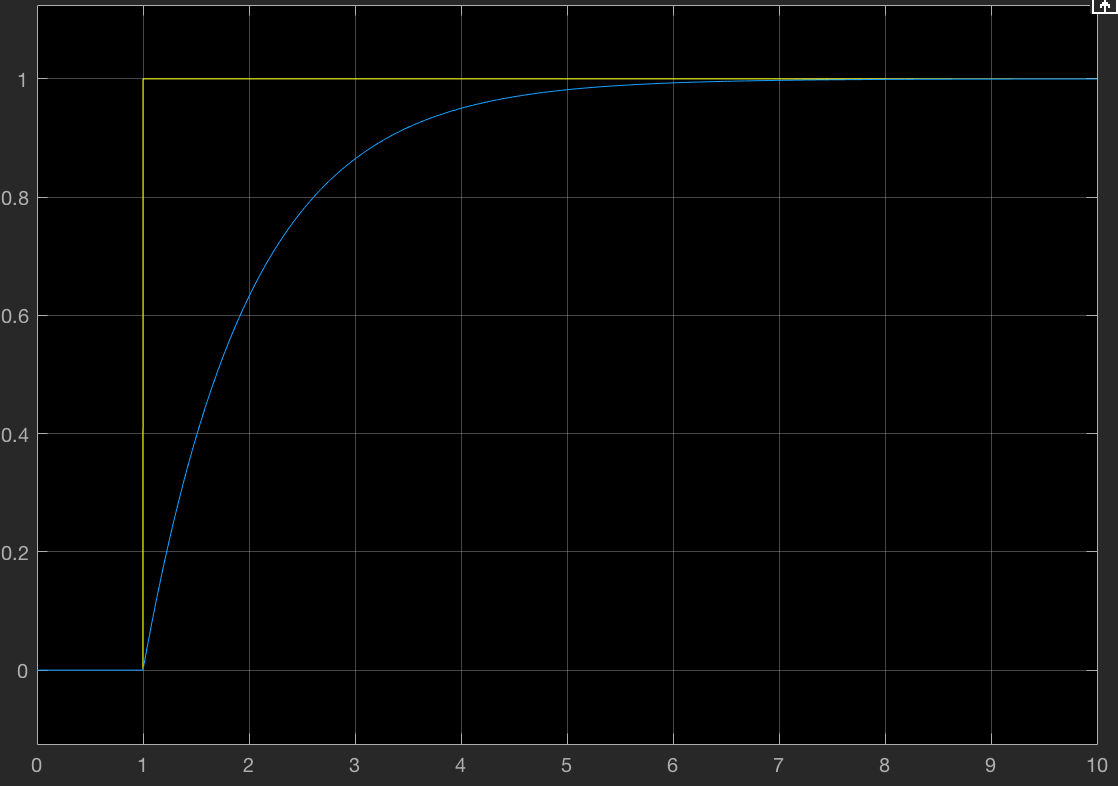
(a)

Use Laplace Transfer:



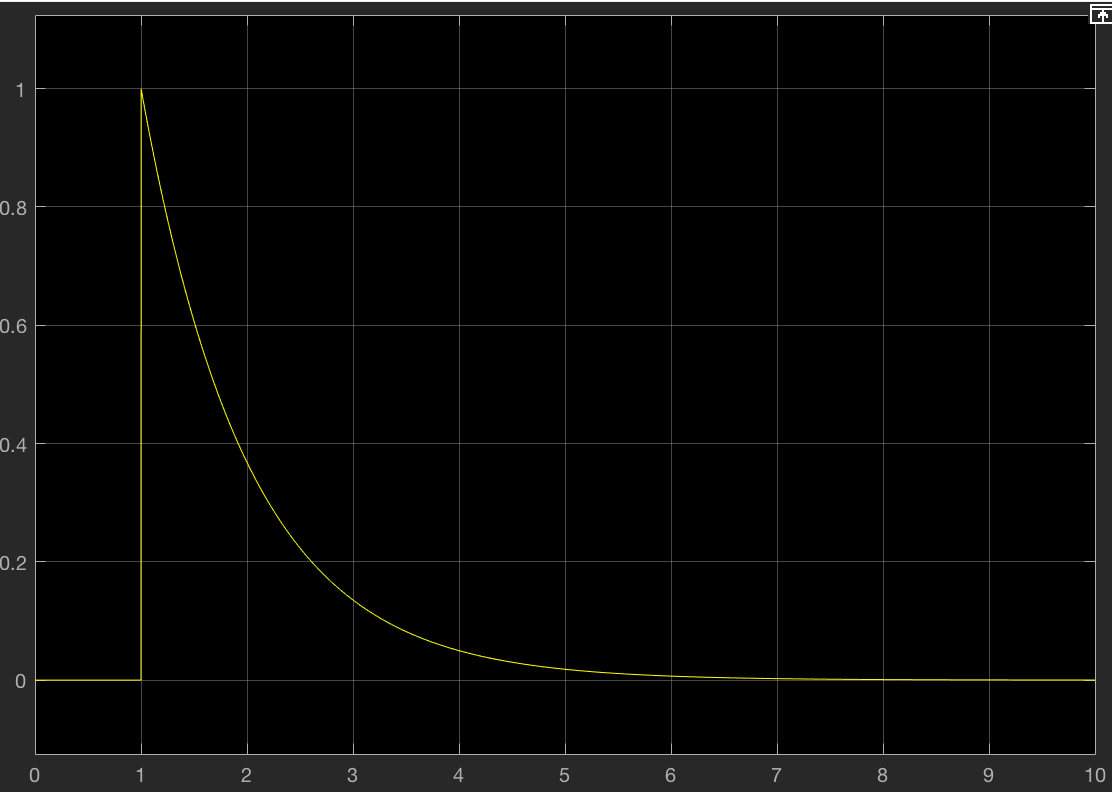
The lines from scope1:

The yellow line represents input. The blue line represents position.



The lines from scope2:

The yellow line represents velocity.

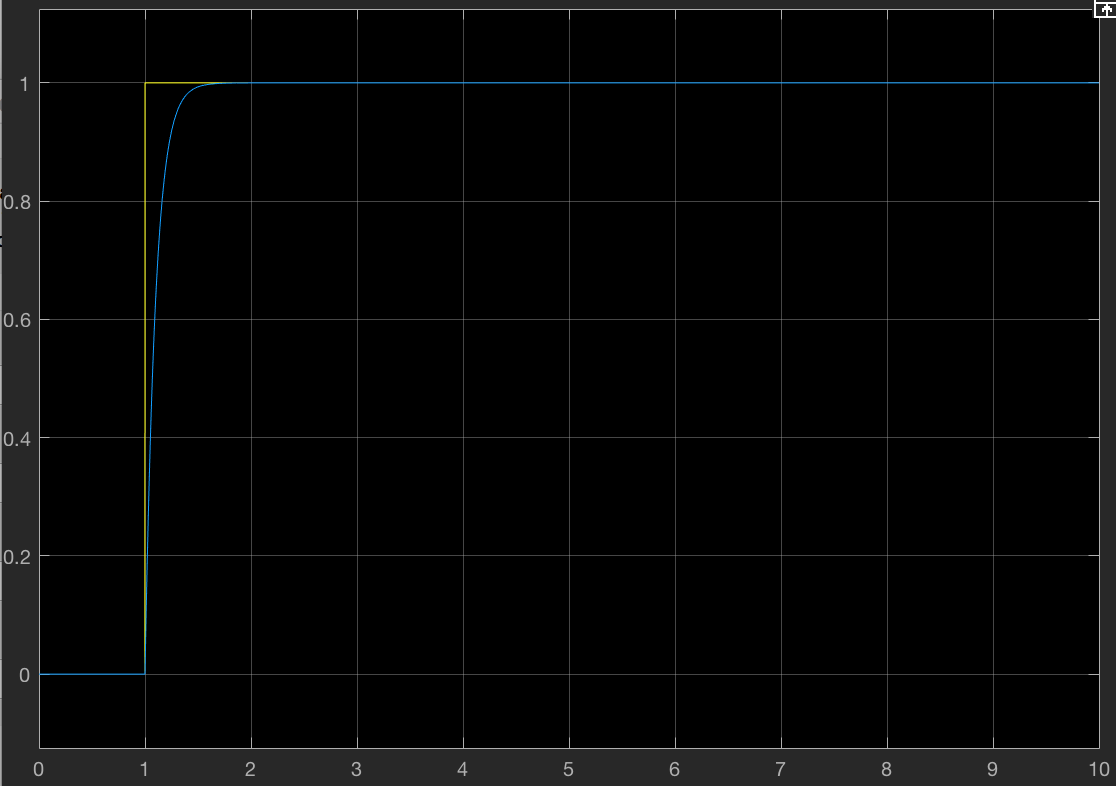


(b)

let

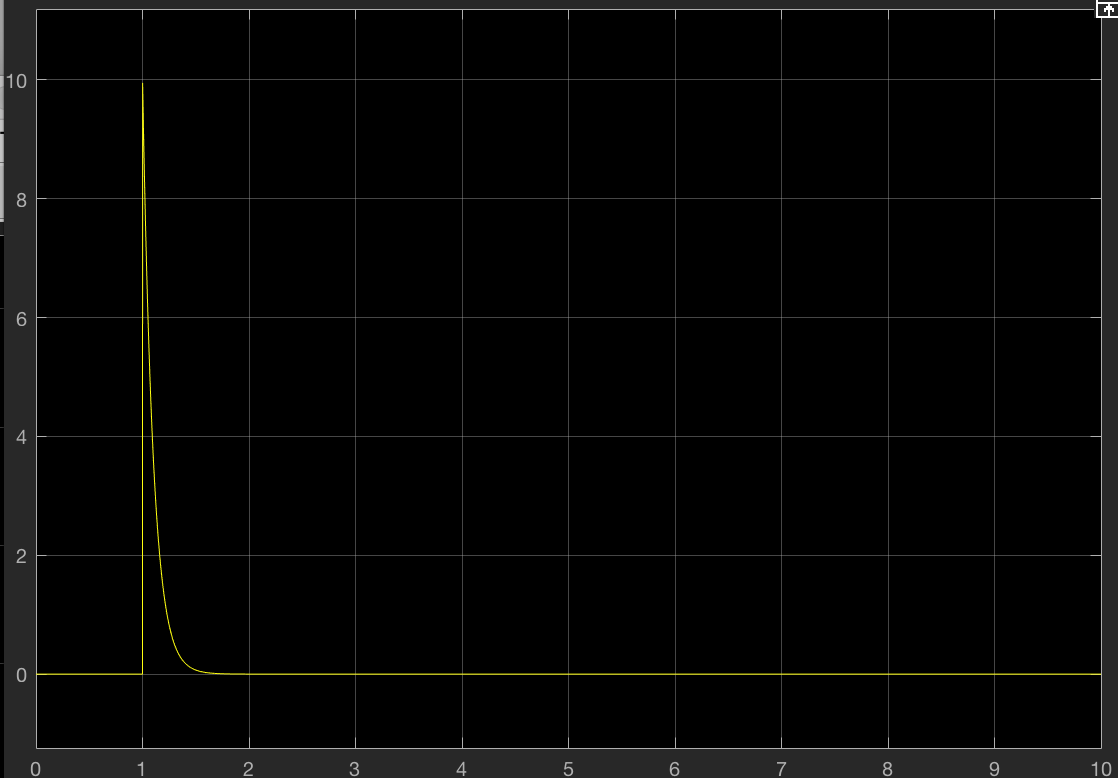
The lines from scope1:

The yellow line represents input. The blue line represents position.



The lines from scope2:

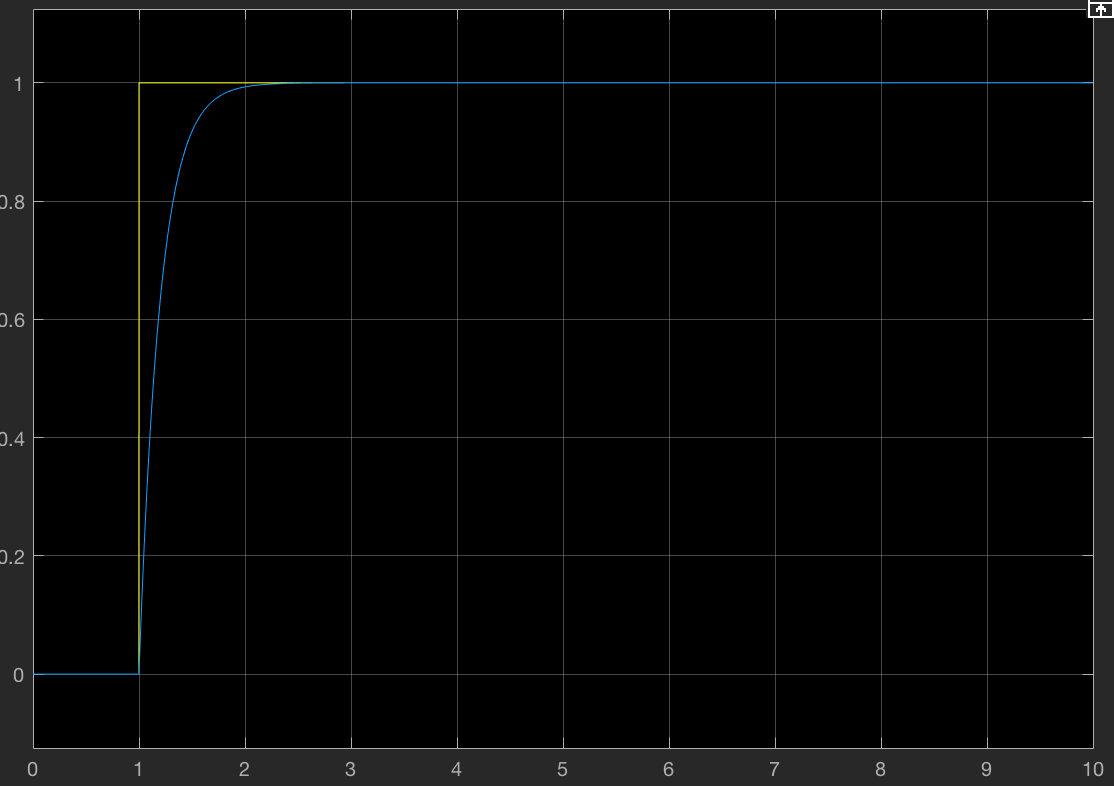
The yellow line represents velocity.



when :

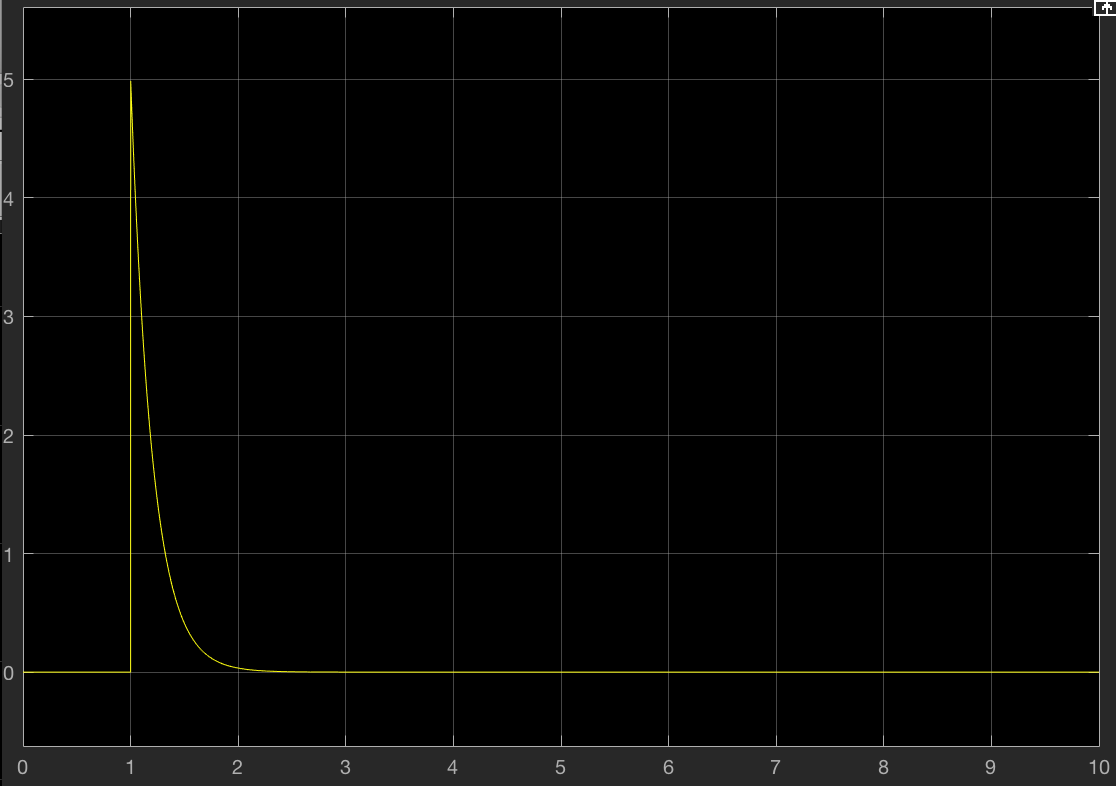
The lines from scope1:

The yellow line represents input. The blue line represents position.



The lines from scope2:

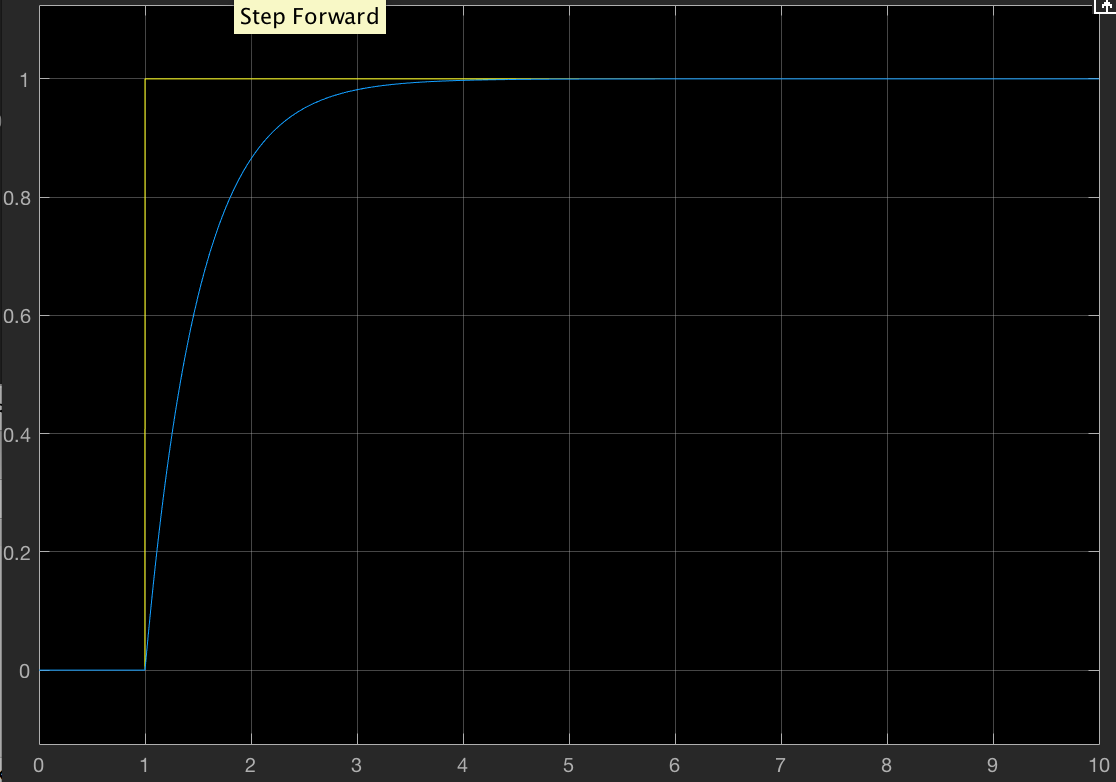
The yellow line represents velocity.



when ,

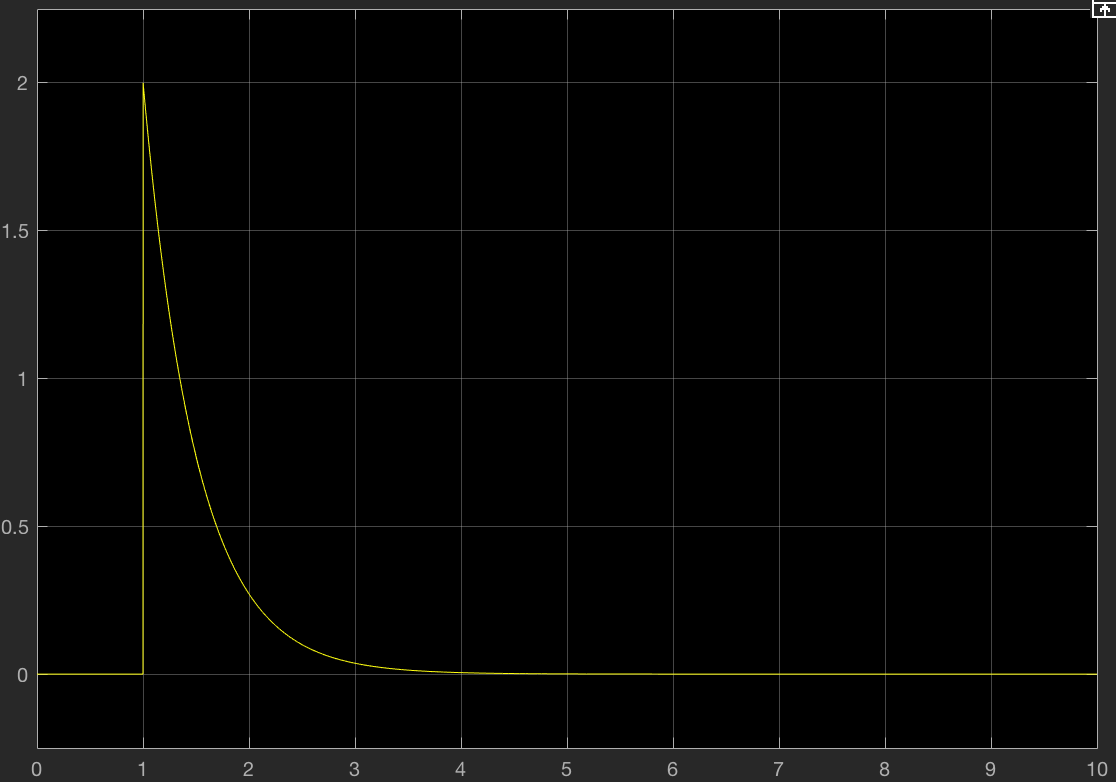
The lines from scope1:

The yellow line represents input. The blue line represents position.



The lines from scope2:

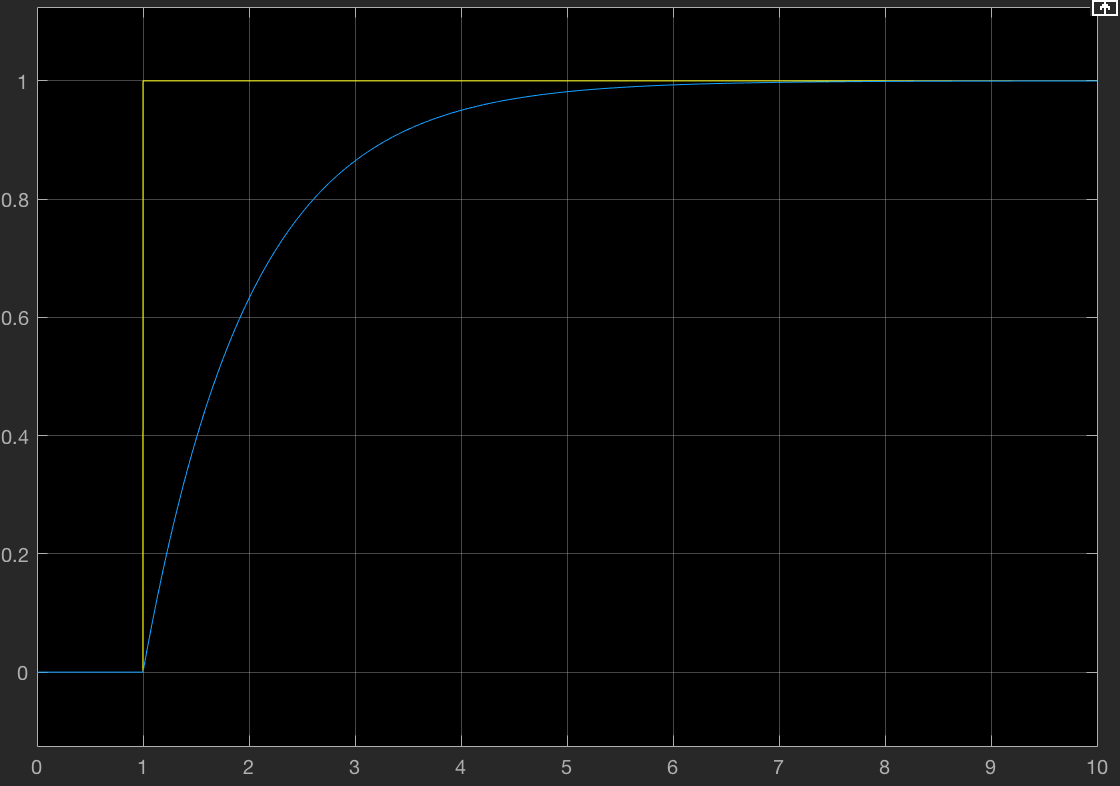
The yellow line represents velocity.



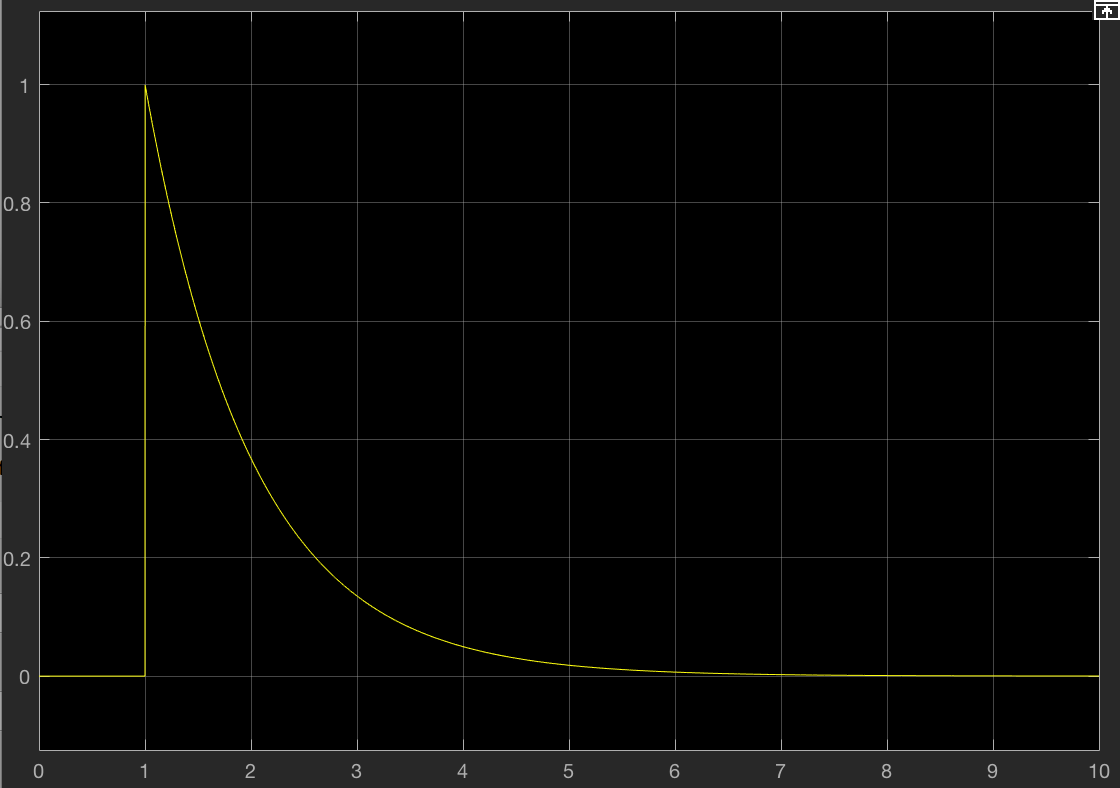
when :

The lines from scope1:

The yellow line represents input. The blue line represents position.



The yellow line represents velocity.



(c)

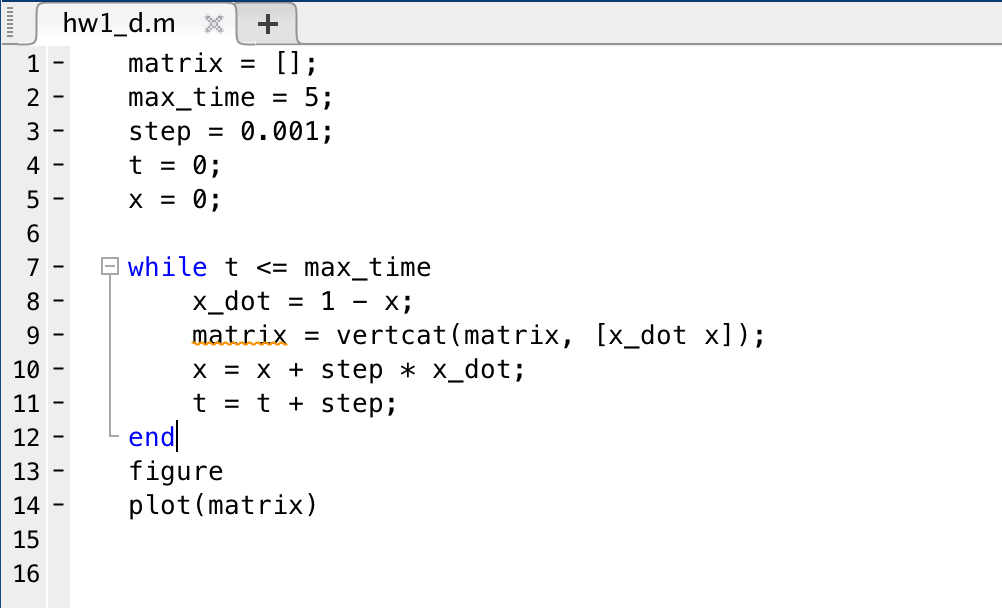
pros:

1. Easy to understand and implement
2. Easy to adjust the system through changing the parameters

cons:

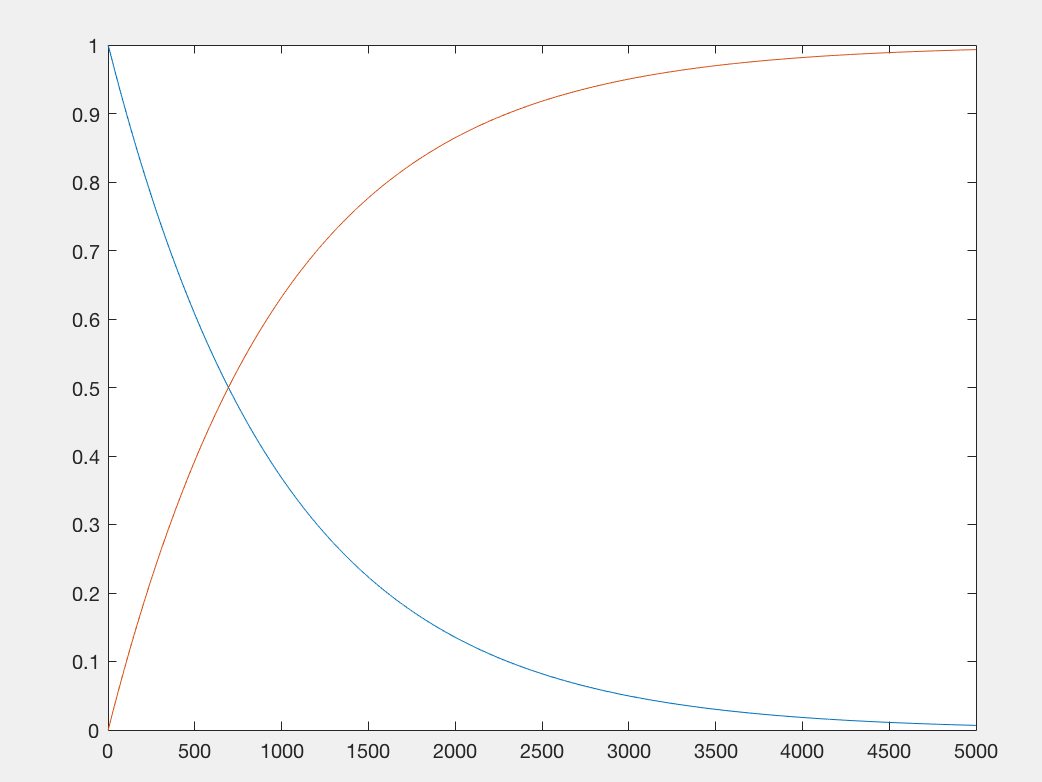
1. System need to get a high speed immediately at beginning, which means the acceleration is extremely large at the beginning, so it is a little impractical.
2. Only approximately approach to the goal, cannot 100% approach to the goal, even though it is really closed to the goal at last.

(d)



Plot:

The red line represents the position. The blue line represents the velocity.

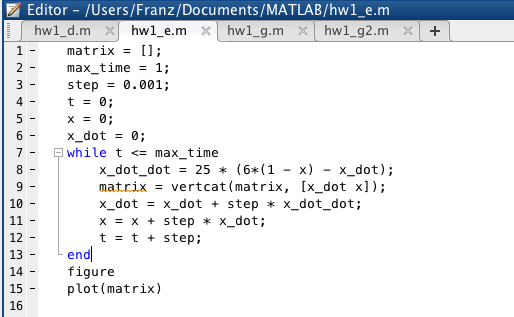


Difference:

For the position, it will move faster when using Euler method than 1a)

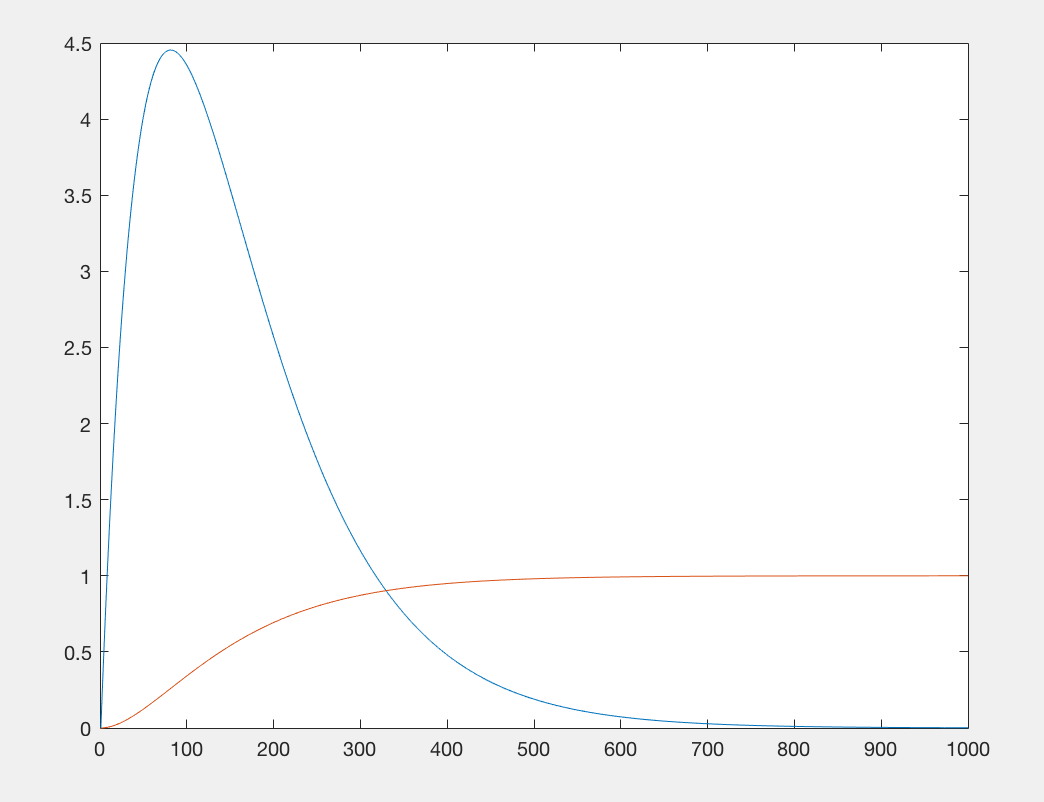
For the velocity, it will slow down faster when using Euler method than 1a)

(e)



Plot:

The red line represents the position. The blue line represents the velocity.



difference:

1. The speed is not from 1 and then decrease, but from 0 and then increase and then decrease
2. It takes less time to make speed approach close to 0 than before
3. The position approach to destination faster than before

Pros:

1. The speed is not from 1, so it is more practical in real life
2. It is faster to approach equilibrium state

Cons:

1. It will reach a larger maximum value of speed
2. The acceleration at the beginning is still very large

(f)

(g)

global xf xf\_dot xf\_dot\_dot

xf = 1;

xf\_dot = 0;

xf\_dot\_dot = 0;

matrix = [];

max\_time = 1;

step = 0.001;

t\_togo = max\_time;

x = 0;

x\_dot = 0;

x\_dot\_dot = 0;

matrix = vertcat(matrix, [x\_dot x]);

while t\_togo >= 0

    x = x + step \* x\_dot;

    x\_dot = x\_dot + step \* x\_dot\_dot;

    x\_dot\_dot\_dot = getX\_dot\_dot\_dot(x, x\_dot, x\_dot\_dot, t\_togo)

    x\_dot\_dot = x\_dot\_dot + step \* x\_dot\_dot\_dot;

    matrix = vertcat(matrix, [x\_dot x]);

    t\_togo = t\_togo - step;

end

figure

plot(matrix)

function res = getC0(x)

res = x;

end

function res = getC1(x\_dot)

res = x\_dot;

end

function res = getC2(x\_dot\_dot)

res = x\_dot\_dot / 2.0;

end

function res = getC3(x, x\_dot, x\_dot\_dot, t)

res1 = 20\*getX(x, x\_dot, x\_dot\_dot, t) - 8\*getY(x, x\_dot, x\_dot\_dot, t) + getZ(x, x\_dot, x\_dot\_dot, t);

res = res1 / 2.0;

end

function res = getC4(x, x\_dot, x\_dot\_dot, t)

res1 = -15\*getX(x, x\_dot, x\_dot\_dot, t) + 7\*getY(x, x\_dot, x\_dot\_dot, t) - getZ(x, x\_dot, x\_dot\_dot, t);

res = res1 / t;

end

function res = getC5(x, x\_dot, x\_dot\_dot, t)

res1 = 12\*getX(x, x\_dot, x\_dot\_dot, t) - 6\*getY(x, x\_dot, x\_dot\_dot, t) + getZ(x, x\_dot, x\_dot\_dot, t);

res2 = 2\*t\*t;

res = res1 / res2;

end

function res = getX(x, x\_dot, x\_dot\_dot, t)

global xf

res1 = 2\*xf - 2\*x - 2\*t\*x\_dot - x\_dot\_dot\*t\*t;

res2 = 2.0 \* t\*t\*t;

res = res1 / res2;

end

function res = getY(x, x\_dot, x\_dot\_dot, t)

global xf\_dot

res1 = xf\_dot - x\_dot - t\*x\_dot\_dot;

res2 = t\*t;

res = res1 / res2;

end

function res = getZ(x, x\_dot, x\_dot\_dot, t)

global xf\_dot\_dot

res1 = xf\_dot\_dot - x\_dot\_dot;

res2 = t;

res = res1 / res2;

end

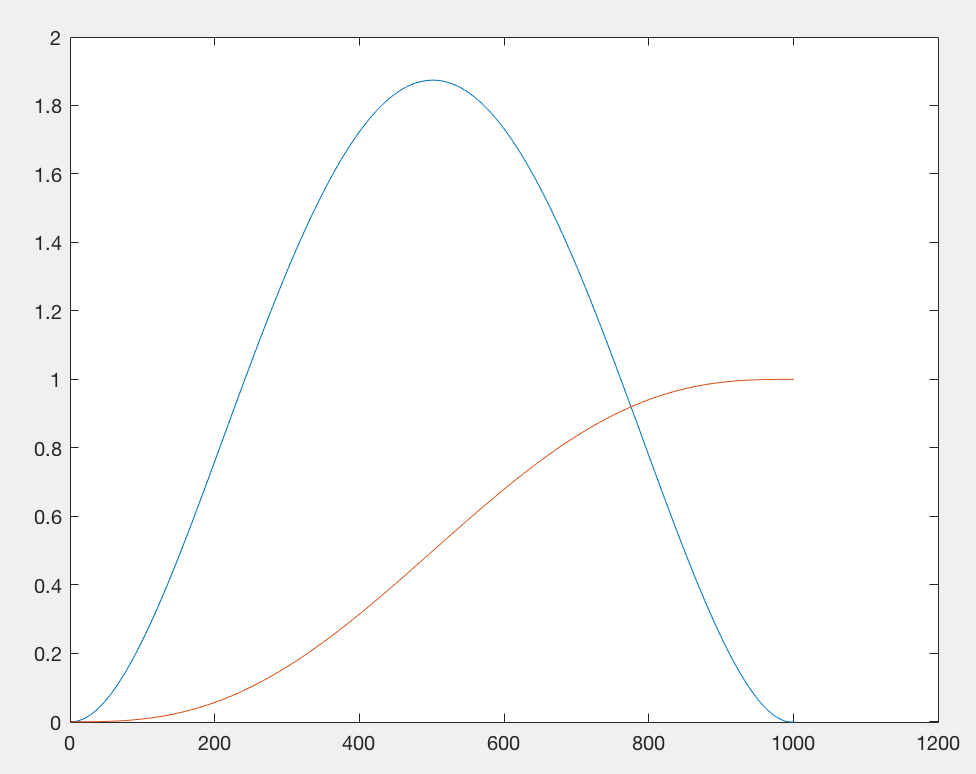
function res = getX\_dot\_dot\_dot(x, x\_dot, x\_dot\_dot, t)

res = 6\*getC3(x, x\_dot, x\_dot\_dot, t) ;

end

Plot:

The red line represents the position. The blue line represents the velocity.



Comparison about results:

1. For position, min-jerk algorithm will arrive destination at 1s, as fast as the method in (e)
2. For velocity, the maximum speed of min-jerk algorithm is smaller than method in (e)
3. In min-jerk, the velocity reaches 0 at 1s, as fast as the method in (e)

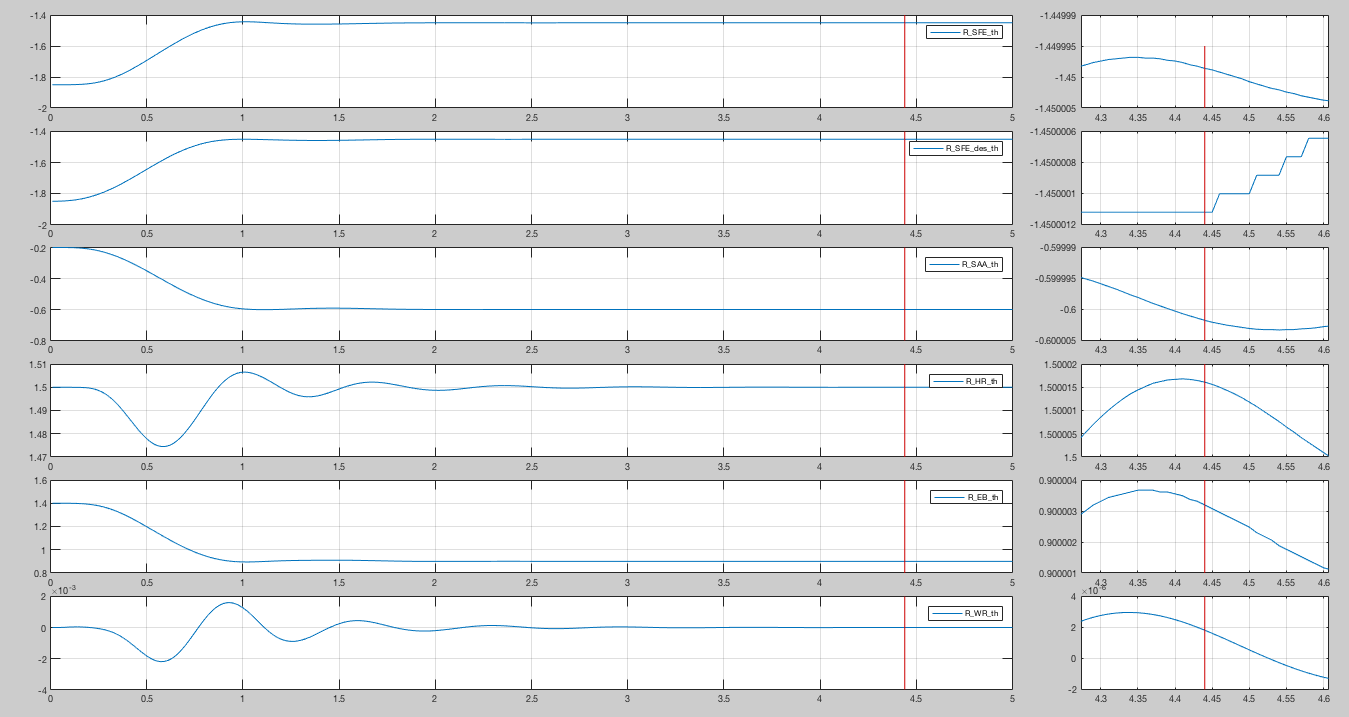
Similarity:

1. The lines of speed in both graphs are increase from 0 and then decrease to 0 smoothly
2. The lines of position in both graphs are increase smoothly and reach 1 at 1s.

Differences:

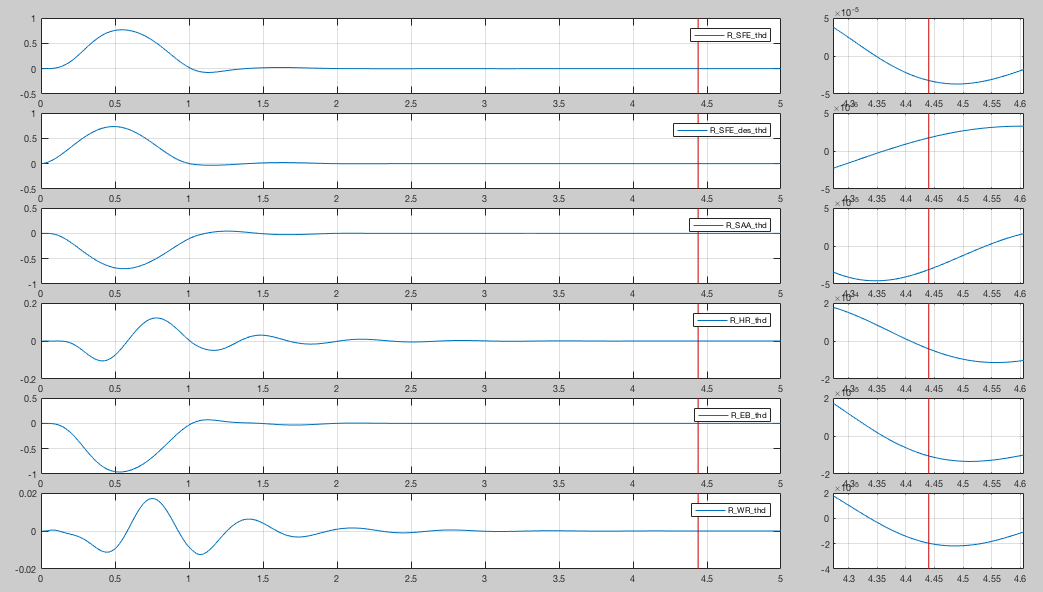
1. In this question, speed increase slower
2. In this question, the maximum number of speed is smaller
3. In this question, the line of speed is about symmetric
4. In this question, the position increase slower

(h)



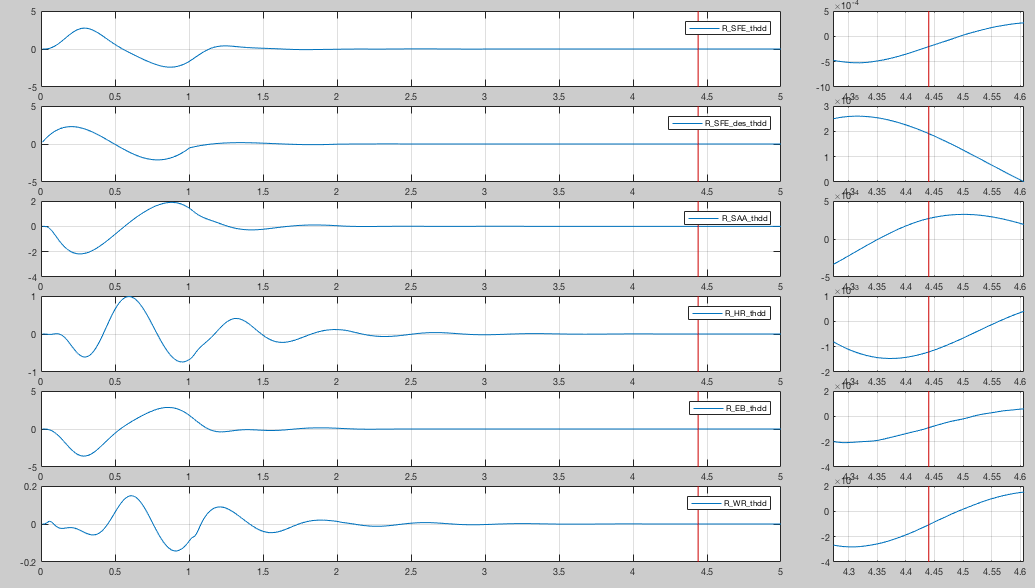
The quality of position of R\_SFE, R\_SFE\_des, R\_SAA, R\_EB are really good. They move pretty smooth and arrive right position really quick

The quality of position of R\_HR, R\_WR are not so good. They moves backwards and then forwards before 1s. But they doesn’t arrive the equilibrium at 1s. So after 1s, they are a little shaking around the equilibrium position. And finally arrive right position.



The quality of speed of R\_SFE, R\_SFE\_des, R\_SAA, R\_EB are really good. They speed up gradually at beginning and slow down at the end, and reach 0 a little after 1s.

The quality of speed of R\_HR, R\_WR are not so good. The speed of them is floating a lot when moving, and reaches 0 at around 2s.

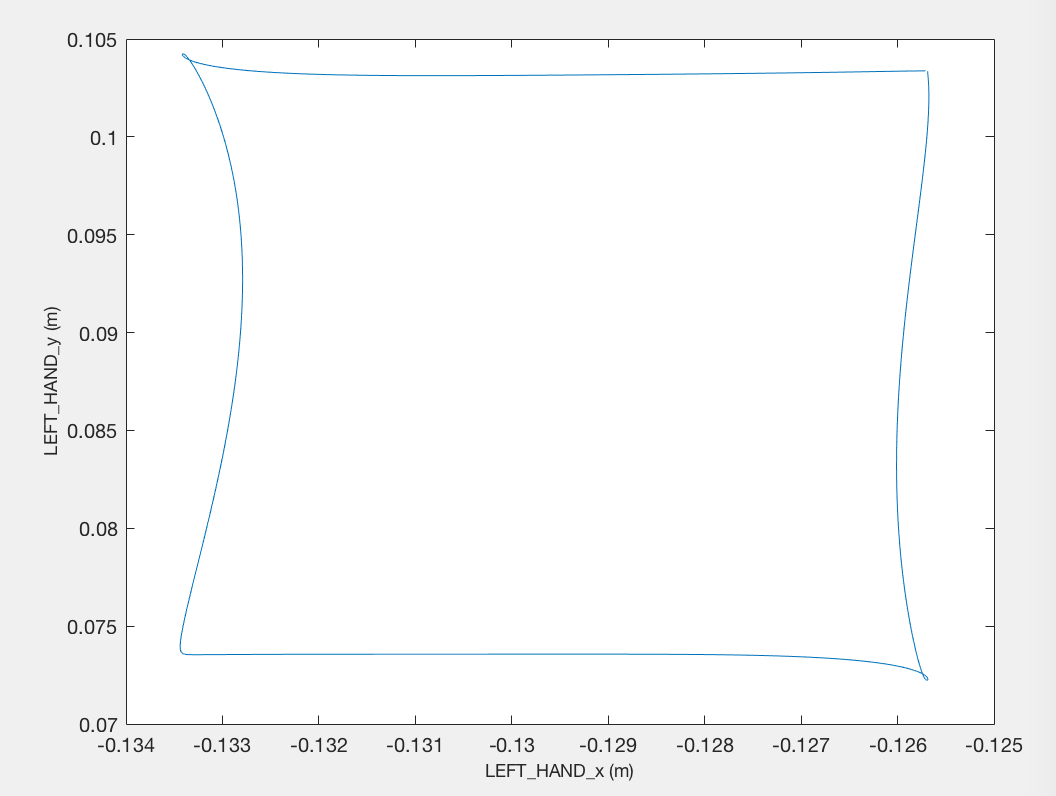


The quality of acceleration of R\_SFE, R\_SFE\_des, R\_SAA, R\_EB are really good. The acceleration float around 0 twice and then reach 0 a little after 1s.

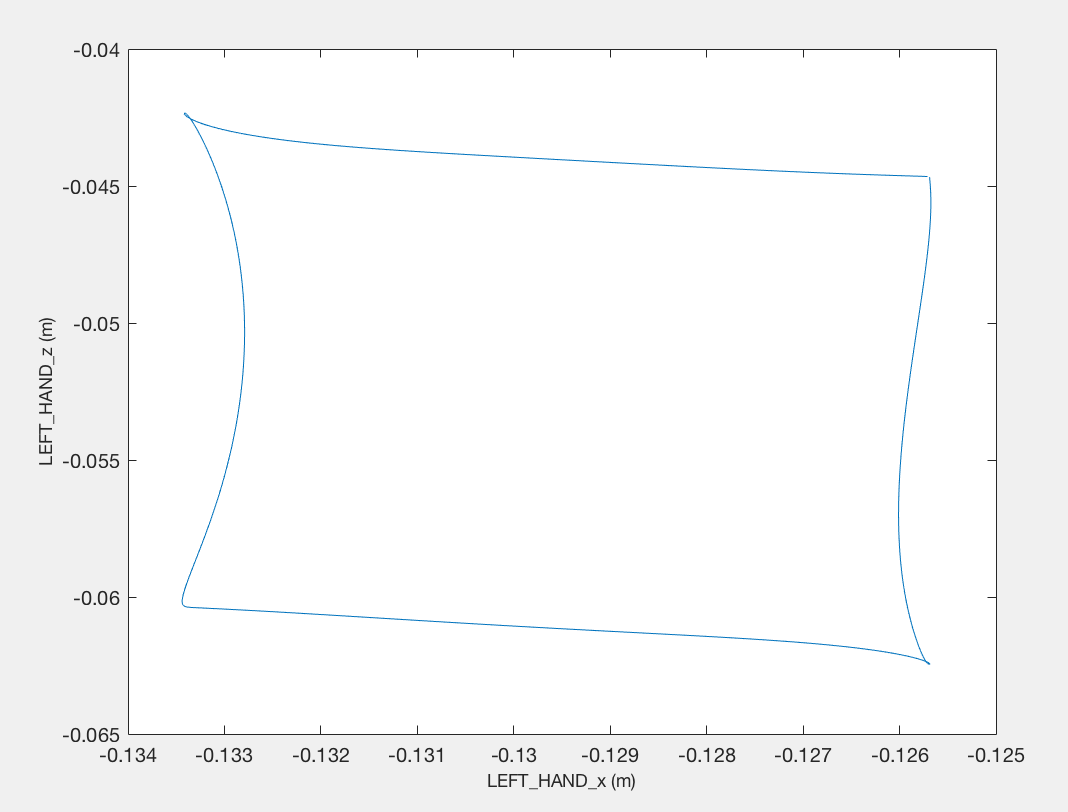
The quality of acceleration of R\_HR, R\_WR are not so good. The acceleration float around 0 many times before reaching 0 at around 2s.

(i)

The plot of x-y:



The plot of x-z:



The quality of square in x-y and x-z is pretty good.

In plot of x-y:

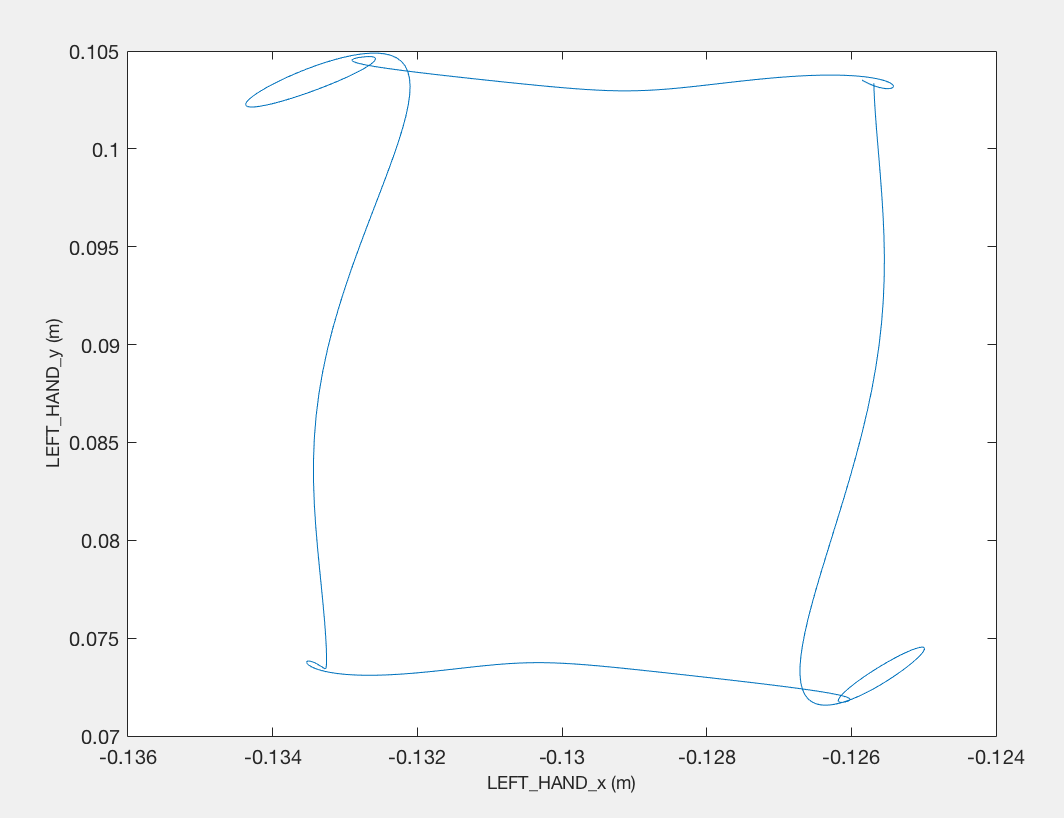
1. The top-line is almost straight and horizon.
2. The left line is basically vertical, but a little curve.
3. The bottom-line is almost straight and horizon
4. The right line is almost straight and vertical.

In plot of x-z:

1. The top-line is almost straight and basically horizon.
2. The left line is basically vertical, but a little curve.
3. The bottom-line is almost straight and basically horizon
4. The right line is almost vertical and basically straight.

(j)

The plot of x-y:



The plot of x-z:

