Variable Structures Systems (VSS)

Variable Structures Systems is another type of controller which uses full-state feedback to make a system behave a desired way. The result is similar to pole placement: the resulting control law is

$$U = \alpha \cdot sign(K_r R - K_x X)$$

where α is a constant determining the maximum and minimum of U. A slight variation is saturating control:

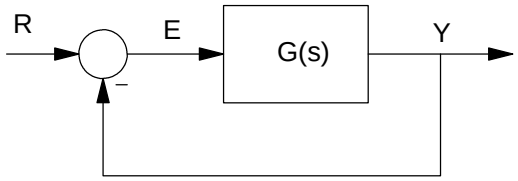
$$U = \text{limit}(-\alpha, K_r R - K_x X, \alpha)$$

The problem with each of these is the control law is nonlinear. To prove stability, we need something other that eigenvalues since eigenvalues do not apply for nonlinear systems.

Unfortunately, there are only three proofs for stability for nonlinear systems:

- Hyperstability,
- H-infinity, and
- Lyapunov.

For the first two, consider the following feedback system:



If the phase shift of G(s) is 180 degrees, you have positive feedback. If the gain of G(s) is 'a' at this frequency, the closed-loop gain is:

$$\text{gain} = a + a^2 + a^3 + a^4 + \ldots$$

This is only finite when a < 1.

In order for the closed-loop system to be stable, the gain must be less than one when the phase shift is 10 degrees.

This leads to two ways to prove closed-loop stability:

H-Infinity: If you can prove the gain of G(s) is always less than one, the closed-loop system must be stable. This is useful when you want to determine how much a system can be perturbed and remain stable. Ideally, the perturbations are zero (less than one). As long as the perturbation are small enough, stability won't be affected (the loop gain is less than one).

Hyperstability: If the phase shift of G(s) never reaches 180 degrees, the closed-loop system must be stable. This is used in one form of model-reference adaptive control.

Lyapunov Stability: Define an energy function which is positive definite. If you can show the change in energy is always negative definite, the system must be stable.

Lyapunov stability is what's used for VSS controllers.

Lyapunov Stability:

Example 1: Use Lyaponov methods to prove the following system is stable:

$$\dot{x} = -3x$$

Step 1: Define a positive definite energy function:

$$V = \tfrac{1}{2}x^2$$

Step 2: Check that the change in energy is negative definite:

$$\dot{V} = x\dot{x}$$

$$\dot{V} = x(-3x)$$

$$\dot{V} = -3x^2$$

This system is stable.

Example 2: Find the range of k which results in a stable system:

$$\dot{x} = -3x + u$$

$$u = -kx$$

Step 1: Define an energy function:

$$V = \tfrac{1}{2}x^2$$

Step 2: Check that the change in energy is negative definite:

$$\dot{V} = x\dot{x}$$

$$\dot{V} = x(-3x - kx)$$

$$\dot{V} = -(3 + k)x^2$$

To be stable

$$3 + k > 0$$

$$k > -3$$

Example 3:

$$\dot{X} = AX + BU$$

Define a sliding surface

$$\sigma = CX$$

Define an energy function

$$V = \tfrac{1}{2}\sigma^T\sigma > 0$$

Pick U so that $\dot{V}$ is negative definite:

$$\dot{V} = \sigma^T\dot{\sigma} < 0$$

Substituting:

$$(CX)^T\left(C\dot{X}\right) < 0$$

$$X^TC^T(CAX + CBU) < 0$$

If

$$|CBU| > |CAX|$$

$$CB > 0$$

then

$$X^TC^T(CBU) < 0$$

$$X^TC^TU < 0$$

Let

$$U = -\alpha \cdot sign(CX)$$

where

$$|CB\alpha| > |CAX|$$

If you add in a set point (R), you get

$$U = -\alpha \cdot sign(CX - K_rR)$$

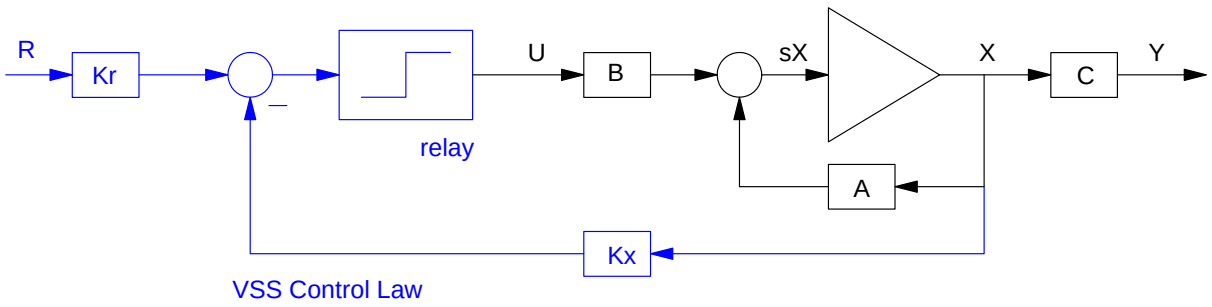
where Kr is picked so that

$$CX - K_rR = 0$$

at steady-state (i.e. Kr makes the DC gain zero to the fictitious output CX)

Note that this is the same as the control law

$$U = \alpha \cdot sign(K_r R - K_x X)$$



Example: Double Integrator:

$$\dot{X} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} X + \begin{bmatrix} 0 \\ 1 \end{bmatrix} U$$

Define the sliding surface to be

$$\sigma = \begin{bmatrix} 1 & 1 \end{bmatrix} X$$

Assume X is bounded by 10

$$|CBU| > |CAX|$$

$$|\alpha| > 10$$

Then

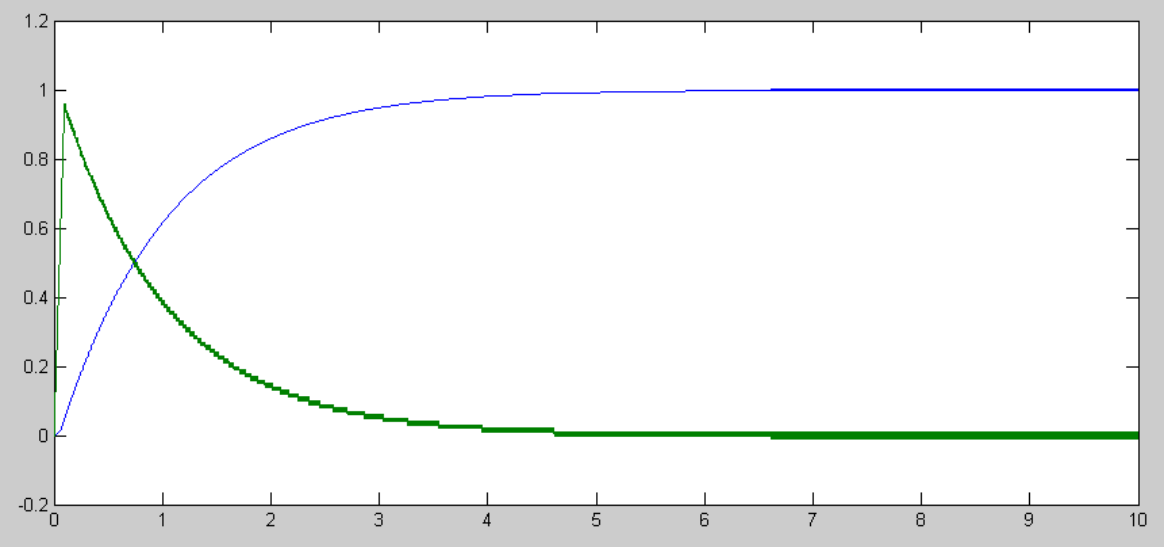
$$U = -10 \cdot sign(CX)$$

Adding in a reference

$$U = -10 \cdot sign((x - R) + (\dot{x}))$$

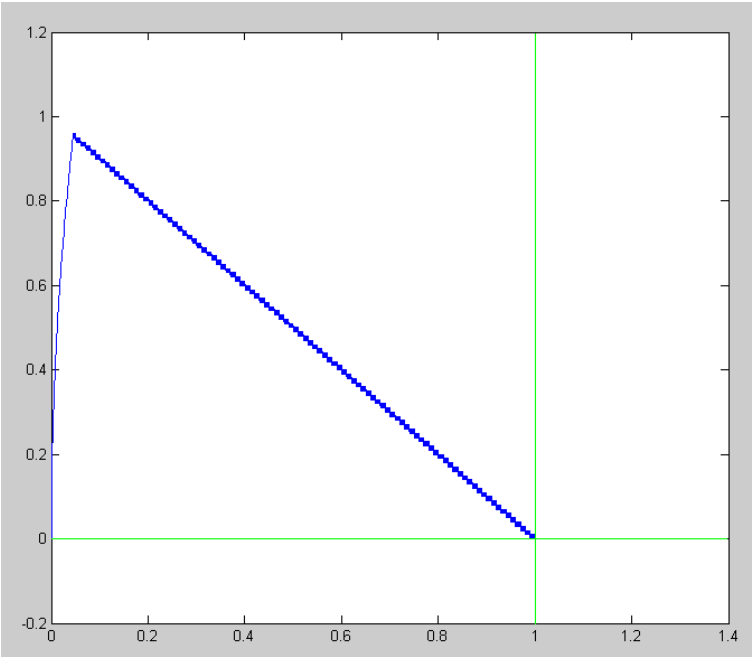
$$U = 10 \cdot sign(R - (x + \dot{x}))$$

The response for a step input is as follows. Note that the system behaves like a system with a pole at -1 (the zero in the transfer function from R to σ)



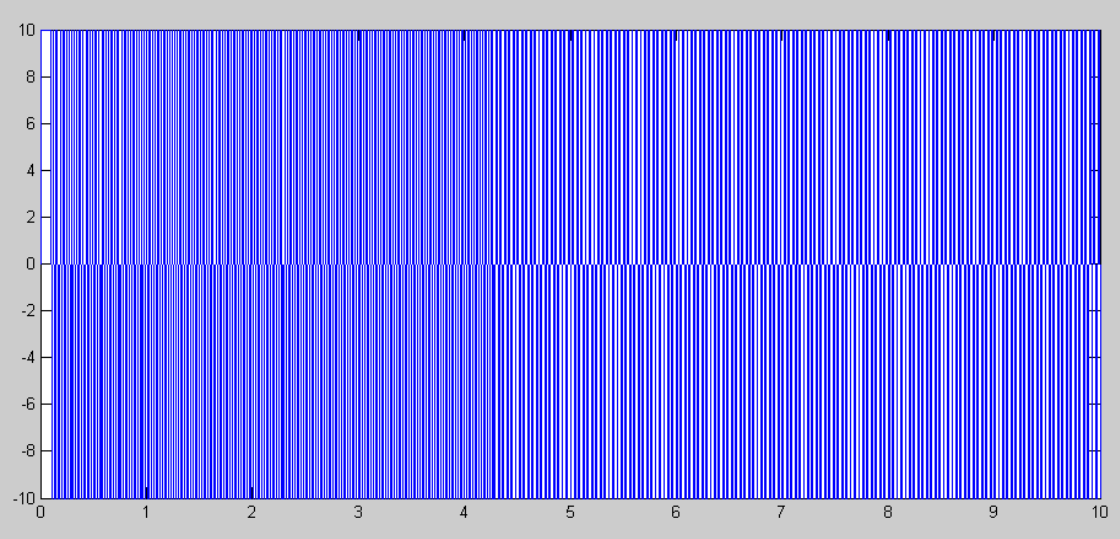
Step Response for a VSS controller with $\sigma = (s+1)X$

If you plot x vs. dx/dt, you get the phase plane (below) you get a sliding mode. The controller pushes the system to the sliding surface defined by the eigenvector associated with the eigenvalue of (s+1).



Phase Plane for y = (s + 1) x along with its sliding surface

If you plot the input, it chatters from -10 to +10 while you're on the sliding surface.



Input u(t). Note that it chatters from -10 to +10 while you're on the sliding surface.

Matlab Code:

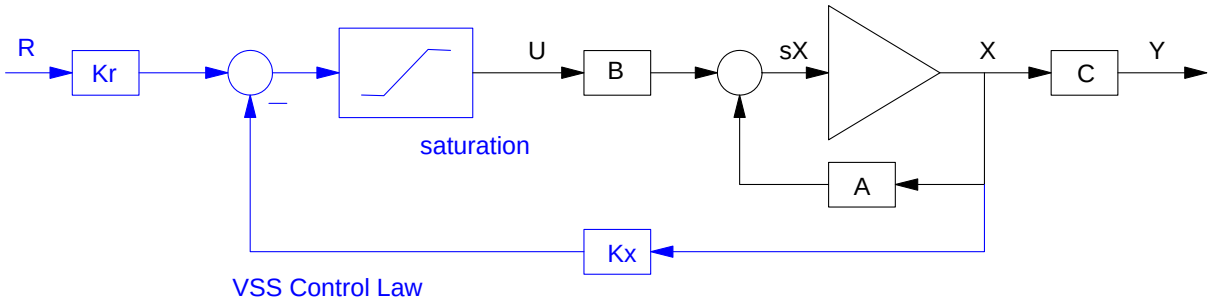
```
>> A = [0,1;0,0];
>> B = [0;1]
>> C = [1,0]
>> t = [0:0.01:10]';
>> Kx = [1,1];
>> DC = -C*inv(A-B*Kx)*B
>> Kr = 1/DC;

>> C = [1,0;0,1]
>> D = [0;0]

>> t = [0:0.001:5]';

>> [y, u] = step_vss(A,B,C,D, t, Kx, Kr);

% time response
>> plot(t,y)

% phase plane
>> plot(y(:,1), y(:,2));

% input
>> plot(t,u)
```

Saturating Control

Rather than using a relay function, a saturating function with a large gain results in

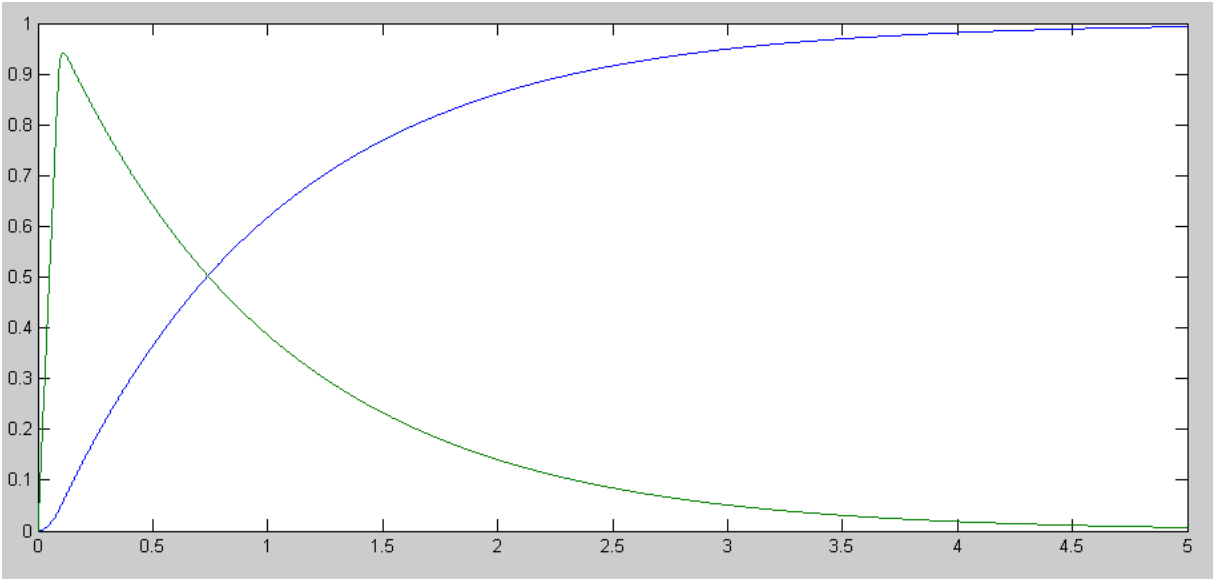
- Almost the same result (same sliding surface, same closed-loop response), but
- The input no longer chatters

$$U = \text{limit}(-10, k(K_r R - K_x R), 10)$$



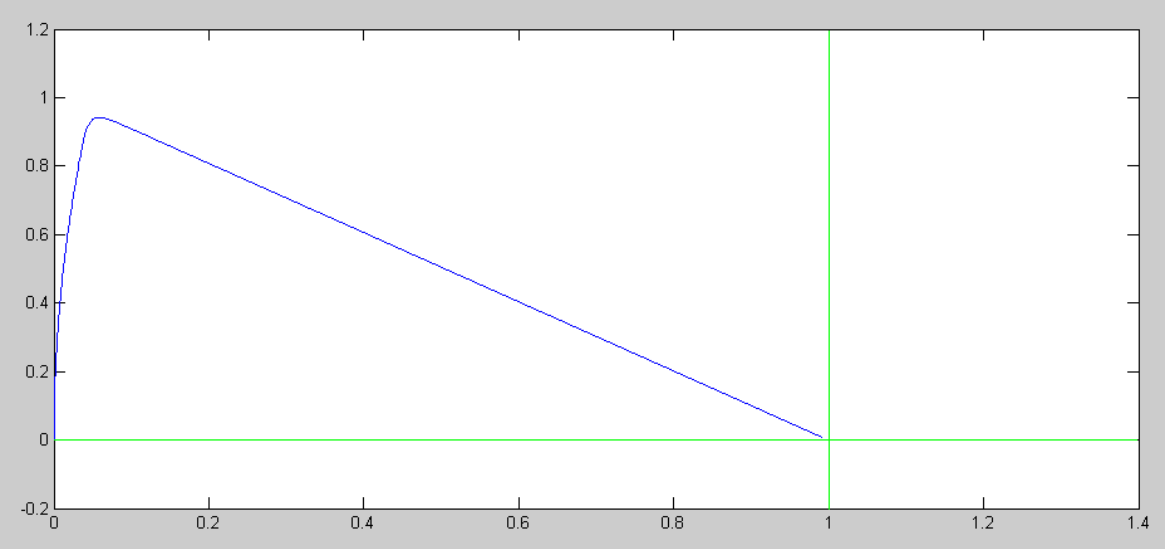
It's a really minor change but that results in an input that no longer chatters. Repeating the previous case with a saturating function results in the following:

The output for a step input is almost the same: it approaches the set point with a pole at s = -1:



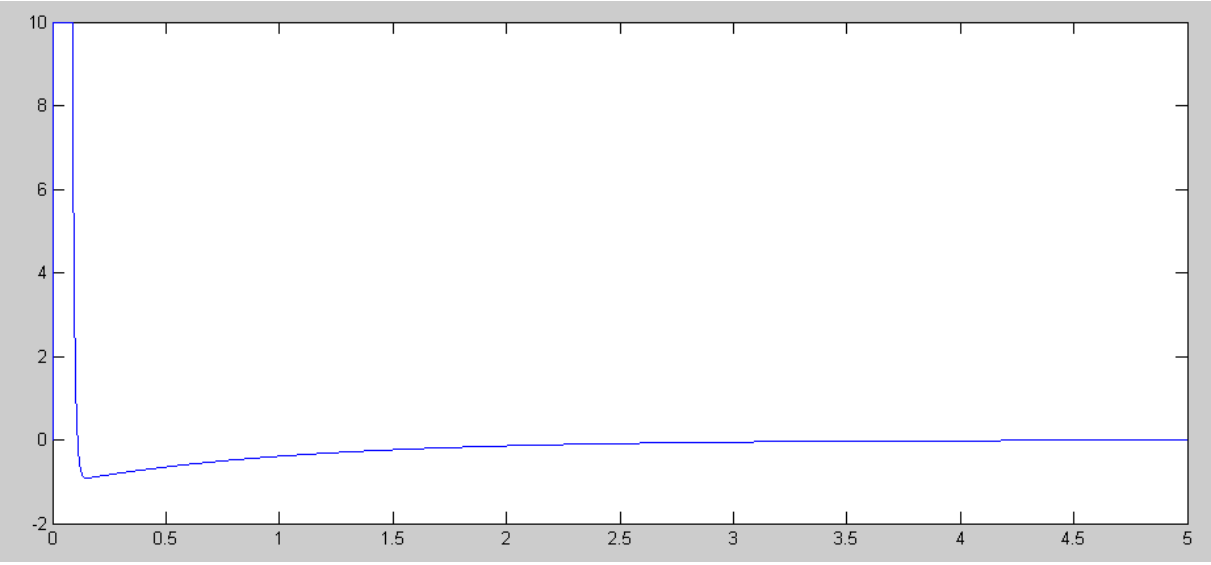
x (blue) and dx (green).
Saturating Control with sigma = (s+1)x

The phase plane shows a sliding mode corresponding to the eigenvector for a pole at s = -1:



Phase Plane: Saturating Control with sigma = (s+1)x

But, the input no longer chatters.



Input (U) for saturating control with sigma = (s+1)x

Note that the zeros determine

- The sliding surface and
- The closed-loop poles

VSS Control for an RC Filter (real zeros)

To illustrate how you can dictate the response by placing the zeros, consider the 4th-order heat equation:

$$sX = \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} X + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} U$$

```
>> A = [-2,1,0,0;1,-2,1,0;0,1,-2,1;0,0,1,-1]
>> B = [1;0;0;0]
```

To place the zero, convert to controller canonical form using Bass Gura:

```
N = length(A);

T1 = [];
for i=1:N
   T1 = [T1, (A^(i-1))*B];
end


P = poly(eig(A));
T2 = [];
for i=1:N
    T2 = [T2; zeros(1,i-1), P(1:N-i+1)];
end

T3 = zeros(N,N);
for i=1:N
    T3(i, N+1-i) = 1;
end

T = T1*T2*T3;
```

Check that this transform (T) does in fact take you to controller canonical form (Ac, Bc):

```
>> Ac = inv(T)*A*T

         0    1.0000         0         0
         0   -0.0000    1.0000    0.0000
         0    0.0000    0.0000    1.0000
   -1.0000  -10.0000  -15.0000   -7.0000

>> Bc = inv(T)*B

     0
     0
     0
     1
```

In controller canonical form the zeros are determined by the C matrix.

```
>> % poles at ( -1, -2, -3 )
>> poly([-1,-2,-3])

     1     6    11     6

>> Cc = [6,11,6,1]

     6    11     6     1
```

Checking: with output feedback, the closed-loop poles go to the zeros

```
>> eig(Ac-Bc*Cc*100)

 -101.0203
   -2.9899
   -1.9898
   -1.0000
```

Convert back to state-variable form with the linear transformation, T

```
>> C = [0,0,0,1];
>> D = 0;
>> Kx = Cz*inv(T)

    1.0000    1.0000    2.0000    2.0000
```

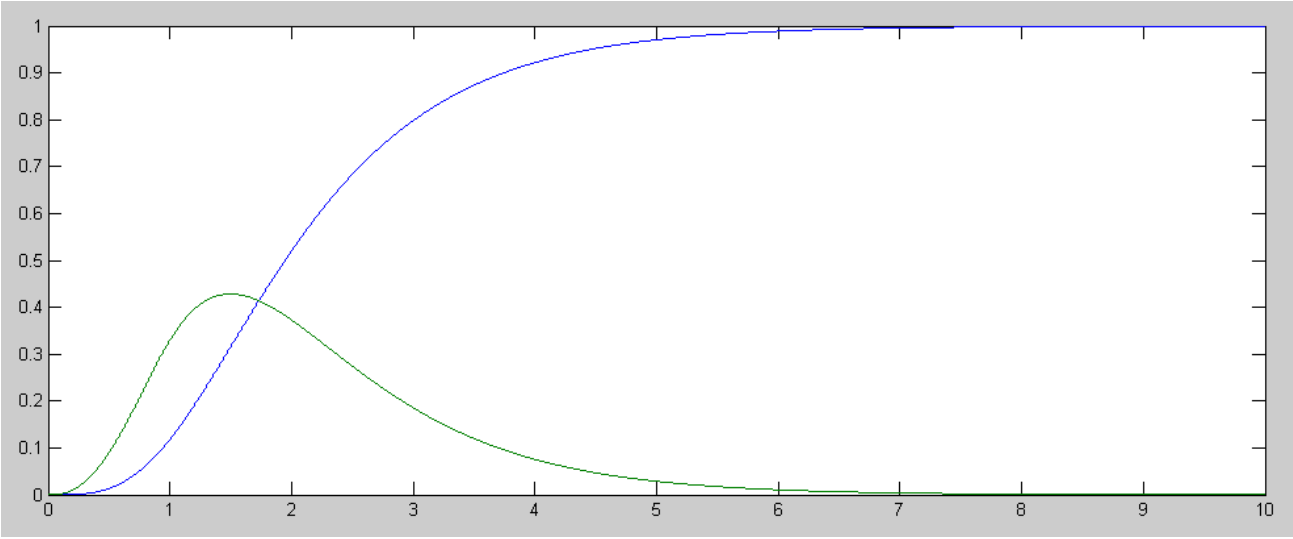
Check that the zeros are at {-1, -2, -3}

```
>> eig(A-100*B*Kx)

 -101.0203
   -2.9899
   -1.9898
   -1.0000

>> Kr = 6;
```

Take the step response and record the position (x) and velocity (CA = dx/dt):

```
>> Cx = [0,0,0,1]

     0     0     0     1

>> C = [Cx; Cx*A]

     0     0     0     1
     0     0     1    -1

>> D = [0;0]

     0
     0
```

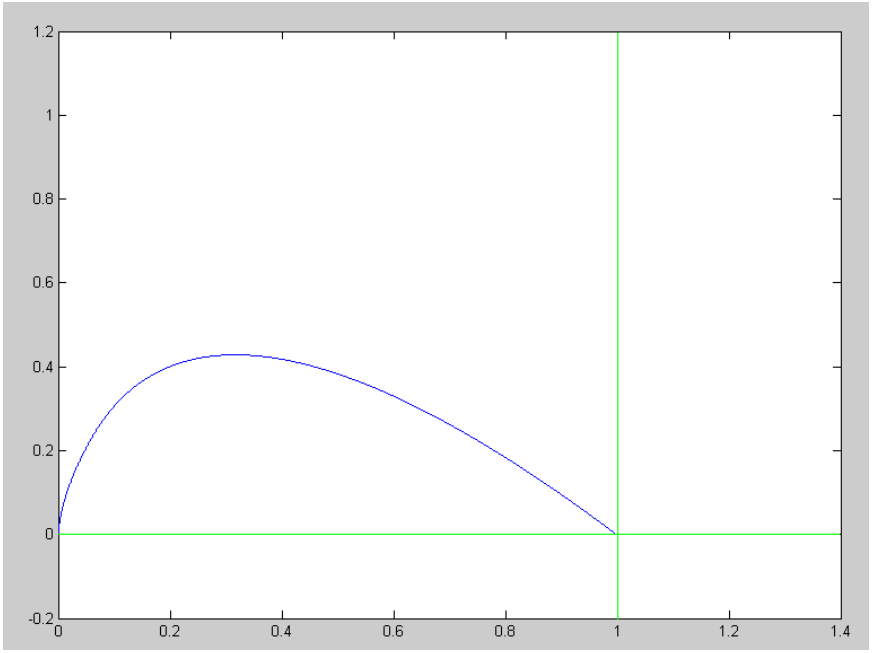
```
>> [y,u] = step_vss(A,B,C,D, t, Kx, Kr);
>> plot(t,y)
```



Step Response with VSS Control: sigma = (s+1)(s+2)(s+3)x
Position (blue) and velocity (green)

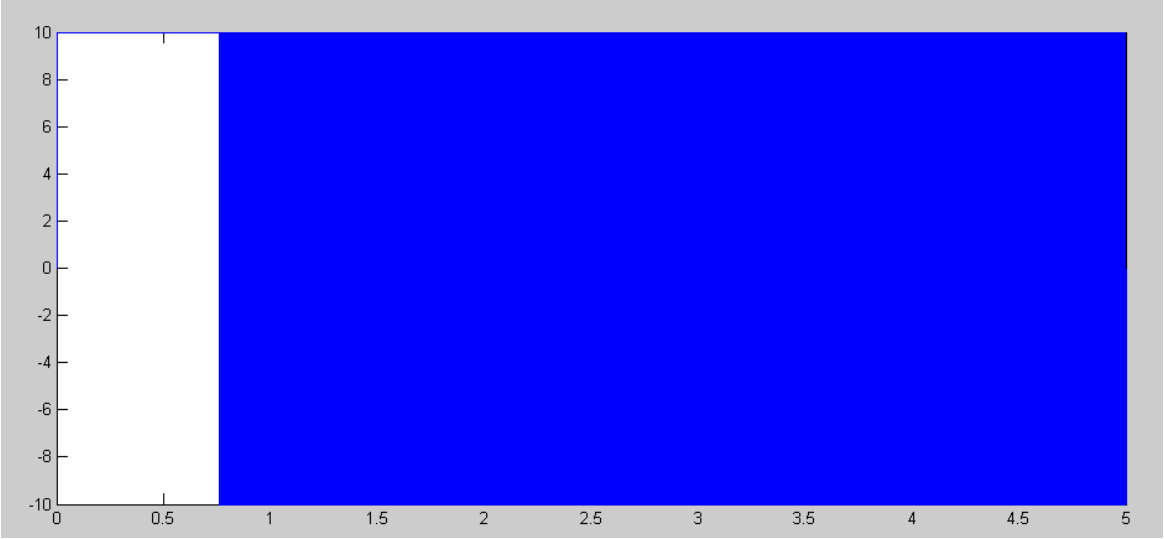
The phase plane shows the dominant pole at s = -1:

```
>> plot(y(:,1),y(:,2))
>> hold on
>> plot([1,1],[-0.2,1.2],'g')
>> plot([0,1.4],[0,0],'g')
```



Phase Plane: position vs. velocity

Again, the input is chattering between -10 and +10 when you hit the sliding surface:



Input u(t) with VSS Control. Once you hit the sliding surface, it chatters between -10 and +10.

Case 2: Saturating Control.

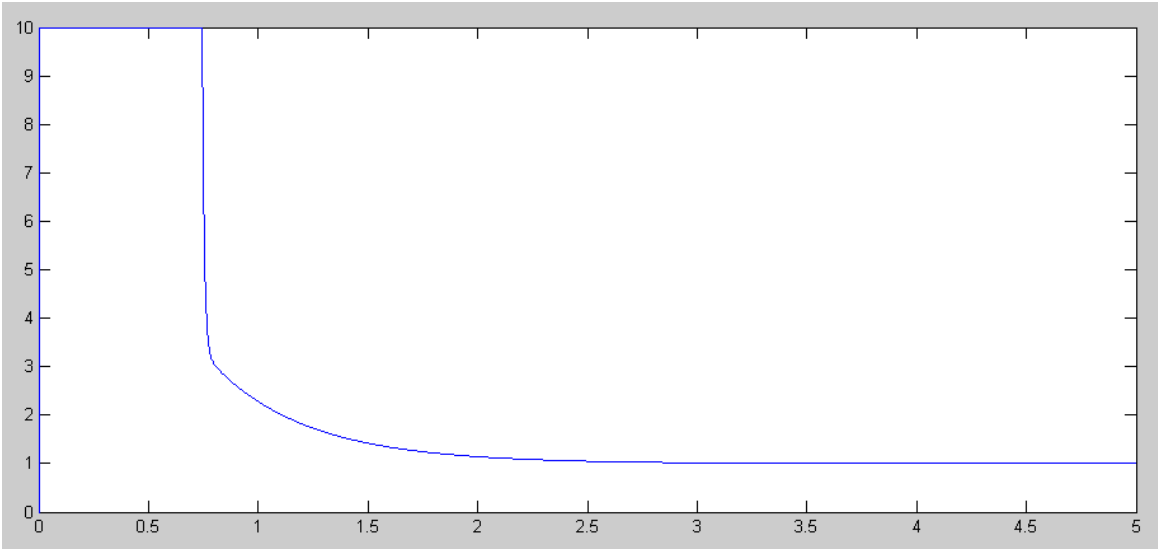
Changing the input from

$$U = 10 \cdot sign(K_r R - K_x X)$$

to

$$U = \text{limit}(-10, 100(K_r R - K_x X), 10)$$

keeps everything almost the same except the input no longer chatters.



Input u(t0 with Saturating Control. Once you approach the sliding surface, u(t) stops clipping.

VSS Control for an RC Filter (complex zeros)

What works with real zeros also works with complex zeros. For example, make the closed-loop system behave like a system with zeros at { -1 + j3, -1 - j3, -3}

```
>> poly([-1+j*3,-1-j*3,-3])

     1     5    16    30

>> Kx = [30, 16, 5, 1]*inv(T)

    1.0000    0.0000   10.0000   19.0000
```

Check: Are the zeros where we wanted:

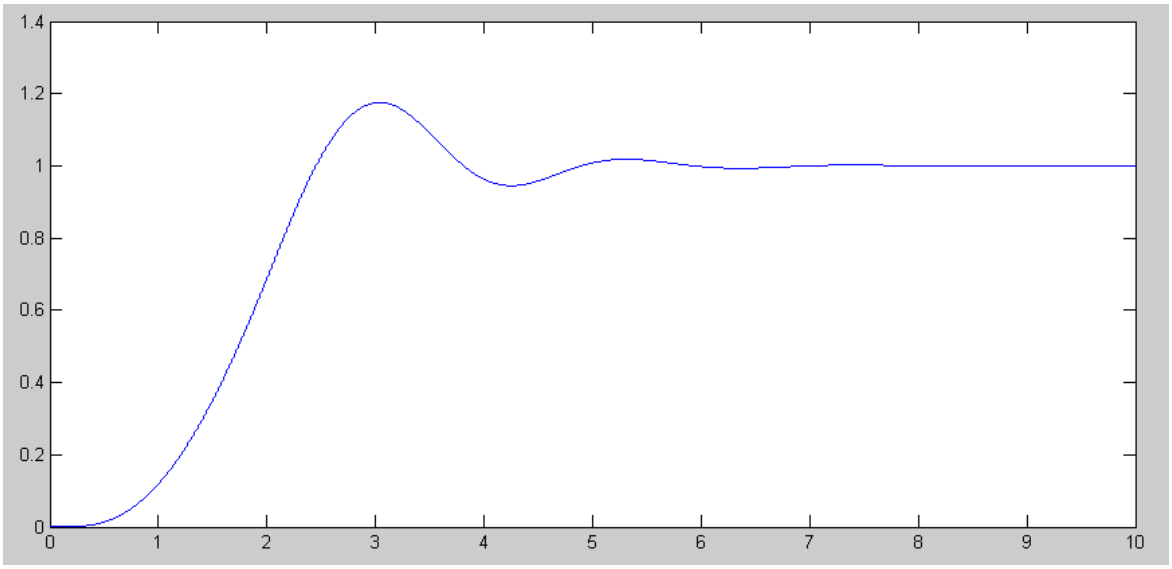
```
>> eig(A-90*B*Kx)

 -92.1196
  -0.9410 + 2.9822i
  -0.9410 - 2.9822i
  -2.9983
```

Add Kr to make the DC gain one:

```
>> DC = -Cx*inv(A-B*Kx)*B

    0.0323

>> Kr = 1/DC

    31
```

Now take the step response with a VSS controller:

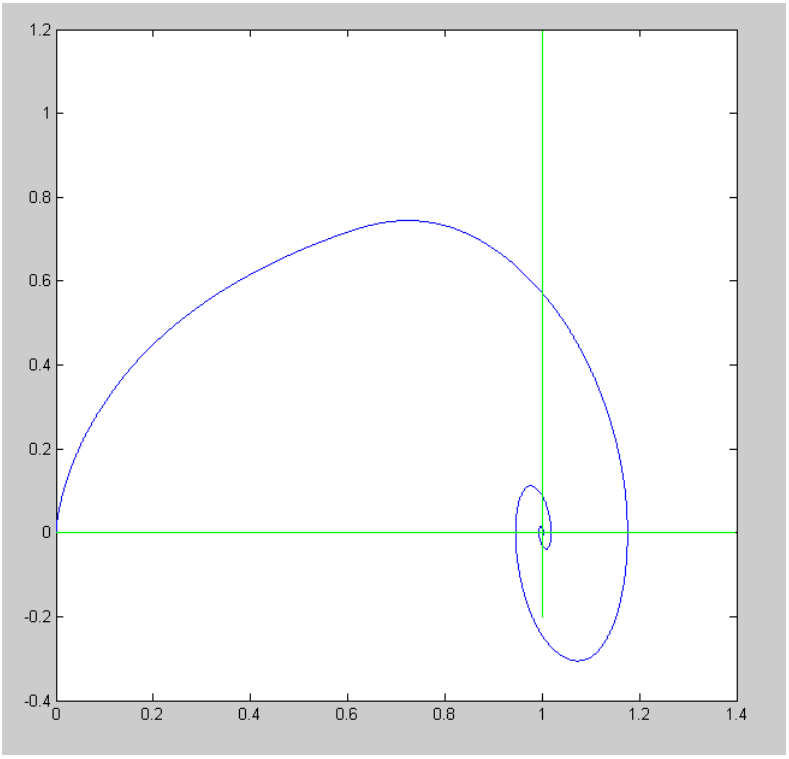
```
>> [y, u] = step_vss(A,B,C,D, t, Kx, Kr);
>> plot(t,y)
```



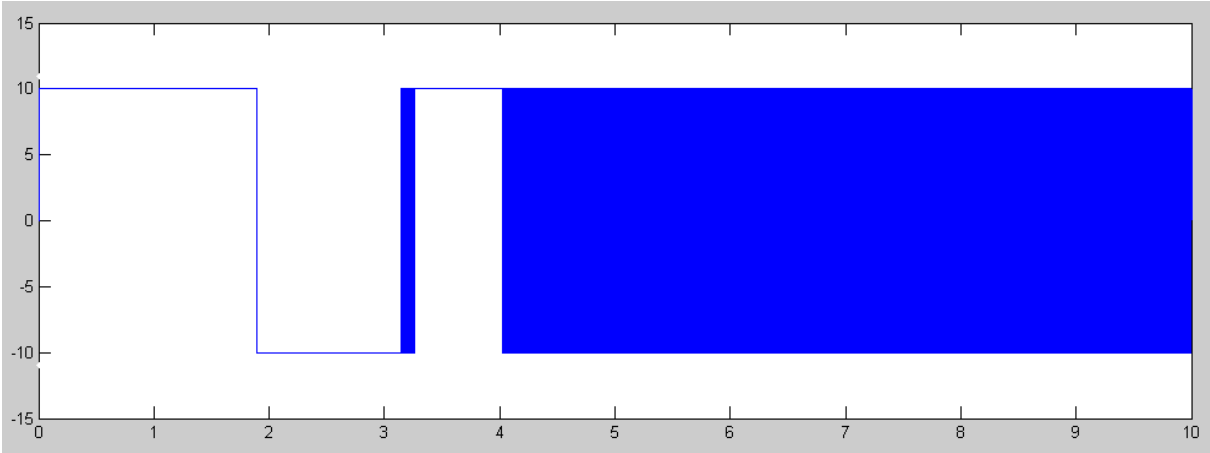
VSS Control: Step Response with zeros at {-1 + j3, -1 - j3, -3}

The phase plane is from

```
>> plot(y(:,1),y(:,2))
>> hold on
>> plot([1,1],[-0.2,1.2],'g')
>> plot([0,1.4],[0,0],'g')
```

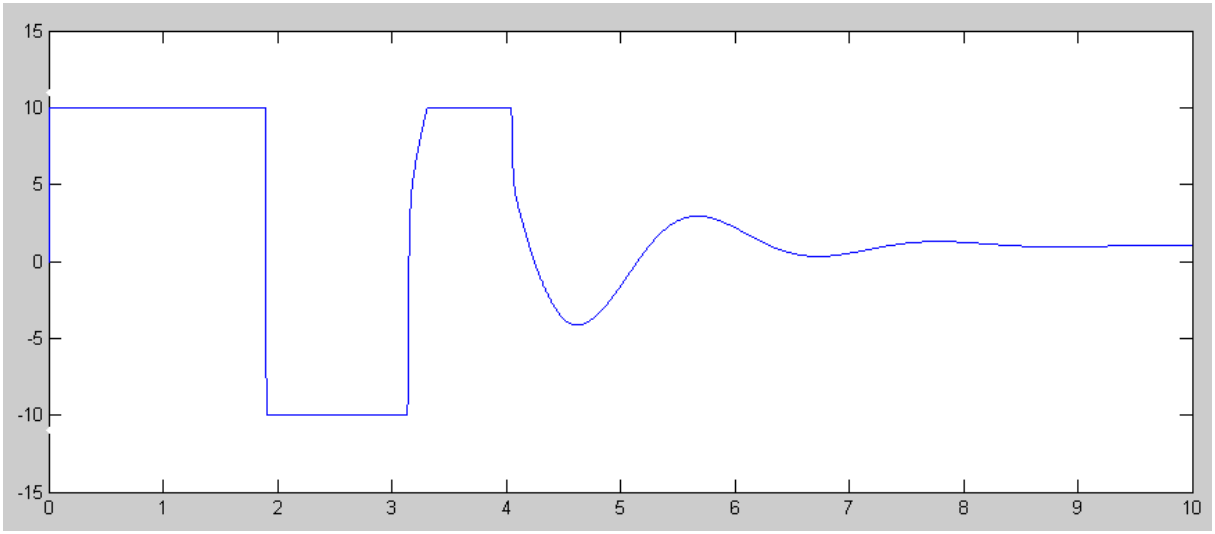


Phase Plane: The log spirals correspond to the complex zeros at {-1 + j3, -1 - j3}



Input, u(t), for VSS control with zeros at {-1 + j3, -1 - j3, -3}. Once you hit the sliding surface, the input chatters between -10 and +10.

If you change to a saturating controller, the response is almost the same except that the input no longer chatters:



Input, u(t), for a Saturating Control with zeros at {-1 + j3, -1 - j3, -3}. Once you hit the sliding surface, the input drops between -10 and +10.

step_vss

```
function [ y, u ] = step_vss( A, B, C, D, t, Kx, Kr )


T = t(2) - t(1);
[m, n] = size(C);

npt = length(t);

Az = expm(A*T);
Bz = B*T;

X = zeros(n,1);

y = zeros(npt, m);
u = zeros(npt, 1);

y(1,:) = (C*X + D)';

for i=2:npt
    % VSS Control
    %    U = 10*sign(Kr*1 - Kx*X);
    % saturating control
        U = max(-10, min(100*(Kr*1 - Kx*X), 10));

    X = Az*X + Bz*U;
    Y = C*X + D;

    y(i,:) = Y';
    u(i) = U;

   end

end
```