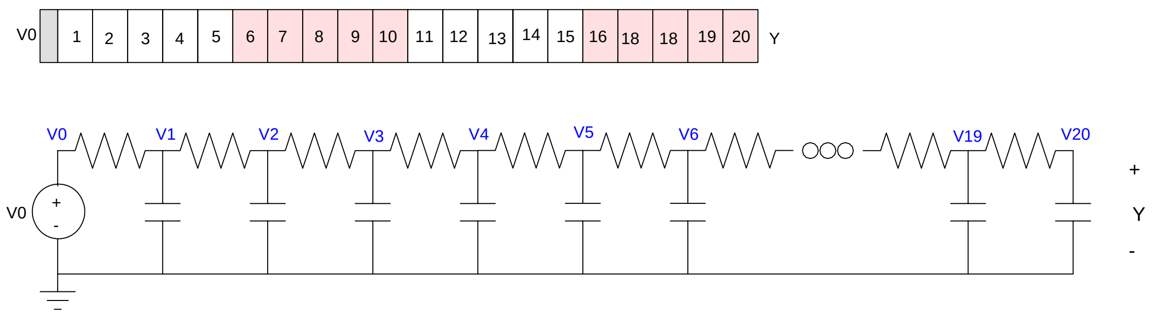
Pole Placement Example: Heat Equation

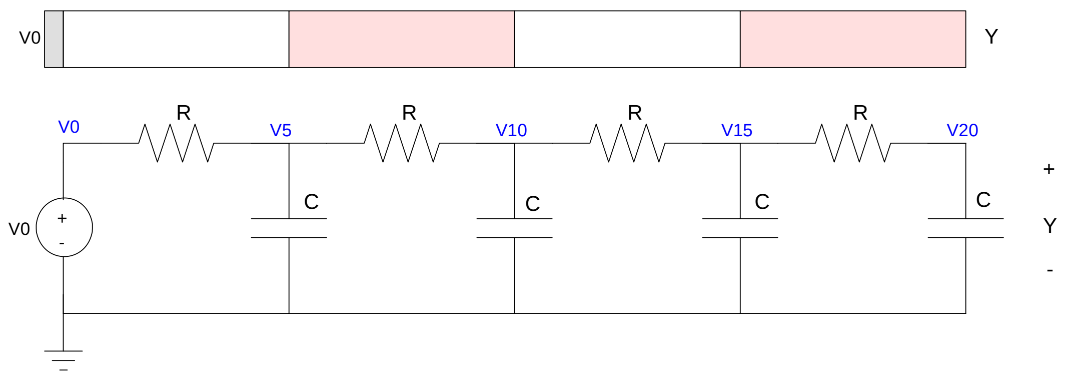
Problem: Design a feedback controller for a 20th-order RC filter so that the closed-loop system has

- A DC gain of 1.00
- No overshoot for a step input, and
- a 2% settling time of 4 seconds.



Temperature Along a Metal Bar: Modeled as a 20-Stage RC Filter: R = 0.2Ω, C = 0.2F

Solution: One problem with pole-placement is it uses full-state feedback. This means for a 20th-order system, you'll have 20 feedback gains. To reduce the number of gains, use a 4th-order approximation for the RC filter. Each finite-element lumps 5 capacitors together, giving it 5x the capacitance



Simplified Model: Lump Five Nodes Together to Give a 4-Stage RC Filter with R = 1Ω and C = 1F

The 4th-order approximation for the 20th-order system is

$$s\begin{bmatrix} V_5 \\ V_{10} \\ V_{15} \\ V_{20} \end{bmatrix} = \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}\begin{bmatrix} V_5 \\ V_{10} \\ V_{15} \\ V_{20} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} V_0$$

$$Y = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} V_5 \\ V_{10} \\ V_{15} \\ V_{20} \end{bmatrix}$$

Step 1: Input the system into Matlab

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

         -2     1     0     0
          1    -2     1     0
          0     1    -2     1
          0     0     1    -1

     >> B = [1;0;0;0]

          1
          0
          0
          0

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

          0     0     0     1
```

Decide where you want to place the closed-loop poles. Since the requirements say no overshoot and a 2% settling time of 4 seconds, the dominant pole belongs at s = -1. Since this is a 4th-order system, you need to place four poles, however.

Somewhat arbitrarily, place these poles at {-1, -2, -3 , 4}

Step 2: Determine the feedback gains, Kx, to place the poles of (A - B Kx) at {-1, -2, -3, -4}

From before,

```
     >> Kx = ppl(A, B, [-1, -2, -3, -4]

         3.00  5.00   7.00  8.00

     >> eig(A - B*Kx)

        -4.0000
        -3.0000
        -2.0000
        -1.0000
```

Step 3: Find Kr so that the DC gain is one. The closed-loop system is

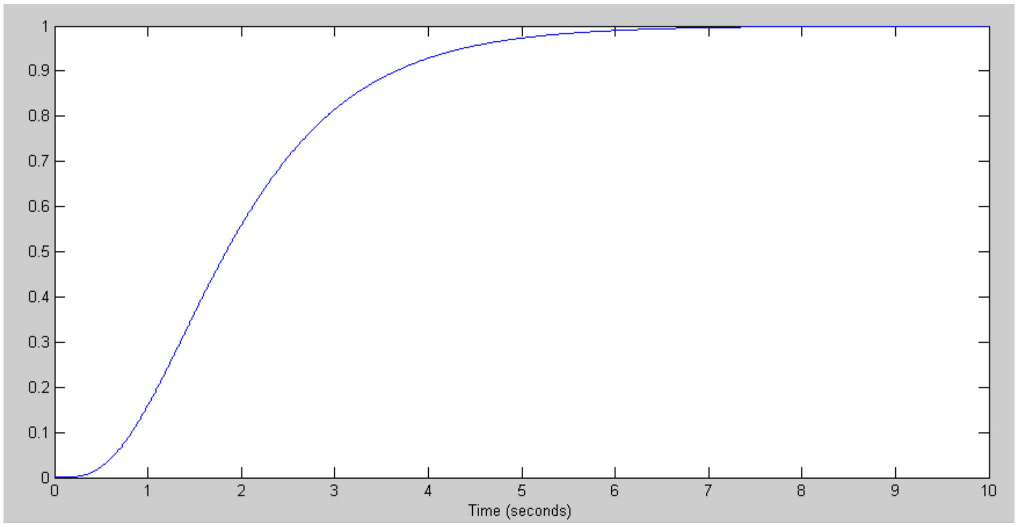
```
>> DC = -C*inv(A - B*Kx)*B

    0.0417

>> Kr = 1/DC


   24.0000
```

This gives you the control law:

$$V_0 = K_r R - K_x X$$

Step 4: Check the step response of the closed-loop system:

```
>> G = ss(A-B*Kx, B*Kr, C, 0);
>> t = [0:0.01:10]';
>> y = step(G,t);
>> plot(t,y);
>> xlabel('Time (seconds)');
```



Step Response of the Closed-Loop System with Poles at {-1, -2, -3, -4}

Note that the settling time isn't quite 4 seconds. This is due to the other three poles affecting the system slightly (the pole at -1 isn't that dominant)

Comparison to the 20th-Order Model

The 4th-order model is an approximation that makes anaysis easier. To see if these gains actually work on the 20th-order system, first in put the 20th-order system:

```
>> A20 = zeros(20,20);
>> for i=1:19
   A20(i,i) = -50;
   A20(i+1,i) = 25;
   A20(i,i+1) = 25;
   end
>> A20(20,20) = -25;
>> B20 = zeros(20,1);
>> B20(1) = 25;
>> C20 = zeros(1,20);
>> C20(20) = 1;
```

Add in the feedback gains, Kx. These are zero execpt at elements {5,10,15,20}

```
>> K20 = zeros(1,20);
>> K20([5,10,15,20]) = Kx;
```

Case 1: Closed-Loop Poles = {-1, -2, -3, -4}

The 4th-order model gives

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

   -4.0000
   -3.0000
   -2.0000
   -1.0000
```

The 20th order system with the same feedback gains give

```
>> K20([5,10,15,20]) = Kx;
>> eig(A20-B20*K20)

  -98.2965 + 1.5098i
  -98.2965 - 1.5098i
  -91.7830
  -79.8836 + 4.9956i
  -79.8836 - 4.9956i
  -68.6791 + 7.0008i
  -68.6791 - 7.0008i
  -57.4661 + 5.1074i
  -57.4661 - 5.1074i
  -50.0000
  -41.9362
  -21.9870 + 4.8499i
  -21.9870 - 4.8499i
  -12.1576 + 5.2447i
  -12.1576 - 5.2447i
   -1.3662 + 1.0369i
   -1.3662 - 1.0369i
   -5.6278 + 3.5131i
   -5.6278 - 3.5131i
```

The 20th-order system's poles are not very close to what the 4th-order system predicted. Either we need a better model or we need to reduce the gains.

Large gains indicate that we're trying to make the system behave a way it doesn't want to. To reduce the gains, let's choose the desired closed-loop poles based upon how the system *wants* to behave. The open-loop eigenvalues are

```
>> eig(A)

   -3.5321
   -2.3473
   -1.0000
   -0.1206
```

Placing the closed-loop dominant pole at -1 means we're trying to make the system 8x faster. That's a lot. Instead, let's try to make the closed-loop system twice as fast and leave the other poles unchanged.

```
>> Kx = ppl(A, B, [-0.25, -1, -2.34, -3.53])

    0.1200    0.2477    0.3296    0.3677
```

The closed-loop poles for the 4th-order sytem with these gains are where we placed them:

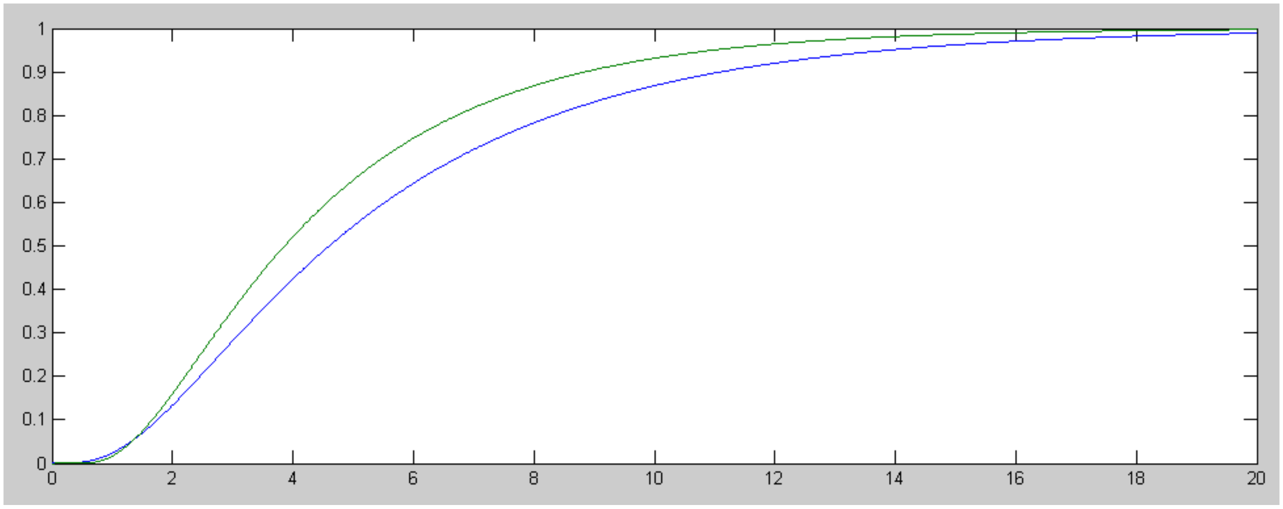
```
>> eig(A-B*Kx)

   -3.5300
   -2.3400
   -1.0000
   -0.2500
```

The closed-loop poles for the 20th-order system are closer:

```
>> K20([5,10,15,20]) = Kx;
>> eig(A20-B20*K20)

  -99.4340
  -97.5251
  -95.2955
  -91.1452
  -84.7344
  -80.9994
  -73.5189
  -67.1925
  -59.3244
  -51.9244
  -44.5394
  -37.9640
  -27.6224
  -23.6016
  -16.3435
   -0.3267
   -1.2062
   -3.7687
   -6.7989
  -11.7347
```

Comparing the two step responses:

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

    0.4842

>> Kr = 1/DC

    2.0651

>> G = ss(A-B*Kx,B*Kr,C,D);
>> y = step(G,t);

>> G20 = ss(A20-B20*K20,B20*Kr,C20,0);
>> y20 = step(G20,t);

>> plot(t,y,t,y20)
```



Step Response of the 4th-Order Sytem (blue) and 20th Order System (green)

Example 2: Can you make an RC filter oscillate? Find the gains, Kx, which place the closed-loop poles at

$$s = \{-1 + j5, \ -1 - j5, \ -5, \ -6 \}$$

Solution: Same as before. Using Bass Gura to place the poles:

```
>> Kx = ppl(A, B, [-1+j*5, -1-j*5, -5, -6] )

    6.0000   33.0000  201.0000  539.0000

>> eig(A - B*Kx)

  -1.0000 + 5.0000i
  -1.0000 - 5.0000i
  -6.0000
  -5.0000
```

Checking with the 20th-order system:

```
>> K20([5,10,15,20]) = Kx;
>> eig(A20-B20*K20)

   1.2002 + 4.4059i
   1.2002 - 4.4059i
  -3.7469 + 7.8668i
  -3.7469 - 7.8668i
 -10.1433 +11.5028i
 -10.1433 -11.5028i

   etc
```

The 20th-order system is unstable - meaning the gains are too large or this model isn't accurate enough for pushing it this hard.

Instead, let's try to not push the system so hard. To reduce the gains, check the open-loop eigenvalues:

```
>> eig(A)

   -3.5321
   -2.3473
   -1.0000
   -0.1206
```

The dominant pole is at -0.12. Let's make the system twice as fast and oscillating. Keep the fast two poles unchanged to keep the gains down

```
>> Kx = ppl(A,B,[-0.2+j,-0.2-j,-2.34,-3.53])

   -0.7300    0.2982    2.8943    5.1281
```

These gains are much more reasonable - so let's go with them. Let's check to make sure Kx places the poles of (A - B Kx) where we want.

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

  -3.5300
  -2.3400
  -0.2000 + 1.0000i
  -0.2000 - 1.0000i
```

Checking the poles of the 20th-order system:

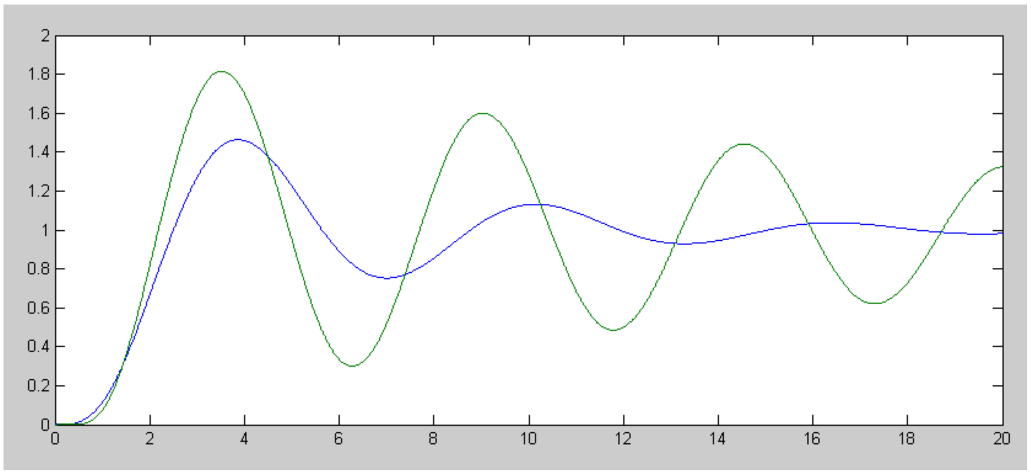
```
  -0.0554 + 1.1391i
  -0.0554 - 1.1391i
  -5.0210 + 1.2985i
  -5.0210 - 1.2985i
 -13.8275 + 2.4604i
 -13.8275 - 2.4604i
      etc
```

Sort of where we put the poles. The difference tells you that the model is marginal for making the system this fast. Now that we know Kx, find Kr to set the DC gain to one:

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

    0.1164

>> Kr = 1/DC

    8.5906
```

Step 3: Check the closed-loop response:

```
>> G = ss(A-B*Kx, B*Kr, C, 0);
>> t = [0:0.01:5]';
>> y = step(G,t);
>> plot(t,y);
>> xlabel('Time (seconds)');
```



Step Response of the 4th-order model (blue) and 20th-order system (green)

Matlab Simulation:

The following code calls the previous routines, *heat_dynamics()* and *heat_display()* and uses the feedback control laws we computed herein. To places the closed-loop poles at {-1, -2, -3, -4}, the control law is

$$U = K_r R - K_x X$$

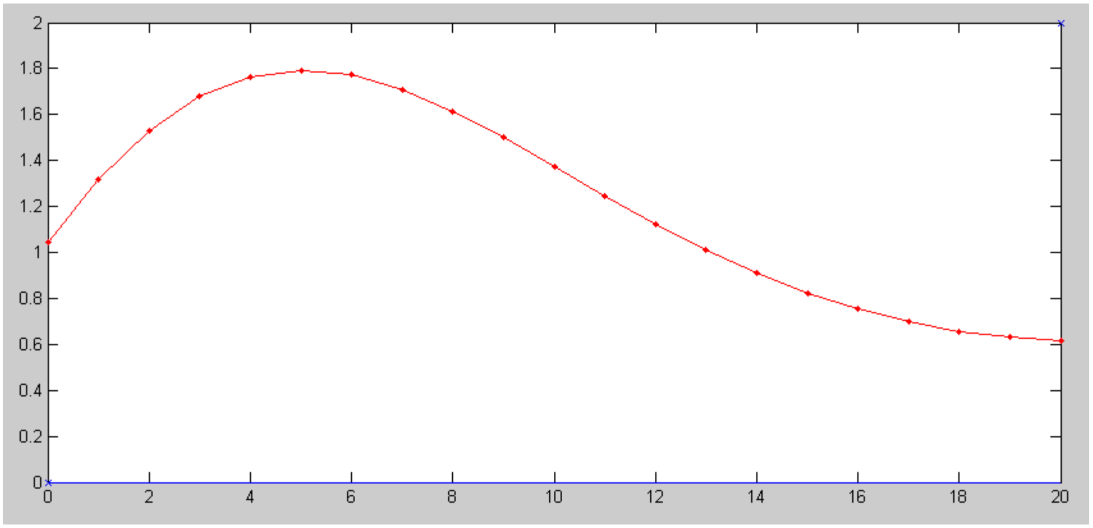
$$K_x = \left[\begin{array}{cccc} 3 & 5 & 7 & 8 \end{array}\right]$$

$$K_r = 24$$

To improve the numerical integration routine, a sampling size of dt = 0.0001 is used.

To speed up the simulation, the display routine is called every 100th iteration (0.01 second)

```
X = zeros(20,1);
Ref = 1;
dt = 0.0001;
t = 0;

% poles at {-1, -2, -3, -4}
Kx = [3   5  7  8];
Kr = 24;


while(t < 100)
   Ref = 0.5 + 0.5 * (sin(0.2*t) > 0);
   for j=1:100
      U = Kr * Ref - Kx * [X(5); X(10); X(15); X(20)];
      dX = heat_dynamics(X, U);
      X = X + dX * dt;
      t = t + dt;
      end
   heat_display(X, U);
   end
```


2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10
12
14
16
18
20