Heat Equation

Pole Placement Example

NDSU ECE 463/663

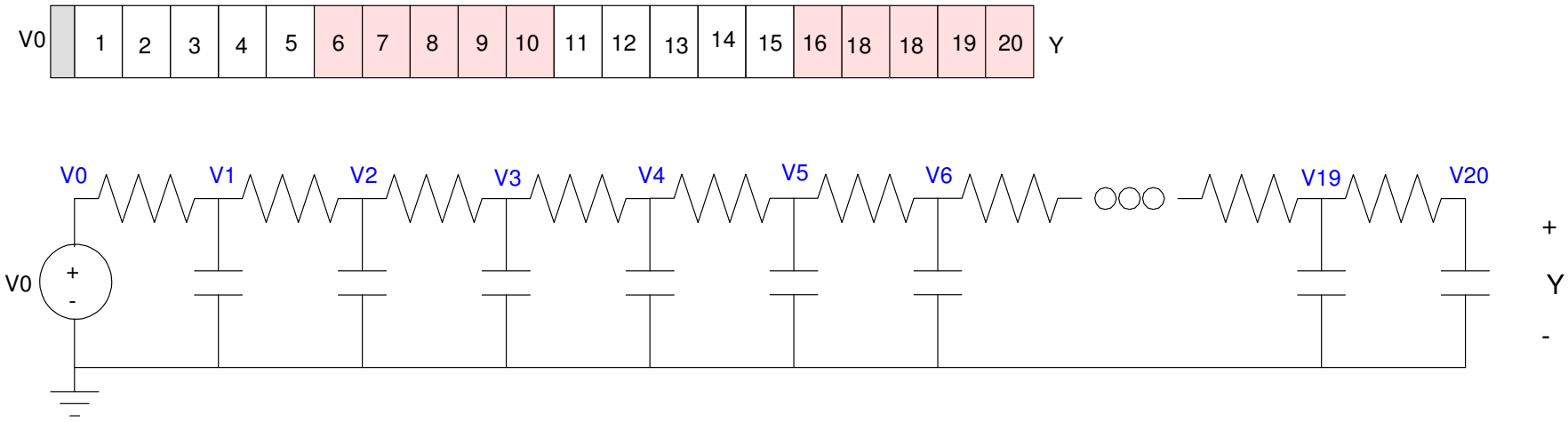
Lecture #14

Inst: Jake Glower

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Problem: Design a feedback controller for a 20th-order RC filter:

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



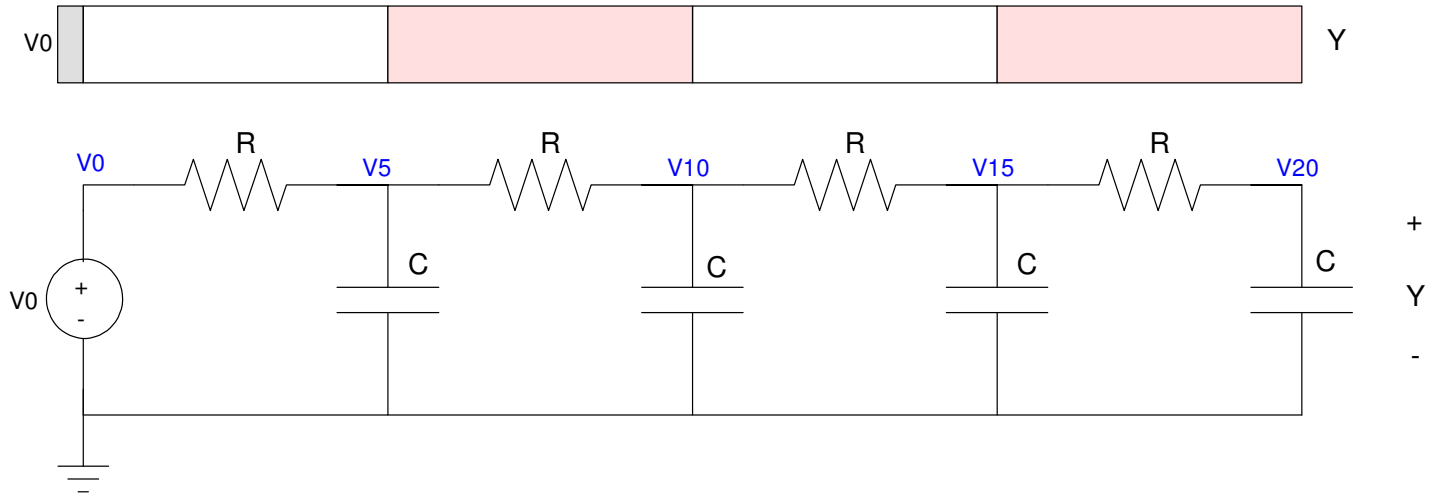
Temperature Along a Metal Bar: Modeled as a 20-Stage RC Filter: R = 0.2Ω, C = 0.2F

If you model this as a 20th order system, you'll get 20 feedback gains

- Meaning you need to use 20 sensors

Approximate this with a 4th-order model

- Lump 5 capacitors & 5 resistors together
- Slightly wrong but now only needs 4 sensors



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

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

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

Now find the feedback gains using Bass Gura

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.

- Dominant pole: s = -1
- Arbitrarily place the other three poles at {-2, -3 , 4}

Step 2: Determine the feedback gains, Kx

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
>> Kr = 1/DC

   24.0000
```

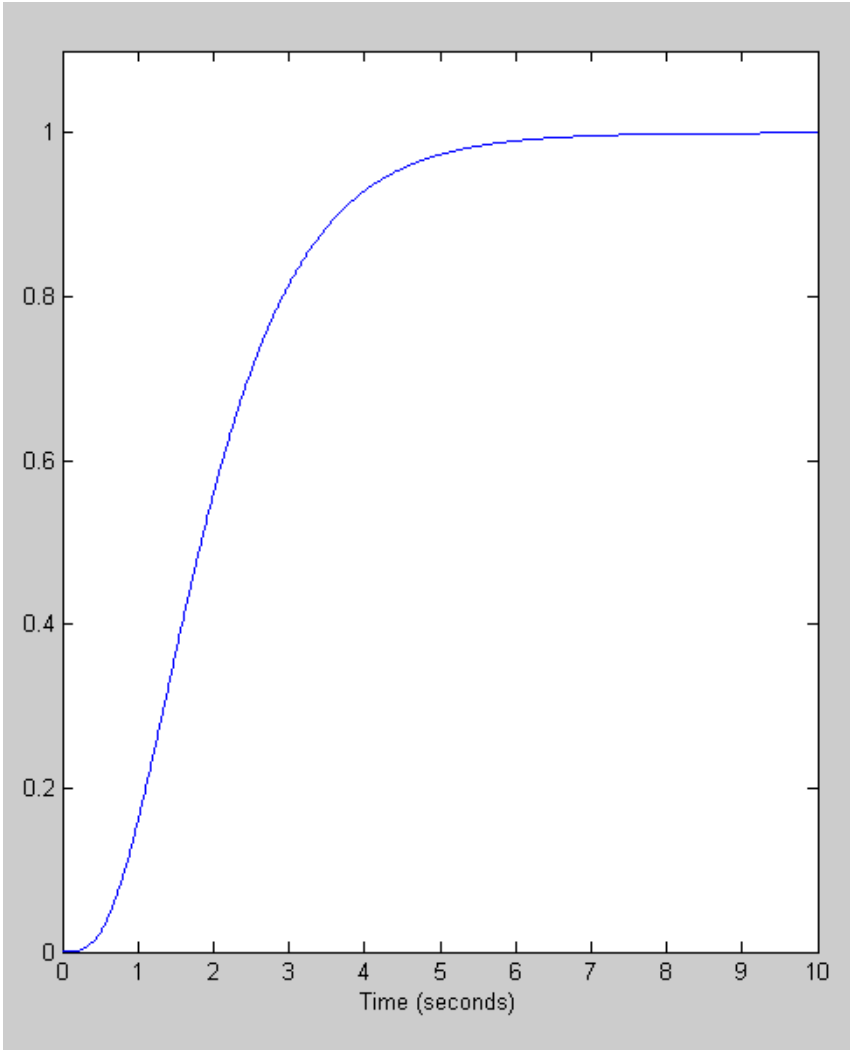
This gives you the control law:

$$V_0 = K_r R - K_x X$$

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)');
```

- DC gain = 1.000
- 2% settling time = 4 seconds (ish)
- No overshoot



Comparison to the 20th-Order Model

The 4th order model is slightly wrong

- Moved C's so they could be lumped

How do these gains work for a 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
```

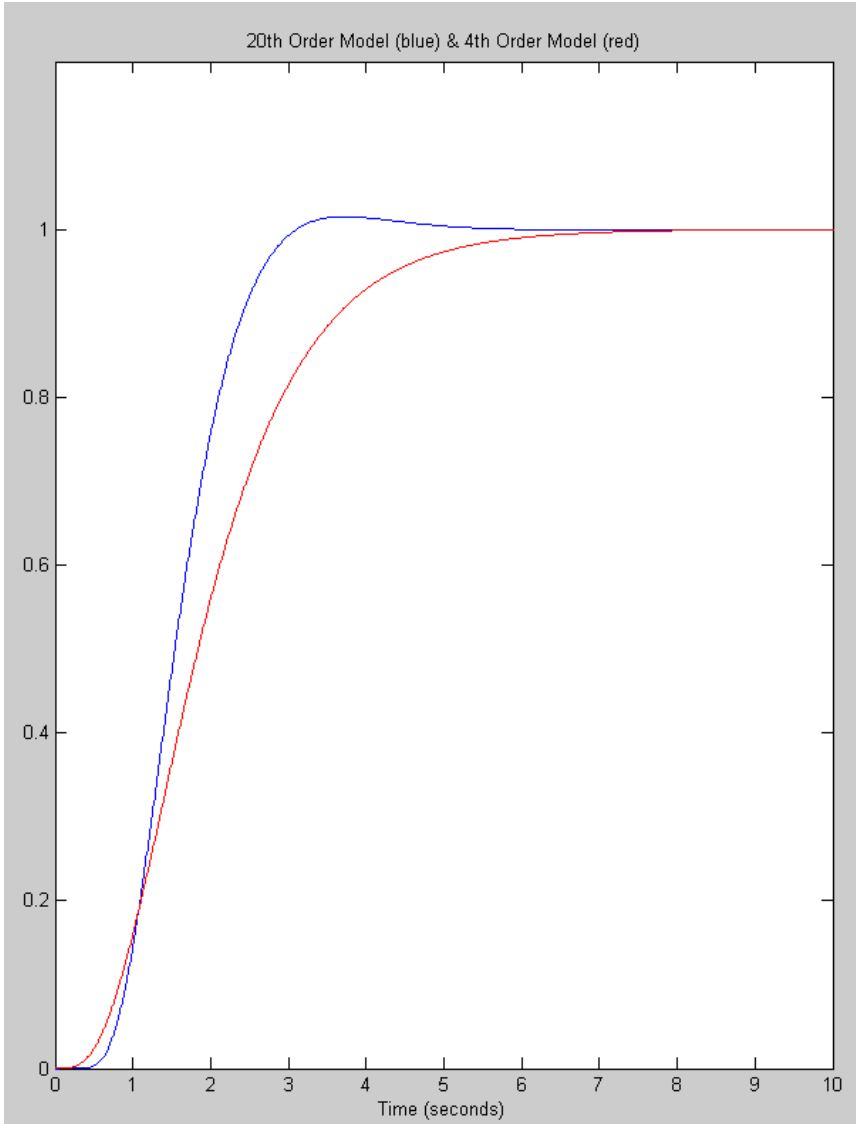
20th order Model

```
eig(A20-B20*K20)

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

Comparing Step Responses

- Blue: 20th Order Model
- Red: 4th Order Model

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

G4 = ss(A-B*Kx, B*Kr, C, 0);
G20 = ss(A20-B20*K20, B20*Kr,
C20, 0);

t = [0:0.01:10]';
y4 = step(G4,t);
y20 = step(G20,t);
plot(t,y20,'b',t,y4,'r');
xlabel('Time (seconds)');
```



Why are the poles wrong?

- Kx is too large
- Large gains indicate you're moving the poles too far
- The desired response is too different from the open-loop response

Check the open-loop poles:

- How the system *wants* to behave.

```
>> 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 4th-order Model

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

  -0.2500
  -1.0000
  -2.3400
  -3.5300
```

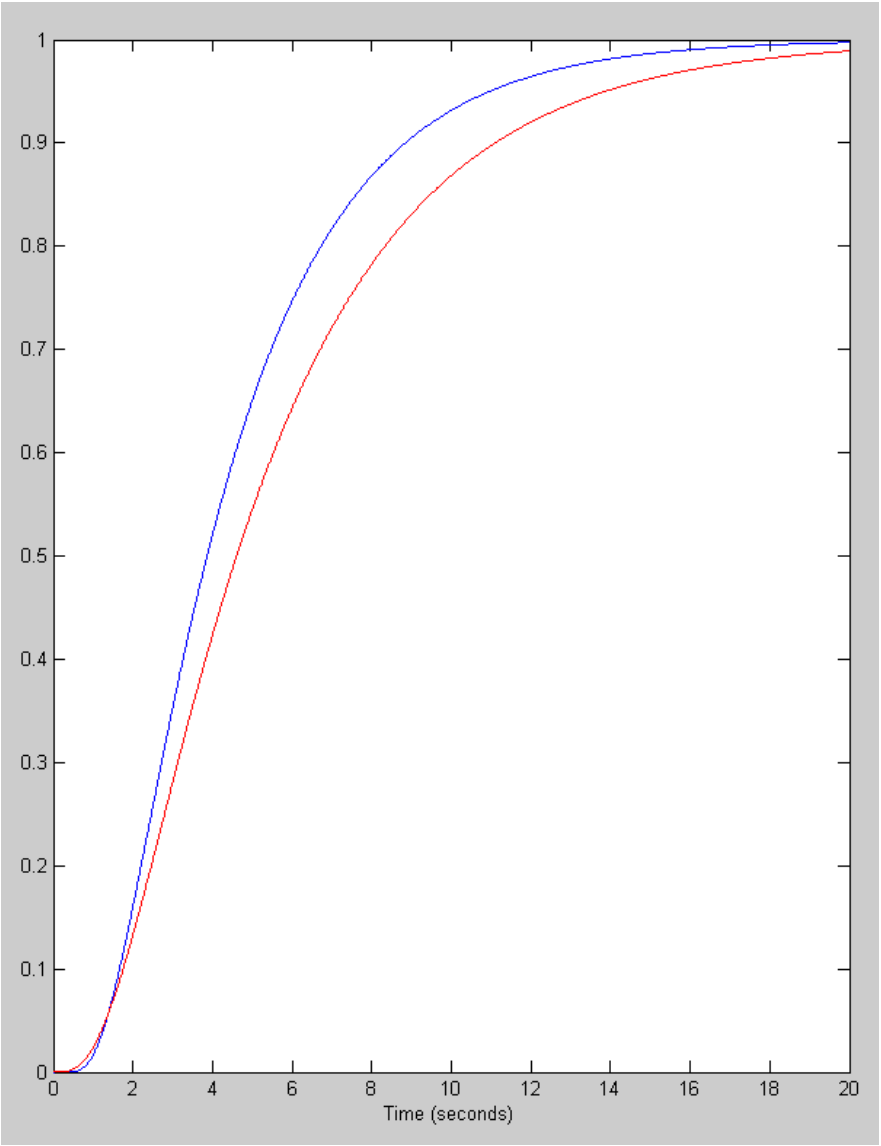
20th order Model

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

Comparing the two step responses:

- Blue: 20th-order model
- Red: 4th-order approximation

```
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)
```



Example 2: Can you make an RC filter oscillate?

- 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

- 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.

- Check the open-loop eigenvalues:

```
>> eig(A)

   -3.5321
   -2.3473
   -1.0000
   -0.1206
```

The dominant pole is at -0.12.

- 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.

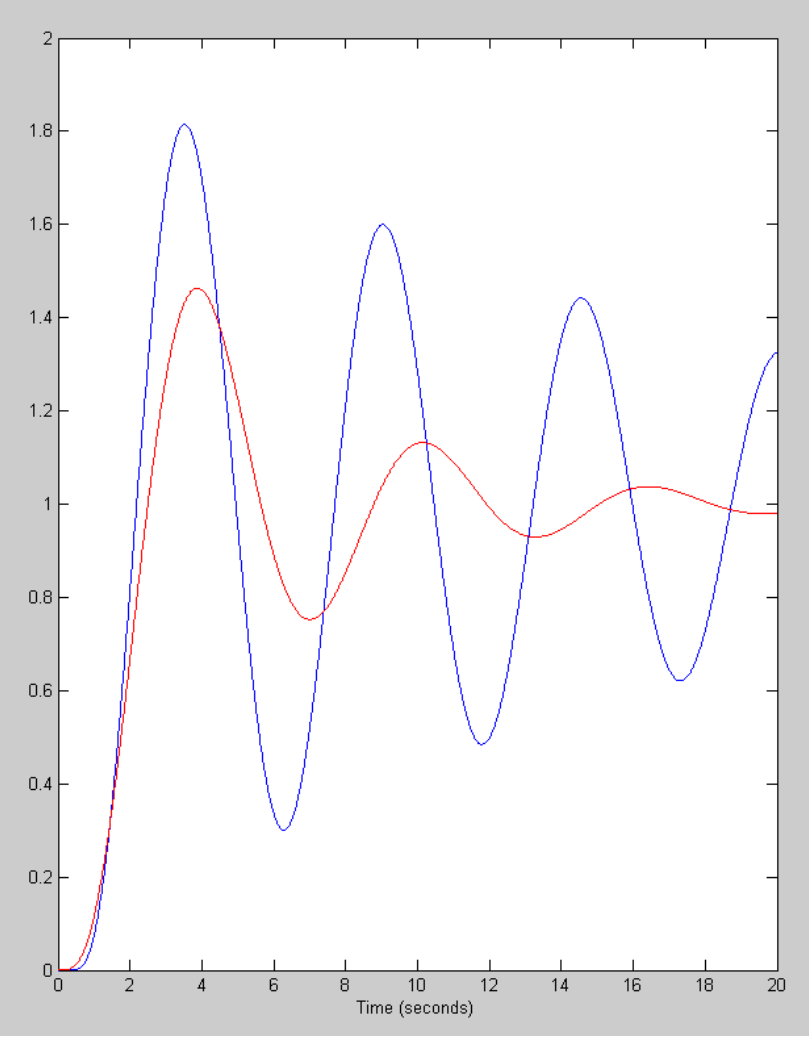
Now that we know Kx, find Kr to set the DC gain to one:

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

    0.1164

>> Kr = 1/DC

    8.5906
```

Step 3: Check the closed-loop response:

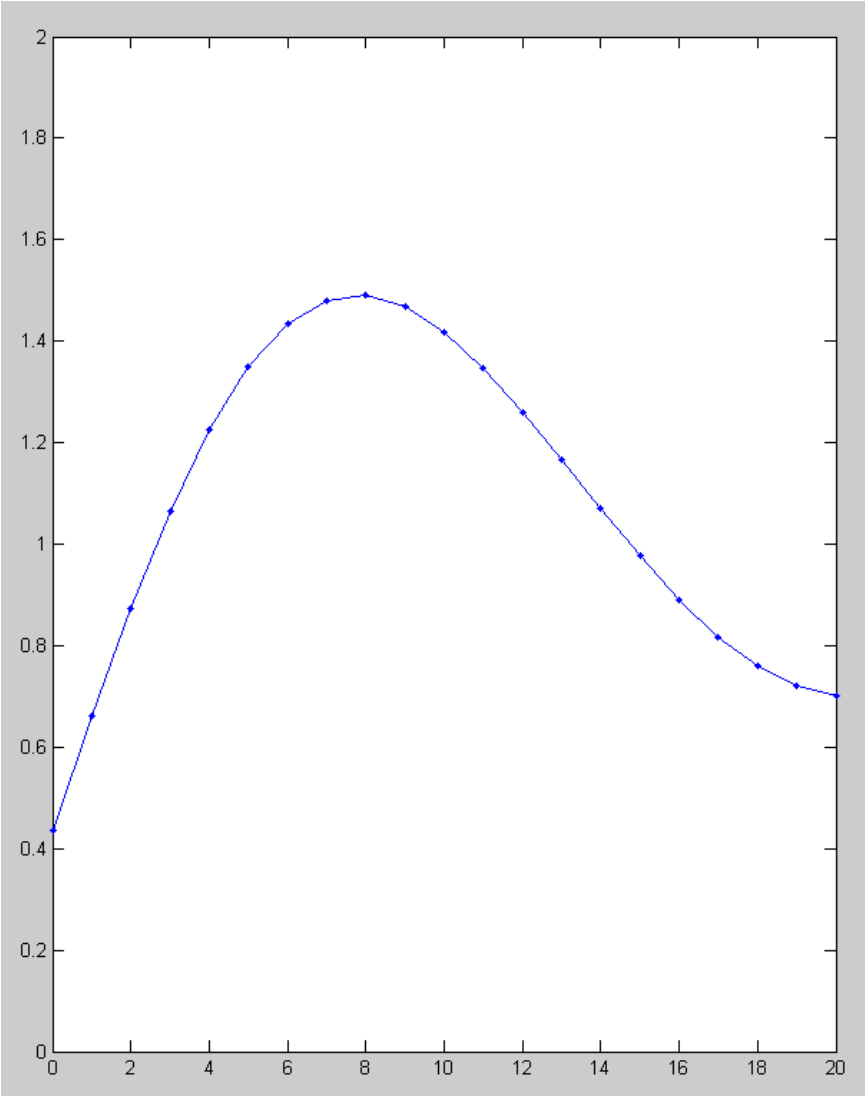
- Blue: 20th-Order System
- Red: 4th-Order Approximation

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



Matlab Simulation: Heat20

```
% 20-stage RC Filter

V = zeros(20,1);
V0 = 1;
DATA = [V0;V];
dV = zeros(20,1);

Ref = 1;
Kx = [3, 5, 7, 8];
Kr = 24;

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

dt = 0.001;
t = 0;
```



```
while(t < 10)

    for i2=1:10
% Control Law
       V0 = Kr*Ref - K20*V;

      dV(1)  = 25*V0   - 50*V(1) + 25*V(2);

      for i=2:19
         dV(i)  = 25*V(i-1) - 50*V(i) + 25*V(i+1);
      end

      dV(20) = 25*V(19) - 25*V(20);

      V = V +  dV*dt;
      t = t + dt;
    end

   plot([0:20], [V0;V], '.-');
   ylim([0,2]);
   pause(0.01);
   end
```