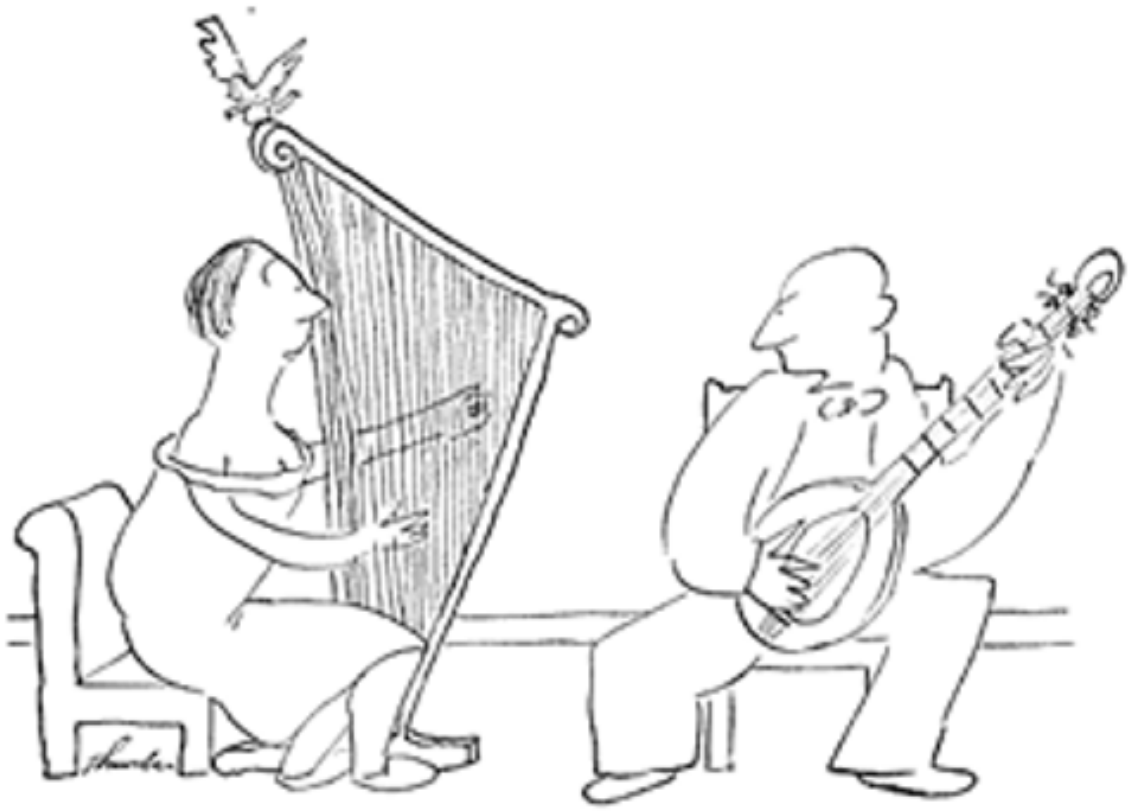

"Come on, get hot!"

EE 311 **April 24, 2019**
Lecture 26

8.9 Limit Cycles

A stable filter which begins in an arbitrary starting state and has zero input, may produce an output for a short period of time but must eventually drop to zero. In some circumstances, where there is round off error or overflow, a system can become nonlinear and the output can oscillate indefinitely even though the input is held at zero. Such cycles are commonly referred to as *limit cycles.*

The state variable representation gives an easy way to produce and observe limit cycles. For a second order system the state variables representation is given by

$$z\mathbf{v} = \mathbf{A}\mathbf{v} + \mathbf{B}x$$
$$y = \mathbf{C}\mathbf{v} + \mathbf{D}x$$

Where $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ represents the state of the system. If the input is zero, the state equation becomes (in the time domain)

$$z\begin{bmatrix} v_1(k+1) \\ v_2(k+1) \end{bmatrix} = \mathbf{A}\begin{bmatrix} v_1(k) \\ v_2(k) \end{bmatrix} \tag{8.33}$$

Equation (8.33) is called the trajectory equation and represents a mapping of states at time *k* to time *k*+1. Since we have only two states we visualize the trajectory in two dimensions as shown in Figure 8.37. For simplicity we limit the valid values of the state variables to the range -1 to +1.

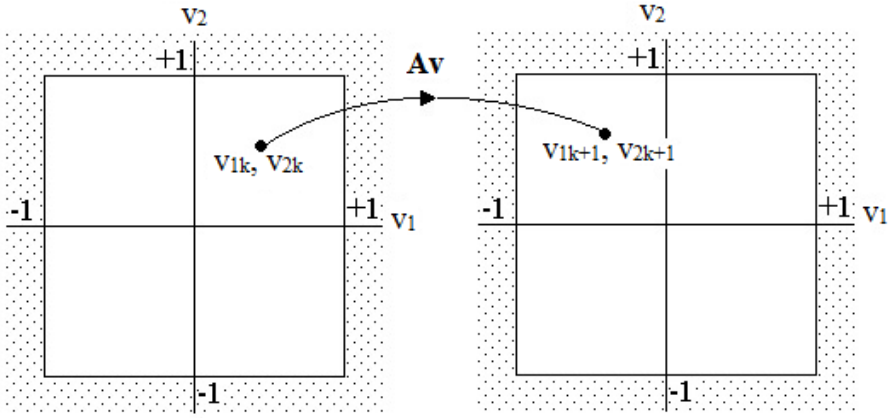

Figure 8.37
State variable trajectory. The input is zero so the next state is given by **Av.** The shaded area represents overflow.

Depending on the values of the elements of the matrix **A**, the trajectory equation may result in the next state being out of bounds or, falling into the shaded areas in Figure 8.37. Since we are unable to represent numbers in these areas, overflow rules need to be applied which determine how an illegal state gets mapped back into the valid range. With fixed point arithmetic we deal with two types of overflow: saturation overflow and two's complement overflow. In saturation overflow any state variable that falls outside the legal boundary is returned to that boundary. For

example if the next state is at v_1, v_2 = 1.3, 0.8, the saturation overflow rules would remap it to v_1, v_2 = 1.0, 0.8. In two's complement overflow, any state variable that exceeds a legal value is wrapped around to the other side of the map. Using two's complement rules for overflow the state v_1, v_2 = 1.3, 0.8 would be remapped to v_1, v_2 = -0.7, 0.8. Figure 8.38 shows these two types of overflow as they apply to a sinusoid.

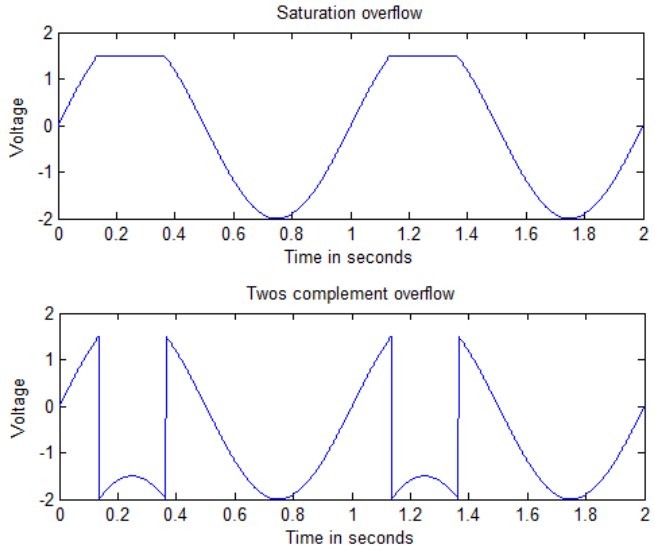

Figure 8.38
The top figure illustrates saturation overflow above 1.5 volts. The bottom figure is an example of two complement overflow.

Consider a second order system represented by the transfer function

$$H(z) = \frac{z^2 + 2z + 1}{z^2 - z + 0.8}$$

Which is a stable second order section with poles at 0.5 ± j0.7416 and two zeros at z = -1. The state variable representation is given by

$$z\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.8 & 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} x$$

$$y = \begin{bmatrix} 2 & 3 \end{bmatrix} + [1]x$$

The trajectory equation is, from (8.33),

$$\begin{bmatrix} v_1(k+1) \\ v_2(k+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.8 & 1 \end{bmatrix}\begin{bmatrix} v_1(k) \\ v_2(k) \end{bmatrix}$$

Figure 8.39 shows how the areas of all numbers which can be represented are mapped by the trajectory equation. Note that there are two triangular areas which represent overflow. When the rules for two's complement overflow are applied the overflow areas are mapped back into the space of representable numbers. It is clear from looking at this map that any state near corner 2 will be mapped by way of the overflow rules to a location near corner 4 and vice-versa. Thus,

this set of equations, when initialized to a state near corner 4, say at v_1, v_2 = 0.95, -0.95, will map alternately to a location near corners 2 and 4 without ever decaying into zero. The result will be that these two state variables will always have some numeric value which will result in some output other than zero. The output variable y can be computed from $y = \mathbf{C}v$**.** Figure 8.40 shows the output, with zero input, when the state variables are initialized to v_1, v_2 = 0.95, -0.95.

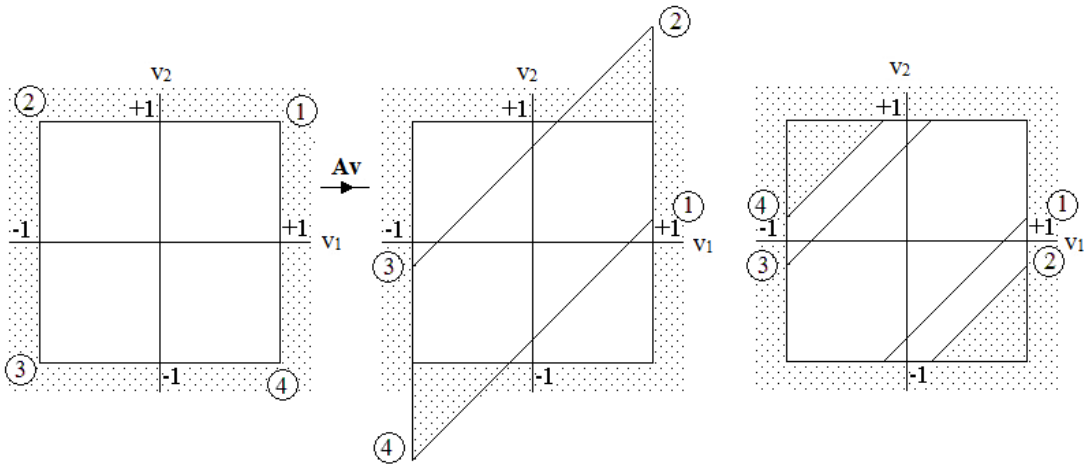

Figure 8.39
(Left) The standard space of numbers that can be represented. (Middle) The standard space is mapped into a parallelogram by **Av**. Note that two triangles are mapped into overflow areas. (Right) The two's complement overflow rules are applied and the overflow areas are wrapped back into the area of representable numbers.

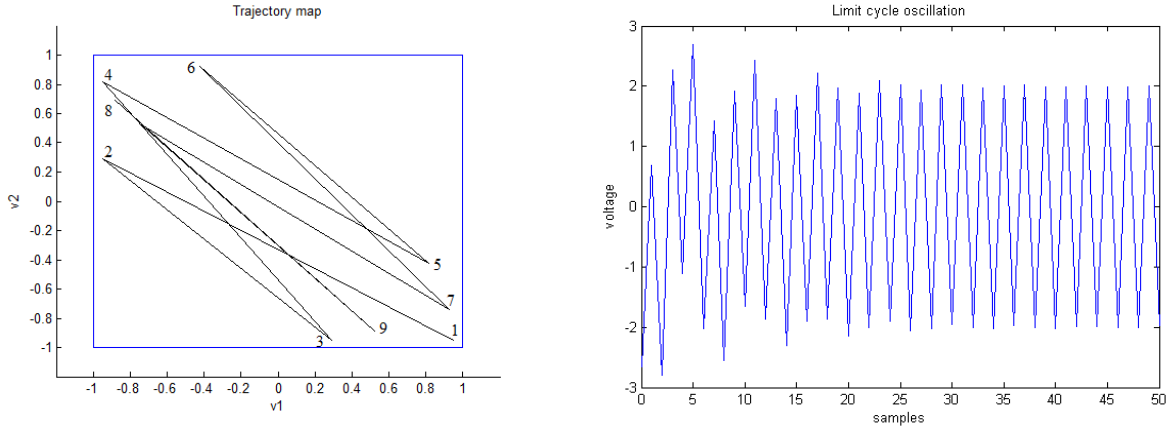

Figure 8.40
{Left) Nine iterations of the trajectory when the starting state is at v_1, v_2 = 0.95, -0.95. (Right) The output y when there is no input but the starting state is at v_1, v_2 = 0.95, -0.95.

There are two ways to eliminate limit cycle oscillation. The first is to use saturation overflow and the second is to do a coordinate transformation and find an appropriate **A** matrix.

Saturation overflow does mostly eliminate limit cycle oscillation but doing it in software is computationally expensive since it requires that every multiplication and addition be checked for overflow. On many modern dedicated processor chips, saturation overflow is implemented in hardware with little or no computational cost.

Modifying the **A** matrix is a viable solution when saturation overflow cannot be done, or when the nonlinear effects that it introduces cannot be accepted.

Viewing overflow oscillation on a trajectory plot provides insight as to how to predict when overflow oscillation will occur. We see from the trajectory that in order for the system to decay to zero the present state must be closer to the origin the previous state. Thus, if we measure the distance from the origin to the present state we can use that information to predict overflow oscillation. The distance from the origin to a state at (v_1 v_2) is given by $\sqrt{v_1^2 + v_2^2}$. If the input is zero, the next state is given by

$\mathbf{A} \cdot \mathbf{v} = \begin{bmatrix} a_{11}v_1 + a_{12}v_2 \\ a_{21}v_1 + a_{22}v_2 \end{bmatrix}$ so that in order that the distance to the next state be less than the distance to the present state we require

$\sqrt{(a_{11}v_1 + a_{12}v_2)^2 + (a_{21}v_1 + a_{22}v_2)^2} < \sqrt{v_1^2 + v_2^2}$ for no overflow oscillation.

This equation becomes

$$a_{11}^2v_1^2 + 2v_1v_2a_{11}a_{12} + a_{12}^2v_2^2 + a_{21}^2v_1^2 + 2v_1v_2a_{21}a_{22} + a_{22}^2v_2^2 < v_1^2 + v_2^2 \quad (8.34)$$

Equation (8.34) is easily solved if a_{12} = -a_{21} and $a_{11} = a_{22}$. In this case we get

$(a_{11}^2 + a_{12}^2)(v_1^2 + v_2^2) < v_1^2 + v_2^2$

This equation is satisfied if $(a_{11}^2 + a_{12}^2) < 1$. For a second order filter stage which has roots at $\alpha \pm j\beta$, a condition for stability is $\alpha^2 + \beta^2 < 1$. Therefore, if we can make $a_{11} = \alpha$ and $a_{12} = \beta$, we could guarantee that if the filter was stable it would also be free of overflow oscillation. The **A** matrix for such a filter would be

$$\mathbf{A} = \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix} \quad (8.35)$$

Since there are very few filters which naturally have an **A** matrix in the form of equation 8.13, the problem becomes one of how to transform the **A** matrix to that of a system which does not suffer from overflow oscillation [3]. Recall from equation (8.25) through (8.28) that we can define a transformation matrix **T** and create a system with an identical transfer function which has an **A** matrix given by

$\mathbf{A} \leftarrow \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$

We want to find a matrix **T** such that

$\overline{\mathbf{T}}^{-1} \cdot \overline{\mathbf{A}} \cdot \overline{\mathbf{T}} = \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$ or,

$$\frac{1}{|T|} \cdot \begin{bmatrix} T_{22} & -T_{12} \\ -T_{21} & T_{11} \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} = \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix} \quad (8.36)$$

The eigenvalues of the **A** matrix which are the roots of the characteristic equation are given by $\left|z\bar{\mathbf{I}} - \overline{\mathbf{A}}\right| = 0$ which leads to

$$\alpha = \frac{a_{11} + a_{22}}{2} \qquad \beta = \frac{\sqrt{-(a_{11} - a_{22})^2 - 4a_{12}a_{21}}}{2}$$

Making these substitutions into equation (8.36) we get (after several pages of algebra)

$$\overline{\mathbf{T}} = \begin{bmatrix} -1/\beta & 0 \\ -\alpha/\beta & 1 \end{bmatrix} \text{ and } \overline{\mathbf{T}}^{-1} = \begin{bmatrix} -\beta & 0 \\ -\alpha & 1 \end{bmatrix} \tag{8.37}$$

Any second order filter stage having complex roots can be transformed using equations (8.25) through (8.28) and (8.37) into a filter stage which is guaranteed to be free of overflow oscillation. It is important to note that this transformation does not guarantee that no overflow will take place—only that the length of the vector from the origin to the next state will be shorter than the length from the origin to the present state.

Example 8.14

Apply equations (8.25) through (8.28) and (8.37) to the system given below to create a new system that is free of overflow oscillation. Plot the zero input trajectory for each system from v_1, v_2 = 0.95, -0.95. Note that this is the same system as that shown in Figure (8.38).

$$z\mathbf{v} = \mathbf{A} \cdot \mathbf{v} + \mathbf{B} \cdot \mathbf{x}$$

$$\mathbf{y} = \mathbf{C} \cdot \mathbf{v} + \mathbf{D} \cdot \mathbf{x}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -.8 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 2 & 3 \end{bmatrix}, \text{ and } \mathbf{D} = \begin{bmatrix} 1 \end{bmatrix}$$

Solution:

We begin by finding values for α and β. The characteristic equation is given by $\left|z\mathbf{I} - \mathbf{A}\right| = 0$ or $z^2 - z + 0.8 = 0$.

The roots of this equation are z = 0.5 ± j0.7416 so that α =0.5 and β = 0.7416. We note that $\alpha^2 + \beta^2 = 0.8$ so that the system is stable. The transforming **T** matrix is given by

$$\mathbf{T} = \begin{bmatrix} -1/\beta & 0 \\ -\alpha/\beta & 1 \end{bmatrix} = \begin{bmatrix} -1.348 & 0 \\ .6742 & 1 \end{bmatrix} \text{ and } \mathbf{T}^{-1} = \begin{bmatrix} -0.7418 & 0 \\ .5 & 1 \end{bmatrix}$$

This gives (from equation 8.17)

$$\mathbf{A} \leftarrow \mathbf{T}^{-1} \cdot \mathbf{A} \cdot \mathbf{T} = \begin{bmatrix} -0.7418 & 0 \\ .5 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ -0.8 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1.348 & 0 \\ .6742 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & -0.7418 \\ 0.7418 & 0.5 \end{bmatrix}$$

$$\mathbf{B} \leftarrow \mathbf{T}^{-1} \cdot \mathbf{B} = \begin{bmatrix} -0.7418 & 0 \\ .5 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{C} \leftarrow \mathbf{C} \cdot \mathbf{T} = \begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} -1.348 & 0 \\ .6742 & 1 \end{bmatrix} = \begin{bmatrix} -4.7186 & 3 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 1 \end{bmatrix}$$

Using the **A** matrix a MATLAB® m-file can be written to plot the trajectory. A segment of this m-file is shown below. This segment plots the first 50 points of the trajectory and draws lines connecting the points in order. The resulting trajectory plot is shown in Figure 8.41.

```
num = [1 2 1];den = [1 -1 .8];
r = roots(den);
alpha = real(r(1,1));beta = imag(r(1,1));
T = [-1/beta 0;-alpha/beta 1];
At = T^-1*A*T;
Bt = T^-1*B;
Ct = C*T;
figure(1);clf;
axis([-1.2 1.2 -1.2 1.2]);
line([1 -1],[1 1]);line([-1 -1],[1 -1]);
line([-1 1],[-1 -1]);line([1 1],[-1 1]);
v = [0.95;-0.95];count = 0;
while(((v(1,1) ~= 0) & (v(2,1) ~= 0)) & count < 50)
   Vnew = At*v;
   % Use twos complement overflow
   Vnew = twosover(Vnew);
   v = [v(1,1) Vnew(1,1)];
   y = [v(2,1) Vnew(2,1)];
   line(x,y,'Color','blue');
   v = Vnew;
   count = count + 1;
end
```

```
%twosover
function s = twosover(x);
  for i = 1:length(x)
      if (x(i)>=1)
        x(i) = x(i) - 2;
      else
         if (x(i) < -1)
           x(i) = x(i) + 2;
         end
      end
  end
s = x;
```

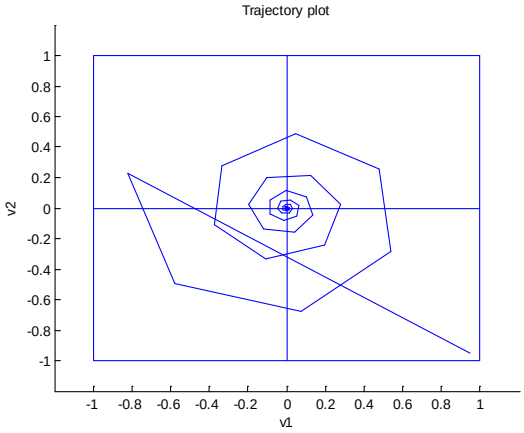

Figure 8.41
A trajectory plot for a transformed system. This is the same system whose trajectory is shown in Figure 8.40. Transforming the **A** matrix has removed the limit cycle oscillation due to two complement overflow.

In Figure 8.41 notice that with the starting state at 0.95, -0.95, and the trajectory spirals into zero. Compare this to the trajectory of Figure 8.40 which oscillates.

Ebert [4] et al and Roberts and Mullis [3] have shown that for a second order system with an A matrix given by

$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, there will be no overflow oscillation if

$a_{12} \cdot a_{21} \geq 0$

or if,

$a_{12} \cdot a_{21} < 0$ *and* $|a_{11} - a_{22}| + |\mathbf{A}| < 1$

There are other solutions to the overflow and overflow oscillation problem. One simple way to avoid overflow oscillation is to make use of saturation overflow characteristics instead of two's compliment. In saturation overflow all overflow results in the number being set to some maximum value. But implementation of saturation overflow requires additional hardware or additional software overhead to check every calculation and make the overflow correction if necessary.

Likewise some filter realizations are inherently free of overflow oscillation such as the lattice structure.

```
//SOSExample.c
April 16, 2009
//Second order sections
//0.041953  0.054929  0.041953  1.000000  -0.935199  0.296056
//1.000000  -0.091121  1.000000  1.000000  -0.986939  0.728775
//Section 1
const float A1[4] = {0, 1.0, -0.2961, 0.9352};
const float B1[2] = {0, 1};
const float C1[2] = {0.0295, 0.0942};;
const float D1 = 0.0420;
//Section 2
const float A2[4] = {0, 1.0, -0.7288,  0.9869};
const float B2[2] = {0, 1};
const float C2[2] = {0.2712, 0.8958};
const float D2 = 1;
int main(void)
  {unsigned int xInt;
   unsigned int dtoaOut;
   float x;
   float Zx11, Zx12, Zx21, Zx22, x11 = 0, x12 = 0, x21 = 0, x22 = 0;
   float y1, y2;
   while(1)
     {//A/D to xInt
      x = (float)xInt/1024.0;
      //Section 1
      Zx11 = A1[0]*x11 + A1[1]*x12 + B1[0]*x;
      Zx12 = A1[2]*x11 + A1[3]*x12 + B1[1]*x;
      y1 =  C1[0]*x11 + C1[1]*x12 + D1*x;
      //Section 2
      Zx21 = A2[0]*x21 + A2[1]*x22 + B2[0]*y1;
      Zx22 = A2[2]*x21 + A2[3]*x22 + B2[1]*y1;
      y2 =  C2[0]*x21 + C2[1]*x22 + D2*u;
      //Output
      //Send y to D/A
      x11 = Zx11;
      x12 = Zx12;
      x21 = Zx21;
      x22 = Zx22;
      //Wait for next cycle
     }
  }
```