Master MOSiG - M1

Report

Signal Fundamentals

Linear systems and Convolution

Computer Exercises N. 2

Authors

Jander Botelho do Nascimento

Raquel Araújo de Oliveira

Grenoble, 03 November 2010

1 Example of linear system: a recursive filter

1.1 On which condition (on y[0]) is this system linear? Prove that this system is linear (under this condition).

We know this:

x1[t] -> H -> y1[t] and x2[t] -> H -> y2[t]

=> x1[t] +x2[t] -> H -> y1[t] + y2[t] => H(x1[n] + x2[n]) = H(x1[n]) + H(x2[n]) = y1[n] + y2[n]

1)In order to prove in which condition of y[0] the system (A) is linear, we will first try to a simpler way, choosing the constant 0 (y[0] = 0).

(A) y[n] = 0.05 x[n] + 0.95 y[n-1]

Let's call P(n) the property that says that the system above is linear.

Suppose n=1

Additivity Property: H(x1[n] + x2[n]) = H(x1[n]) + H(x2[n])

Applying in the system (A):

H(x1[n] + x2[n])

=0.05(x1[n]+x2[n]) + 0.95 H(x1[n-1]+x2[n-1])

=0.05(x1[1]+x2[1]) + 0.95 H(x1[0]+x2[0])

=0.05(x1[1]+x2[1]) + (0.95 x 0)

=0.05x1[1]+0.05x2[1]

=H(x1[1])+H(x2[1])

So, P(1) is true.

2) Suppose P(n) is true for all n <= k

Then, for n = k + 1:

H(x1[n] + x2[n]) = H(x1[n]) + H(x2[n])

H(x1[n] + x2[n])

=0.05(x1[n]+x2[n]) + 0.95 H(x1[n-1]+x2[n-1])

=0.05(x1[k+1]+x2[k+1]) + 0.95 (H(x1[k])+H(x2[k]))

=0.05(x1[k+1]+x2[k+1]) + 0.95(y1[k] + y2[k])

=0.05x1[k+1] + 0.05x2[k+1] + 0.95y1[k] + 0.95y2[k]

=(0.05x1[k+1] + 0.95y1[k]) + (0.05x2[k+1] + 0.95y2[k])

=H(x1[k+1])+H(x2[k+1])

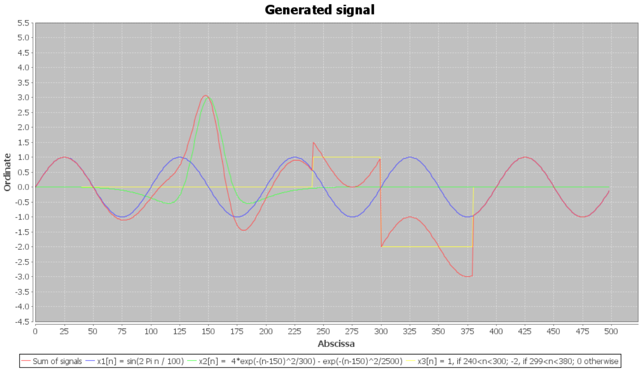
So, P(k+1) is true.

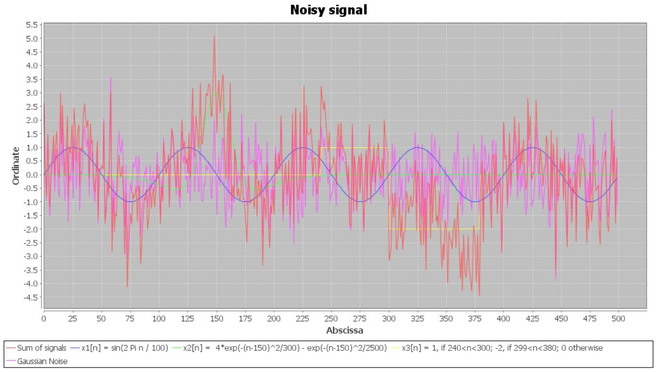
According of Mathematical Induction, if P(1) is true and P(k+1) is true, so P(n) is true for all n>=1.

1.2 Complete the corresponding Java code creating these input signals.

```
/**
 * Generates and returns the signal
 * x1[n] = sin(2 &Pi;n / 100)
 * @param nbSamples number of samples of the signal.
 * @return x1[n] = sin(2 &Pi;n / 100)
 */
public static Signal generateX1(int nbSamples) {
  Signal x1 = new Signal();
  x1.settName("x1[n] = sin(2 Pi n / 100)");

  // Write your code here
  double y;
  for(int i=0;i<nbSamples-1;i++){
      y=Math.sin((2.0 * Math.PI * i) / 100.0);
      x1.addElement((double) i, y);
  }
  return x1;
}

/**
 * Generates and returns the signal
 * x2[n] =  4*exp(-(n-150)^2/300) - exp(-(n-150)^2/2500)
 * @param nbSamples number of samples of the signal.
 * @return x2[n] =  4*exp(-(n-150)^2/300) - exp(-(n-150)^2/2500)
 */
public static Signal generateX2(int nbSamples) {
  Signal x2 = new Signal();
  x2.settName("x2[n] =  4*exp(-(n-150)^2/300) - exp(-(n-150)^2/2500)");

  // Write your code here
  double part1, part2, y;
  for(int i=0;i<nbSamples-1;i++){
      part1=i-150.0;
      part1=part1 * part1;
      part1=part1/300.0;
      part1=part1*(-1);
      part1=Math.exp(part1);
      part1=4*part1;
      part2=i-150.0;
      part2=part2 * part2;
      part2=part2 / 2500.0;
      part2=part2*(-1);
      part2=Math.exp(part2);

      y=part1-part2;
      x2.addElement((double) i, y);
  }
  return x2;
}

/**
 * Generates and returns the signal
 * x3[n] =  1 for 240 &lt; n &gt; 300
 *         -2 for 299 &lt; n &gt; 380
 *          0  otherwise
 * @param nbSamples
 * @return x3
 */
public static Signal generateX3(int nbSamples) {
  Signal x3 = new Signal();
  x3.settName("x3[n] = 1, if 240<n<300; -2, if 299<n<380; 0 otherwise");

  // Write your code here
  double y;
  for(int i=0;i<nbSamples-1;i++){
      if (240<i && i<300)
          y=1;
      else if(299<i && i<380)
          y=-2;
      else
          y=0;
      x3.addElement((double) i, y);
  }
  return x3;
}
```

1.3 Test and sum these 3 signals by generating 500 samples signals. (Menu 1: Generate x1, x2, x3, gaussian noise and sums) and save each signal.



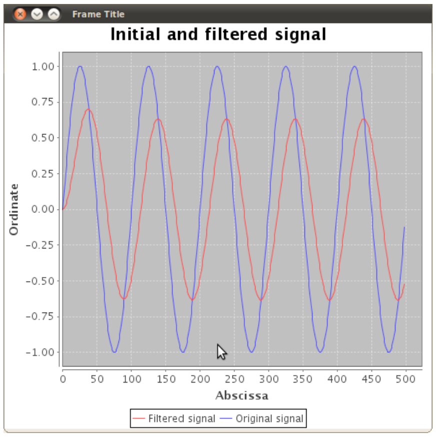


1.4 Complete the corresponding Java code.

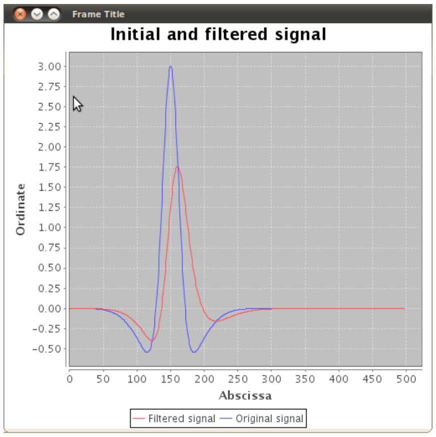
```
public Signal singlePolLowPassFilter() {
  Signal result = new Signal();
  result.settName("Filtered signal");
  // Write your code here
  result.addElement(0, 0);
  for(int n=1;n<getNbSamples();n++){
      double yOfNMinusOne=result.getValueOfIndex(n-1);
      double xOfN=this.getValueOfIndex(n);
      result.addElement(n, 0.05*xOfN+0.95*yOfNMinusOne);
  }
  //end of my code
  return result;
}
```

1.5 Pass each signal x1, x2, and x3 through the linear system. What do you observe?

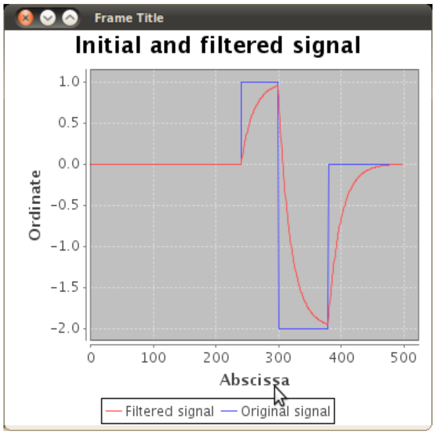
x1



x2



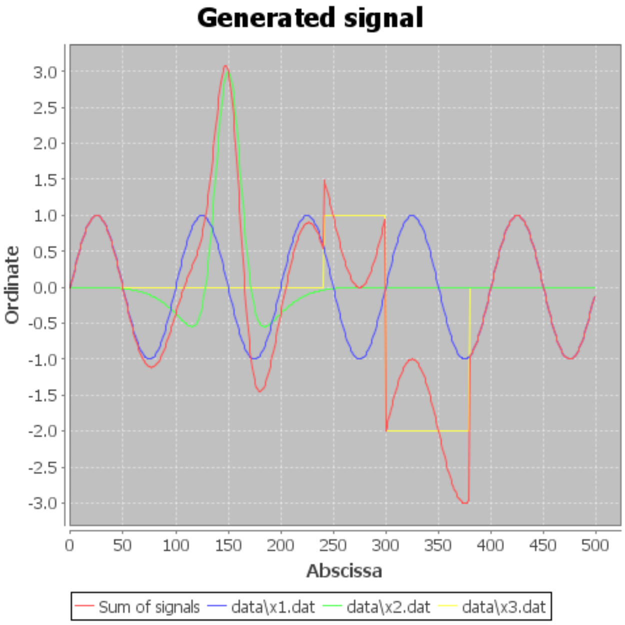
x3



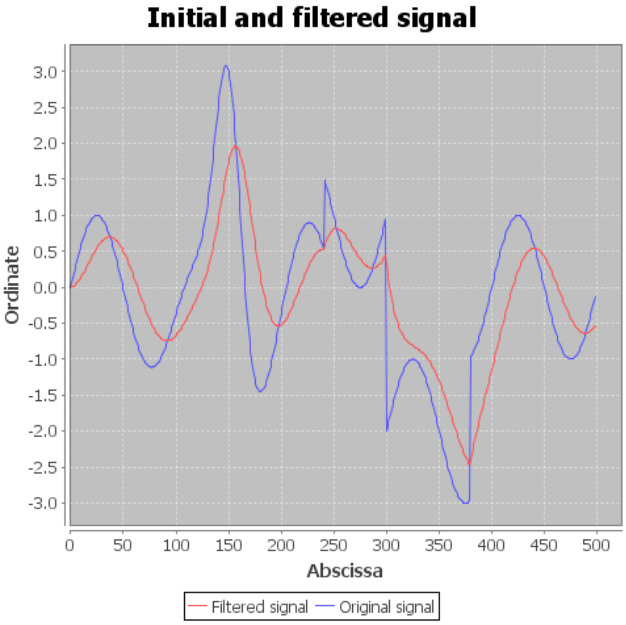
It's possible to see that the filter smooths all signal edges, reducing the noisy of the signals.

1.6 Pass the sum x = x1 + x2 + x3 through the linear system. Sum the output signals y= S(x1) + S(x2) + S(x3) = y1 + y2 + y3 and compare it to x. What do you observe? Why?

x (in red)



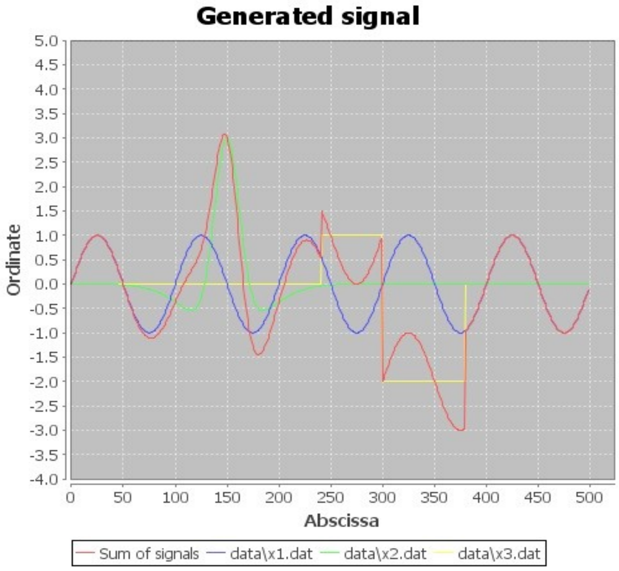
y (in red)



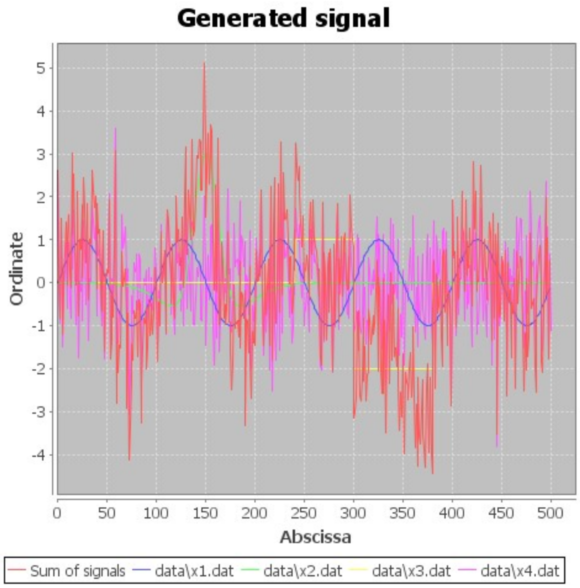
We can see that they are similar in their shape, but the result signal is smoothed. That happens because the linear system acts as a filter, smoothing out short-term fluctuations and highlighting longer-term trends or cycles.

1.7 Generate a signal xNoise = x1 + x2 + x3 + x4 and compare it with x.

x



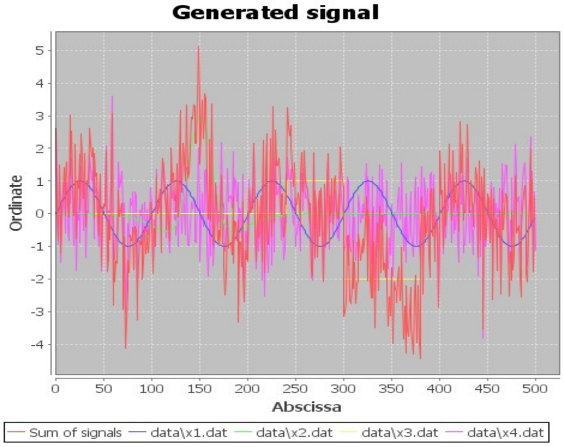
xNoise

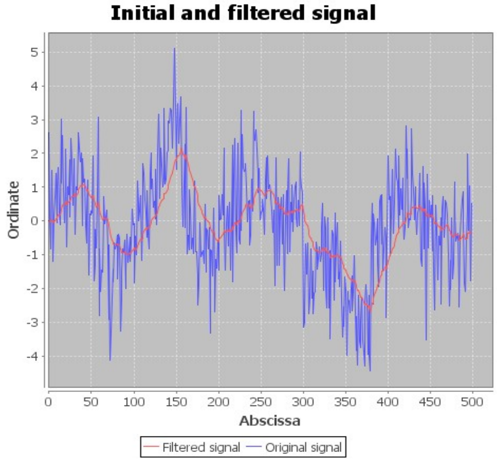


When we add the signal x4 (Gaussian), that represents the noise, the resultant signal is less clean, as if the noise pollute the signal. Furthermore, in the result signal the frequency is bigger than in the original signal and it is lesser smooth.

1.8 Pass xNoise through the linear system. What do you observe?

xNoise





After the signal through into the linear system, the signal becomes smoother, excesses are trimmed.

2 1D Convolution

2.1 Which condition on K ensures you the same number of operations between the 2 implementations?

When K is equal to N, the 2 implementations ensures us the same number of operations (where K is the size of the kernel and N is the size of the example signal).

2.2 Complete the corresponding Java code.

```
 /**
  * Truncates linear convolution of the current signal with the given kernel
  * (Note: the output signal has the same number of samples
  *  than the current signal)
  */
public Signal linearConvolve(Signal kernel) {
   Signal result = new Signal();
   result.settName("Convolved signal");
   int nbSamples = this.getNbSamples(); //N
   int kernelSize = kernel.getNbSamples(); //K

   // Write your code here
   for(int n=0;n<=nbSamples-1;n++){
      double y=0.0;
      for (int k=0;k<=kernelSize-1;k++)
         if (this.isInSeries(k) && kernel.isInSeries(n-k))
            y+= (this.getValueOfIndex(k) * kernel.getValueOfIndex(n-k));
      result.addElement(n, y);
   }
   return result;
 }

 /**
  * Circular convolve the current signal with the given kernel
  * (Note: the output signal has the same number of samples
  *  than the current signal)
  */
 public Signal circularConvolve(Signal kernel) {
   Signal result = new Signal();

   result.settName("Convolved signal");
   int nbSamples = this.getNbSamples();
   int kernelSize = kernel.getNbSamples();
```

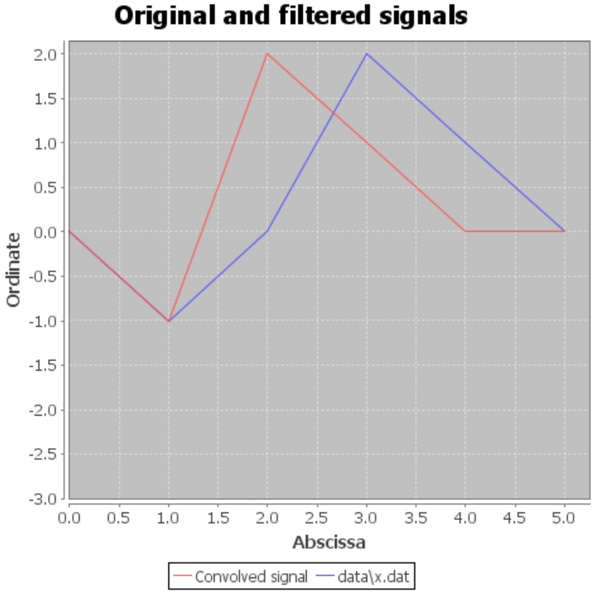
```
    // Write your code here
    //to swap the kernel
    Signal kernel2 = new Signal();
    for(int n=0;n<=kernelSize-1;n++)
      kernel2.addElement(n, kernel.getValueOfIndex(kernelSize-1-n));

    for(int n=0;n<=nbSamples-1;n++){
      double y=0.0;
      for (int k=0;k<=kernelSize-1;k++){

        int aux= ( n - (kernelSize/2) + k );
        if (this.isInSeries(aux) && kernel2.isInSeries((int)k))
            y+= (this.getValueOfIndex(aux) * kernel2.getValueOfIndex((int) k));
      }
      result.addElement(n, y);
    }
    return result;
  }
}
```

2.3 Create 2 signal files and check the result of truncated linear convolution and circular convolution.

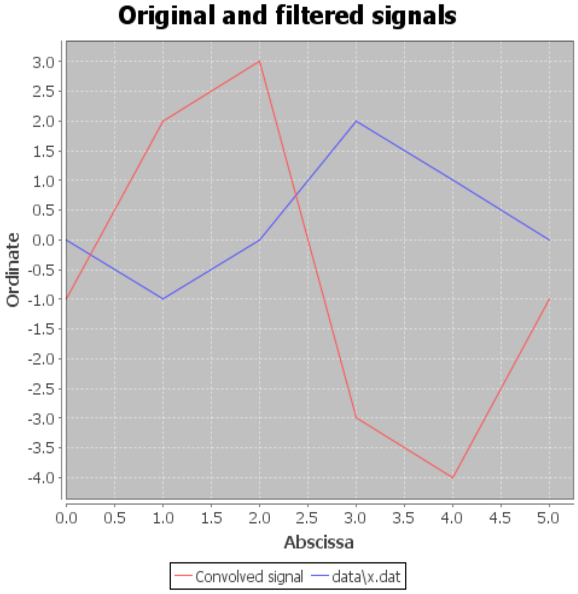
Files created.

2.4 What are the differences between both results ? Explain these differences with the equations 1 and 2.

Linear Convolution



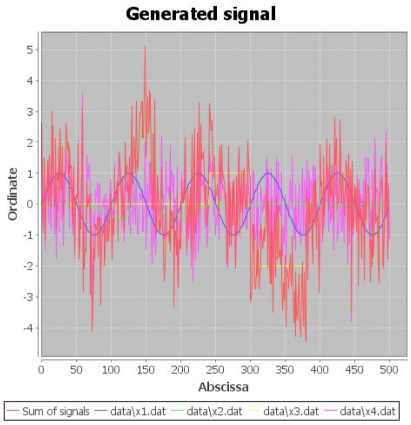
Circular Convolution



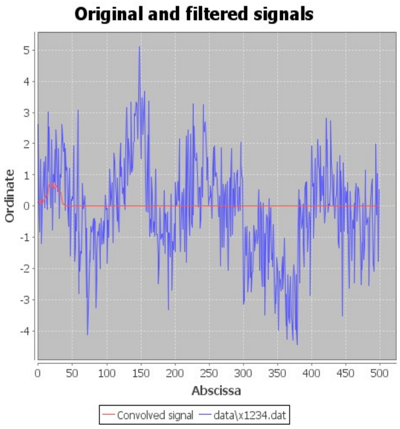
Convolution is the mathematical process that relates the output, y(t), of a linear, time-invariant system to its input, x(t), and impulse response, h(t). The continuous time relationship is described by the familiar convolution integral. For infinite discrete sequences, designated by x(n), y(n), and h(n), this integral relation reduces to another familiar expression, the convolution sum. For two finite discrete sequences of length N_x and N_h, the linear or aperiodic convolution sum takes on a slightly different form, reducing the bounds of the sum from the infinity to the length of the sequence. Essentially, convolution is a filtering process. Or, more appropriately, filtering is an application of convolution. For the discrete case, multiplication in the frequency domain translates to *circular* convolution in the time domain. Note how trailing samples that were left alone in linear convolution are in the circular convolution wrapped around, resulting in the shorter, different result.

2.5 Create a file to store the kernel of a 20-points filter. Convolve this kernel with the previous noisy signal x_{Noisy}. What do you observe?

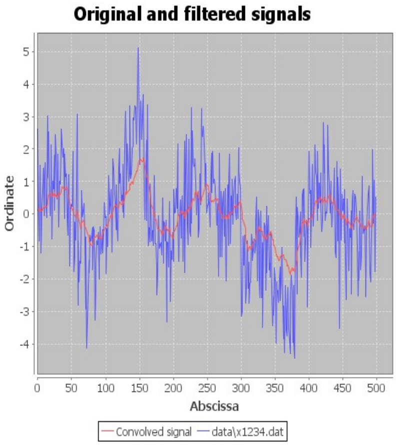
xNoise



Linear Convolution



Circular Convolution



Convolution is used to smooth signals contaminated by noise. This reduces the harmful influence of the noisy component, which often presents in the mix with the main signal.