Project Ts: Sampling and Reconstruction Demonstration

Cole Sheyka

February 2023

1 Abstract

The goal of this report is to explain the Shannon Sampling Theorem and provide insight to the theoretical and real methods used to attain a digital signal from a continuous, real-time signal. Ideally, we want to sample a continuous signal to create a reconstruction with the same properties as the original signal.

2 Introduction

The signal being studied can be seen in Figure 1 below.

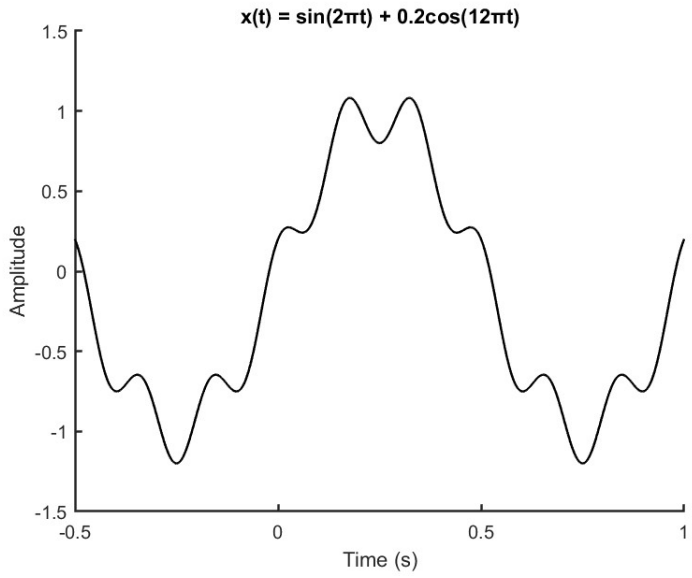

Figure 1: The Continuous signal

Figure 1 was generated in MATLAB by passing a time vector, t, into the original signal. The size of this time vector can be seen in the following code:

```
dt = 0.0001;     % small time step
t = -20:dt:20;   % time vector
```

Such a small time step ensures that $x(t)$ digitally behaves as the original continuous signal. Although in reality, Figure 1 is not truly continuous.

To analyze this signal with a digital device, the original signal, $x(t)$, is sampled at discrete points, kTs. Here, $k\epsilon\mathbb{Z}$ and Ts is the sampling period. Once we have a discrete-time sampled vector, with convolution we can digitally reconstruct the original signal. Since time in $x^*(t)$ is an integer with value kTs as $k \to \infty$, we can write

$$x^*(t) = x(kTs) = x[k]$$

where $x[k]$ is shorthand for a discrete vector of sampled $x(kTs)$.

Below, we will discuss sampling and reconstruction methods with two sampling frequencies. That is, $Ts = 0.17$ and $Ts = 0.017$. These frequencies were specifically chosen to observe aliasing and decent reconstruction simultaneously. By convolving data sampled with these two frequencies and a specified filter, we can observe a resulting reconstruction of the original signal.

3 Unit Impulse (Delta Function)

The unit impulse, or Dirac delta signal can be defined as follows,

Let

$$d_\epsilon = \begin{cases} \frac{1}{\epsilon} & \frac{-\epsilon}{2} \leq x \leq \frac{\epsilon}{2}, \\ 0 & otherwise. \end{cases}$$

The unit impulse signal is

$$\delta(x) = \lim_{\epsilon \to \infty} d_\epsilon(x).$$

As $\epsilon \to \infty$, the area under the curve is always unity. Therefore, the area under the unit impulse is unity.

Moreover, if we define the unit step signal as

$$u(x) = \begin{cases} 0 & x < 0, \\ 1 & x \geq 0 \end{cases} \tag{1}$$

Then,

$$\int_{-\infty}^{\infty} \delta(v)dv = u(x) \tag{2}$$

Equation 3 is another way to interpret the limit from above.

In terms of complex exponentials, the unit impulse can be derived by

$$\delta(x) = \frac{1}{2\pi} \sum_{-\infty}^{\infty} e^{\pm jkx} dk$$

4 The Dirac Comb

The comb signal, or $\tilde{\delta}_{T_s}$, is a periodic signal with unit impulses spaced apart by the sampling frequency.

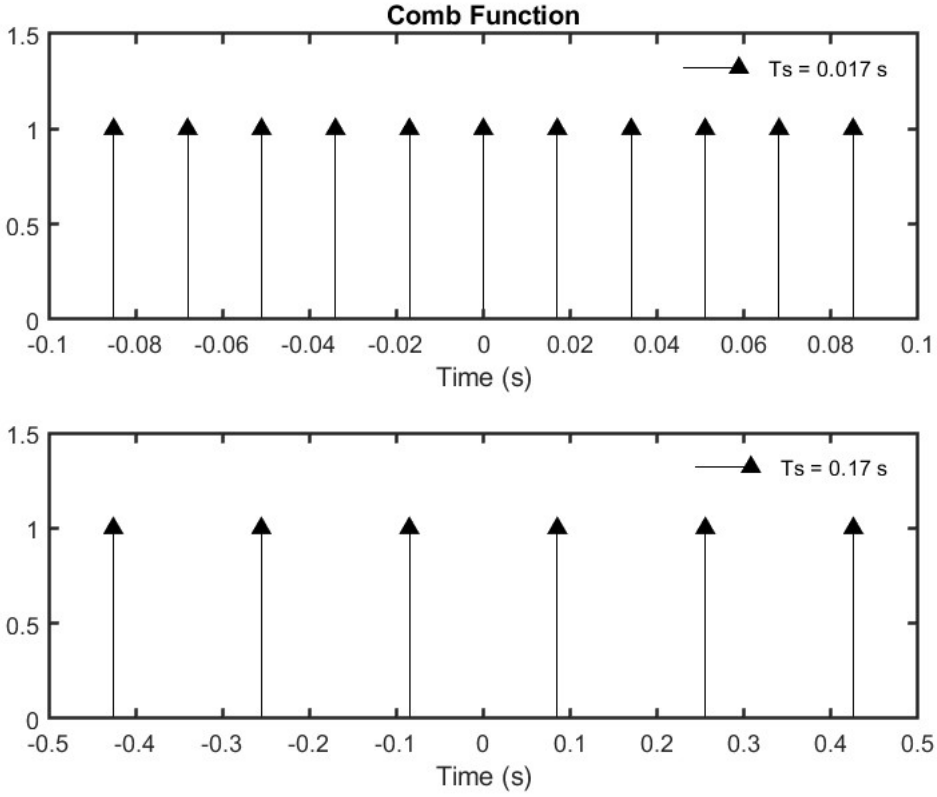

Figure 2: Two respective comb signals for each sampling frequency

Figure 2 shows two Dirac comb signals with respect to the sampling period in time. Multiplication in time with a continuous signal and $\tilde{\delta}_{T_s}$ results in a sampled signal. Convolution with the aforementioned allows periodicity with the sampled function.

5 Convolution

Convolution with a comb function can be written as

$$x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t-\tau)d\tau \tag{3}$$

where τ is a dummy variable. Equation 3 says that the original function, $x(t)$, is equal to the sum of infinite delta functions separated by an infinitesimally small distance.

Convolution also allows us to sample $x(t)$ at a specific t_0. It follows that for any continuous signal $x(t)$,

$$X(t_0) = \int_{-\infty}^{\infty} x(t)\delta(t-t_0)dt \tag{4}$$

Equation 4 says that the magnitude of $X(t_0)$ is equal to the convolution of a continuous-time signal with a unit impulse time-shifted to t_0.

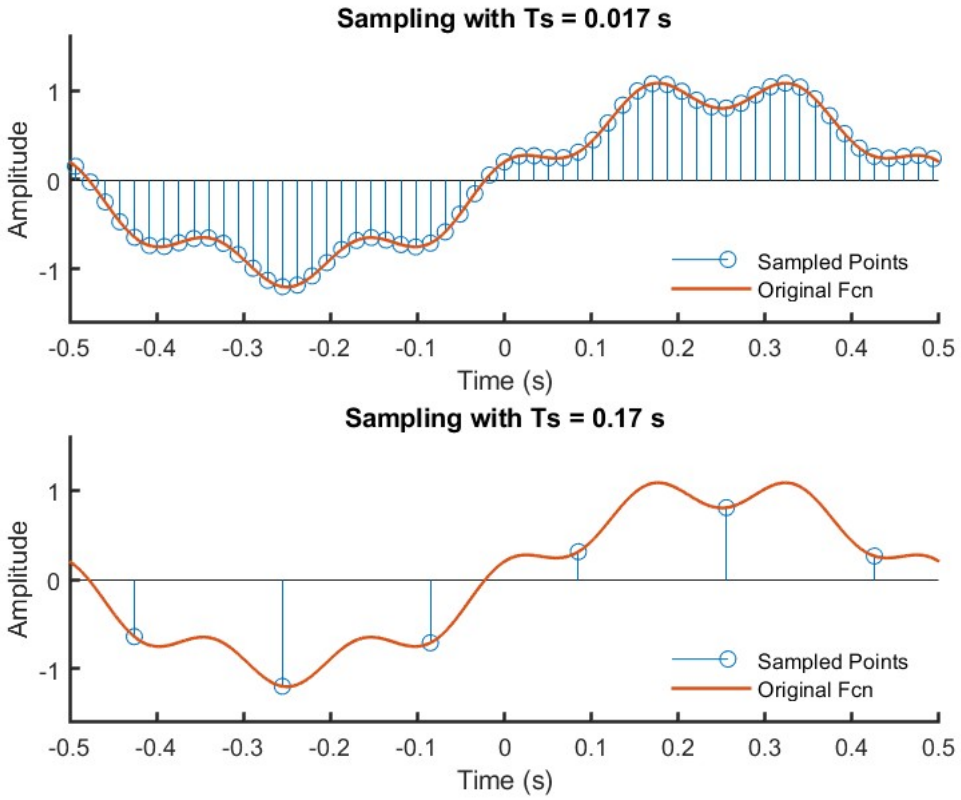

Figure 3: Sampling the original continuous-time signal at two different sample rates

In Figure 3, we see that the higher sampling frequency results in $x[k]$ having more data points to represent the continuous-time signal. We could also code $x[k]$ with the following commands in MATLAB:

```
Ts = 0.17;            % Example sample period
tTs = -20:Ts:20;      % Build sample vector
xk = sin(2*pi*tTs) + 0.2*cos(12*pi*tTs);     % x*(t)=x(kTs)=x[k]
```

In this code example, xk is a vector of discrete-time points 'sampled' from the original signal.

Later on we will model reconstruction in time by modifying Equation 3 and saying that

$$x_R(t) = \int_{-\infty}^{\infty} x^*(t)h(t-\tau)d\tau \tag{5}$$

where $h(t)$ is the impulse response (the model) of a filter. It follows that convolution of sampled data with the impulse response of various filters results in some form of reconstruction, $x_R(t)$.

6 Fourier Analysis

A Fourier Transform decomposes a function of time into sine and cosine waves. Like Taylor Series which approximates functions as a sum of polynomials, Fourier series approximates continuous and periodic signals as a sum of trigonometric functions. Consider the following infinite series:

$$\tilde{x}(t) = A_0 + A_1 cos(1t) + A_2 cos(2t) + ... + B_1 sin(1t) + B_2 sin(2t) + ...$$

$$= A_0 + \sum_{k=1}^{\infty} A_k cos(nt) + \sum_{k=1}^{\infty} B_k sin(nt)$$

The Fourier Series equation is more commonly written as:

$$\tilde{x}(t) = \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k cos(k\omega t) + \sum_{k=1}^{\infty} B_k sin(k\omega t) \tag{6}$$

where $f(t)$ is a periodic and continuous function of time. If the signal is periodic, then the synthesis equation can be used to map the discrete points in the frequency domain back to continuity in the time domain by virtue of a sum of cosine and sine signals. The process of summing these trigonometric signals is defined in the Fourier series equation.

It turns out that the result in Equation 6 is a pretty good approximation for real time signals. With Euler's identity, we can rewrite Equation 6 as:

$$\tilde{x}(t) = \sum_{k=-\infty}^{\infty} C_k e^{j\omega_0 kt} \tag{7}$$

where

$$C_k = \frac{1}{T_0} \int_{T_0} \tilde{x}(t) e^{-j\omega_0 kt} dt \tag{8}$$

In Equation 8, we are integrating over one period, T_0. Equations 7 and 8 are known as the Synthesis equation and Analysis equation, respectively. Rewriting Equation 8 we have

$$T_0 C_k = \int_{-\infty}^{\infty} \tilde{x}(t) e^{-j\omega_0 t} dt$$

As $T_0 \to \infty$,

$$\begin{aligned} T_0 C_k &= X(\omega_0) \\ X(\omega) &= \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \end{aligned} \tag{9}$$

This result is known as the forward Fourier Transform (FT). $X(\omega)$ in Equation 9 tells the magnitude and phase of the continuous signal $x(t)$ by breaking it up into its real and imaginary parts. Rewriting Equation 9 we have

$$X(\omega) = \int_{-\infty}^{\infty} x(t) cos(\omega t) dt + j \int_{-\infty}^{\infty} x(t) sin(\omega t) dt \tag{10}$$

Equation 10 is equivalent to equation 9, but it helps to visualize $X(\omega)$ in its $a + jb$ form. The integral on the left is the magnitude of the real part of $x(t)$ and the right is the magnitude of the imaginary part. Using the Pythagorean theorem, we can visualize magnitude and phase of the original signal in this way.

Likewise, with the Synthesis equation,

$$\tilde{x}(t) = \sum_{k=-\infty}^{\infty} C_k e^{j\omega_0 kt}$$

$$= \sum_{k=-\infty}^{\infty} TC_k e^{j\omega_0 kt} \frac{1}{T}$$

$$= \sum_{k=-\infty}^{\infty} TC_k e^{j\omega_0 kt} \frac{\omega_0}{2\pi}$$

$$= \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} X(\omega) e^{j\omega_0 kt} \Delta\omega$$

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{j\omega_0 t} d\omega_0 \tag{11}$$

The result in Equation 11 is known as the inverse Fourier transform (IFT). Like in Equation 10, the tilde in the periodic $\tilde{x}(t)$ is dropped because both the IFT and FT work on any continuous signal.

Somehow, $x^*(t)$ can be explained as the product of the $x(t)$ and $\tilde{\delta}_{T_s}$ where

$$\tilde{\delta}_{T_s} = \sum_{k=-\infty}^{\infty} \delta(t - kTs).$$

Since $\tilde{\delta}_{T_s}$ is periodic, then we can equate it to the synthesis equation. Now,

$$\tilde{\delta}_{T_s} = \sum_{k=-\infty}^{\infty} C_k e^{j\omega_0 kt}.$$

Rewriting the analysis equation, we have

$$C_k = \frac{1}{T_0} \int_{T_0} \sum_{k=-\infty}^{\infty} \delta(t - kTs) e^{-j\omega_0 kt} dt.$$

Since there is only one Dirac function within the range of the integral, the equation reduces to

$$C_k = \frac{1}{T_0} \int_{T_0} \delta(t) e^{-j\omega_0 kt} dt.$$

$$C_k = \frac{1}{T_0}.$$

Therefore the Fourier series of the impulse train is

$$\tilde{\delta}_{T_s} = \sum_{k=-\infty}^{\infty} \delta(t - kTs) = \frac{1}{T_0} \sum_{k=-\infty}^{\infty} C_k e^{j\omega_0 kt} \tag{12}$$

This equation is also known as the discrete time Fourier transform (DTFT).

7 Nyquist-Shannon Sampling Theorem

Rewriting the FT to represent a discrete-time signal we have

$$X^*(\omega) = \int_{-\infty}^{\infty} x^*(t) e^{-j\omega_0 t} dt$$

$$= \frac{1}{T_s} \int_{-\infty}^{\infty} x(t) [\sum_{k=-\infty}^{\infty} e^{j\omega_s kt}] e^{-j\omega_0 t} dt$$

$$= \frac{1}{T_s} \sum_{k=-\infty}^{\infty} \int_{-\infty}^{\infty} x(t) e^{-j(\omega_0 - \omega_s k)t} dt$$

$$= \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X(\omega)\Big|_{\omega=\omega_0 - k\omega_s}$$

$$X^*(\omega) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X(\omega_0 - k\omega_s) \tag{13}$$

where $\omega_0 = \omega_{max}$ and ω_s is the sampling frequency.

Equation 18 represents a discrete reconstruction of the original sampled signal in the frequency domain. It effectively creates a discrete spectrum of the original by shifting by $k\omega_s$.

It is extremely critical that $\omega_s > 2\omega_0$. However, in practice it's best to choose $\omega_s = 10\omega_0$. Otherwise, the reconstruction of the original $X(\omega)$ will have overlap, or barley resemble the original signal. This phenomena of overlapping in reconstruction is called aliasing. Aliasing causes the reconstruction to inaccurately copy the original signal. In practice, the original signal is passed through a filter with a specified cutoff frequency such that most of the desired information is captured.

Considering the original signal

$$x(t) = sin(2\pi t) + 0.2cos(12\pi t)$$

the highest frequency is $0.167rad/s$. Therefore, $\omega_0 = 0.167rad/s$. Choosing to sample at $\omega_s = 0.17rad/s$ will result in aliasing, and choosing $\omega_s = 0.017rad/s$ should result in a nice periodic reconstruction of the original signal.

8 The Ideal Low Pass Filter (ILP) and the Sinc Function

This section will discuss different methods to reconstruct a signal.

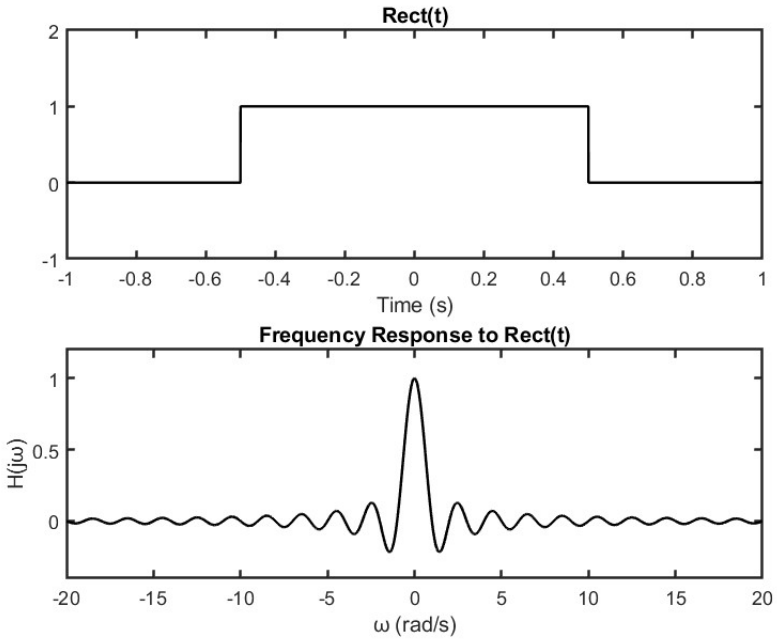

Figure 4: Rectangular function (top) versus its impulse response (bottom)

The impulse response to the unit pulse can be derived as follows. First, let the unit pulse be defined as

$$h(t) = \begin{cases} 1 & \frac{-1}{2} < t < \frac{1}{2}, \\ 0 & otherwise \end{cases}$$

Then using the IFT,

$$H(\omega) = \frac{1}{2\pi} \int_T e^{j\omega t} d\omega H(\omega) = sinc(\omega) \tag{14}$$

Figure 4 shows that the FT of the rectangular function is the sinc function in frequency. Since the Fourier equations have some sense of duality, we can construct a filter, which is similar to the rectangular function, in the frequency domain. This filter is known as the ideal low pass filter, and it can be seen in Figure 6.

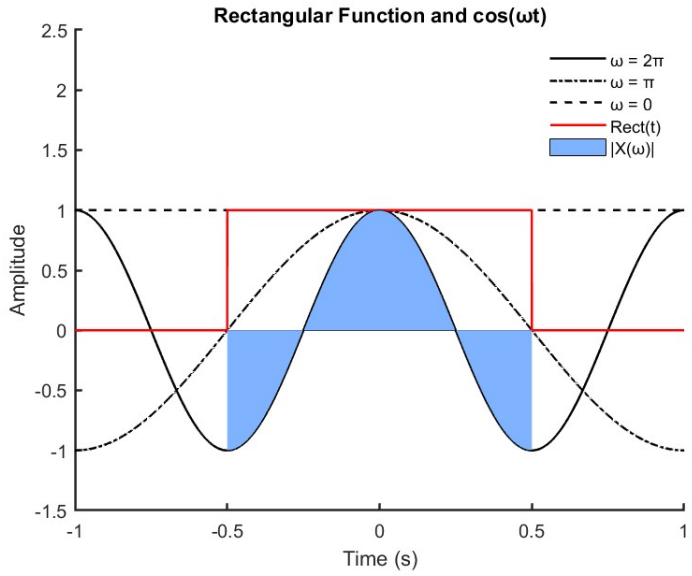

Figure 5: Area under $cos(\omega t)$ when multiplied by $rect(t)$

With Equation 12, we can visualize the magnitude of sinc as a function of ω. Considering Figure 5, since the sine wave is an odd function, integrating it over one period is always zero. Figure 5 says that the convolution of the rectangular function with $cos(\omega_k t)$ is equal to the magnitude of $sinc(\omega_k)$.

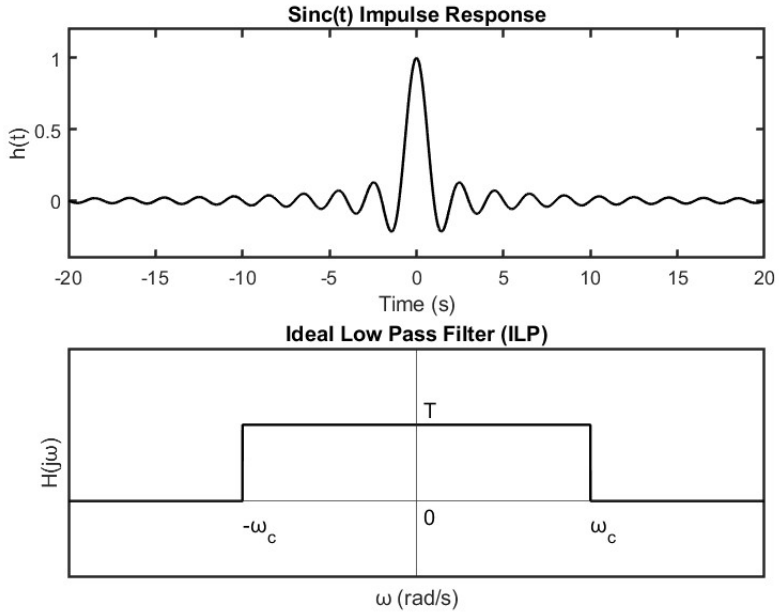

Figure 6: Applying the ILP to filter a signal results in perfect reconstruction of the original signal

In Figure 6, ω_c is the cutoff frequency and can be defined as $\omega_c = \omega_o = \frac{\omega_s}{2}$. Let the ILP be defined as

$$H(j\omega) = \begin{cases} T_s & \frac{-\omega_s}{2} < t < \frac{\omega_s}{2}, \\ 0 & otherwise \end{cases}$$

Then by the IFT,

$$h(t) = \frac{1}{2pi} \int_{\frac{-\omega_s}{2}}^{\frac{\omega_s}{2}} T_s e^{j\omega t}$$

$$= (\frac{2}{2}) \frac{1}{j\omega_s t} [e^{j\omega t}]_{\frac{-\omega_s}{2}}^{\frac{\omega_s}{2}}$$

$$= \frac{2}{\omega_s t} (\frac{\frac{e^{j\omega_s t}}{2} - \frac{e^{2j\omega_s t}}{2}}{2j})$$

$$= \frac{sin(\frac{\omega_s t}{2})}{\frac{\omega_s t}{2}}$$

$$h(t) = sinc(t). \tag{15}$$

This filter is called the 'ideal' low pass filter because the impulse response in non-causal meaning that it isn't real. The sinc function in time cannot be achieved with digital systems because it is dependent on present and past inputs. Since we cannot predict the future, the ILP is not actually used in practice. Although, it does give some good insight to ideal reconstruction.

9 Zero Order Hold (ZOH)

ZOH is probably the most common method used to reconstruct a signal because it is the simplest to perform. A computer can perform a ZOH on a continuous signal by sampling once every Ts. For $Ts_k \leq t < Ts_{k+1}$ the magnitude of $x[k]$ is constant.

Notice in Figure 7 that the ZOH with a sampling rate of 0.017 s does a pretty good job at reconstructing the original signal.

Figure 8 is the ZOH filter which samples at the faster $Ts = 0.017s$. At time 0, the magnitude of the original signal is sampled and scaled by 1, then the magnitude remains constant until $Ts > 0.017s$.

Interestingly, the ZOH filter is what's used in the digital-to-analog converter (DAC) on the Arduino DUE. In a separate experiment a Simulink program was built to upload the original signal to output the reconstruction through an Arduino DUE's DAC0 pin. The results in the figure below were first saved by

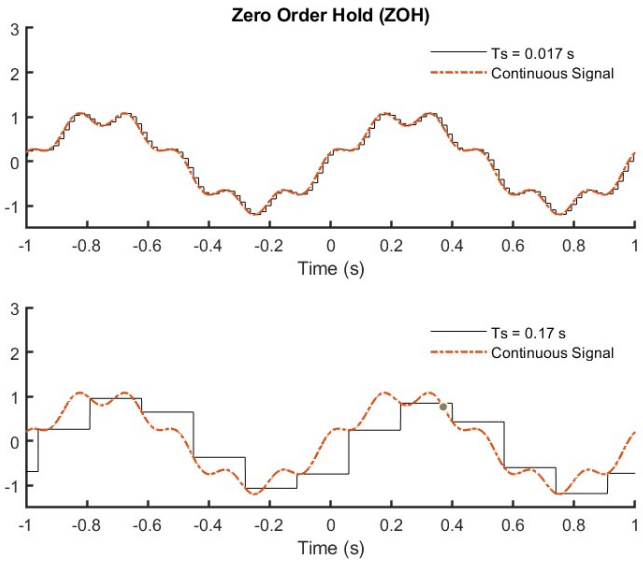

Figure 7: Using ZOH to reconstruct the original signal

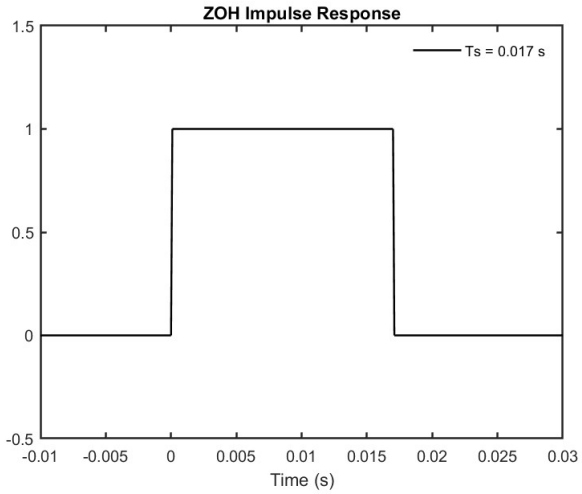

Figure 8: Impulse response of the ZOH

measuring the output voltage on an oscilloscope then plotting the data points in MATLAB.

The results in Figure 9 are extremely similar to the plot seen in Figure 7.

The sine wave function in Simulink is a 12-bit signal, meaning that it sends a decimal value between 0 and 4095 to a 16-bit unsigned integer converting block. The ARM processor on the Arduino DUE is a 32 bit register for the DACCs. The first 16 bits are for DAC0 and the second 16 bits are for DAC1. However, these pins on the DUE have 12 bits of resolution with an output range of 0.55 V to 2.75 V. Therefore, the range of bits between the 2^{16th} bit and the 2^{12th} bit map to 2.75 V following the data type conversion, and the first bit maps to 0.55 V. In Simulink, the DAC analog output block expects a uint16 data type, but any value above 4095 saturates the voltage output of a DAC pin on the Arduino

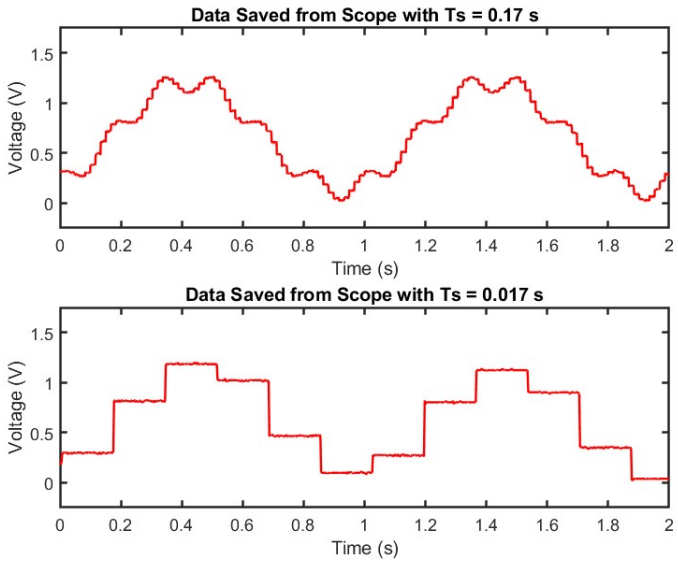

Figure 9: An Arduino DUE's reconstruction of the original signal

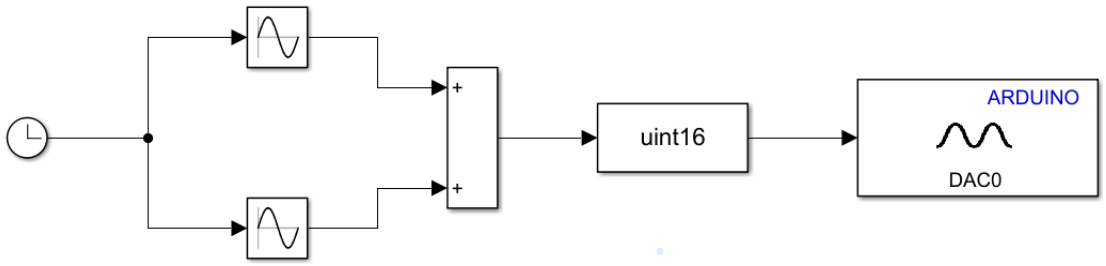

Figure 10: Simulink block diagram showing the set up to upload the original function to the Arduino DUE

DUE.

10 First Order Hold (FOH)

Another model for reconstruction is the FOH. With this filter, the reconstruction is a piecewise linear approximation of the original sampled signal. In real time, the FOH filter cannot be used without a delay because the algorithm inherently requires a sample of time before that sample can happen. Given that the function being analyzed has already been passed through time, it is possible to plot the reconstruction of the original signal using the FOH filter.

With the higher sampling rate, we were able to achieve basically a perfect reconstruction of the original signal. Although, if the Arduino DUE was to apply this filter in its DAC circuit, then the output signal would have to be phase shifted by $\frac{\omega_s}{2}$.

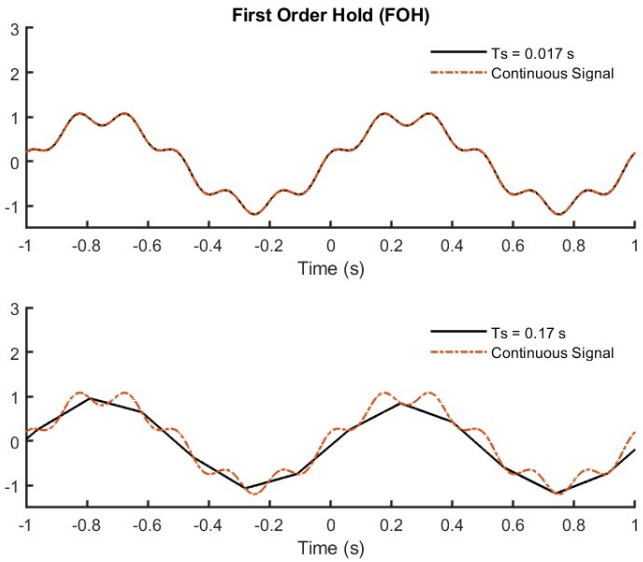

Figure 11: FOH reconstruction of the original signal

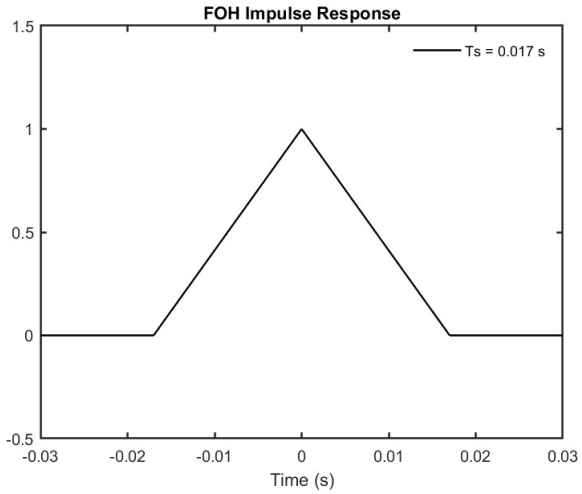

Figure 12: FOH impulse response

In figure 12, we see the actual FOH filter. This is also known as the triangular function. Applying this filter to a continuous signal basically connects the dots between each $x[k]$.

11 Predictive First Order Hold (PFOH)

The last type of filter to be discussed is the PFOH. The PFOH uses linear interpolation to predict what the next value of $x*(t)$ is. Its element-wise form can be written as

$$x[k+1] = x[k] + \frac{x[k] - x[k-1]}{Ts}t \tag{16}$$

where t is the variable interpolating between adjacent points. This model is causal, so it can be used to reconstruct a signal in real time.

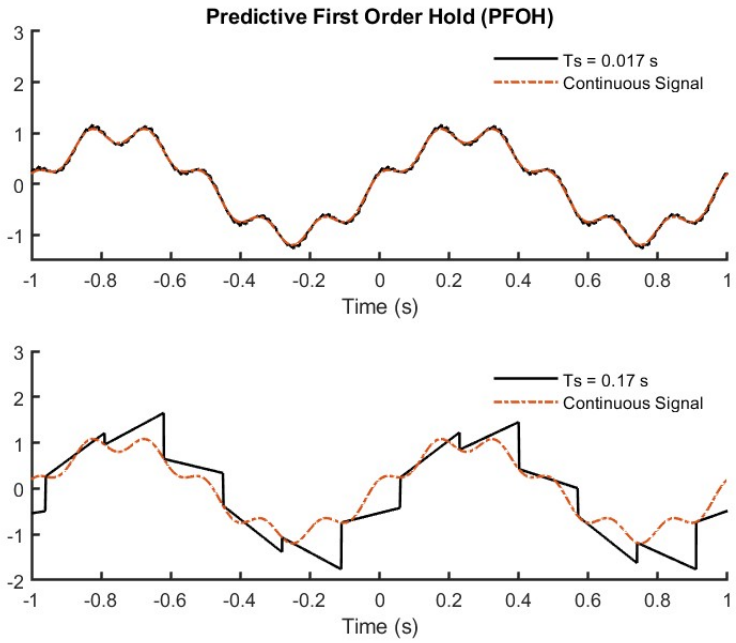

Figure 13: PFOH reconstruction of the original signal

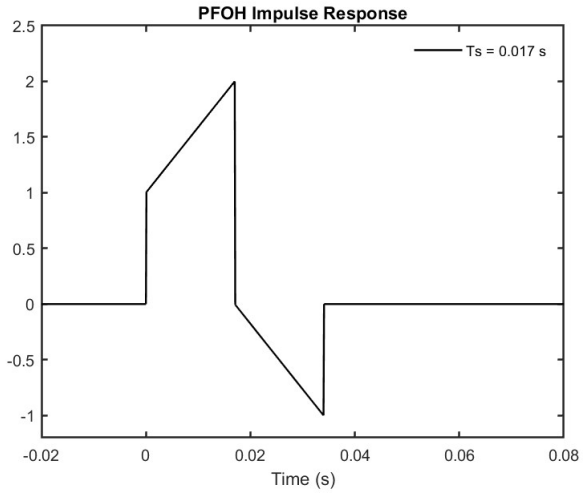

Figure 14: PFOH reconstruction of the original signal

The impulse response of the PFOH is seen in Figure 14. Notice that in Figure 14, the slope of the top and bottom curves are $\frac{1}{Ts}$. This filter approximates the original function by interpolating between the current point in time and the lase point in time to 'predict' what the magnitude of $x[k]$ is.

12 Conclusion

The FT theoretically decomposes a continuous time signal into an infinite amount of discrete Dirac delta functions. The cutoff frequency is the maximum frequency in an analog signal, and the Nyquist-Shannon Sampling theorem helps to identify the proper sampling rate to avoid aliasing. Using a ILP with a signal will result in perfect reconstruction because all points in the original signal are stored for replication and periodicity. Since digital systems can only sample in finite time, then other models must be used to interpret real time signals. The ZOH, PFOH, and a phase-shifted FOH are causal filters, and they can reconstruct a real-time signal fairly well.

13 MATLAB Code

```
%-----------------------------------------------------------------
% Automation Project 2
% Written by Cole Sheyka 4/3/23
%-----------------------------------------------------------------
clc, clear, close all;

% Read data collected from ARD
Ts_data = readmatrix("Ts.csv");
TsA_data = readmatrix("TsA.csv");

Vpps = max(Ts_data(:,2)) - min(Ts_data(:,2));

%-----------------------------------------------------------------
dt = 0.0001;    % small time step
t = -20:dt:20;  % time vector

% Sample periods
Ts = 0.017; % no aliasing
TsA = 0.17; % aliasing

% sample freqs
ws = 1/ Ts;  % no ailiasing
wA = 1/ TsA; % aliasing


% Set up sample vectors in time
ts_ = -20:Ts:20;
tsA_ = -20:TsA:20;

X_Ts = linspace(-20,20,length(ts_));
X_TsA = linspace(-20,20,length(tsA_));
```

```
% Sampled vectors from radio fcn
xk = sample(Ts,t,@radio);
xkA = sample(TsA,t,@radio);

% % FT of original fcn
% tw = -1/ws:1-1/ws;
% FT = fft(radio(tw));
%
% N = length(FT);
% f = (-N/2:N/2-1)/N*ws;
% stem(f)


%-----------------------------------------------------------------
% plot original fcn
figure;
hold on
title('x(t) = sin(2 t ) + 0.2cos(12 t )')
xlim([-.5,1])
plot(t,radio(t),'linewidth',1.2,'color','k')
xlabel('Time (s)')
ylabel('Amplitude')
set(gca,'linewidth',1.2)
hold off

%-----------------------------------------------------------------
% plot the combs
s1 = ones(size(X_Ts));
s2 = ones(size(X_TsA));

figure;
subplot(2,1,1)
stem(X_Ts,s1,"filled",'^','Color','k')
```

```
xlim([-.1,.1])
ylim([0,1.5])
title('Comb Function')
xlabel('Time (s)')
legend('Ts = 0.017 s')
legend boxoff
set(gca,'linewidth',1.2)

subplot(2,1,2)
stem(X_TsA,s2,"filled",'^','color','k')
xlim([-.5,.5])
ylim([0,1.5])
xlabel('Time (s)')
legend('Ts = 0.17 s')
legend boxoff
set(gca,'linewidth',1.2)

%-----------------------------------------------------------------
% plot x*(t)
figure;
subplot(2,1,1)
hold on
stem(X_Ts,radio(X_Ts))
plot(t,radio(t),'linewidth',1.2)
xlim([-.5,.5])
ylim([-1.6,1.6])
title('Sampling with Ts = 0.017 s')
xlabel('Time (s)')
ylabel('Amplitude')
legend("Sampled Points",'Original Fcn','Location','southeast')
legend boxoff
set(gca,'linewidth',1.2)
hold off
```

```
subplot(2,1,2)
hold on
stem(X_TsA,radio(X_TsA))
plot(t,radio(t),'linewidth',1.2)
xlim([-.5,.5])
ylim([-1.6,1.6])
title('Sampling with Ts = 0.17 s')
xlabel('Time (s)')
ylabel('Amplitude')
legend("Sampled Points",'Original Fcn','Location','southeast')
legend boxoff
set(gca,'linewidth',1.2)
hold off

%-----------------------------------------------------------------
% sinc function with t vector
% Impulse response of ILP
y = sin(pi*t)./(pi*t);

figure;
subplot(2,1,2)
syms x
fplot(rectangularPulse(x), [-1 1],'linewidth',1.2,'color','k')
ylim([-1,2])
title('Ideal Low Pass Filter (ILP)')
yline(0)
xt = [-.5 .5 0.02 0.02];
yt = [-.35 -.35 1.2 -.2];
xl = xline(0);
str = {'- _c ',' _c ','T','0'};
text(xt,yt,str)
xlabel('   (rad/s)')
```

```
ylabel('H( j  )')
set(gca,'linewidth',1.2)
set(gca,'XTick',[], 'YTick', [])

subplot(2,1,1)
plot(t,y,'linewidth',1.2,'Color','k')
title('Sinc(t) Impulse Response')
ylabel('h(t)')
xlabel('Time (s)')
ylim([-.4,1.2])
set(gca,'linewidth',1.2)

figure;
subplot(2,1,2)
plot(t,y,'linewidth',1.2,'Color','k')

% Normalized Sinc(t) With Spectral Frequency Components
title('Frequency Response to Rect(t)')
ylabel('H( j  )')
xlabel('   (rad/s)')
ylim([-.4,1.2])
set(gca,'linewidth',1.2)

subplot(2,1,1)
fplot(rectangularPulse(x), [-1 1],'linewidth',1.2,'color','k')
ylim([-1,2])
title('Rect(t)')
xlabel('Time (s)')
set(gca,'linewidth',1.2)

%-------------------------------------------------------------
% ILP time domain
Lp = -.5:dt:.5;
```

```
figure;
hold on
title('Rectangular Function and cos( t )')

plot(t,cos(2*pi.* t),'linewidth',1.2,'color','k')
plot(t,cos(pi.* t),'-.','linewidth',1.2,'color','k')
plot(t,cos(0.* t),'--','linewidth',1.2,'color','k')

syms x
fplot(rectangularPulse(x), [-1 1],'linewidth',1.2,'color','r')

area(Lp,cos(2*pi.*Lp),'facecolor',"#80B3FF")
% stem([-.5 .5],-[cos(2*pi/-.5) cos(2*pi/.5)],".","--",'color','k')

xlim([-1,1])
ylim([-1.5,2.5])
xlabel('Time (s)')
ylabel('Amplitude')
legend('   = 2  ','   =   ','   = 0','Rect(t)','|X(  )|')
legend boxoff
set(gca,'linewidth',1.2)
hold off

%-----------------------------------------------------------------
% Zero Order Hold (ZOH)
h1 = ZOH(t,Ts);
h2 = ZOH(t,TsA);

figure;
plot(t,h1,'color','k','linewidth',1.2)
title('ZOH Impulse Response')
xlabel('Time (s)')
```

```
xlim([-.01,.03])
ylim([-.5,1.5])
legend('Ts = 0.017 s')
legend boxoff
set(gca,'linewidth',1.2)


X_ZOH = conv(xk,h1,'same');
X_ZOHA = conv(xkA,h2,'same');

figure;
subplot(2,1,1)
hold on
plot(t,X_ZOH,'color','k')
plot(t,radio(t),'-.','linewidth',1.2)
title('Zero Order Hold (ZOH)')
xlim([-1,1])
ylim([-1.5,3])
xlabel('Time (s)')
set(gca,'linewidth',1.2)
legend('Ts = 0.017 s','Continuous Signal')
legend boxoff
hold off

subplot(2,1,2)
hold on
plot(t,X_ZOHA,'color','k')
plot(t,radio(t),'-.','linewidth',1.2)
xlim([-1,1])
ylim([-1.5,3])
xlabel('Time (s)')
set(gca,'linewidth',1.2)
legend('Ts = 0.17 s','Continuous Signal')
```

```
legend boxoff
hold off

%-------------------------------------------------------------
% First Order Hold (FOH)
h3 = FOH(t,Ts);
h4 = FOH(t,TsA);

figure;
plot(t,h3,'color','k','linewidth',1.2)
title('FOH Impulse Response')
xlabel('Time (s)')
xlim([-.03,.03])
ylim([-.5,1.5])
legend('Ts = 0.017 s')
legend boxoff
set(gca,'linewidth',1.2)

X_FOH = conv(xk,h3,'same');
X_FOHA = conv(xkA,h4,'same');

figure;
subplot(2,1,1)
hold on
plot(t,X_FOH,'color','k','linewidth',1.2)
plot(t,radio(t),'-.','linewidth',1.2)
title('First Order Hold (FOH)')
xlim([-1,1])
ylim([-1.5,3])
xlabel('Time (s)')
set(gca,'linewidth',1.2)
legend('Ts = 0.017 s','Continuous Signal')
legend boxoff
```

```
hold off

subplot(2,1,2)
hold on
plot(t,X_FOHA,'color','k','linewidth',1.2)
plot(t,radio(t),'-.','linewidth',1.2)
xlim([-1,1])
ylim([-1.5,3])
xlabel('Time (s)')
set(gca,'linewidth',1.2)
legend('Ts = 0.17 s','Continuous Signal')
legend boxoff
hold off

%-------------------------------------------------------------
% Predictive First Order Hold (PFOH)
h5 = PFOH(t,Ts);
h6 = PFOH(t,TsA);

figure;
plot(t,h5,'color','k','linewidth',1.2)
title('PFOH Impulse Response')
xlabel('Time (s)')
xlim([-.02,.08])
ylim([-1.2,2.5])
legend('Ts = 0.017 s')
legend boxoff
set(gca,'linewidth',1.2)

X_PFOH = conv(xk,h5,'same');
X_PFOHA = conv(xkA,h6,'same');

figure;
```

```
subplot(2,1,1)
hold on
plot(t,X_PFOH,'color','k','linewidth',1.2)
plot(t,radio(t),'-.','linewidth',1.2)
title('Predictive First Order Hold (PFOH)')
xlim([-1,1])
ylim([-1.5,3])
xlabel('Time (s)')
set(gca,'linewidth',1.2)
legend('Ts = 0.017 s','Continuous Signal')
legend boxoff
hold off

subplot(2,1,2)
hold on
plot(t,X_PFOHA,'color','k','linewidth',1.2)
plot(t,radio(t),'-.','linewidth',1.2)
xlim([-1,1])
ylim([-2,3])
xlabel('Time (s)')
set(gca,'linewidth',1.2)
legend('Ts = 0.17 s','Continuous Signal')
legend boxoff
hold off


%-------------------------------------------------------------
% Plot the data collected from the DUE
figure;
subplot(2,1,1)
plot(Ts_data(:,1)+1,Ts_data(:,2)-.5,'color','r','linewidth',1.2)
title('Data Saved from Scope with Ts = 0.17 s')
ylim([-.25,1.75])
```

```
xlim([0,2])
ylabel('Voltage (V)')
xlabel('Time (s)')
set(gca,'linewidth',1.2)

subplot(2,1,2)
plot(TsA_data(:,1)+.3,TsA_data(:,2)-.5,'color','r','linewidth',1.2)
title('Data Saved from Scope with Ts = 0.017 s')
ylim([-.25,1.75])
xlim([0,2])
ylabel('Voltage (V)')
xlabel('Time (s)')
set(gca,'linewidth',1.2)



%-----------------------------------------------------------------
% FCNs
%-----------------------------------------------------------------
% fcn to analyze
function xt = radio(t)
    xt = sin(2*pi*t) + 0.2*cos(12*pi*t);
end

%-----------------------------------------------------------------
% comb fcn
function D = comb(Ts,t)
    dt = t(2)-t(1);
    D = zeros(1,length(t));
    Ns = round(Ts/dt);

    for i = 1:Ns:length(t)
        D(i) = 1;
    end
```

```
end

%-------------------------------------------------------------
% sample fcn
function X = sample(Ts,t,fcn)
    dt = t(2)-t(1);
    X = zeros(1,length(t));
    Ns = round(Ts/dt);

    for i = 1:Ns:length(t)
        X(i) = fcn(t(i));
    end

end

%-------------------------------------------------------------
% ZOH
function h = ZOH(t,Ts)
    h = zeros(1,length(t));
    for i = 1:length(t)
        if  (t(i)>0 && t(i) <= Ts)
            h(i) = 1;
        end
    end
end

%-------------------------------------------------------------
% FOH
function h = FOH(t,Ts)
    h = zeros(1,length(t));

    for i = 1:length(t)
```

```
        if (t(i)>-Ts && t(i) <= 0)
            h(i) = 1/Ts*(t(i)+Ts);
        end

        if (t(i)> 0 && t(i) <= Ts)
            h(i) = -1/Ts*t(i)+1;
        end
    end
end

%-------------------------------------------------------------
% PFOH
function h = PFOH(t,Ts)
    h = zeros(1,length(t));

    for i = 1:length(t)
        if (t(i)>0 && t(i) <= Ts)
            h(i) = 1/Ts*t(i)+1;
        end

        if (t(i)> Ts && t(i) <= 2*Ts)
            h(i) = -1/Ts*(t(i)-Ts);
        end
    end
end
```