ELEC 421
Digital Signal and Image Processing

Siamak Najarian, Ph.D., P.Eng.,

Professor of Biomedical Engineering (retired),

Electrical and Computer Engineering Department,

University of British Columbia

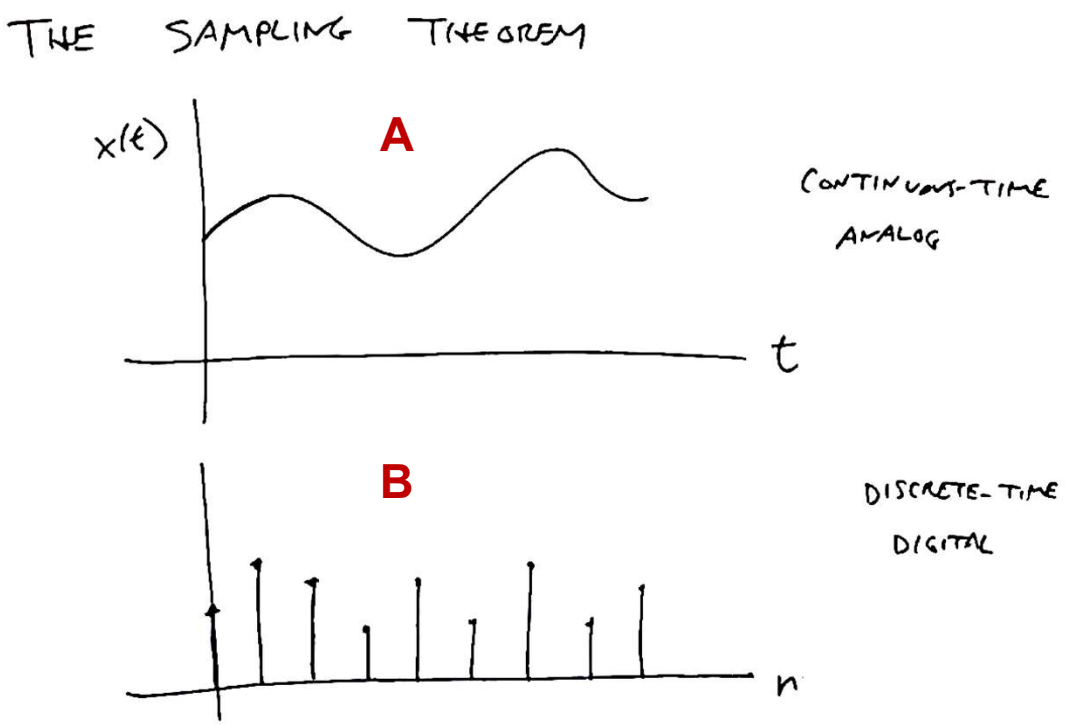
Course Roadmap for DSP

Lecture	Title
Lecture 0	Introduction to DSP and DIP
Lecture 1	Signals
Lecture 2	Linear Time-Invariant System
Lecture 3	Convolution and its Properties
Lecture 4	The Fourier Series
Lecture 5	The Fourier Transform
Lecture 6	Frequency Response
Lecture 7	Discrete-Time Fourier Transform
Lecture 8	Introduction to the z-Transform
Lecture 9	Inverse z-Transform; Poles and Zeros
Lecture 10	The Discrete Fourier Transform
Lecture 11	Radix-2 Fast Fourier Transforms
Lecture 12	The Cooley-Tukey and Good-Thomas FFTs
Lecture 13	The Sampling Theorem
Lecture 14	Continuous-Time Filtering with Digital Systems; Upsampling and Downsampling
Lecture 15	MATLAB Implementation of Filter Design

Lecture 13:
The Sampling Theorem

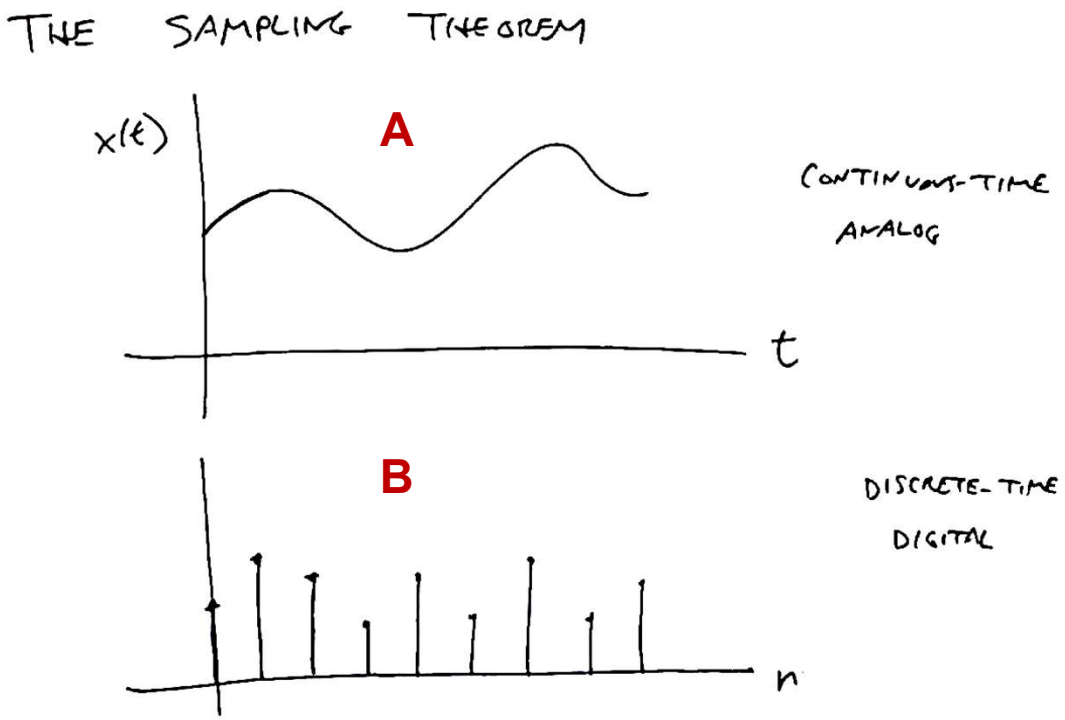
Table of Contents

The sampling theorem



- In this lecture, we want to talk about one of the most fundamental concepts in digital signal processing which is **sampling**. We will mainly talk about classical sampling theorem, just as a reminder for how that all works. In this setup, let us start by discussing the difference between continuous time signals and discrete time signals.
- A **continuous time signal**, **A**, which we might also think of as an **analog signal**, is a function of **t** and the assumption here is that we can choose any real number **t** we want and get back any signal value, **x(t)** that we want. However, in the **digital signal processing** (DSP) world, we have to deal with an ***approximation*** to this underlying continuous time signal. The way we do that is with a **discrete time** or **digital signal**, **B**, where we get only certain values of the signal and presumably at regular units. For example, MATLAB assumes that these units are spaced out by **1** unit on the **n**-axis. So, if we do not tell MATLAB anything else, MATLAB will get things done by one unit, that is, **x[1]**, **x[2]**, **x[3]**, and so on.

The sampling theorem



- The process of going from a continuous time signal to a discrete time signal is called **sampling**. The sampling process is definitely an **x**-axis issue i.e., a time-domain axis issue.
- There is also an issue on the **y**-axis called **quantization**. We will also talk about that briefly here. The idea is that we may not necessarily get infinite resolution in the heights of the signal values in **B**. It could be that we are also limited by some sort of a chopped up version of the **y**-axis and have to round these values to the nearest thing. That is the meaning of quantization.
- **Conclusion:** Both **sampling** and ***quantization*** are ways of thinking about digitizing either the ***domain*** or the ***range*** of a signal.

Periodic sampling of a continuous-time signal

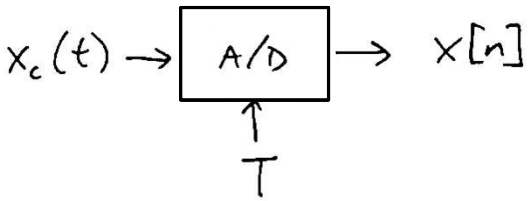
PERIODIC SAMPLING:

$$x[n] = x_c(nT) \quad (1) \qquad n \text{ AN INTEGER}$$

T = SAMPLING PERIOD

$\omega_s = \frac{2\pi}{T}$ = SAMPLING FREQUENCY (RADIANS) (2)

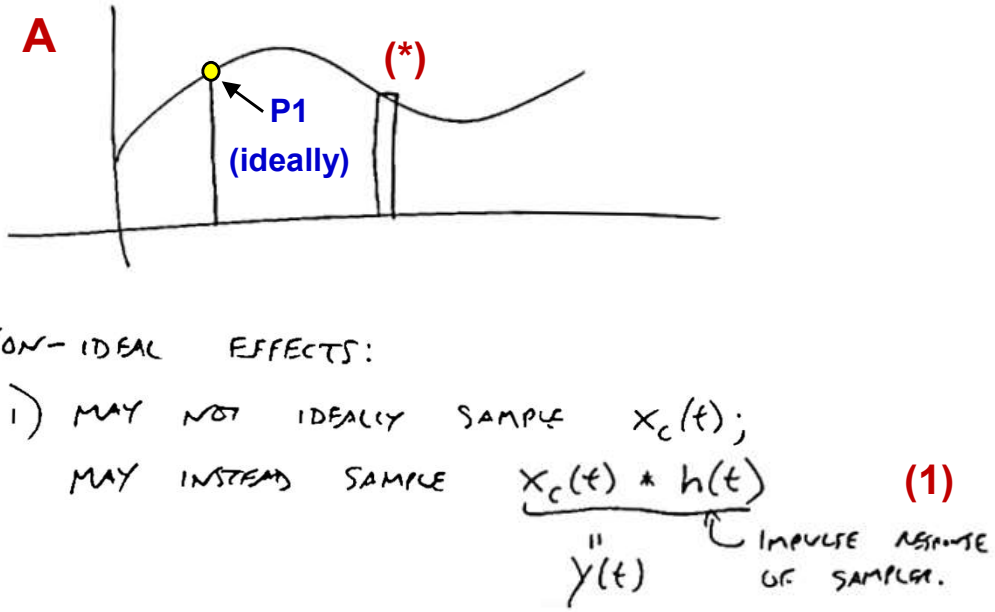
$\frac{1}{T}$ = SAMPLING FREQUENCY (HERTZ, Hz) (3)



(4)

- Most of the time, we think about what we call **periodic sampling**. Periodic sampling is simply saying that we obtain a discrete time signal by sampling a continuous time signal every **T** units, **(1)**. Here, **n** is an integer and **T** is called the **sampling period**. We also have **2π/T**, which we sometimes call **omega s** (ω_s) or the **sampling frequency**, **(2)**. This sampling frequency is measured in radians. If we wanted to talk about Hertz, we could just look at **1/T** and that is the sampling frequency in Hertz, **(3)**, which is just the reciprocal of **T**.
- Most of the time, in our lectures, we are going to talk about ***radians*** because that is where the mathematics is more natural.
- Sometimes the process of obtaining the discrete time signal from the continuous time signal is called **A to D conversion**, **(4)**. Going backwards is called **D to A conversion**. For example, we might take a continuous time signal, and feed it into an analog to digital converter (or sometimes called **the continuous to discrete converter**). As one of the inputs to this A/D converter is the sampling period, **T**. What comes out is the discrete time signal, **x[n]**.

Non-ideal effects

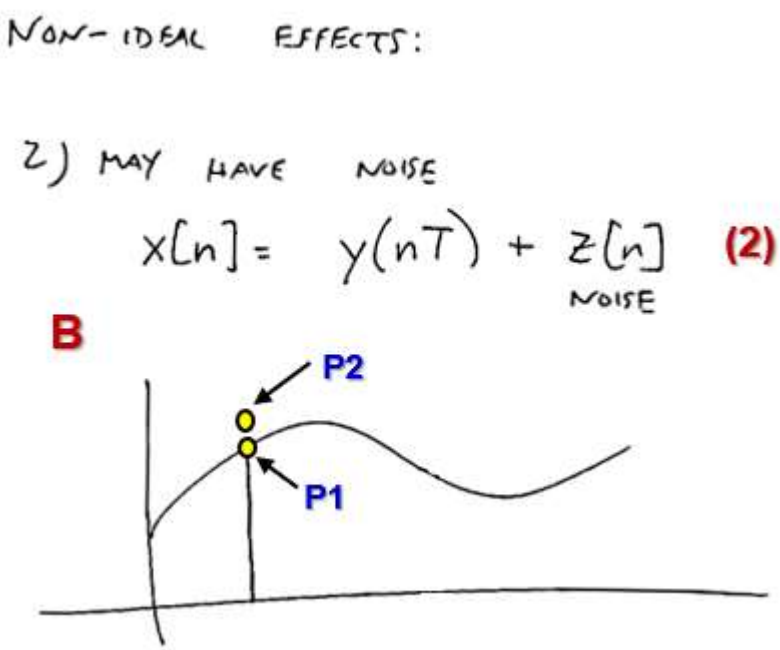


- The process of averaging the signal over a short time interval can be modeled as a convolution of the continuous-time signal with a rectangular pulse (the ***aperture function***). The aperture function can be approximated as a short duration pulse, similar to an impulse response.

- In the real world, there are often some **non-ideal effects** that we can still model with the machinery that we have learned so far.

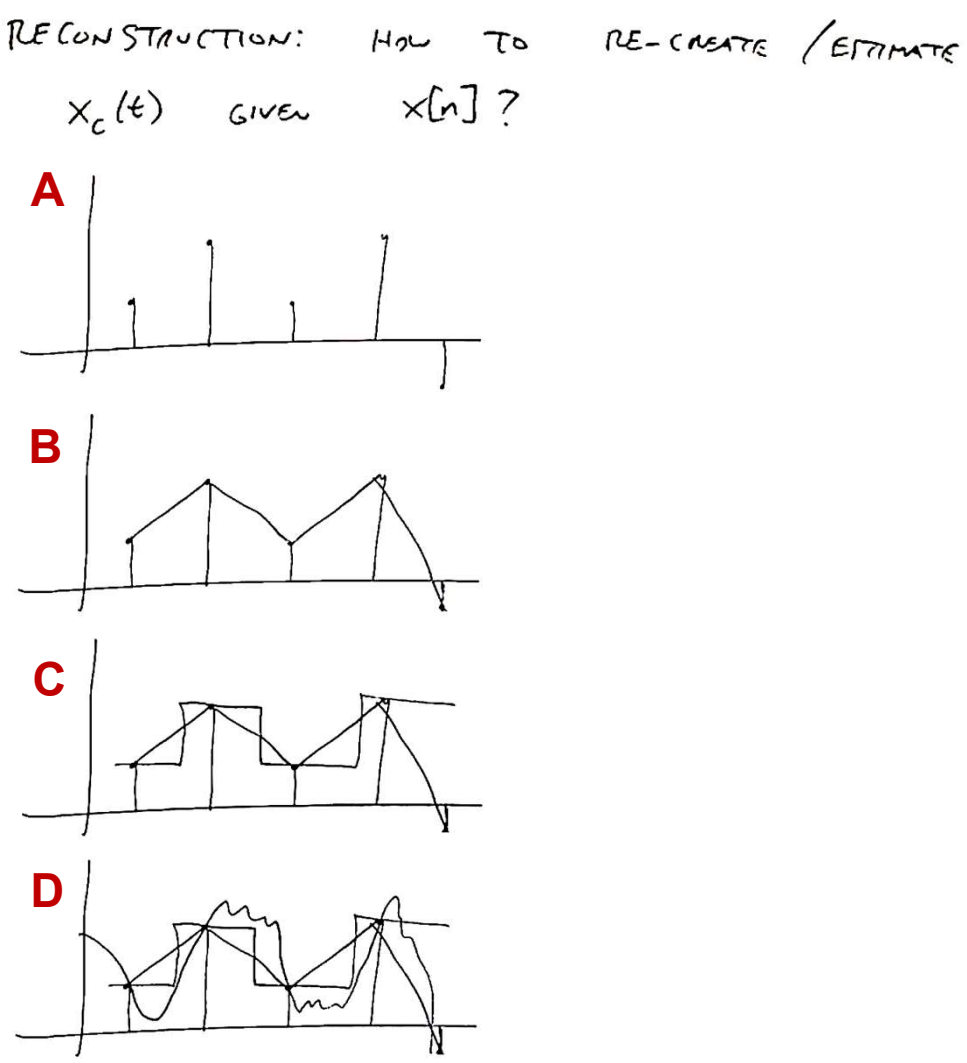
- **Non-ideal Effect #1:** For example, it is not necessarily always true that our **sampler** takes a continuous time signal and ***ideally*** just picks off one value, as shown in **A**. If **A** is the continuous time signal, ideally, what we want is getting exactly the value of point **P1**. What we may actually get instead is basically ***the average signal over a very very short time period***, shown by **(*)**. So, we mean that it may not be physically possible for the sampler to pick off one value, it may picks off the ***average of a fraction of a millisecond of values***, for example. In that case, we may not ideally sample the continuous time signal, $\mathbf{x_c(t)}$, we may instead sample some signal that is ***convolved*** with some kind of tiny ***impulse response***, **h(t)**, as shown in **(1)**. In practice, this means that we may instead get samples of this ***slightly blurred out signal***. But again, knowing that we know how to handle convolved signals, that is not really a big issue for us. So, going forward, we are just going to assume that we ***do*** get the ideal samples but we could also analyze these slightly blurry samples in this way.

Non-ideal effects



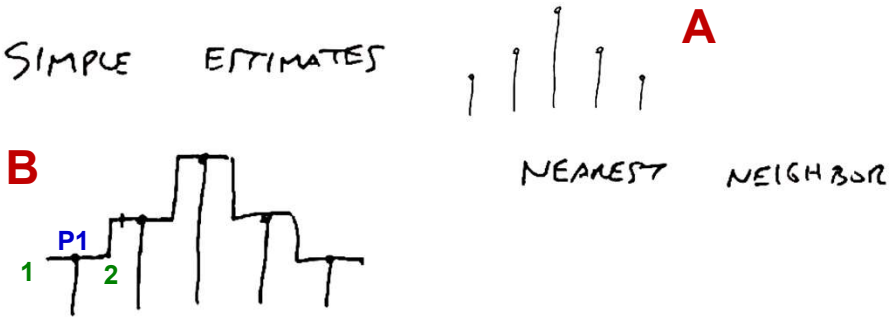
- **Non-ideal Effect #2:** The other non-ideal effect is that we may have **noise**, which means that instead of getting exactly the value of **P1**, we may get the value, **P2**, instead, as shown in **B**. So, just due to noise in the system, we might not get exactly what we asked for. What this means is that our discrete time signal, **x[n]**, as represented in **(2)**, may be some sort of a sampled version of a blurry signal, **y(nT)**, plus some sort of a noise term, **z[n]**, that we cannot control.
- So, life is not always so ideal, but in this case, we are going to assume everything is good. It is kind of straightforward about how we do the sampling. We just design a suitable sampler that can pick off the values. Here, the topic that is of interest to us, at this stage, is **how would we go back from the digital values to the continuous time signal?**

Ways of reconstructing a continuous signal from discrete samples



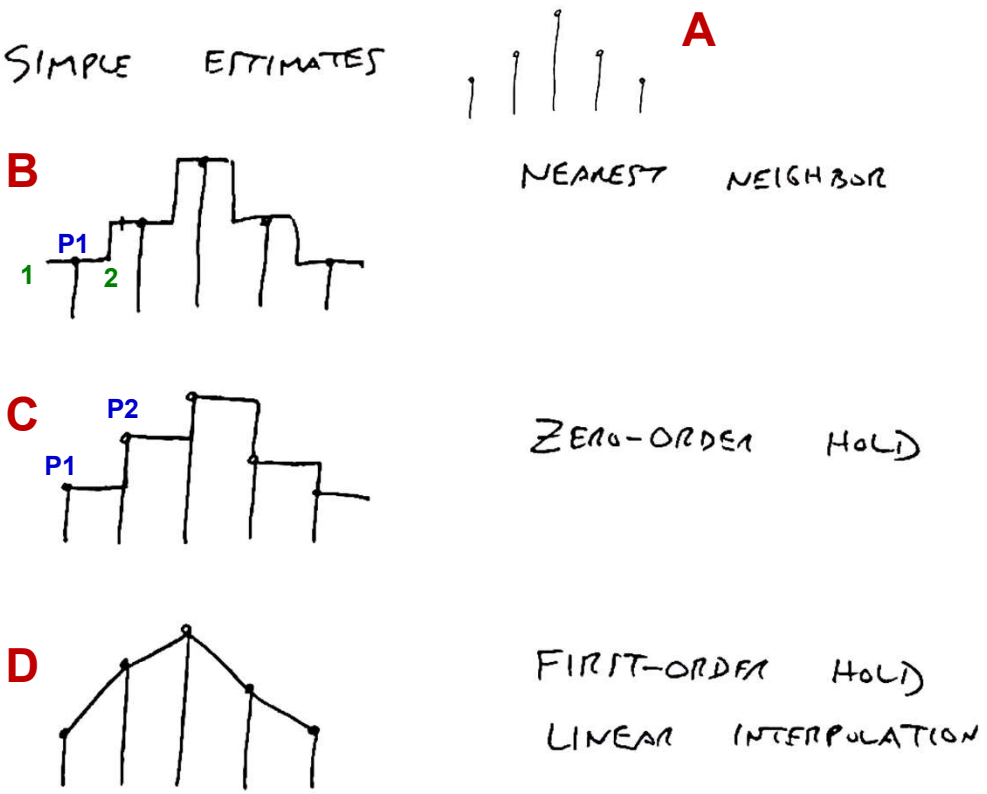
- The core hypothesis is that we can use DSP, implemented in MATLAB or hardware, as a substitute for the ideal processing of continuous-time signals. To do that, we need some confidence that the digital signals adequately represent the continuous time signal. This is a central topic called **signal reconstruction**. This basically talks about how to ***re-create*** or ***estimate*** the continuous time signal, $\mathbf{x_c(t)}$, given the discrete time samples, **x[n]**.
- That is clearly a case of **one-to-many** kind of problem. This means that if we are given some samples, shown in **A**, there are lots of ways that we could hypothesize a continuous time signal that went through those samples. This is like an **interpolation problem**. We could do something really simple like just connect these samples with **lines**, shown in **B**. Or, we could try and make these into **bars** like a bar graph, shown in **C**. Or, we could even make some sort of **oscillatory signal** that could do basically anything in between the samples, shown in **D**.
- **Conclusion:** Just from the samples alone, there is no way to disambiguate which of these cases we might be in. And, that is the fundamental challenge of the sampling theorem. Namely, **what is *the* signal that we want to reconstruct from a given set of samples?**

Nearest neighbor



- Certainly, there is this sense that if we had sampled the signal fast enough, then maybe connecting the dots or making the bar graph (also called **staircase graph**) is good enough. Here, we are going to draw a signal, **A**, in three different ways, and then use three simple estimates that also have names.
- **Nearest Neighbor:** One natural way is the nearest neighbor, **B**. This estimate says if we want to fill in the time gaps, we should just take the closest sample that we saw in the discrete world. This means that we would have something like a **bar graph**. This says, for example, all the samples closest to **P1**, i.e., in the range of **1** to **2**, belong to the horizonal line of **1-2** and then we have a jump, and so on. This is called **nearest neighbor interpolation**.

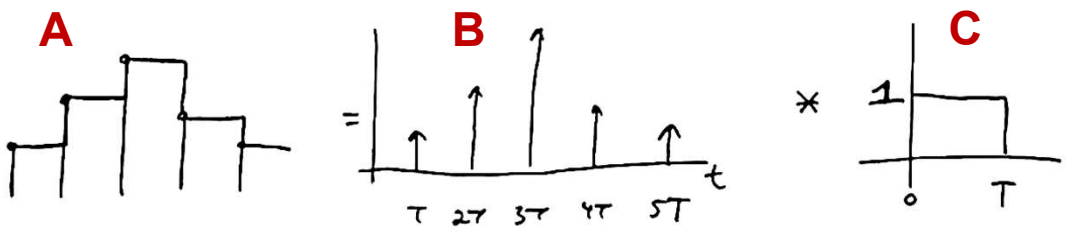
Zero-order hold; First-order hold (linear interpolation)



- **Zero-order Hold:** The natural modification of the nearest neighbor interpolation is to reconstruct the signal and just use the most recent value that we saw, **C**. We basically just hold on to point **P1** (i.e., we go horizontally) and when we see a new signal, **P2**, we hop up to this new value. This is like the nearest neighbor but offset by half a sample. It is called **zero-order hold**.
- **First-order Hold (Linear Interpolation):** At this stage, we can start to think about the fact that we could do better by just ***connecting the dots***, shown in **D**, instead of having these stair steppy functions in **C**. Connecting the dots gives us perhaps a slightly better approximation. This is called **first-order hold** (also called **linear interpolation**).
- In computer graphics, even higher order interpolations are used, for example, a cubic spline. These splines are used to connect up the discrete points and turn them into a smoother looking continuous function. In that case, it is really like a cubic polynomial that goes through the points. In order to make that cubic spline, we are not just using the nearest neighbors, we are also using ***multiple neighbors*** that are further away from us. This way we can make smoother and smoother interpolations if we are willing to consider more and more of the neighbors.

Each reconstruction algorithm corresponds to filtering a set of impulses with a specific filter

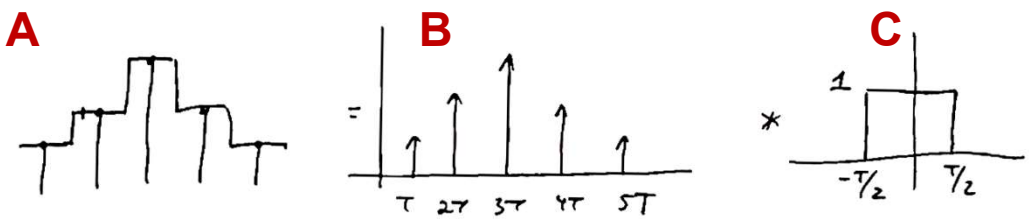
Zero-order Hold:



- One thing that is interesting to think about and that plays into the way that we talk about the sampling theorem is that in each of these reconstructions, we are taking the original samples and ***convolving*** them with some special kind of baseline, like a **kernel**.
- Let us start by looking at **zero-order hold**, **A**. What we can do here is to draw **A** as **delta functions**, instead of dots. This is shown in **B**. This is the critical thing that lets us convert between digital and continuous signals. The idea is that in MATLAB, we have the continuous time set of stick figures (the stem plots) and to think about analyzing that in a more principal way, we can look into the relationship between those samples and delta functions in continuous time. These delta functions just happened to fire at those sample locations and have the same heights. Let us take **B** and convolve it. So, if we think about **B**, as a function of **t**, we can convolve it with **C**. Here, **C** is a little pulse, that is **T**-units wide. By convolving **B** with **C**, we are going to produce exactly the signal in **A**. So, in some sense, this zero-order hold is a reconstruction that we can obtain by taking the original samples and convolving them with this special kind of baseline simple signal.

Each reconstruction algorithm corresponds to filtering a set of impulses with a specific filter

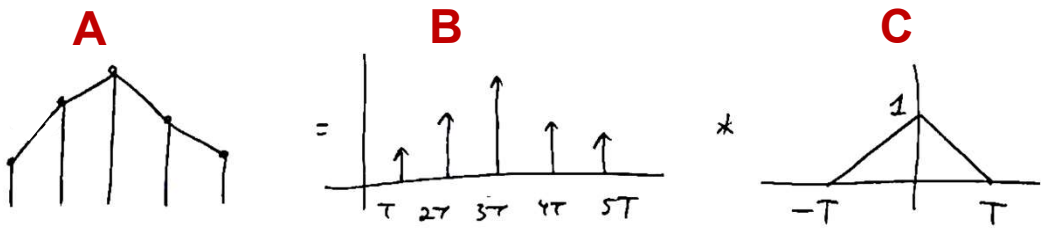
Nearest Neighbor:



- In a similar way, we would get **A**, if I had taken **B** (the same delta functions) and instead convolved it with **C**. One way of thinking about this is that it is the impulse response of a filter that does the reconstruction.

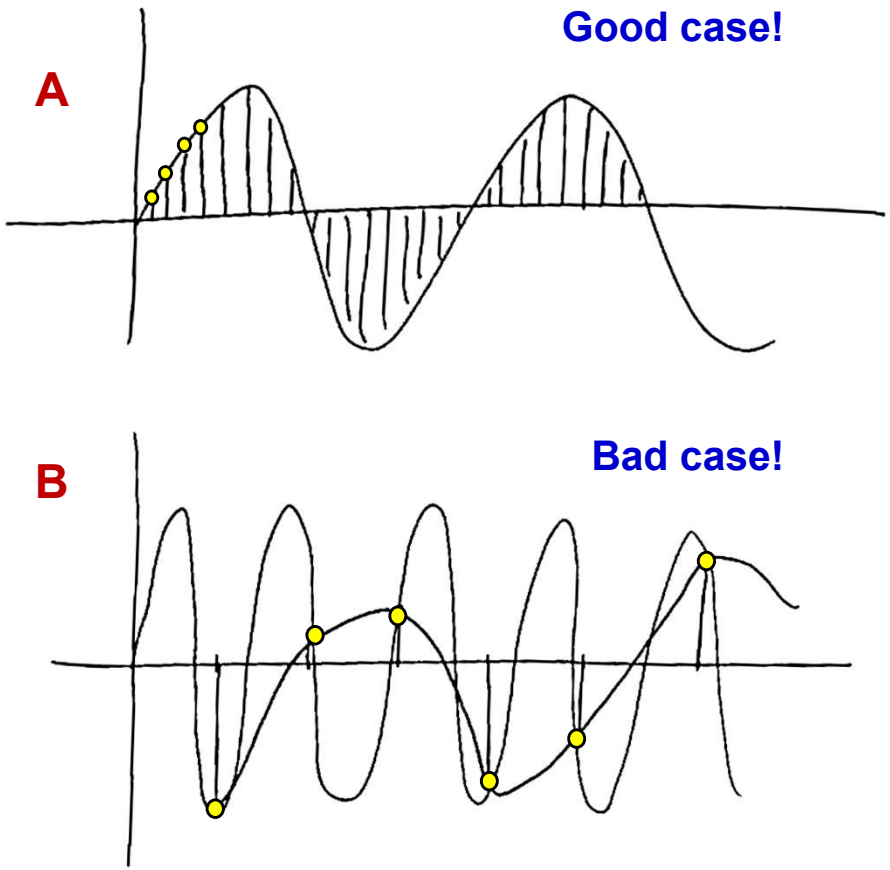
Each reconstruction algorithm corresponds to filtering a set of impulses with a specific filter

First-order Hold:



- For first-order hold, it turns out that the corresponding signal for this kind of impulse response of the sampler looks like **C**.
- **Conclusion:** The idea is that all we are doing here is choosing the correct kernel to convolve the time domain impulses with in order to get the reconstructed signal.

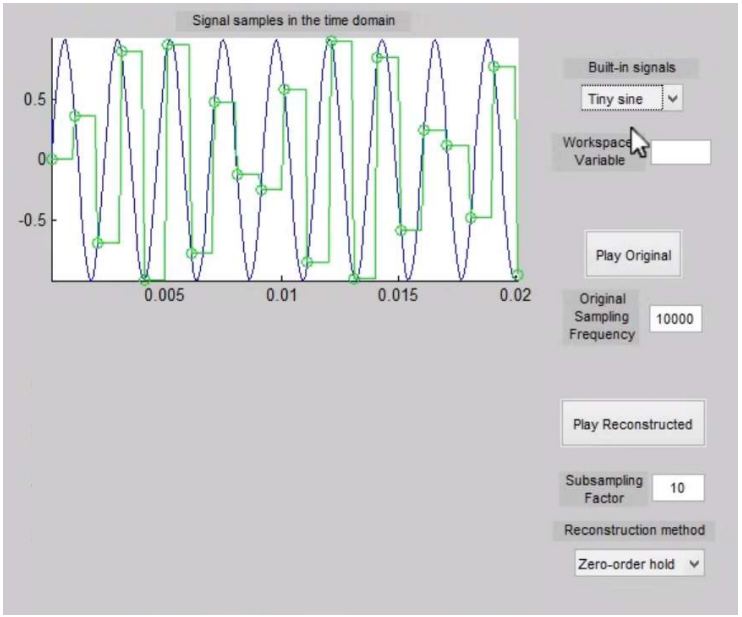
What can go wrong with interpolating samples?



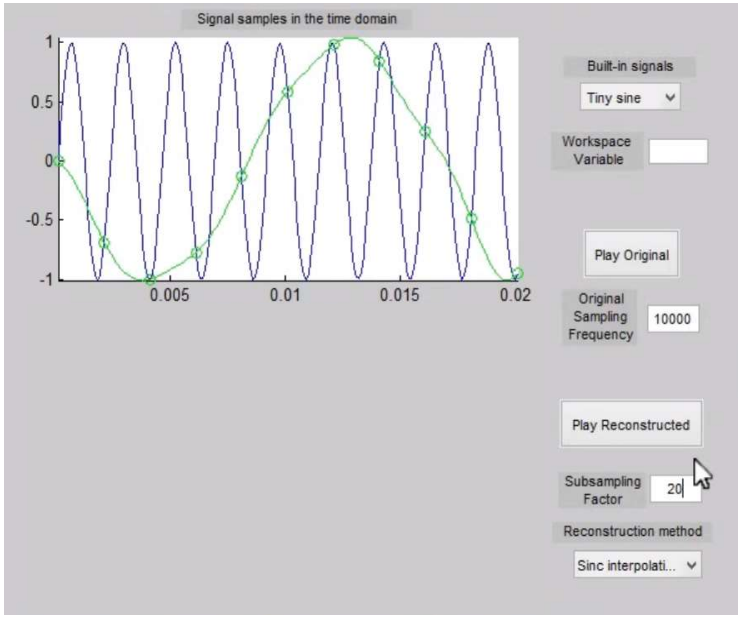
- **Goal:** Now, we want to talk about ***perfect reconstruction***. As our main goal, we would love to be able to take our original samples and ***get back exactly*** the original time domain signal.
- Let us suppose that we have a sinusoid, shown in **A**. Now, we sample it super finely, shown by yellow dots at the end of vertical lines in **A**. This looks pretty good. In fact, it seems like a pretty credible argument that if these samples were close enough together then connecting the dots should get us pretty close to the original signal. If they were super close together, then maybe we would not get exactly the same signal but we would get something that was pretty close. Because of this, we are going to call this the **good case**. So, life is good here since if the samples were close enough together, there would not be any real ambiguity about what the underlying signal was.
- However, the problem is that there is also a **bad case**. Let us suppose that we have a ***really high frequency sinusoid*** and we sample it every once in a while. Here, let us sample really far apart, as shown in **B**. Now, if we were to naively connect the dots, this connecting of the dots is giving us some sort of a weird signal that appears to have ***lower frequency*** than the underlying sinusoid. Here, we would not be able to capture all the ***oscillations*** that is going on between the dots.

MATLAB example of sampling and reconstruction of a sine wave

A

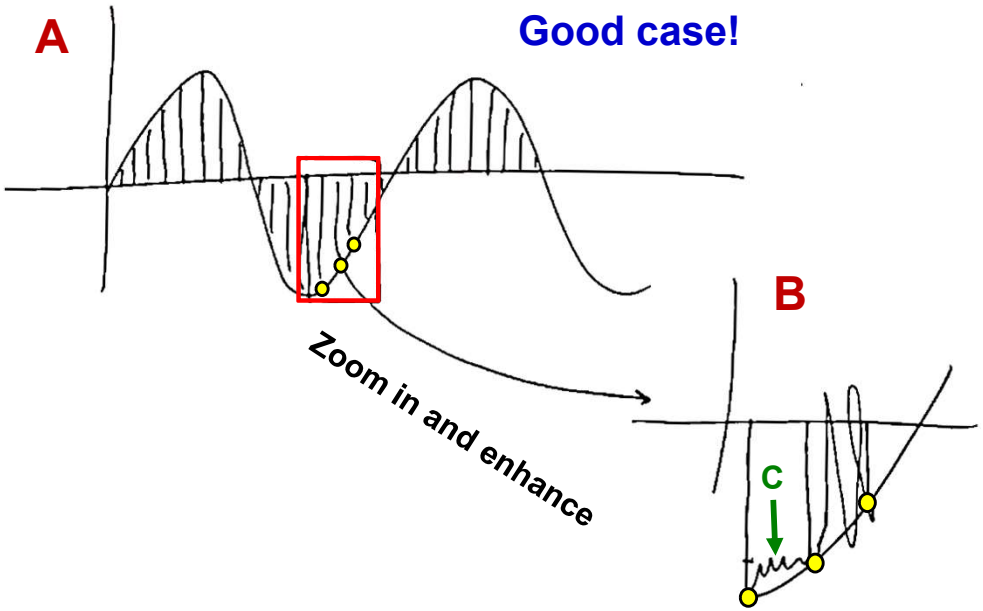


B



- **Example in MATLAB:** Here is a tiny sinusoid in MATLAB. We are going to look at different possibilities for what the reconstruction looks like. In both **A** and **B**, the **blue** curve is the original signal and the **green** curve is the reconstructed signal under different circumstances, i.e., using different reconstruction methods.
- In **A**, the green is simply the **zero-order hold,** meaning that we hold the sample that we were on. We can see that we get a crummy reconstruction.
- In **B**, we are sampling ***too slow***. Here, we are using a different reconstruction method (**sinc interpolation**, not discussed yet). The green samples are hitting the blue sinusoid at the correct places, but when we connect the dots, what we get is this apparently ***low frequency sinusoid***. And, that is exactly the problem. The problem is that without knowing anything about the underlying signal, there is no way for us to tell ***a priori*** whether we are in the good case, where it appears the samples are really close together, or the bad case, where the samples are really far apart. Put it differently, the problem is we do not know anything about what is happening between the samples.

MATLAB example of sampling and reconstruction of a sine wave



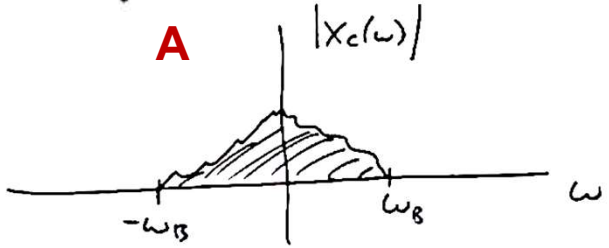
- As mentioned earlier, the problem is that we do not know anything about what is happening between the samples. To show this, let us go back to our previous sketch about the good case, **A**.
- Suppose that we got our samples and they look like graph **A**. Let us zoom in and enhance the picture in the **red** box. This is shown in **B**. It could be that in between the region in the box (i.e., between the yellow dots) our samples are comparatively far apart. And, who knows what this signal could be doing between the samples. That is, between two consecutive yellow dots in **B**, the signal **C** could be oscillating wildly in between the samples. The problem is that we do not know the shape of the signal without putting more ***restrictions*** on the problem. That is, **are we in the good case or are we in the bad case?** So, the only way out of this problem is to put some restrictions on the input signal. That is, we need to say, e.g., we are not going to tolerate any crazy wiggling between our samples. That means we have to say that our ***input signal is restricted*** in some way. And, that is what is called the **bandlimited assumption**, which we are going to talk about shortly.

Bandlimited signals

A SIGNAL IS BANDLIMITED IF THERE'S SOME FREQUENCY ω_B S.T.

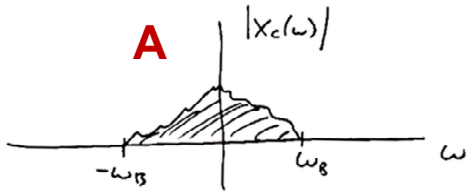
$$|X_c(\omega)| = 0 \quad \text{FOR} \quad |\omega| > \omega_B$$

A



- Here is the assumption that makes everything come together and that is the following. We are going to say a signal is **bandlimited** if there is some frequency **omega B** ($\boldsymbol{\omega_B}$) such that the continuous time Fourier transform, $|\mathbf{X_c(\omega)}| = \mathbf{0}$, for omega outside the interval of $|\boldsymbol{\omega}| > \boldsymbol{\omega_B}$. This means that our continuous time Fourier transform, $|\mathbf{X_c(\omega)}|$, looks something like **A**. So, in general, a bandlimited signal is a signal that has a finite bandwidth.
- So, between $\mathbf{-\omega_B}$ and $\mathbf{+\omega_B}$, we have some content, shown by the shaded area and outside of that range, we have nothing. Again, in all the stuff we are going to draw next, the signals are sketched as ***approximately symmetric***. This is because we assume that we have an approximately real signal and what we are drawing here, shown in **A**, is the magnitude of the continuous time Fourier transform. We know that there are these symmetries in the Fourier transform. That means these are actually complex numbers.

Statement of the sampling theorem; The Nyquist rate



SAMPLING THEOREM — SHANNON, NYQUIST

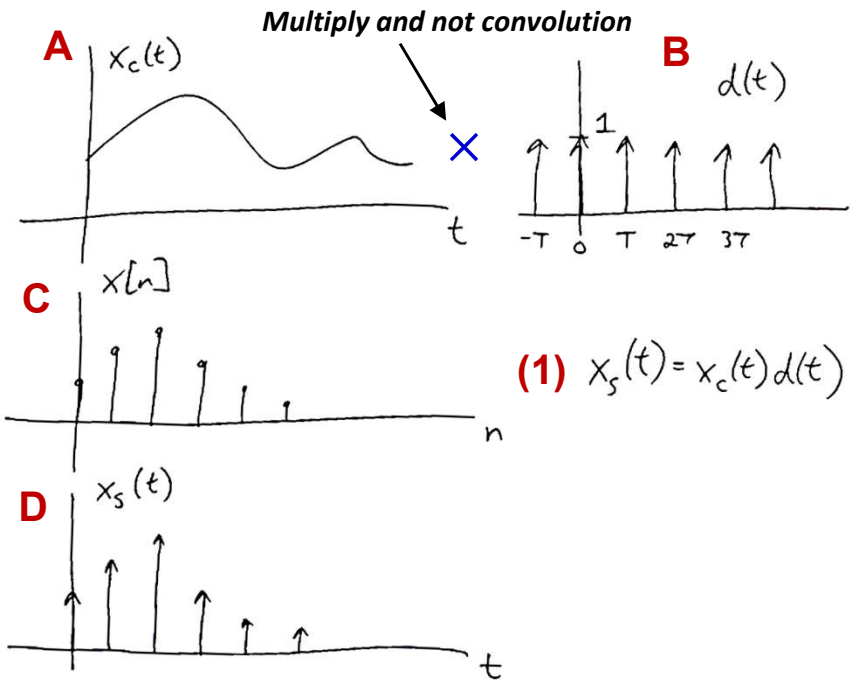
A BANDLIMITED SIGNAL WITH MAX FREQUENCY ω_B CAN BE PERFECTLY RECONSTRUCTED FROM EVENLY-SPACED SAMPLES IF THE SAMPLING FREQUENCY ω_S SATISFIES

$$\omega_S > 2\omega_B$$

$2\omega_B$ IS CALLED THE NYQUIST RATE.

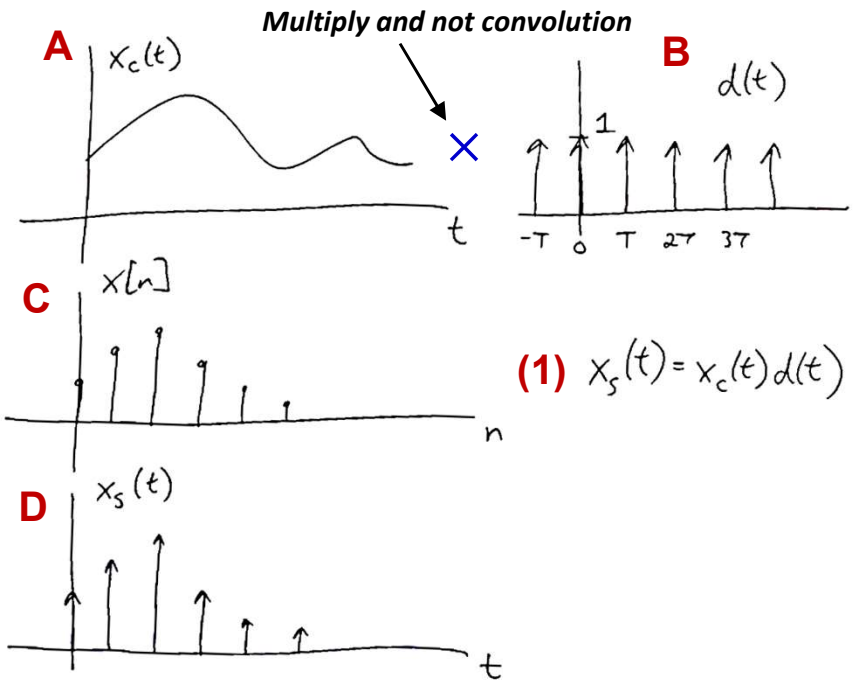
- Let us talk about what happens when we sample a signal that looks like **A**. This is called the **sampling theorem**. This was attributed widely to Claud Shannon (the father of information theory) and also to Nyquist who talked about this idea even earlier than Shannon.
- The theorem says that a bandlimited signal with **maximum frequency omega B** (ω_B) can be ***perfectly*** reconstructed from evenly-spaced samples if the **sampling frequency omega S** (ω_S) satisfies the condition of $\omega_S > 2\omega_B$. Here, $2\omega_B$ is called the **Nyquist Rate (NR)**. This is another way of saying that a signal can be reconstructed from samples if we sample above the Nyquist Rate. The NR is related to the highest frequency present in the underlying signal.
- Let us, number one, prove why this is true, and number two, show explicitly how that reconstruction could be obtained.
- An important note to make is that ***we have to sample faster than the Nyquist Rate***. We cannot sample ***at*** the NR. This is sometimes good enough, but sometimes will not do it for us. We will look at an example of why sometimes sampling at the NR could be bad.
- **Conclusion:** The key idea is we have to always sample greater than Nyquist Rate.

Impulse-train version of sampling



- Let us say we have a continuous time signal, $\mathbf{x_c(t)}$, shown in **A**. Here, $\mathbf{x_c(t)}$ is our ***underlying continuous time signal***. Our discrete time signal, **x[n]**, is shown in **C**, which is basically sampling $\mathbf{x_c(t)}$, every **T** units. Our proxy for the discrete time signal of **x[n]** is going to be a signal, $\mathbf{x_s(t)}$, where we multiply the continuous time signal by a set of delta functions. Let us ***multiply*** $\mathbf{x_c(t)}$ by **d(t)** shown in **B**, i.e., multiply $\mathbf{x_c(t)}$ by a set of impulse functions, **d(t)** in the time domain. This is shown by **(1)**. Here, we mean ***multiply*** and ***not convolution***. Note that this set of impulse functions are separated out by **T** and they have height **1**. The effect of multiplying **A** by **B** is simply to pick off the values that we want. That is, we want to get graph **D**, which is our $\mathbf{x_s(t)}$.
- So, here, what we are actually doing is making a **proxy continuous time signal** or $\mathbf{x_s(t)}$ (shown in **D**) that clearly is basically the same as our discrete time signal, **x[n]**, just with the delta function arrows replacing our stem plot dots of **x[n]**.
- **Goal:** What we are trying to do is to figure out how we could get from signal $\mathbf{x_s(t)}$ back to signal $\mathbf{x_c(t)}$. Ultimately, we are going to convolve this time domain signal of $\mathbf{x_s(t)}$ with some ***special reconstruction filter***, which is going to get us back to $\mathbf{x_c(t)}$.

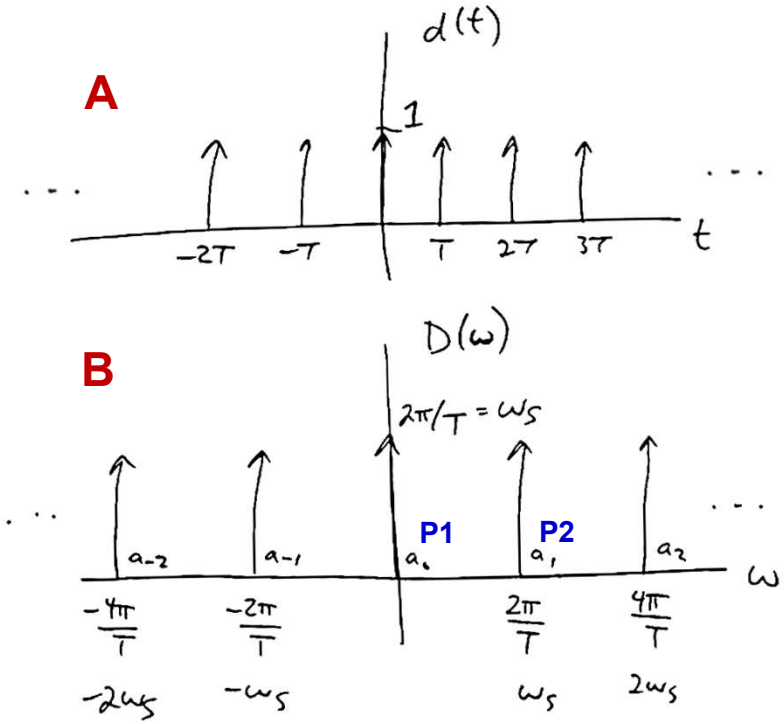
Impulse-train version of sampling



(1) $x_s(t) = x_c(t)\,d(t)$

- We know that our sampled signal $\mathbf{x_s(t)}$ is like our continuous signal $\mathbf{x_c(t)}$ times our impulse train **d(t)**. This is shown in equation **(1)**.
- In order to think about these things in the frequency domain, we need to know what the **FT of d(t)** is. This is because here in **(1)**, what we are doing is multiplying in the time domain and that means we are convolving the original signal in the frequency domain with whatever the **FT of d(t)** is.

The FT of an impulse train is also an impulse train



- **Goal: What is the frequency domain version of d(t)?** As a reminder, let us look at **A**, which is **d(t)** in the time domain, and sketch its frequency domain version, **B**. Remember that everything in **A** is in continuous time. So, the graph is an infinitely long extending ***impulse train*** in time. In **B**, the omega axis goes off infinitely far, too. We showed earlier that the **FT of d(t)** is actually ***another impulse train*** in the frequency domain, **B**, and it corresponds to impulses that look like vertical arrows in a graph of **D(ω)** vs. **ω**. The height of the impulses is **2π/T**, which is actually the same as the sampling frequency, or $\omega_s = 2\pi/T$. The separation between the impulses in **B** is every **2π/T** units or we can think about these as basically multiples of the sampling frequency, ω_s, $2\omega_s$, $3\omega_s$, etc.
- Remember that the way we get to graph **B** is by thinking about **A** like a periodic continuous time signal. We showed that the Fourier series of **d(t)** is a constant. This is because we know that delta functions correspond to constants. Also, we showed that when we have a periodic signal, and we take the regular FT of it, what we get, e.g., at **P1**, is the a_0 coefficient multiplied by a constant. At **P2**, we have a_1, etc. These a_k's are the Fourier series coefficients multiplied by something, which turns out to be **2π/T**. These coefficients occur at multiples of the sampling frequency.

The FT of the (continuous time) sampled signal

$$x_s(t) = x_c(t)\, d(t) \quad (1)$$

$$X_s(\omega) = \frac{1}{2\pi} X_c(\omega) * D(\omega) \quad (2)$$

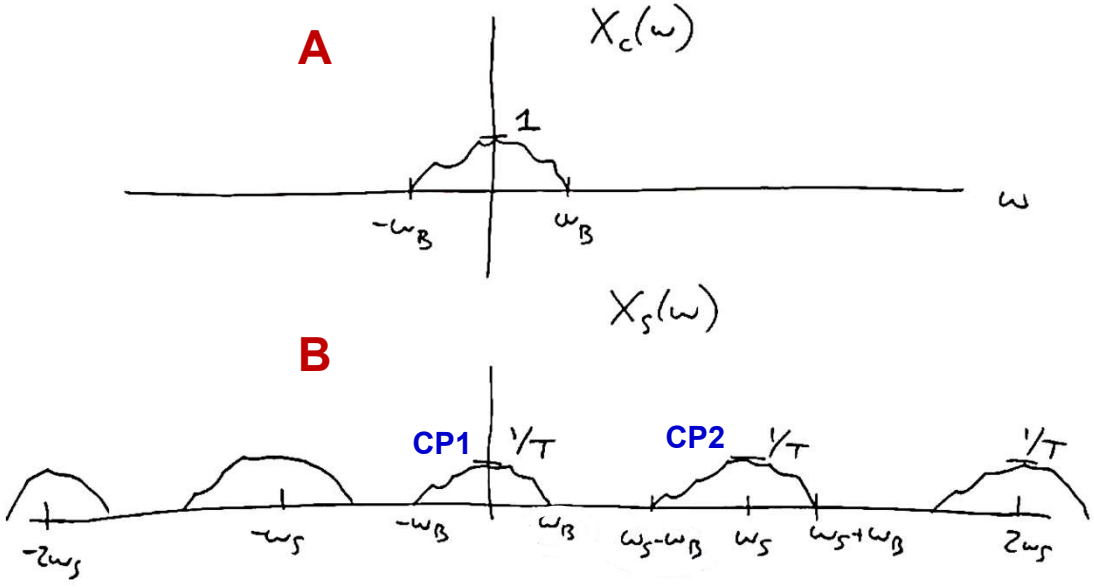
$$= \frac{1}{2\pi} X_c(\omega) * \left[\frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta(\omega - k\omega_s) \right] \quad (3)$$

$$X_s(\omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c(\omega - k\omega_s) \quad (4)$$

- In **(1)**, we have our sampled continuous time signal, $\mathbf{x_s(t)}$, which is equal to our underlying continuous time signal, $\mathbf{x_c(t)}$, multiplied by the impulse train, **d(t)**.
- In the frequency domain, we have **(2)**. Here, $\mathbf{X_s(\omega)}$ corresponds to a convolution. Note that there is a $\mathbf{2\pi}$ in the denominator of **(2)**. By writing the equation for **D(ω)**, we see that the **k** in the delta function and in the sum goes from minus infinity to positive infinity, and that these delta functions are spaced-out every multiple of the sampling frequency, i.e., at every $\mathbf{k\omega_s}$.
- **What happens when we convolve a signal with a delta function?** This convolution occurs in **(3)** and we know that it just shifts the signal. This is like saying we are going to get a bunch of copies of the original signal at all the places where the delta function fires, shown in **(4)**. Put it differently, this is like saying we are going to get a whole bunch of periodic copies of the original signal.

Sampling a bandlimited signal: copies in the frequency domain

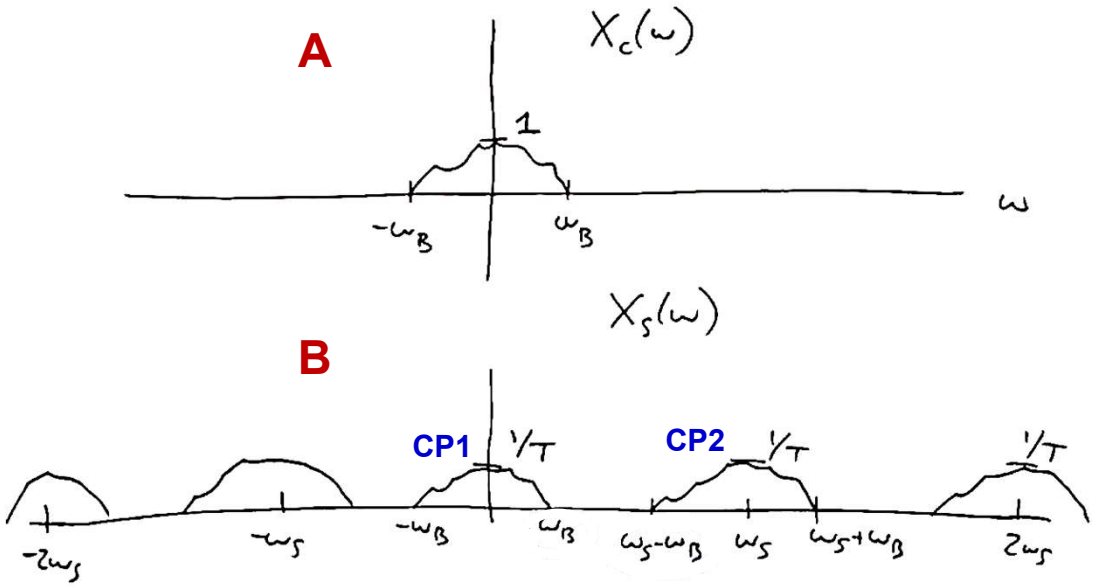
$$X_s(\omega) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c(\omega - k\omega_s) \quad (1)$$



- **Goal (Proof of Nyquist Theorem):** Let us first sketch out $\mathbf{X_S(\omega)}$, (1), and interpret the meaning of it. To do that, suppose we have our original signal $\mathbf{X_c(\omega)}$, shown in **A**. As we are trying to prove the NR theorem, we are going to assume that this signal, $\mathbf{X_c(\omega)}$, is bandlimited between $\mathbf{-\omega_B}$ and $\mathbf{+\omega_B}$. Graph **A** shows the original spectrum of the continuous time signal.
- **What does equation (1) tell us?** This equation predicts that we are going to get a whole bunch of copies of $\mathbf{X_c(\omega-k\omega_S)}$ and that they are going to be **1/T** as high as they used to be, and finally, they are going to occur at multiples of the sampling frequency, i.e., at $\mathbf{k\omega_S}$.
- In graph **A**, since $\mathbf{X_c(\omega)}$ has a height **1**, the sampled spectrum, $\mathbf{X_S(\omega)}$, is going to look like **B**, with a height of **1/T**. In **B**, let us suppose that we sampled at $\mathbf{\omega_s}$, $\mathbf{2\omega_s}$, $\mathbf{3\omega_s}$, etc. Our first copy, **CP1**, is centered at **0**, because that is definitely an example of a multiple of the sampling frequency, i.e., $\mathbf{k\omega_S = (0)(\omega_S) = 0}$. Our second copy, **CP2**, is centered at $\mathbf{\omega_s}$, and so on.

Sampling a bandlimited signal: copies in the frequency domain

$$X_s(\omega) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c(\omega - k\omega_s) \quad (1)$$



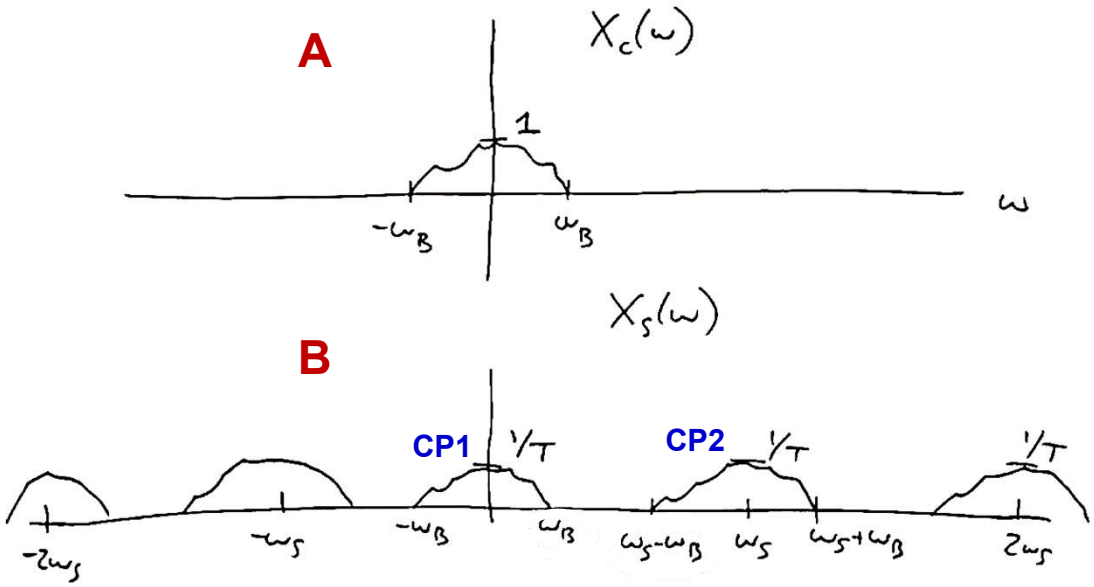
GOOD CASE: $\omega_s - \omega_B > \omega_B$

$\omega_s > 2\omega_B$

- All the copies in graph **B** are going to have height **1/T**. The first copy has a band between $-\omega_B$ and $+\omega_B$. The second copy has a band between $\omega_s - \omega_B$ and $\omega_s + \omega_B$.
- Now, graph **B** is really all we need to help us prove the sampling theorem. Since graph **B** is the spectrum of the sampled signal, it is pretty obvious that all we need to do is to notch out the middle copy (**CP1**). That is, all we need is a **filter** that crops out the middle copy and multiplies it by **T** in the frequency domain. By doings this, we get back exactly the signal in **A**. So, all we need to do is lowpass filter the sampled signal, $X_s(\omega)$, to get back the original signal, $X_c(\omega)$.
- **Conclusion:** We can see in this picture why the Nyquist theorem is what it is. Because what we need is to make sure that **CP1** ***does not overlap*** with **CP2**. So, the good case is when $(\omega_s - \omega_B) > \omega_B$ or when the difference between the two omegas is ***strictly greater*** than ω_B. Alternatively, it means that $\omega_s > 2\omega_B$. Here, the final outcome is that copies do not overlap and that is exactly the statement of the sampling.

Sampling a bandlimited signal: copies in the frequency domain

$$X_s(\omega) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c(\omega - k\omega_s) \quad (1)$$

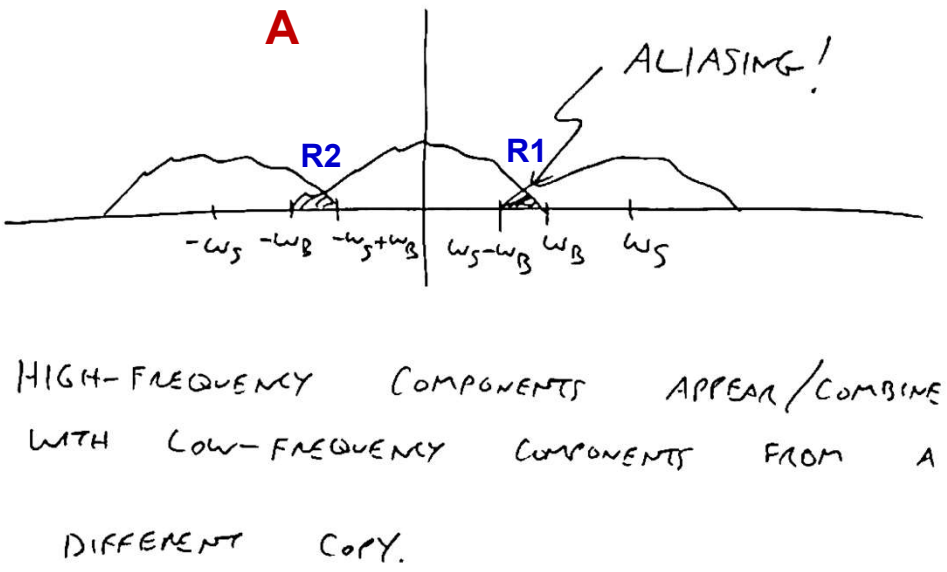


GOOD CASE: $\omega_s - \omega_B > \omega_B$

$\omega_s > 2\omega_B$

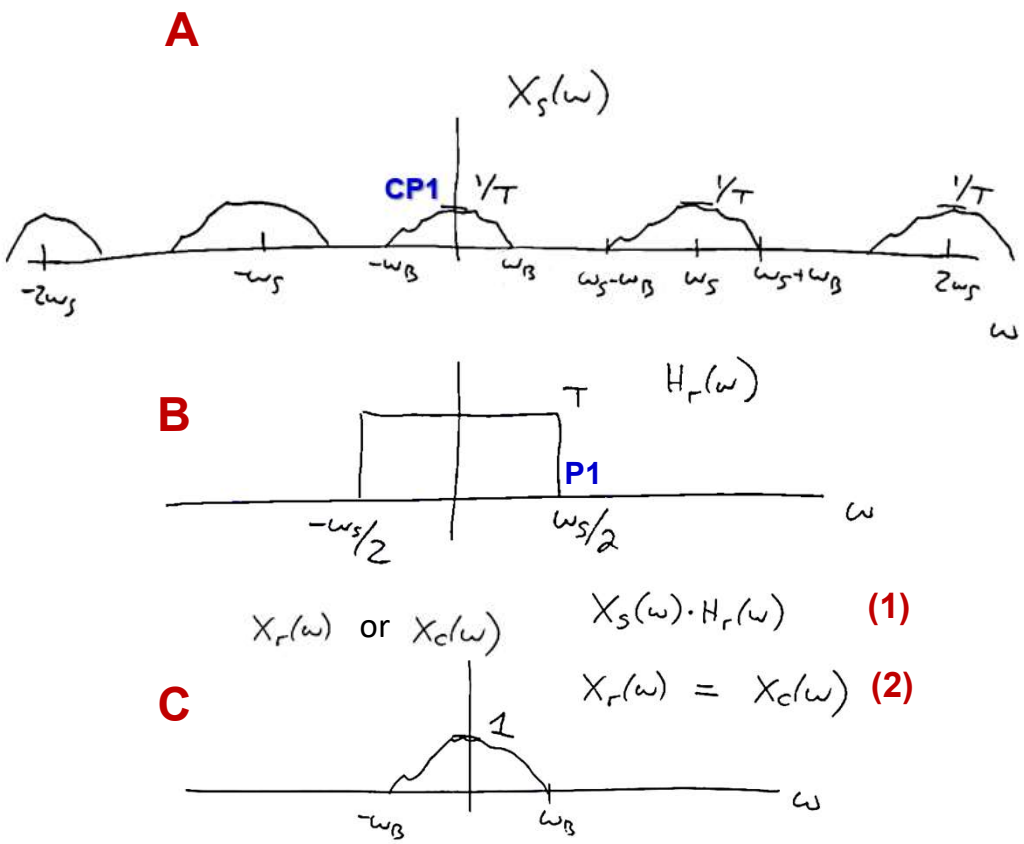
- In graph **B**, the Nyquist theorem is telling us that the ***edge*** $\boldsymbol{\omega_s - \omega_B}$ and the ***edge*** $\boldsymbol{\omega_B}$ should stay away from each other.
- The example we used here is definitely the good case, and conversely, the bad case is when we have copies that are overlapping. That is, when we sample too slowly.
- It is maybe not very clear from the picture in **B** why it would be a problem ***if the copies of CP1 and CP2 were right on top of each other***. We are going to present an example of that shortly.

Aliasing: overlapping copies in the frequency domain



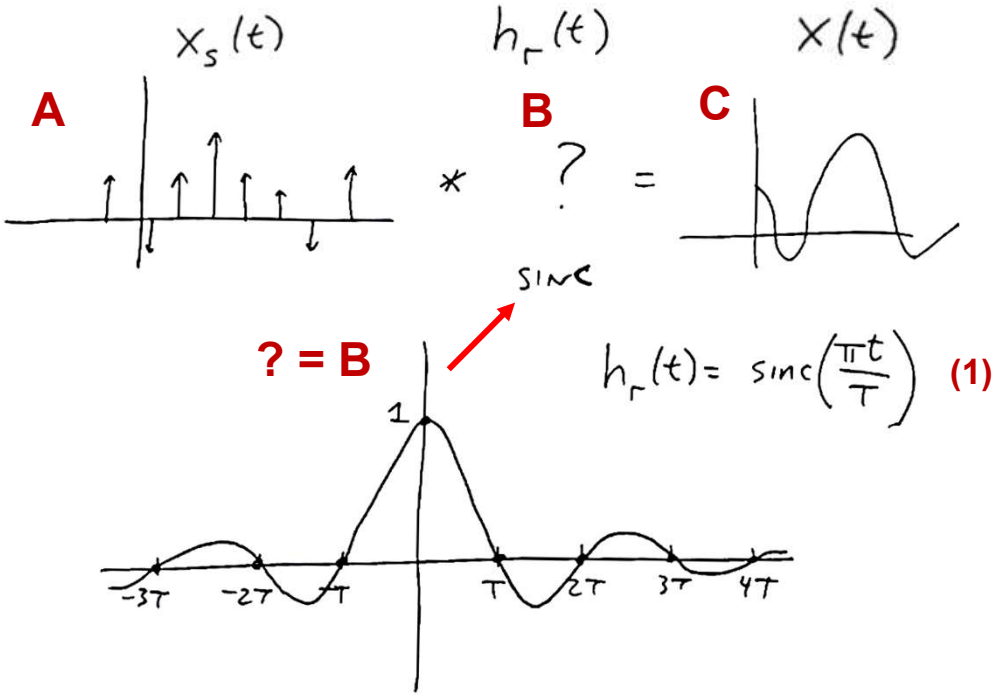
- Now, let us take a look at the bad case. In graph **A**, ω_B is the **bandlimit**, and ω_S is the **sampling frequency**. When we look at this combination of signals in **region 1** or **region 2** (**R1** or **R2**), unfortunately, these signals have overlapped in a way that we cannot undo. So, this is bad! This is called **aliasing**. That means that if we sample too slowly, we kind of garble up high frequencies to the original signal and they get mixed up with lower frequencies of the lower copy in the frequency domain. And, once we have added this stuff together, shown in the shaded area of **R1**, there is really nothing we can do to recover it. So, that is not good!
- **Conclusion:** The idea is that high frequency components appear as low frequency or maybe appear that they have combined with low frequency components from a different copy. So, we do not want to be in this bad case. That means we have to be careful to sample fast enough.

The ideal reconstruction filter in the frequency domain: a pulse



- Assuming that we ***do*** sample fast enough, we are in the good case, shown in **A**. In this case, all we need to do in order to reconstruct things is to take the signal, $\mathbf{X_S(\omega)}$, and multiply it by some nice **reconstruction filter**, such as the one shown in **B**, $\mathbf{H_r(\omega)}$. Here, we would say that we are going to notch **CP1** out, and hence, we are going to multiply $\mathbf{H_r(\omega)}$ by $\mathbf{X_S(\omega)}$ in the frequency domain.
- We can choose this reconstruction filter to be as wide as we want (wider than $\mathbf{-\omega_B}$ to $\mathbf{+\omega_B}$ in graph **A**) as long as we do not hit any of the edges in graph **A** signal. A safe choice, if we have done it right, should be half the sampling frequency. In general, the point of **P1** in **B** would be like some cut off, but here we are going to choose this particular case of $\mathbf{+\omega_S/2}$. That means we are going to just notch out **CP1**.
- When we multiply in the frequency domain, **(1)**, $\mathbf{X_S(\omega).H_r(\omega)}$, what we should get is exactly our original signal shown in **C**. That is, $\mathbf{X_r(\omega) = X_c(\omega)}$. The notation $\mathbf{X_r(\omega)}$ means the reconstructed signal. Again, in **C**, we are going to get back the right height in the frequency domain, **1**. This is exactly equal to the original time domain signal. So life is good!

The ideal reconstruction filter in the time domain: a sinc

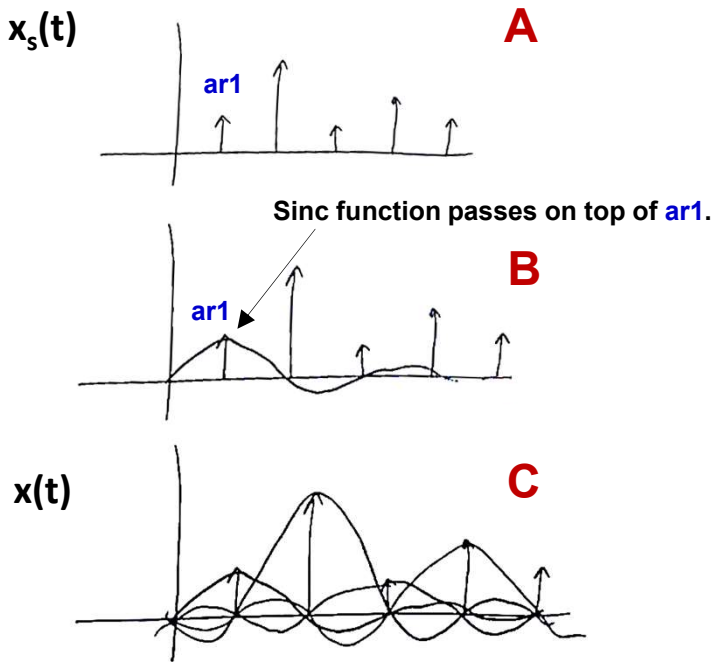


- Note that what we are doing in $\mathbf{X_s(\omega).H_r(\omega)}$ is that we are multiplying $\mathbf{X_s(\omega)}$ by a pulse in the frequency domain (**lowpass filtering**), $\mathbf{H_r(\omega)}$. We know that ***a pulse in one domain corresponds to a sinc in the other domain.***
- Let us go back and think about ***what*** is being convolved with the original signal, $\mathbf{x_s(t)}$. Here, that is like asking **if we have our original signal, $x_s(t)$, A, what are we convolving it with, B, to get back our continuous time signal, x(t), shown in C?** This question mark (**?**) is going to be some sort of a **sinc function**, **B**, and it turns out that the sinc has a special structure where it hits **zero** (i.e., zero crossings) on the **t**-axis at every one of the places that is multiples of **T**, and it is **1** at **t = 0**, as shown in **B**.
- In the time domain, $\mathbf{h_r(t)}$ looks like the sinc presented by **(1)**. Here, at multiples of **T**, in the numerator, we have a sine of some multiple of **π**, which is always **0**, and only at **t = 0** we might have a problem. We know that the sinc function at **0** is equal to **1**.

The ideal reconstruction filter in the time domain: a sinc

$$x(t) = x_s(t) * h_r(t) \quad (1)$$

$$x(t) = \sum_{n=-\infty}^{\infty} x[n]\ \mathrm{sinc}\left(\frac{\pi}{T}(t-nT)\right) \quad (2)$$

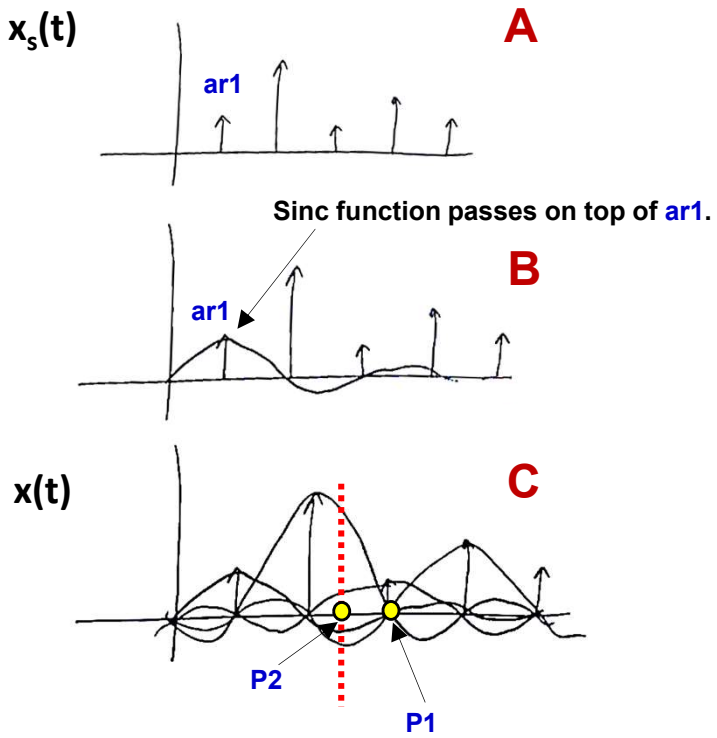


- Putting it all together, what we could say is that if we did not want to go into the frequency domain and just stay in the time domain, we could represent the reconstruction process as a **convolution**.
- We could say **$x(t) = x_s(t) * h_r(t)$**, **(1)**, that is, **x** of **t** should be equal to this kind of sampled version of **t**, **$x_s(t)$**, convolved with the reconstruction filter, **$h_r(t)$**, **(2)**. This is a different way of saying we are going to get a sum from **$n = -\infty$** to **$+\infty$**. In the sum, **x[n]**'s are our samples and the sum is basically a ***convolution sum*** of **x[n]** and a sinc function, **(2)**. All we are doing in **(2)** is using the sinc function and creating the output, **x(t)**.
- Let us start with signal **A**, **$x_s(t)$**. It means the output, **x(t)**, is actually the sum of a bunch of sinc functions that happen ***to pass right on tops of the vertical arrows***. One sinc function that passes on top of one of the arrows, **ar1**, is shown in **B**. By the same token, we can draw other sinc functions, each passing through the top of only one specific arrow. The final graph, for a limited number of sinc functions, is shown in **C**.

Sketch of how sinc functions add up between samples

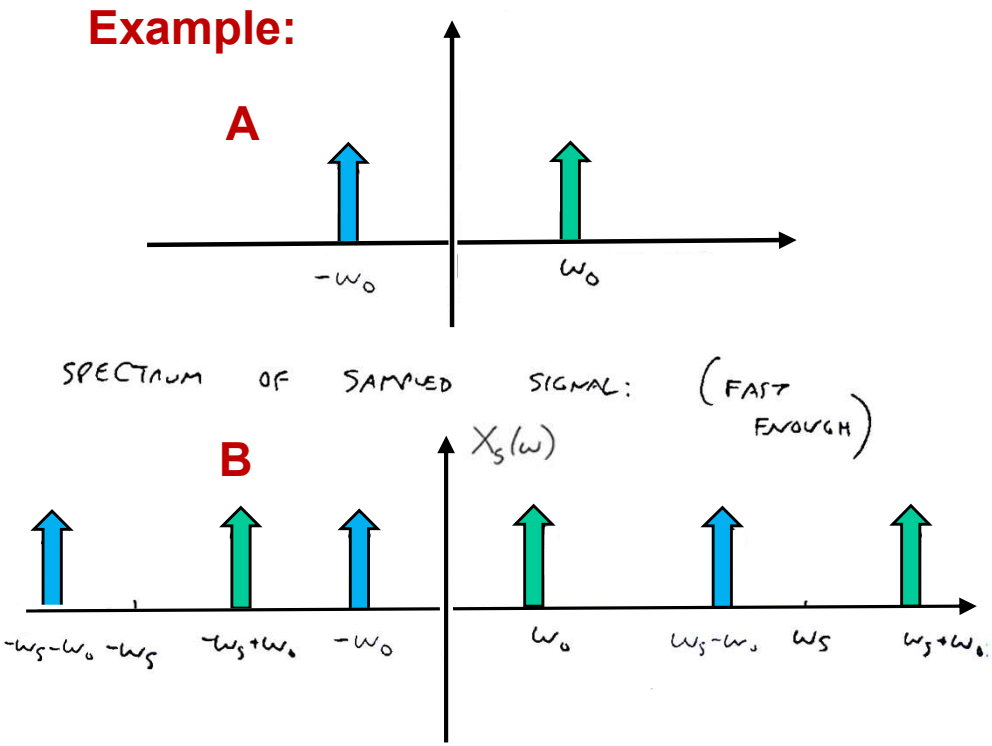
$$x(t) = x_s(t) * h_r(t) \quad (1) \longrightarrow$$

$$x(t) = \sum_{n=-\infty}^{\infty} x[n] \, \text{sinc}\left(\frac{\pi}{T}(t - nT)\right) \quad (2)$$



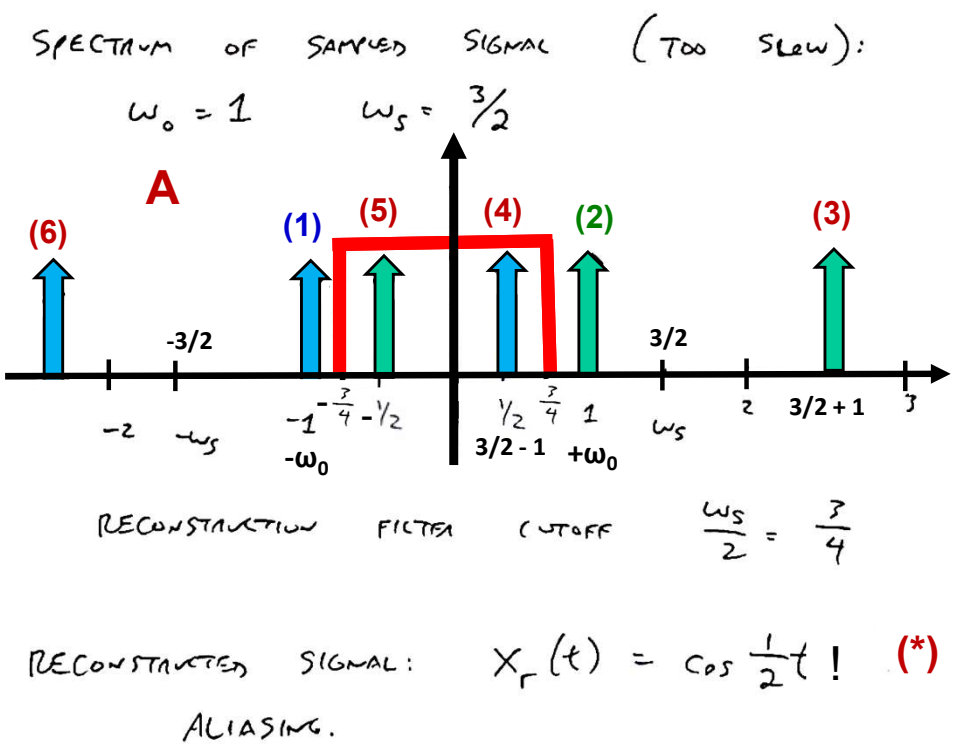
- In **C**, what is happening is that in order to reconstruct the output, what we are doing is centering this special sinc function over every sample and then we are adding those things up. We can see that at each sample (at each arrow and at a point like **P1**), the only sinc function that is non-zero is the one that is sitting on top of the sample. So, we are adding a bunch of zeros plus the original sample.
- In between, such as at the point **P2** shown by the **red** vertical dashed line in **C**, to get the value of **x(t)**, we are adding a whole bunch of sinc functions that magically add up to the right value, which is quite neat! However, the downside is that we can see that, in theory, to get any of these intermediate values, such as at **P2**, we have to include potentially an infinite number of sinc functions, represented by the intersection of each sinc function with the **red** vertical dashed line. Here, we could argue that after we have added up, say **50** sinc functions, the sinc functions are going to be pretty small. So, in practice, maybe we can cut off those at some point.
- **Conclusion:** All the samples that we are going to use for digital signal processing can be related back to the original continuous time samples, under the assumption that we have sampled fast enough. That is, we need to make sure that we never run into problems by sampling too slowly.

Example: sampling a cosine



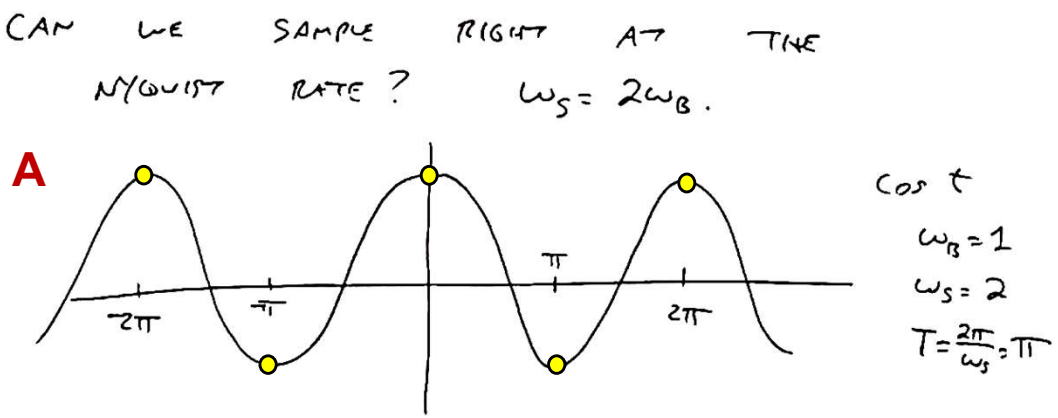
- Let us just do an example. Suppose that we have a cosine, $\mathbf{x(t) = \cos(\omega_0 t)}$. Its Fourier transform is shown in **A**. The sampling theorem tells us that if we need to reconstruct the signal, we have to sample faster than twice this frequency, $\boldsymbol{\omega_0}$. As we see here, there is only really one frequency present in the signal. So, that is pretty straightforward.
- The **spectrum** (i.e., the **Fourier transform**) of our sampled signal looks like **B**. In **B**, we are assuming that we are sampling fast enough.

Example: sampling a cosine



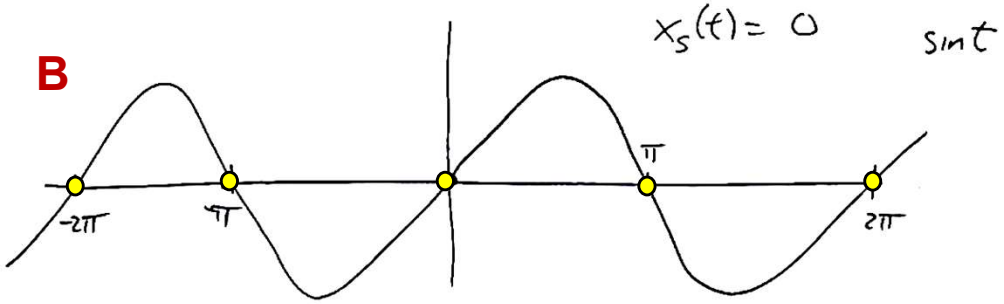
- Now, let us think about the bad case. Suppose that we had a bad situation, as shown in **A**. We look at the spectrum of the sampled signal, we see it is too slow. For example, let us say that $\omega_0 = 1$ and that we choose the sampling frequency, $\omega_S = 3/2$. We know by Nyquist that we should be sampling at least **2** times ω_0 or $\omega_S = (2)(1) = 2$ to avoid any problems. So, **3/2** is too slow!
- Our original signals are at **(1)** and **(2)**, which are at $\omega_0 = \pm 1$. Because of our sampling frequency of **3/2**, we are going to have copies of **(2)**, at **(3)** and **(4)**, i.e., at $\omega_S \pm \omega_0$. Also, we will have copies of **(1)**, at **(5)** and **(6)**.
- Here, we see exactly the **aliasing problem**. Because a copy of a high frequency **(3)**, gets aliased down to **(4)**. In other words, what used to be a cosine at a high frequency, **(3)**, now looks like **cosine** of **1/2t** at **(4)**, **cos(1/2t)**. That is bad!
- If we chose our reconstruction filter cut off to be or $\omega_c = \omega_S/2 = 3/4$, then we will have this filter as the **red** rectangle in graph **A**. Now, if we were to just go ahead and reconstruct our reconstruction signal, $x_r(t)$, this signal would be **cos(1/2t)** and not **cos(t)**, equation **(*)**. This is an example of aliasing.
- In practice, the problem is that the D/A converter is just assuming that everything is going according to plan. It does not know what signals are coming into it. So, if it is just using its standard reconstruction filter, when we feed something in and it was sampling too slow, it will give us back something that is not our original signal. And, here, there is nothing that we can do about it and there is no way we can disambiguate whether **cos(1/2t)** is coming in or **cos(t)** is coming in.

Why cannot we sample exactly at the Nyquist rate?



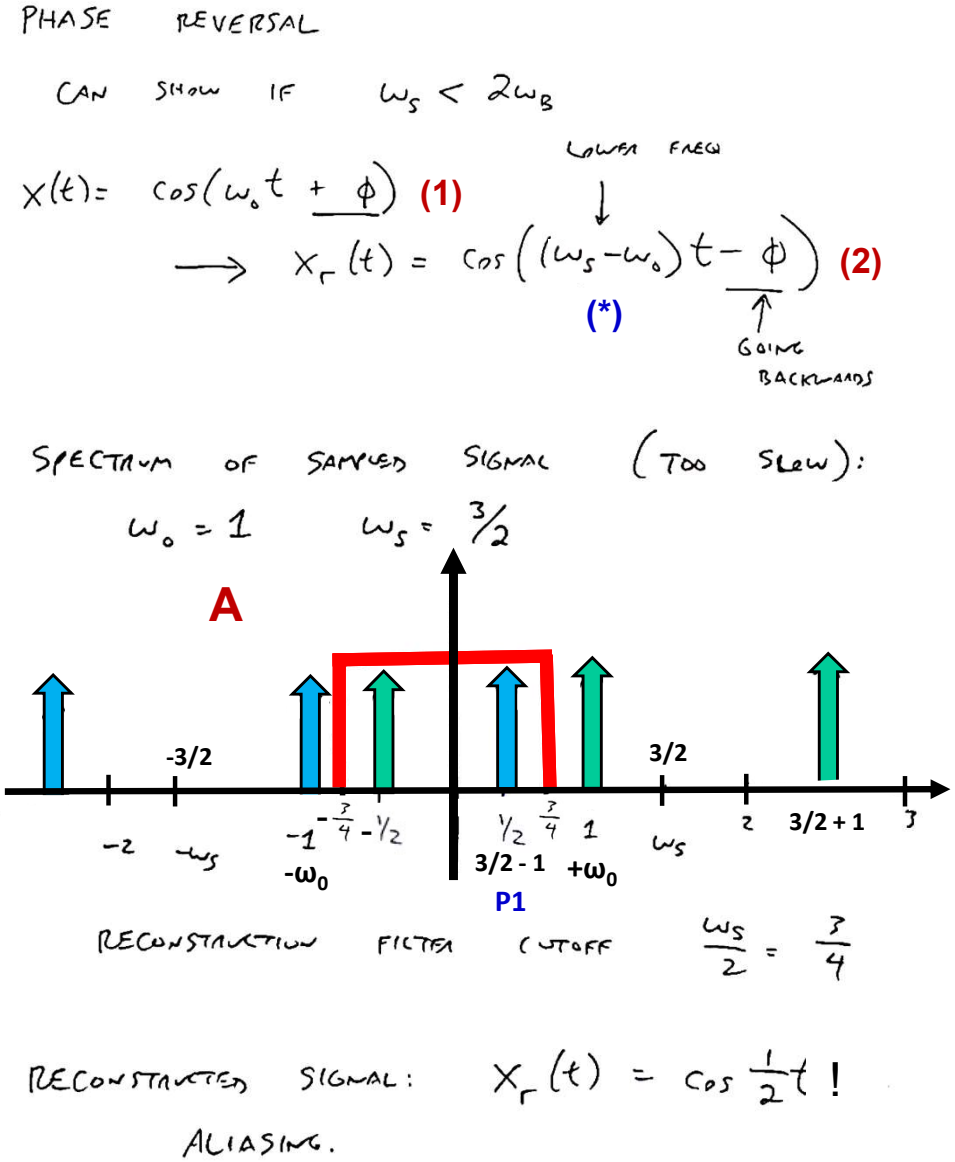
- **What could go wrong if we sample at the Nyquist rate?** In other words, we want to know why we cannot have the sampling frequency exactly equal to the Nyquist rate. When we sample right at the NR, that is like saying that our ω_S is exactly equal to twice the highest frequency.
- As an example, let us think about a cosine again. Suppose that we have $\cos(\omega_0 t) = \cos(\omega_B t) = \cos(1.t)$. That means our omega bandlimit, ω_B, is **1**. So, let us say that our sampling frequency is $\omega_S = 2\omega_B$. This means that $\omega_S = 2$, and the sampling period, $T = 2\pi/\omega_S = (2\pi)/\omega_S = \pi$. By referring to **A**, this does not look so bad.

Why cannot we sample exactly at the Nyquist rate?



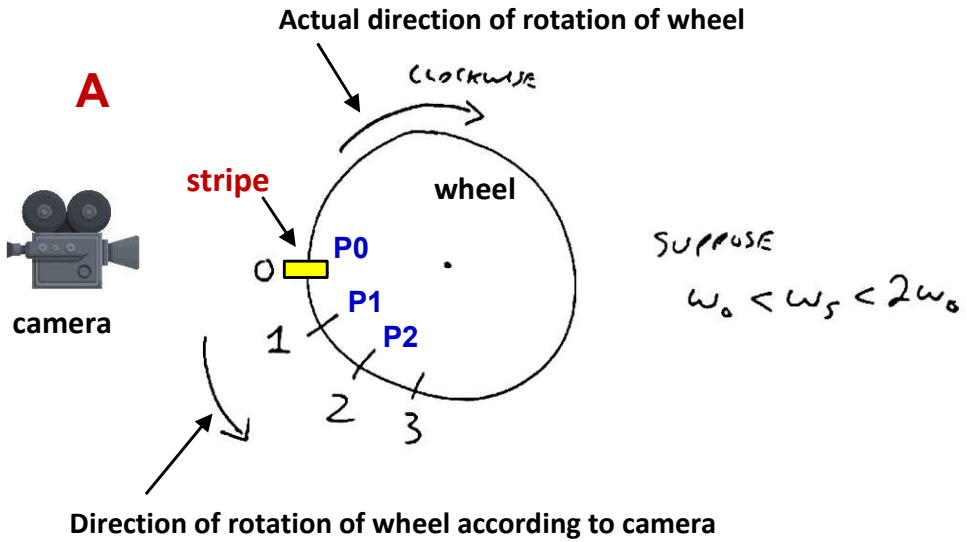
- Let us suppose instead of **cos(t)**, we have **sin(t)**, as shown in **B**. Again, we are going to sample every **π** units. Here, if we sample this signal at exactly the NR, we would get samples every **π** units. So, our sampled signal would be identically **0**, or $\mathbf{x_s(t) = 0}$. That is like saying that every time we fire the delta function, it just happens to hit the zero of the sine function.
- So, if we were to reconstruct that, there is no way to successfully reconstruct the original signal. Put differently, there is no way to disambiguate the reconstructive signal from just being a constant zero, i.e., we get the same constant value of **0**.
- This admittedly is like a worst case scenario. But the problem is when we look at an input signal, we do not really know what combination of sines and cosines it is. We could get lucky in one case, such as **A**, but we could also get unlucky, such as in **B**. In case **B**, we have an extreme case of the signal seeming to vanish when we sample .
- **Conclusion:** Worst case scenario, we cannot sample at the NR.

Phase reversal (the "wagon-wheel" effect)



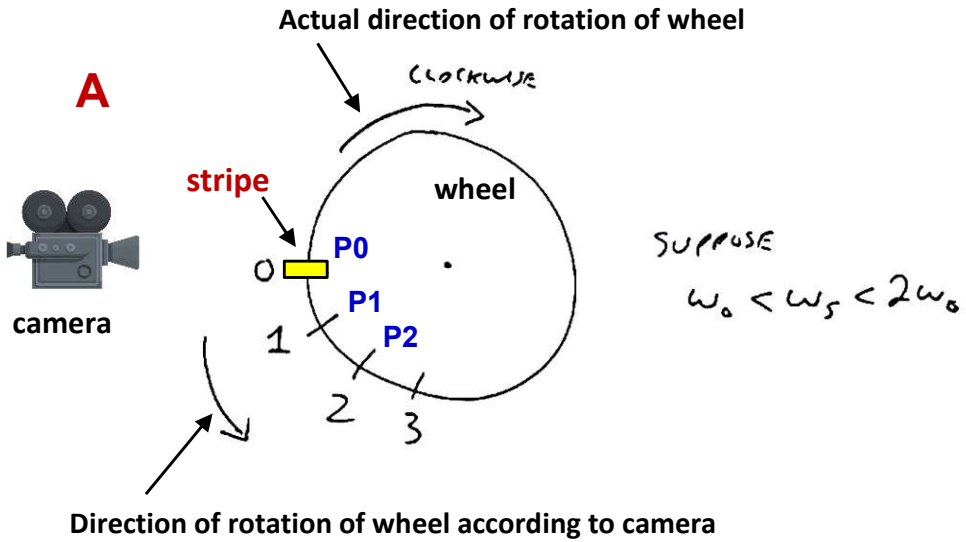
- **Phase Reversal:** Another interesting property, when we do not sample enough, is called **phase reversal**. We can show that if our sampling frequency, ω_S, is less than twice the bandwidth, ω_B, $\omega_S < 2\omega_B$, and if we have **x(t)** being some cosine at a frequency of ω_0, the reconstructed signal, $x_r(t)$, is a cosine that is aliased, **(2)**.
- First of all, we already knew from graph **A** that what happens when things go wrong. Here, when things go wrong, the number at **P1** is $\omega_S - \omega_B = 3/2 - 1 = 1/2$. This tells us that when we sample too slowly, the highest frequency appears to be $\omega_S - \omega_B$. That is exactly the same thing as **(*)** in equation **(2)**. This **(*)** is like "***lower frequency***", assuming that we sample too slowly.
- Besides aliasing, we see another phenomenon, which is the phase reversal. When comparing **(1)** and **(2)**, the fact that in **(1)** we have $+\phi$ and in **(2)** we have $-\phi$ means that ***we are going backwards***.

Phase reversal (the "wagon-wheel" effect)



- Here is a real world example. Let us suppose we have got a camera and that we are filming a wheel. This wheel has a **stripe** on it, shown by the yellow rectangle. In diagram **A**, the wheel is rotating clockwise.
- The Nyquist rate tells us that we need to sample quickly relative to this wheel. **What happens if we sample a little bit slowly?** Suppose $\omega_0 < \omega_S < 2\omega_0$. That is, our sampling frequency is between the frequency of the wheel and twice the frequency of the wheel. What that means is that we are sampling the wheel a little bit less than once every rotation. So, the idea is that when the wheel rotates clockwise, we sample it again ***before*** it gets to the starting point of **P0**. So, we are sampling it less than once per rotation.
- If **P0** is the ***starting image***, then as it rotates around, we would sample it again at **P1** and as it goes around, we would sample it again at **P2**, and so on. So, according to the camera, it would look like the wheel was going the other way (i.e., counterclockwise). This is because the stripe would appear to be going in the reverse direction.

Phase reversal (the "wagon-wheel" effect)



- This phenomenon is exactly what we see when we look at old movies. Here, it looks like the wagon wheels going the wrong way. Or, when we look at a car commercial, it looks like the wheels are going the wrong way. Even our eyes can sometimes fool us like that. Because our eyes have an ***innate sampling frequency***.
- **Conclusion:** In diagram **A**, if the camera shutter does not sample fast enough and the wheel is going too fast, we can easily get phase reversal phenomenon.

MATLAB examples of sampling and reconstruction

DIAL TONE FREQS:

350 Hz

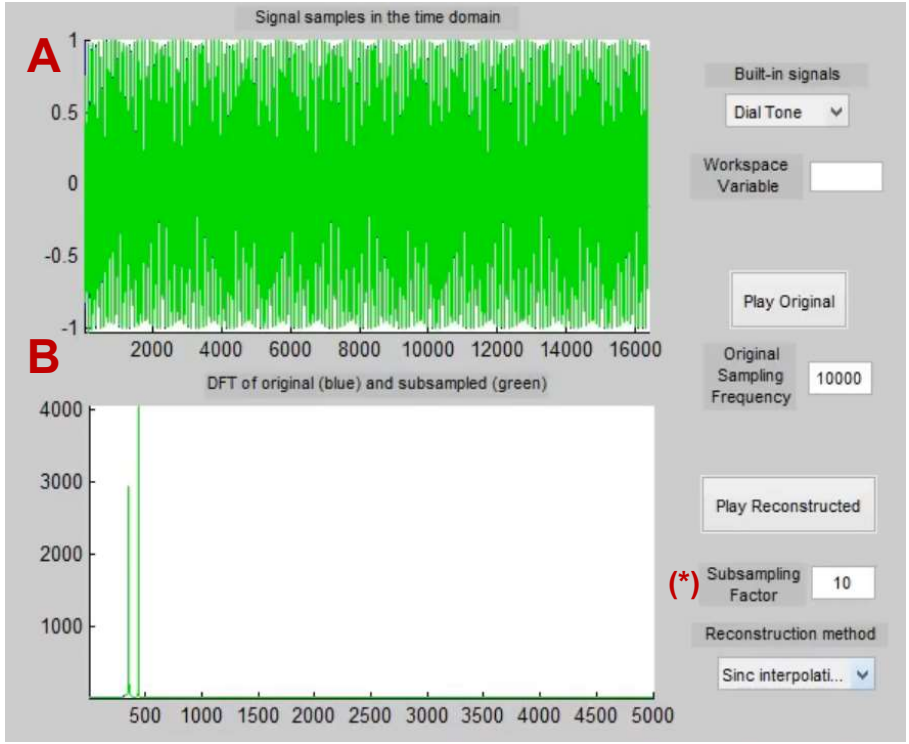
440 Hz $= \omega_B$

$2\omega_B = 880$ Hz

$10000/10 = 1000$ (1)

$10000/11 = 909$ (2)

$10000/12 = 833 = \omega_S$ (3)



- Let us say we have a signal (here, a dial tone) that has two component frequencies or two pure tones (or two sines) at **350 Hz** and **440 Hz**. So, here, ω_B **= 440 Hz**, which is the highest frequency in our signal. This tells us that our NR is $2\omega_B$ **= 2(440) = 880 Hz**. Based on NR of **880 Hz**, we know that we need to sample at least **880 Hz**. In our original signal, we are sampling at **10,000 Hertz**. So, the original signal (the **blue** curve) was way oversampled. So, the reconstructed signal (the **green** one) and the original signal (the **blue** one) overlap. Here, we are using sinc interpolation as our reconstruction method.
- Box **(*)** shows the **factor of subsampling (FOS)**. This is like saying, suppose we take every **10th** sample, then **FOS = 10**. If we do that, we are sampling at **10,000/10 = 1,000 Hertz**. Given that **NR = 880 Hz**, this should be perfectly fine (no aliasing).
- For **FOS = 10**, in **A**, the **blue** and the **green** curves, the original and the reconstructed signals overlap. In **B**, we look at both the original spectrum and the reconstrued spectrum.

MATLAB examples of sampling and reconstruction

```
>> % Parameters for generating the sound
fs = 10000;             % Sampling frequency (Hz)
total_duration = 10;  % Total duration of the ringing (10 seconds)
t = 0:1/fs:total_duration;  % Time vector for 10 seconds

% Frequencies for the ringing tone (e.g., 350 Hz and 440 Hz)
f1 = 350;              % Frequency of the first cosine wave (Hz)
f2 = 440;              % Frequency of the second cosine wave (Hz)

% Generate the two cosine waves
signal1 = cos(2 * pi * f1 * t);  % First cosine wave
signal2 = cos(2 * pi * f2 * t);  % Second cosine wave

% Combine the two signals
ringing_signal = signal1 + signal2;

% Modulate the signal (to mimic the on-off pattern of a ringing phone)
on_duration = 1;       % Ringing for 1 second
off_duration = 2;      % Pause for 2 seconds
modulation = [ones(1, fs * on_duration), zeros(1, fs * off_duration)];
modulated_signal = repmat(modulation, 1, ceil(total_duration / (on_duration + off_duration)));

% Ensure the signal and modulation are the same length
modulated_signal = modulated_signal(1:length(ringing_signal));  % Trim or extend modulation

% Multiply the ringing signal by the modulation pattern
final_signal = ringing_signal .* modulated_signal;

% Play the sound
sound(final signal, fs);
```

Prefiltering to avoid aliasing

DIAL TONE FREQS:

350 Hz

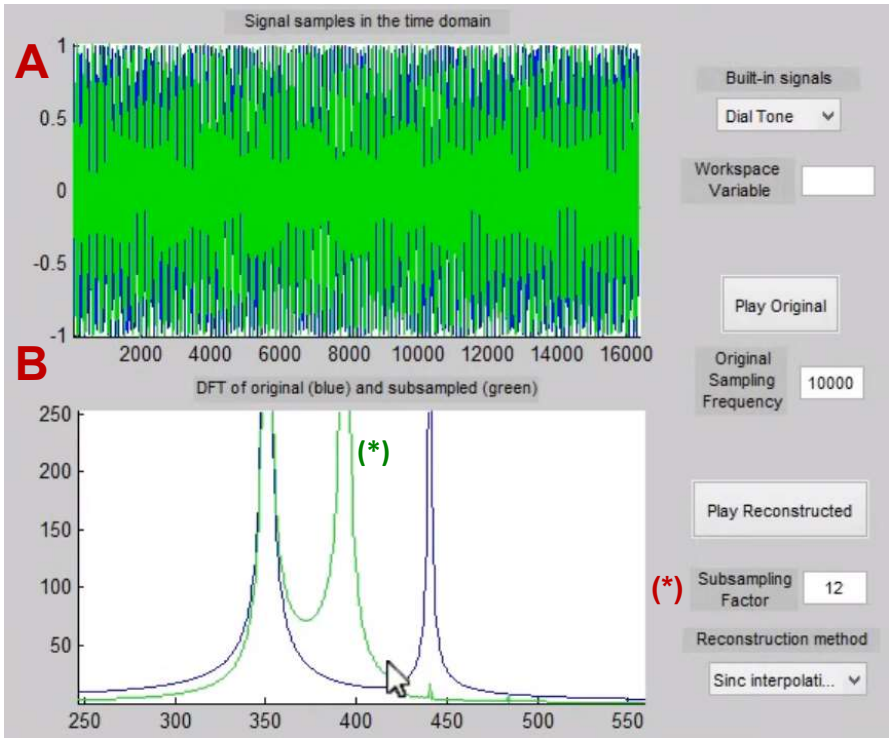
440 Hz $= \omega_B$ → 833 − 440 = 393 Hz

$2\omega_B$ = 880 Hz

10000/10 = 1000 (1)

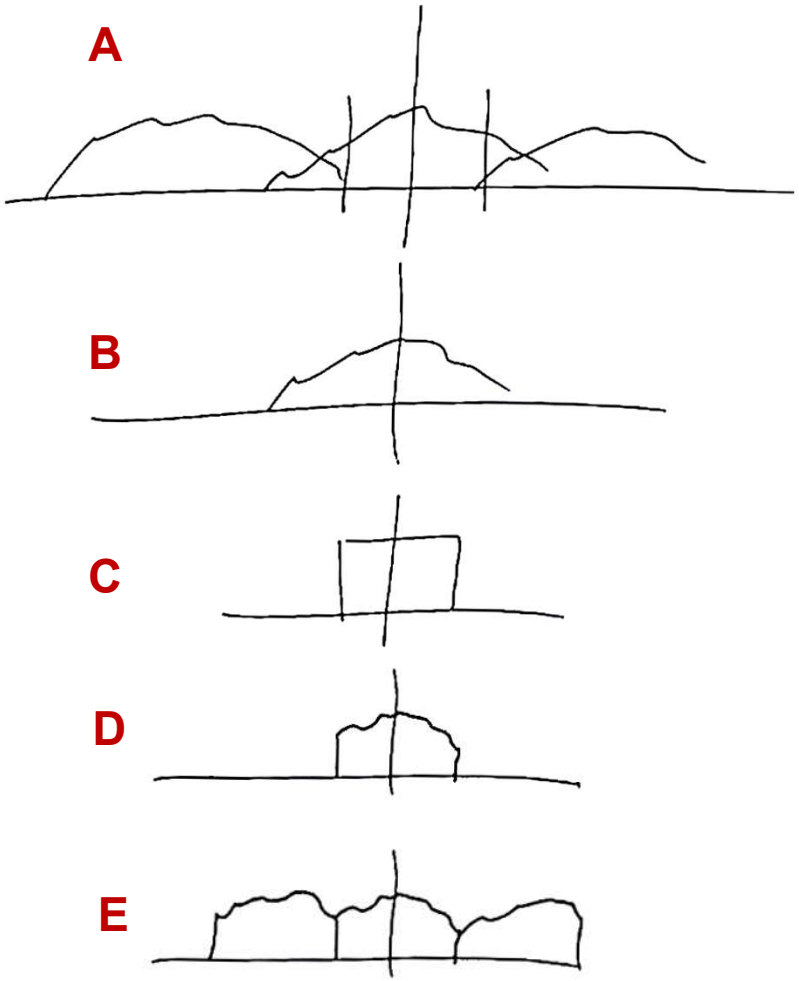
10000/11 = 909 (2)

10000/12 = 833 $= \omega_S$ (3)



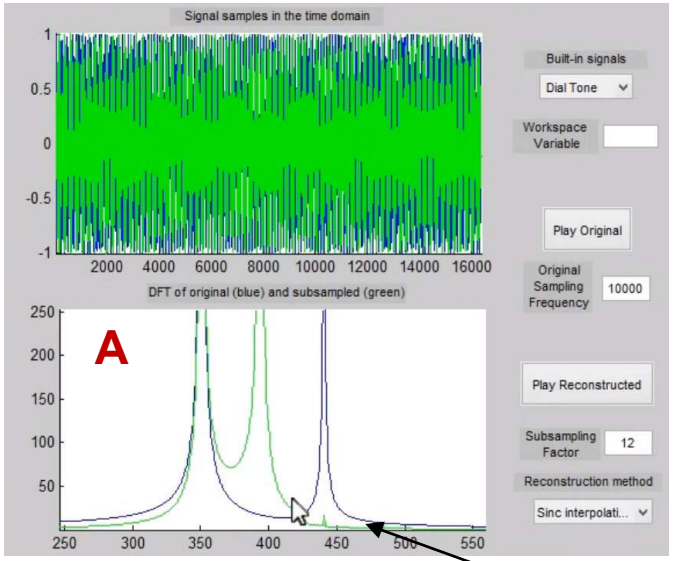
- However, if we subsample by **12** (**FOS = 12**), we get **10,000/12 = 833 Hz**. Here, we should start to have problems due to aliasing, shown in **A** and **B**.
- The aliasing problem we are going to have is that the **440 Hertz** sinusoid is going to alias into something that sounds like a lower tone. We can ***predict*** that when the sampling rate is ω_S **= 833 Hz**, and the bandwidth is **440 Hz** or ω_B **= 440 Hz**, then the **440 Hz** tone will turn into $\omega_S - \omega_B$ **= 833 - 440 = 393 Hz**. So, the **440 Hz** sine (the higher tone) is going to get shifted down to **393 Hz**.
- In graph **A**, when we subsample by a factor of **12** (**FOS = 12**), we can see that the blue and the green curves are no longer the same. In **B**, when we look at the peaks, we can see that the lower frequency sinusoid, which was originally at **350 Hz** did not get damaged by anything because we were sampling fast enough for that sinusoid. But for the sinusoid that was at **440 Hz**, it got aliased down to the green peak, **(*)**, that was not in the original signal. This **green peak** in **B** is exactly where we predicted it would be. It is roughly at **393 Hz**.
- **Conclusion:** To mitigate against this aliasing effect, we can ***prefilter the signal*** to definitely conform to the NR ***before*** we do the sampling. So, if we wanted to make sure that we did not introduce this distortion, what we could do before we did the sampling was ***lowpass filter the signal*** to prevent any of those high frequencies from getting involved in the sampling.

Prefiltering to avoid aliasing

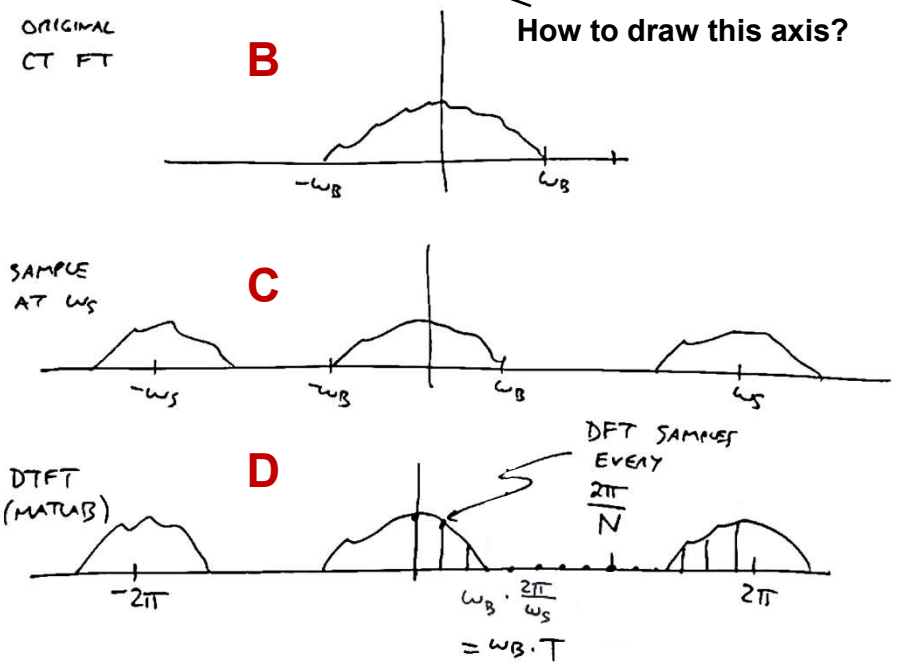


- Let us see what it means ***to prefilter the signal***. We know that we do not want to see **A**. So, what we could do *a priori* is that now that we know what the worst case would be, we are going to take the original signal, shown in **B**, and then prefilter it by an ideal reconstruction filter, shown in **C**. Then, we are going to pass only **D** within the sampler. That means that when we reconstruct, shown in **E**, we get copies of that. The worst case scenario is that they line up ideally as shown in **E**. Here, we do not get any aliasing.
- **Conclusion:** We can prevent aliasing by prefiltering with the right bandwidth.

Conversions between continuous time and discrete time; what sample corresponds to what frequency?

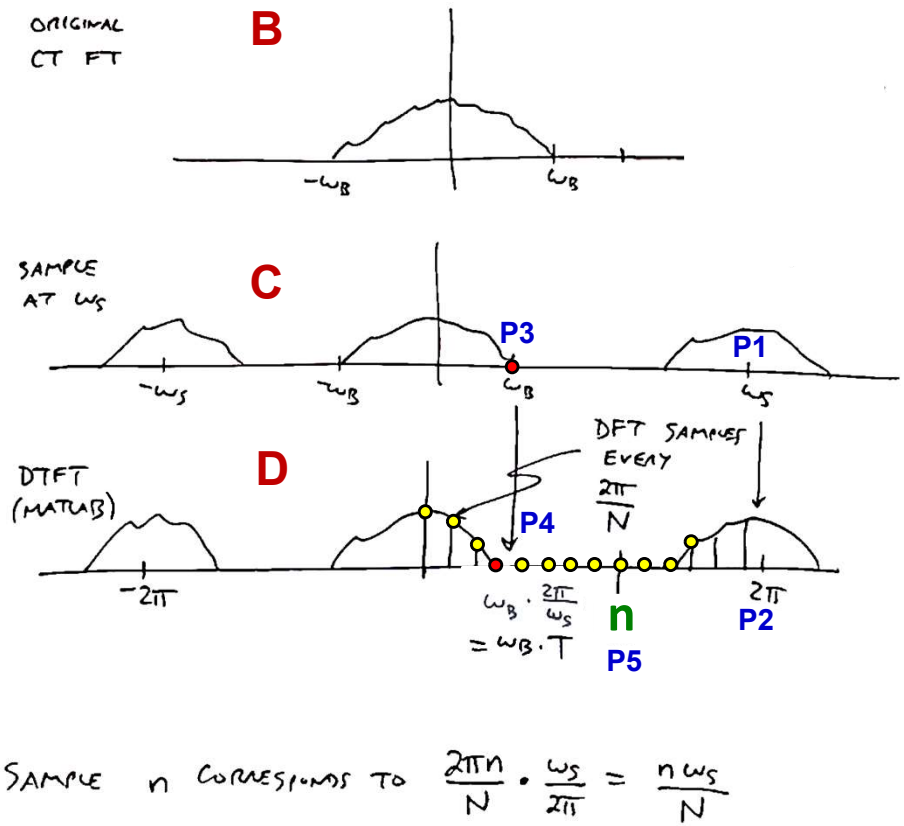


How to draw this axis?



- Let us now fully draw the connections between continuous time and discrete time, which is where we wanted to go from the very beginning (i.e., the big picture). The big picture is that we want to know how to represent continuous time signals. If we look at the MATLAB demo, that is exactly what we are doing here. Having this underlying continuous time signal, **how can we be drawing these axes that show what is happening in continuous time in Hertz?** Let us just be really clear about the ***conversion***. Let us say we have our original continuous time Fourier transform signal, **B**, and that this continuous time signal is bandlimited. Now, if we sample at some **sampling frequency**, ω_S, what we get is copies of that signal centered at the sampling frequency, shown in **C**. Graph **C** is saying that we assumed that our input signal samples are spaced apart by **T**, the **sampling period**.
- If we think about that signal as a discrete time signal in MATLAB, in MATLAB those signals are going to be spaced apart by units of **1**. In MATLAB, the DTFT is like graph **D**. We know that the DTFT's are **2π-periodic**. That means that we are still going to get these periodic looking signals shown in **D**, but the copies always occur at multiples of **2π**. Because we know that the signal is **2π-periodic**.

Conversions between continuous time and discrete time; what sample corresponds to what frequency?

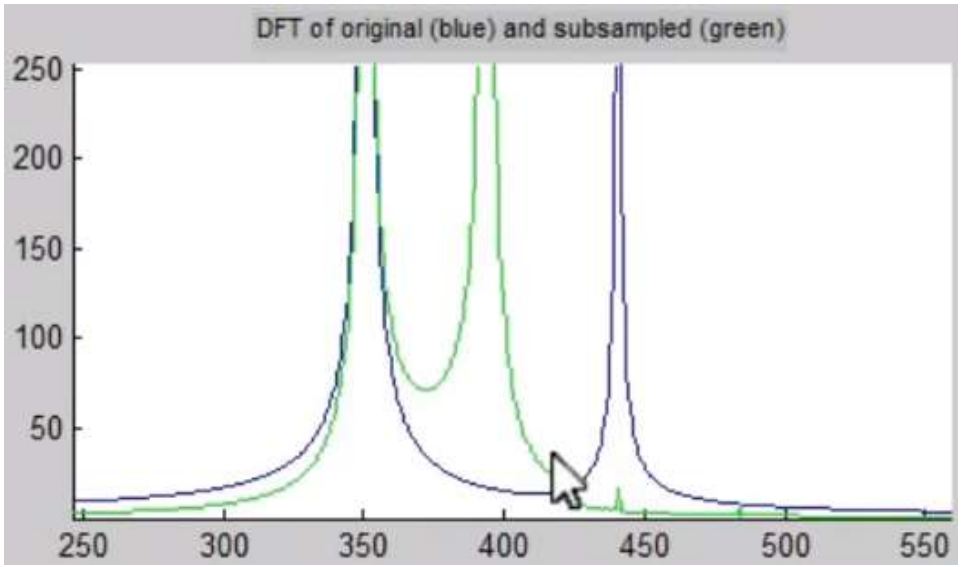


SAMPLE n CORRESPONDS TO $\frac{2\pi n}{N} \cdot \frac{\omega_S}{2\pi} = \frac{n\omega_S}{N}$

- What we are doing in **D** is that we are taking the original signal and we are modifying the frequency axis. So, what used to be ω_S in graph **C** (at **P1**) becomes 2π (at **P2**) in graph **D** and what used to be ω_B in graph **C** (at **P3**, **red** circle) becomes $(\omega_B)(2\pi/\omega_S)$ in graph **D** (at **P4**, **red** circle). Point **P4** can also be re-written as $(\omega_B)(2\pi/\omega_S) = \omega_B.T$.
- In Graph **D**, the **DFT samples** would be like sampling the curve every $2\pi/N$ units (shown by the yellow points). So, each yellow sampled points are like **DFT samples** every $2\pi/N$.
- If we wanted to draw an approximation to the FT of the original signal and we took the DTFT and the DFT, as an example, **what does sample n (at point P5) in graph D, correspond to in actual Hertz?** The answer is that **sample n** in **D** corresponds to the DTFT sampled at $(2\pi n)/N$. And, to convert back to Hertz, we have to multiply it by $(\omega_S/2\pi)$. So, we have $(2\pi n)/N.(\omega_S/2\pi) = (n\omega_S)/N$. Here, **N** is the length of our DFT.

Conversions between continuous time and discrete time; what sample corresponds to what frequency?

A



- **Conclusion:** This conversion, $(n\omega_s)/N$, is exactly what we used in order to be able to draw graph **A** in true units of Hertz. In other words, this conversion tells us how we can draw the frequency axis in Hertz.
- Now, we have all the pieces in place. We are now able to talk about DFT's and immediately through this chain, go back to what it actually corresponds to in terms of Hertz or radians, assuming that we sampled fast enough.
- This is something that ties a bow around the whole continuous time and discrete time relationship.

End of Lecture 13