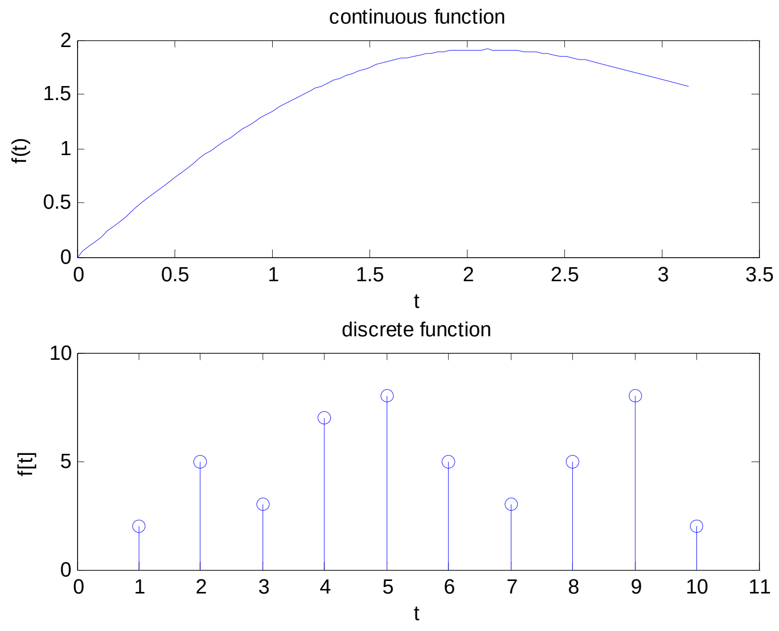
# 20.309 Tutorial Sheet #2
# discrete vs. continuous functions, periodicity, sampling

We will encounter two classes of signals in this class, continuous-signals and discrete-signals. The distinct mathematical properties of each, especially in regards to Fourier transforms, requires us to have at least a basic understanding of the differences between them. First I will present some mostly non-mathematical intuition, then discuss some of the theory behind it. This material is meant to help you understand the labs and homeworks.

Continuous-signals have, as their name suggests, a continuous independent variable. For example, the velocity of a moving body, like you moving around campus, could be considered a continuous signal – at any time t the velocity is clearly defined. Notation is the common f(t), which you are used to from mathematics classes.

In contrast, discrete signals are only defined at discrete independent variables. For example, the daily high temperature during a week is only defined for the 7 days. It would make no sense to compare the daily high of day 1 (say Monday) to day 1.5. Other examples of discrete signals include census data and class attendance; if you think about it there are many others. Notation for discrete numbers is typically f[n], with the brackets emphasizing that n is a sequence.

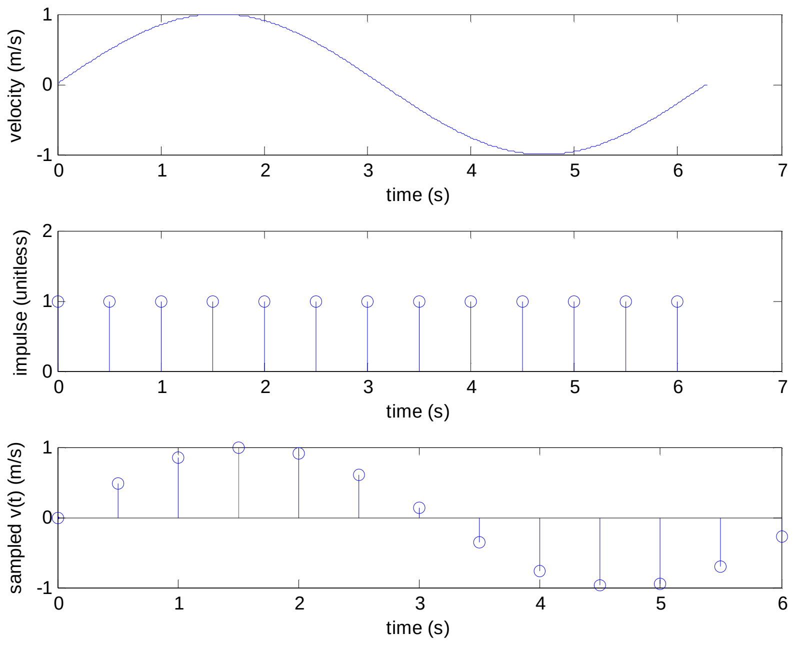
Graphically:



The distinction is very important to us because when we use computers to numerically perform mathematical operations we are confined to dealing with the properties of discrete numbers. That is because typical computers are built with digital logic dealing with ultimately binary data. MATLAB can approximate a continuous function by generating values at small intervals of the independent variable, but there is a limit to how much such approximations actually mirror the real thing. It is that limit we are interested in.

## Sampling

So how do we go from continuous to discrete signals? In the case of the moving body, if you were to measure the velocity with a radar gun and write down your reading every 0.5 seconds, you would in fact now have discrete data. We would say that your sampling frequency is 2 Hz, or 2 per second. Ideally, you would write down values at exactly every 0.5 seconds. Mathematically we can define this operation as multiplying an "impulse train" with your data, then passing it through a “continuous to discrete converter” or “C/D converter”. Let’s examine the situation where you are on a swing, tracing the motion of a pendulum – thus your velocity would be a sinusoid. Your velocity v(t) is thus equal to sin(t).



Signal: $v(t) = \sin(t)$

Impulse train: $p(t) = 1$ for $t = 0, \pm 1, \pm 2, \ldots$

$p(t) = 0$ otherwise

Sampled: $v_p(t) = v(t)\,p(t)$

Performing this operation is known as "sampling" the function v(t) at the sampling frequency of 1/ΔT, where T is the period of the impulse train. Note that an impulse train can also be written as

$$p(t) = \sum_{-\infty}^{\infty} \delta(t - kT), T = 0, \pm 1, \pm 2, \ldots$$

If you sample $v_p(t)$ by multiplying with p(t) you will obtain a sampled signal. Note that the impulse train is still a continuous function although it looks discrete, since it’s defined for every t. Also note that we are multiplying two signals in time, which by the convolution theorem which we saw in class means the transform of $v_p(t)$ equals the convolution of the signals in frequency: $V_p(\Omega) = V(\Omega) * P(\Omega)$. $v_p(t)$ becomes a discrete signal $v_p[t]$

after running the signal through a C/D converter which will non-dimensionalize time into sample numbers (more on this and the convolution in frequency later).

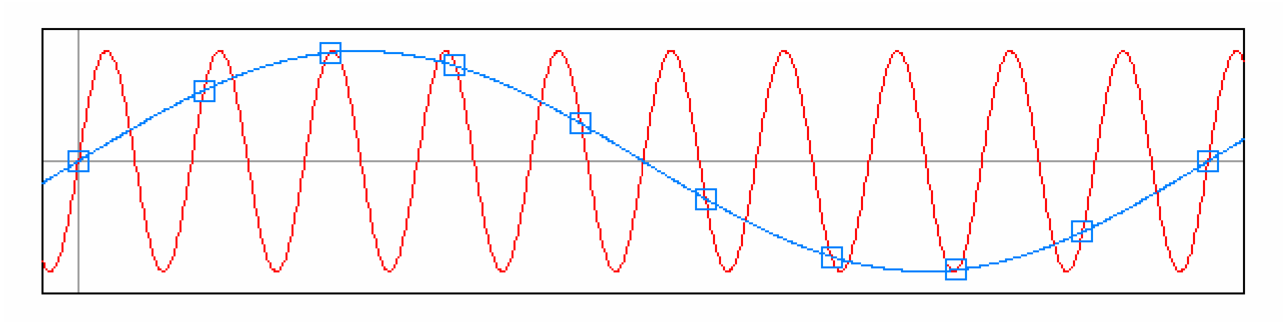
If v(t) is changing slowly enough for our sample rate to capture all the information contained in the signal, we can completely reconstruct v(t) based on the discrete sequence $v_p[t]$. Stop and think about this for a while – you can exactly reconstruct a function v(t) by knowing an infinitesimally smaller subset of information, as long as your sampling frequency is great enough. The cutoff point is called the "Nyquist sampling frequency", and it is exactly twice the highest frequency you can reconstruct in your signal. This is known as the "sampling theorem":

Sampling at $F_{nyquist}$ with $F_{nyquist} > 2*f_{max}$ (where $f_{max}$ is the highest frequency in your signal of interest) allows complete reconstruction of the signal from its sampled values. A more formal way to say this (T = period):

Sampling Theorem, Oppenheim and Willsky, "Signals and Systems" page 518

Let x(t) be a band-limited signal with $X(\Omega) = 0$ for $|\Omega| > \Omega_{max}$. Then x(t) is uniquely determined by its samples x(nT), n = 0, ±1, ±2, …, if $\Omega_s > 2\Omega_{max}$ where $\Omega s = 2\pi/T$. Given these samples, we can reconstruct x(t) by generating a periodic impulse train in which successive impulses have amplitudes that are successive sample values. This impulse train is then processed through an ideal lowpass filter with gain T and cutoff frequency greater then $\Omega_{max}$ and less then $\Omega_s - \Omega_{max}$. The resulting output signal will exactly equal x(t).

If your signal contains frequencies higher then half your sampling rate, then you are "undersampling" the signal and will not be able to reconstruct it fully. If you try to reconstruct it anyways you will encounter "aliasing", or distortion of the original signal. It's really useful to look at the graph below to see what this means – if you aren't sampling fast enough you have no way of knowing if the signal you are observing is the red or blue one.



"Two different sinusoids that give the same samples; a high frequency $f > f_s/2$ (red) and an alias at $f_s - f$ (blue)." (http://en.wikipedia.org/wiki/Aliasing, accessed 10/05/06).

Later we will see what aliasing looks like in the frequency plane.

**Background Theory**

I will present some basics to help understand hwk2, but the full mathematics behind it is outside the scope of this course. A course entitled "Signals and Systems", or the textbook of the same name by Oppenheim and Willsky (available in the lab) can fill in the blanks if you are interested.

In class Prof. Manalis introduced the Fourier Transform for continuous signals, and its inverse. We will specify this transform as the CTFT, or Continuous Time Fourier Transform. Because MATLAB doesn't work with continuous signals, when you want MATLAB to perform a Fourier Transform it actually performs the Discrete Fourier Transform, or DFT.

In class we saw this table:

| | **Time** | **Frequency** |
|---|---|---|
| **Fourier series (CTFS)** | Continuous (t) | Discrete ($w_n$) |
| **Fourier integral (CTFT)** | Continuous (t) | Continuous ($\Omega$) |
| **Discrete Time Fourier Transform (DTFT)** | Discrete [t] | Continous (w) |
| **Discrete Fourier Transform (DFT)** | Discrete [t] | Discrete [2*pi*k/N] |

(I've added the DTFT)

The CTFS maps a continuous, periodic signal to its harmonic (sinusoidal) components. This means that we are trying to mimic our signal with sines and cosines of integer multiples of the frequency (in radians) $\omega_n = n\Omega =$ n*2pi/T. The series of frequencies thus presents a discrete mapping in frequency space. Note from the formula that the CTFS is aperiodic.

CTFS:

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\,dt$$

$$a_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\cos(\omega_n t)\,dt$$

$$b_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\sin(\omega_n t)\,dt$$

$$\omega_n = n\frac{2\pi}{T}, n = 1,2,3...$$

CTFS$^{-1}$ (inverse):

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(\omega_n t) + b_n \sin(\omega_n t)$$

The CTFT is a generalization of the CTFS to include all frequencies in the transform, not just integer multiples. We are taking the formula for CTFS and taking the limit as T goes to infinity. Thus the signal in time no longer needs to be periodic (since the period T is going to infinity), and in turn we need to use all frequencies in order to mimic the signal. The CTFT of a signal is thus continuous, and also not necessarily periodic. The CTFT transforms a continuous time signal to a continuous spectrum in frequency. We represent that frequency as $\Omega$.

CTFT: $$H(\Omega) = \int_{-\infty}^{\infty} h(t)e^{-iwt}\,dt$$

CTFT$^{-1}$: $$h(t) = \int_{-\infty}^{\infty} H(\Omega)e^{iwt}\,d\Omega$$

The DTFT is symmetrical to the CTFS. Recall the CTFS maps a periodic, continuous signal in time to a discrete signal in frequency. The DTFT maps a discrete signal in time to a periodic, continuous signal in frequency. The connection to the CTFT is that the DTFT acts on a sample of a continuous signal. If we were to rewrite the CTFS in terms of exponentials (using Euler's Formula, $e^{i\omega} = \cos\omega + i\ \sin\omega$) the symmetery would be more apparent, I'll leave that to you, or you can look up the Strang reference "Fourier Transforms" on page 5.

DTFT: $$H\left(e^{i\omega}\right) = \sum_{-\infty}^{\infty} h[t]e^{-i\omega n}$$

DTFT$^{-1}$: $$h[t] = \frac{1}{2\pi}\int_{2\pi} H\left(e^{iw}\right)e^{i\omega n}\,d\omega$$

All these transforms involve continuous signals in either time or frequency space. However as you know computers in general can only work with discrete information, and additionally we can only sample for a finite time. Thus the best we are able to do numerically with transforms is a discrete approximation in frequency to a discrete signal in time. This brings us to the DFT. The reference in the 20.309 schedule page titled "FFT reference material" contains a more detailed explanation if you desire.

**The Discrete Fourier Transform (DFT)**

Why DFT if we have DTFT? Recall that DTFT provides a continuous frequency spectrum provided that we are given an "infinitely long" discrete-time signal. However, in real life we do not have the luxury of sampling a signal forever. Therefore DFT approximates the frequency spectrum of a DTFT using discrete samples. The DFT maps a discrete, aperiodic and finite signal in time to a discrete, periodic signal in frequency. This is something computers can do. The DFT of a length N signal in time will also be of length N in frequency. In otherwords, if we have a signal of length 10, we could only estimate the entire frequency spectrum with 10 samples. Note that we have a limited series of frequency values in the DFT, namely N. If a signal f[t] contains a frequency $\omega_x$ which is not close to one of the frequency values available in the DFT, the frequency spectrum will have to use a combination of the available frequencies to substitute, resulting in "spectral leakage". Thus when performing a DFT in MATLAB it can be extremely helpful to know what frequency you are looking for, and adjust input parameters accordingly. Modern implementations of the DFT use an algorithm called the Fast Fourier Transform, or FFT, and use clever windowing techniques to reduce the inherent problems of the DFT approximations. Windowing refers to taking subsets of the signal and performing the FFT multiple times – you can see how this can affect your result based again on the number N of frequency values being the same as the number of time values.

DFT: $$H[k] = \frac{1}{N}\sum_{n=0}^{N-1} h[n] e^{-ik\left(\frac{2\pi}{N}\right)n}, k = 0,1,...,N-1$$

DFT$^{-1}$: $$h[n] = \sum_{k=0}^{N-1} H[k] e^{ik\left(\frac{2\pi}{N}\right)n}, n = 0,1,...,N-1$$

Now let's look at a sampling example…see next document, hand-written pages.