The Intuitions of Signal Processing (for Motion Editing)

This chapter will be an appendix of the book *Motion Capture and Motion Editing: Bridging Principle and Practice*, by Jung, Fischer, Gleicher, and Thingvold to be published Summer 2000 by A. K. Peters publishers. This chapter is reprinted by permission.

Michael Gleicher
Department of Computer Sciences
University of Wisconsin, Madison
gleicher@cs.wisc.edu
http://www.cs.wisc.edu/gleicher

March 22, 2000

Preface

Signal Processing is a subject that is extremely useful for people working with motion for animation. In fact, Signal Processing is a subject that is incredibly useful across an amazing range of fields, including Computer Graphics, Electrical Engineering, Mechanical Engineering, and Physics. Because of this utility, almost all engineers and mathematicians get at least some exposure to signal processing during their training. One outcome of this is that the basics concepts and vocabulary of signal processing are often used in discussion of topics where they apply, such as when we talk about motion editing.

The goal of this paper is to provide a basic introduction to some of the foundational intuitions and vocabulary of Signal Processing, in a minimally mathematical way. It is specifically targeted at people who might want to apply these towards being conversant in motion editing. My goal is to serve two needs:

1. Some people have not had exposure to Signal Processing, and therefore need a quick introduction to the basic concepts and intuitions so they know what the words mean when we discuss motion editing.
2. Other people (like me) have been exposed to signal processing in an introductory electrical engineering (or other discipline) class. Often, these classes stress the mathematical formalisms without giving the intuitions. For this audience, I aim to provide a refresher of the intuitions, and provide some connection to motion editing.

This paper is not meant to provide an introduction to even the basics of Signal Processing. For this, I recommend one of the many textbooks on the subject (since this is a standard course that almost all engineers and mathematicians take, there are a lot of books). Many textbooks are too mathematical and abstract for my tastes.

The books I have referred to in preparing this document are:

- DSP First: A Multimedia Approach [DSPFirst], which seems to be a good balance of theory, intuitions, and examples.
- Signals and Systems: Continuous and Discrete [Ziemer], which was the text I used in my undergraduate Electrical Engineering signal processing course.

- A Wavelet Tour of Signal Processing [Mallat], which I like because it covers newer analytical tools, such as wavelets. Because it has this cross-section of tools, including those described in this document, it is forced to discuss their similarities and differences.

Alternatives to Signal Processing books are books on Image Processing (since an image is just a special kind of signal). Image Processing is probably more familiar to the graphics audience. One good text on the subject is Image Processing for Computer Graphics by Gomes and Velho [GomezVelho]. This book has a nice chapter that summarizes the basics of signal theory, at a good level of mathematical detail. Patrick Hanrahan has created a set of introductory notes on signal processing similar to this document [Hanrahan]. His notes are targeted at graphics students, and are therefore more applied to images.

Introduction: What is a Signal?

A signal, quite broadly, is defined as something that carries information. The dictionary definition for the word used in the sense that we mean it is:

> a detectable physical quantity or impulse (as a voltage, current, or magnetic field strength) by which messages or information can be transmitted [Webster].

A signal is something that has a value (it is measurable) that may change, and by examining this value, we can extract some meaning.

This definition of signal is sufficiently vague that it may seem meaningless. However, it is this generality that gives the study of signals its power. Many of the tools for examining signals are defined in ways that are independent of what kinds of signals we are dealing with.

Some examples of signals include speech, images, video, audio, electrical, and vibrational. For example, the voltage across two points in a circuit, or a current that flows through a wire, could be considered an electrical signal. These electrical signals are things that we can measure that can possibly change over time. By observing how the value changes over time, we can get information. A different type of signal is an image that might measure the intensity of light, which changes over the area of the image. These two examples illustrate the two "parts" to a signal: a domain, the thing that the signal is measured over (time and position respectively); and the range, the values that are measured (voltage and intensity).

Most often, we consider signals with one parameter for the range, and typically, this parameter is time. We call such signals time-varying. The motion for character animation is a time-varying signal: we are interested in conveying information by observing how the pose of the character changes over time. At any instant in time, we have a number of things we may measure about the character, such as positions or joint angles.

Abstractly, we can think of a signal as a function that maps from the domain (time for our examples) to a value. That is, a signal is something of which we can ask the question "what is your value at time t." The beauty of signal theory is that everything is based at this abstract level. The methodology and intuitions do not care if the values represent voltages across a wire, pressure waves in the air, or angles on an animated character.

In all of the examples we described, the "values" that signals provide are individual real numbers (scalars). For some applications, we might want the values to have other forms. For example, if we are doing image processing, we might want each position in the image to map to a color (perhaps represented as a triple of scalars). Or, for motion editing, we might want to map each time to a configuration of the character, which would be a vector of numbers including the position of the character and its joint angles. The standard theory of signal processing focuses on the scalar-valued signals. When vector valued signals are treated, they are typically handled by treating each individual scalar independently. So, in the case of motion editing, we would treat each individual scalar independently as a separate scalar signal, which belies the complex interaction of parameters.

Time Domain Analysis

Suppose we have a time-varying signal, so we can determine what the value is for any given time. Mathematically, we like to think of time as a continuous parameter, that is we can ask for the signal's value at any time, and that for any two times that we might ask about, we can ask about times in between.

When dealing with signals in a digital system, such as a computer, we often do not have this luxury. We must deal with cases where the signal's value can only be known at a specific set of instants. We call such discrete-time. Note that most signals begin as continuous-time, and that discrete-time is merely an artifact of trying to represent the continuous process on a computer. For example, a joint angle signal in a motion-captured motion began as a continuous-time physical signal (the measurement of the performer's joint), but is represented in the computer by the values at a list of specific times. We will begin our discussion of signals by considering continous-time, and return to the realities of discrete-time systems later.

A time domain representation of a signal is something that allows us to ask "what is the value at time t." Generally, this means that we can represent it as a mathematical function to evaluate

$$v = f(t).$$

How we actually choose to represent the function is actually unimportant for now - we can think of it as a black box that simply answers questions. The typical way to visualize a signal is as a waveform, for example:

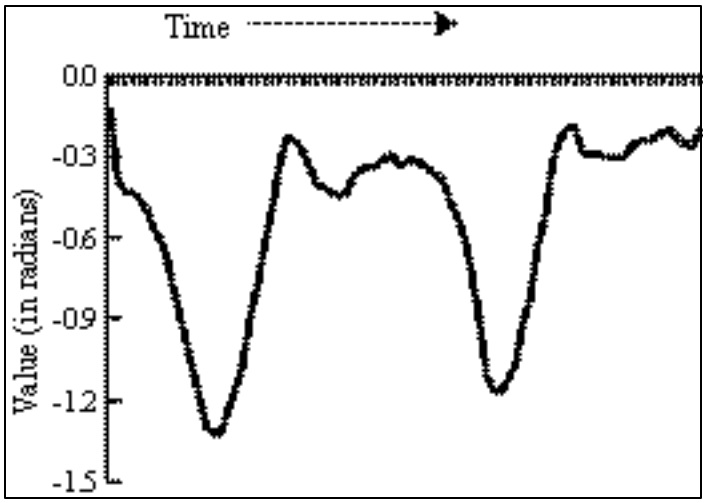


Figure 1: A signal, in a time domain or waveform view.

If we take this abstract view of a signal, it should be obvious that we really can't know much about the signal. If we try to analyze what is going on and are limited to only asking about specific times, the kinds of things we can deduce about the signal are very limited. If we ask what the value is at one particular time, we get little idea of what happens before or after. To put it a different way, if we are limited in the questions we can ask about a signal, we might have to ask a lot of questions. For instance, if we want to ask "does the signal have a value greater than 5 at any time," we would have to check a lot of times.

While this example is contrived, it is meant to motivate the need for a different way to view signals. The idea is that if we look at a signal a different way, there will be a different set of questions that we can answer easily.

Views of a signal can be thought of as a set of simple "building block" signals that are combined to make the signals that we are interested in. To analyze a signal, we break it down into a combination of the building blocks. Different sets of building blocks provide different views of the signal. The time domain is actually

an example of this, although we usually don't think of it this way. The building blocks of the time domain would be the unit impulses, which intuitively can be thought of as signals that have value 1 at one instant in time, and have value 0 at all others (more precisely, a unit impulse has value 0 everywhere except it's "location" and has unit area underneath the curve). We can create any signal by adding together multiples of these building blocks.

Before introducing Fourier analysis, which is simply a different set of building block signals, we first need to review some other signal terminology.

Periodic Signals

A periodic signal is a signal that repeats itself over and over. That is, if you know that value of a signal at a given time, then you know what the value of that signal is some time in the future (when the signal repeats itself). In fact, you know the value at many times in the future (each repetition). Mathematically, we can write this definition as:

$$f(t) = f(t + k\,p)$$

where k is any integer, and p is the amount of time that the signal needs to repeat itself. The smallest non-zero value for p is called the period.

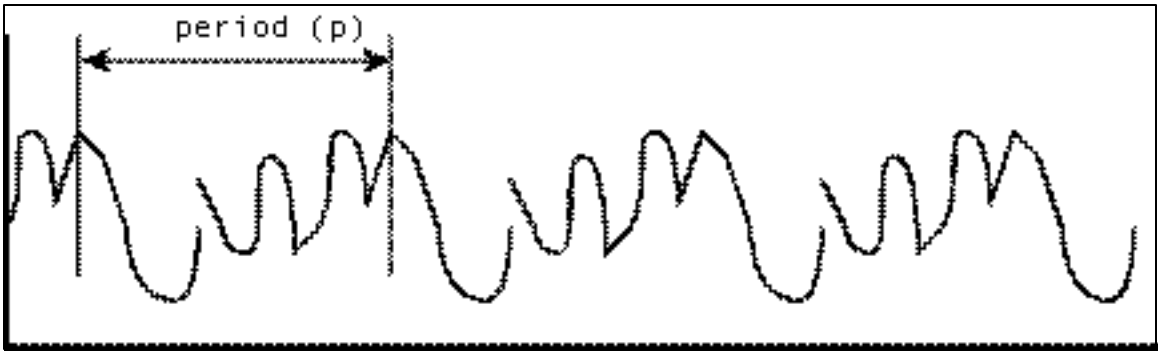


Figure 2: A periodic signal with its period show.

The amount of times that a signal repeats itself in a unit of time is called the frequency. Frequency is the inverse of period

$$\omega = 1/p.$$

Frequency is measured in repetitions per unit of time. The standard measure is the number of repetitions per second, called a Hertz.

In the real world, few things are perfectly periodic. However, we discuss periodic signals because they are easier to deal with mathematically. The analytical tools we are about to introduce for periodic signals have extensions for dealing with more general signals. These extensions are more difficult to explain, but share the same basic terms and intuitions.

One of the most basic signals is a sine (or cosine) signal. These signals have the mathematically simple form

$$f(t) = A \sin(2\,\pi\,\omega + \phi)$$

where A is an amplitude to scale the signal to the “size” we need (since sin always gives a number between -1 and 1), omega is the frequency of the signal (which is multiplied by 2 π since this is how long it takes for the sin function to repeat itself), and phi is called the phase shift which allows us to “start” the repeating bit at someplace other than 0.

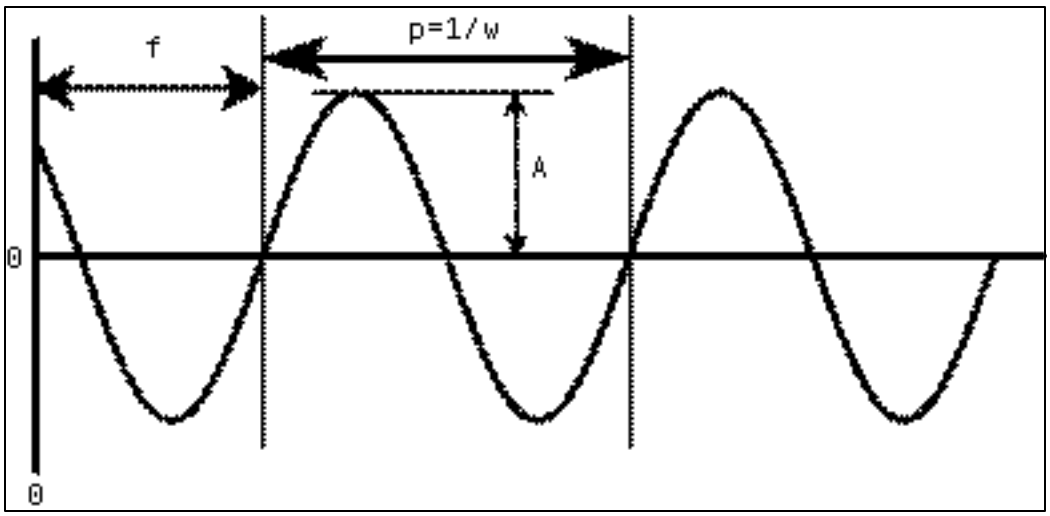


Figure 3: A Sinusoidal Signal.

An alternate way to represent a sin wave is to write it as the magnitude of the exponential of a complex number. Typically, this notation is used in signal processing literature because it is easier to perform certain mathematical operations on. For our discussion, we will continue to use the trigonometric functions since we won't be doing the mathematical derivations.

Another basic signal is a square wave. This signal has the value 1 for the first half of its period and -1 for the second half of its period.

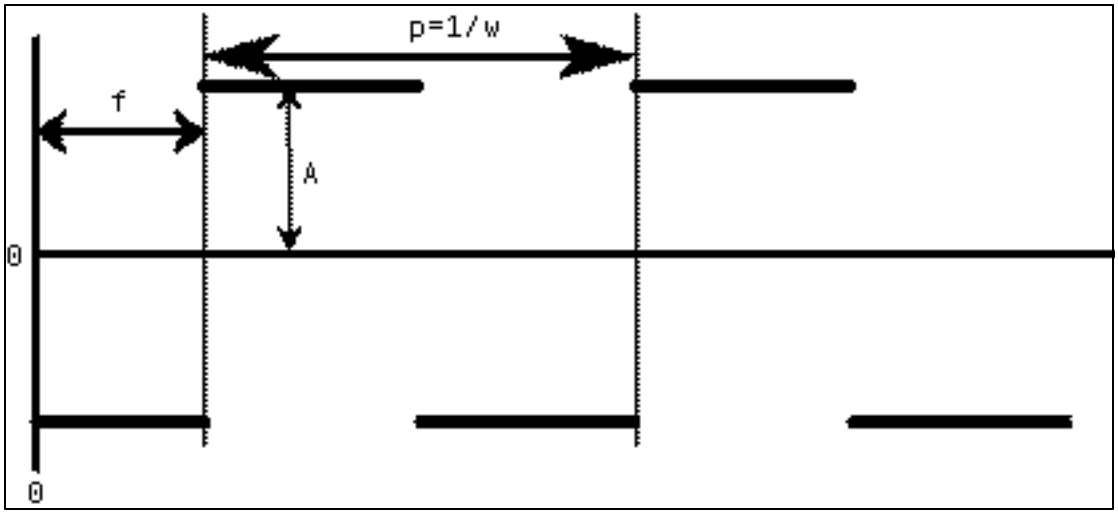


Figure 4: A Square wave signal. Note: the signal only have values of -1 and 1 and never has values in between.

An unusual thing about a square wave is that it is discontinuous: the signal instantaneously changes from one value to another. Often when we draw a picture of a square wave we draw the vertical connection, but this is only an artifact of how we draw the picture.

Frequency Domain Analysis

Sine signals have a number of interesting properties that make them useful building blocks for analyzing signals. One obvious observation is that if we add two sine waves of the same frequency and phase shift together, the result is a sine wave of the same frequency and phase shift. In fact, if we use the complex

number notation mentioned above, even the phase shifts can be handled. Therefore, in this discussion we will simply assume the phase shifts are zero to make our notation easier, with the caveat that we are making some simplifications so our discussion better appeals to intuition.

If sine signals have different frequencies, however, addition cannot combine them. This turns out to be a useful property because if we combine sine signals to make a blend of them, the result can later be broken back down into its pieces. Therefore, if we want to represent a blend of sine signals, we can describe the blend by what the combination is, for example by keeping a list of pairs of frequency and amplitude. This is a frequency domain representation of the signal as it specifies the signal by stating what frequencies it contains and how much of each, rather than explicitly saying what happens at each time. It is easy to figure out what the time-domain representation is (by summing up all of the different component sine signals). For example, we might describe a signal as being 2 of f=1, 3 of f=3, and 2 of f=4. Just as we graph the time domain representation, we might graph the frequency domain representation.

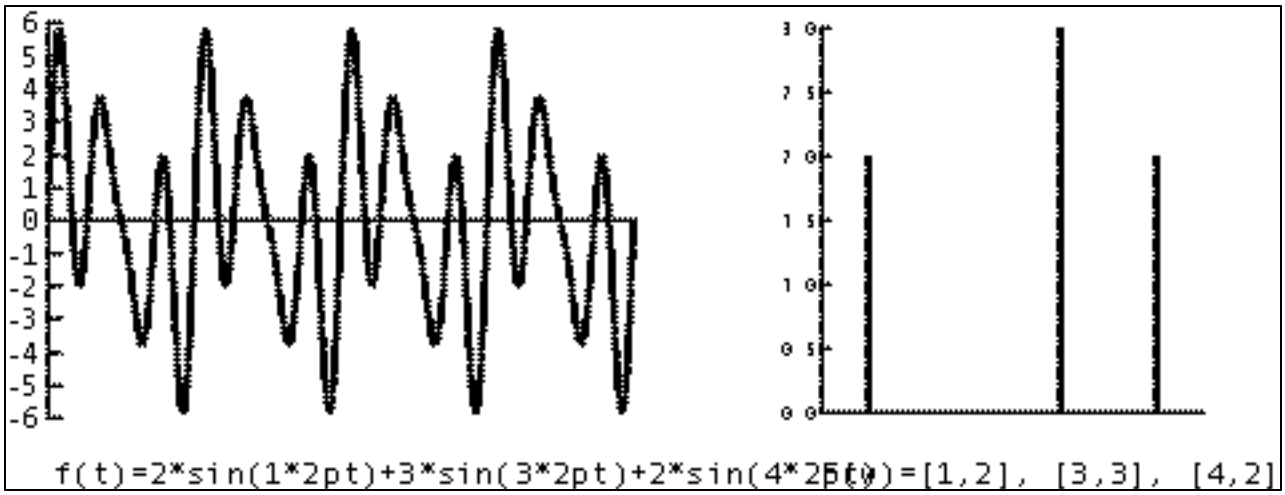


Figure 5: A signal and in both its time domain and Fourier domain representations.

The utility of the frequency representation is comes from the fundamental Theorem of Fourier Analysis. The most basic form of the theorem states that any (with some caveats about continuity) periodic signal can be made up of a blend of sine signals of frequencies that are multiples of the original. It may take an infinite number of these sine signal components to make up the original signal, but it still can be made up of sine signals. Technically, if the signal is not symmetric (that is f(t)=f(-t)), the phase shifts will not all be zero. This can be accounted for by either including a phase shift with each frequency, or by using both a sine and cosine at each frequency.

The Fourier Theorem tells us that any signal f(t) with period p (and therefore, frequency ω=1/p) can be written as

$$f(t) = a_0 + a_1 \sin(2\pi\omega t) + a_2 \sin(2*2\pi\omega t) + a_3 \sin(3*2\pi\omega t) + \ldots$$
$$+ b_1 \cos(2\pi\omega t) + b_2 \cos(2*2\pi\omega t) + b_3 \cos(3*2\pi\omega t) + \ldots ,$$

or, to use a nicer notation

$$f(t) = a_0 + \sum a_i \sin(i*2\pi\omega t) + b_i \cos(i*2\pi\omega t),$$

or, if we prefer to work with phase shifts (storing a and ϕ, rather than a and b),

$$f(t) = a_0 + \sum a_i \sin(i * 2\pi\omega t + \phi_i).$$

Therefore, one way to describe any periodic signal is by specifying the amounts of each different sine signal component (the as and bs, or as and ϕs). Representing a signal this way is called a frequency domain representation or a Fourier Series. The process of determining the coefficients given a time domain representation is called the Fourier Transform, and the process of converting from the frequency representation back to a time domain representation is the Inverse Fourier Transform.

Commonly, the need to represent two numbers per frequency causes signal processors to speak of the phase relation by giving a positive and negative value for each frequency. The notation makes much more sense if we write the equations using the notation of complex exponentials, which are a different way to describe the sine functions. This way, the Fourier representation of a signal can be given as a single graph on the real line. For simplicity in our discussion, we omit the negative frequency terms.

The Fourier Series gives us one way to view a signal, by providing a set of basic building block signals (the sine waves of differing frequencies) that we can decompose any other signal into. The set of basic building block signals must have some special properties (so that we know we will be able to build the signals of interest out of them), which the Fourier Series does. There are many other useful sets of signal building blocks, each providing a different type of analytical method for viewing signals.

As an example of the Fourier Series representation, consider a square wave. This signal does not resemble a sine signal, however, it can be represented by the series (a1 = 1, a3 = 1/3, a5 = 1/5, ...) or,

$$a_n = \begin{matrix} \frac{4}{\pi n} \text{ if } n \text{ is even} \\ 0 \text{ if } n \text{ is odd} \end{matrix}.$$

The process of determining these values is a standard exercise in any introductory signal processing class.

If a signal is not periodic but is defined over a limited time period, one way to handle it with Fourier Analysis is to simply copy it over and over. This is called periodic extension.

The intuition to take away from this is that any signal can be represented by a combination of simple signals of varying frequencies. Even if the signal itself is "low frequency," when we look at it in the frequency representation, it may contain higher frequencies. For example, if we have a low frequency square wave signal (for example with frequency of 1), this signal "contains" high frequencies.

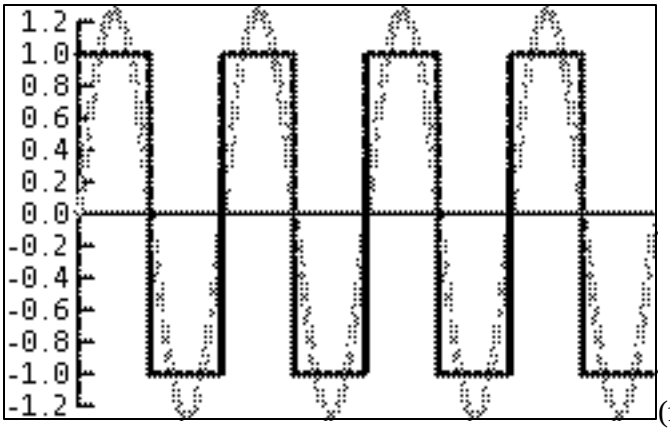
The second intuition is that Fourier analysis gives us a different "view" of a signal than time-domain. Depending on the kind of questions we want to answer about the signal, one or the other representation may be better. For example, if we want to know the value of the signal at a particular time, the time domain representation is better. To get an idea of what the frequency domain representation may be good for, we need to gain intuitions of what it means for a signal to "have" high frequency content.

Approximations with Fourier Series

The Fourier series representation may seem cumbersome: it needs a (potentially) infinite number of terms to describe a signal. This is actually no worse than the time domain representation that would have to specify an infinite number of values for all of the different times in the period.

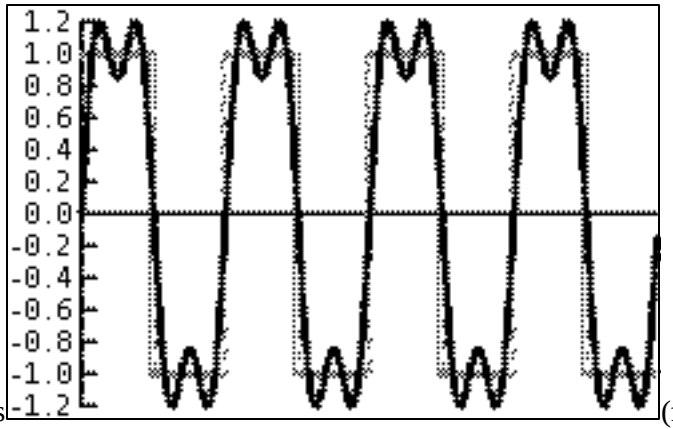
There are some signals that have a compact Fourier series representation. For example, a sine signal consists of only a single Fourier term. Signals that can be represented by a finite number of Fourier terms are called band limited because their frequencies are contained in a range or band of the total range of possible frequencies.

As we saw before, a square wave is not a band limited signal because it requires an infinite number of Fourier Series terms to be represented exactly. If we only chose a subset of these terms, we would only get something that approximates a square wave. If we choose only one term, we get something that doesn't approximate the square wave very well. As we use more and more terms, we get something that more closely resembles the square wave. In the limit, when we include "all" infinity of terms, we get the square wave. Note: when we draw the original square wave in the illustrations, we include the vertical lines for clarity. The actual square wave does not have any values other than -1 and 1.

With one term 

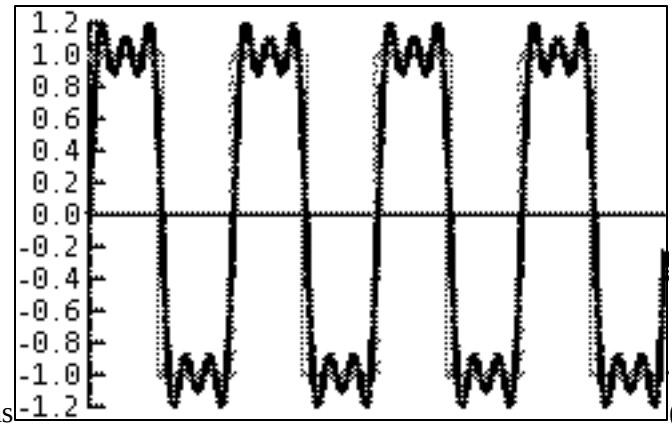
(first harmonic)

Figure 6: Approximating a square wave with 1 harmonic.

With two terms 

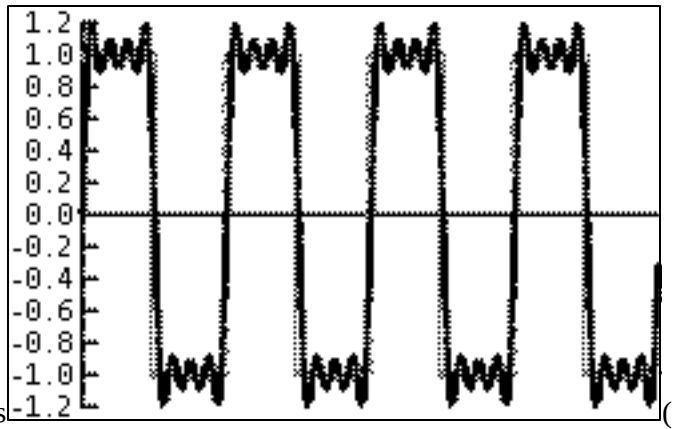
(first and third harmonic)

Figure 7: Approximating a square wave with 2 harmonics (1st and 3rd).

With three terms 

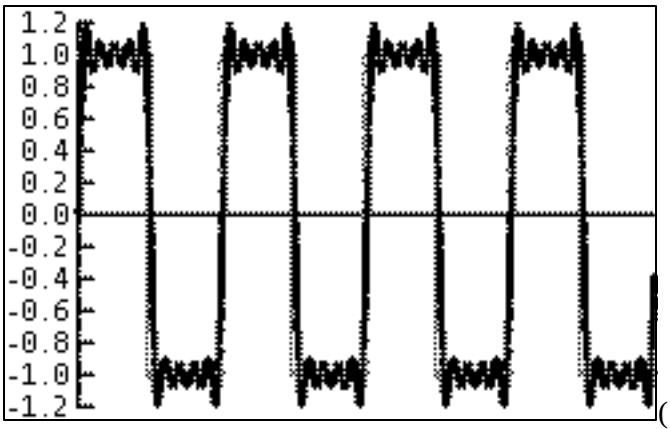
(1^{st}, 3^{rd}, and 5^{th} harmonic)

Figure 8: Approximating a square wave with three (1st, 3rd and 5th) harmonics.

With four terms 

(1st, 3rd, 5th, and 7th harmonics)

Figure 9: Approximating a square wave with four (1st, 3rd, 5th and 7th harmonics).

With five terms 

(1st, 3rd, 5th, 7th, and 9th harmonics).

Figure 10: Approximating a square wave with five (1st, 3rd, 5th, 7th and 9th) harmonics.

When we say that a signal has high frequencies, we are basically saying that its Fourier representation requires high frequencies to approximate the signal well.

Examining the approximations to a square wave gives us insight into what the presence of high frequencies do in the signal.

1. Notice that the approximations do not have the sharp "edge" that the square wave does. As we add more and more terms, the approximation's "edge" gets sharper and sharper.
2. Notice that the approximations overshoot the square wave. This is called the Gibbs phenomenon.
3. Notice that the approximations bounce up and down more than the square wave does. Paradoxically, you may see more high frequency "jiggles" called ringing in the band limited approximation than in the square wave (which contains infinitely high frequencies).

The first observation is particularly important: if we want to represent a signal with very sharp changes, we need to include high frequencies. Because sinusoids are smooth, to make a signal that is non-smooth, we need to add in sinusoids that create the sharp edges because they change very fast (e.g. have high

frequencies). This is one of the most important intuitions to gain: it is the sharp changes (or non-smoothness) of a signal that causes it to have high frequencies.

The existence of a high frequency tells us that a signal has sharp changes. It doesn't necessarily tell us when those sharp changes happen. In fact, for the square wave example, if we shift the square wave, the frequency content doesn't even change (the phase does). This is the power of frequency analysis: it allows us to talk about what happens in a signal, without necessarily talking about when it happens.

To relate these ideas motion we consider some examples. If we have a motion with abrupt changes, such as the snap of a karate kick or the impact of a chef's knife during chopping, these motions are signals with high frequencies. A smooth motion, such as a graceful dive or pirouette, would not have high frequencies. If we tried to represent the karate kick in a way that didn't take its high frequencies into account or damaged its high frequencies, we would probably lose its crisp snap, and might find other problems like ringing and overshoot as well.

The power of frequency analysis is that it gives us a way to discuss what happens in a signal, without much discussion of when it happens. For example, we can say that a signal does make fast changes (e.g. has high frequencies), without saying when these occur. In contrast, time domain analysis is very good at saying when things happen, but not very good at describing what kinds of things happen. In analysis terms, we might say that Fourier analysis provides no time localization, while Time Domain analysis provides poor frequency localization. Other analysis methods, which are basically created by defining a different set of building blocks, provide control over these tradeoffs. Examples of tools that mix time and frequency localization are Wavelets and Gabor Transforms.

Discrete Time Signals

To this point, we have considered signals that are continuous in time. Usually, we only know that value of a signal at a finite set of specific times. Such signals are called discrete-time signals. Often, a discrete time signal is created when we try to represent a continuous, physical process on a computer (which can only measure the original signal at specific instants). The conversion from a continuous time signal to a discrete time signal is called sampling. The inverse process is called reconstruction. For example, with motion capture, our initial signals (the positions and joint angles of the performer) are continuous-time. The capture device samples these signals to create a discrete-time representation.

The idea of sampling is that we can only know that value at certain times. For this discussion, we consider the case of uniform sampling when we take samples periodically at uniform increments. The most common source of sampled signals in animation is motion capture. Keyframed animation is not actually a sampled representation because we typically construct a continuous curve (such as a spline) through the keys.

The obvious problem with sampling is that without other information, we have no idea what happens in between the samples. Many possible signals all look the same when sampled.

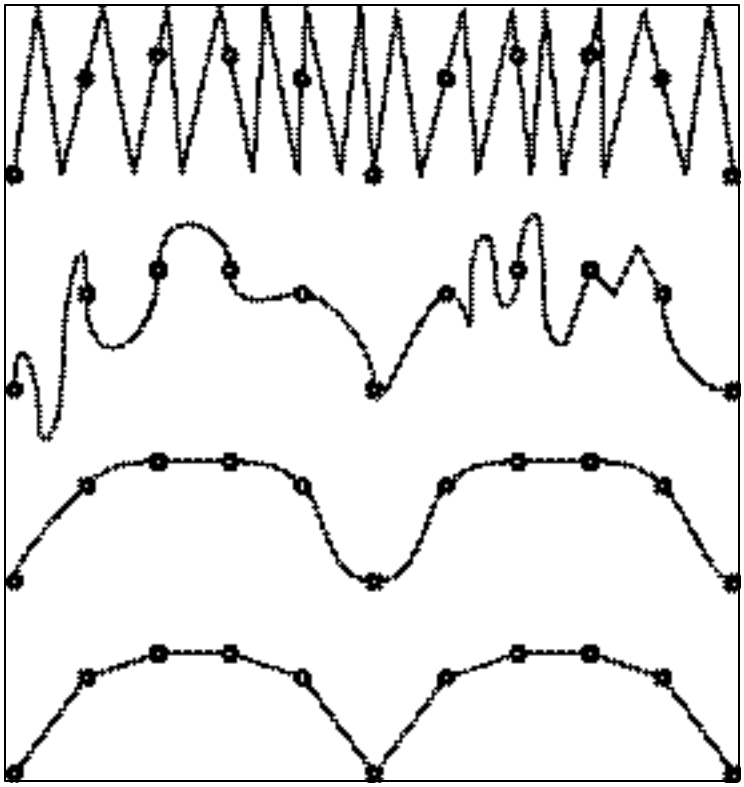


Figure 11: Several different signals possible signals that could have lead to the same set of samples.

Suppose the signal "turns around" between two samples. We really have no way to know if the signal turned around once, twice, three times, or not at all.

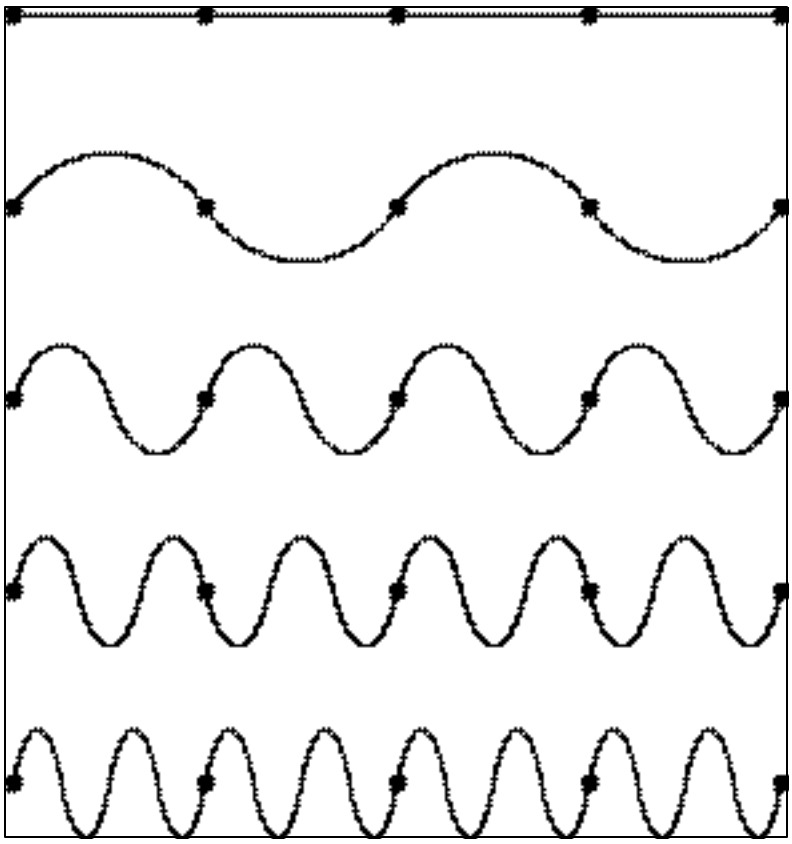


Figure 12: Several different sinusoids that all lead to the same set of samples. Each has a period that is a multiple of the sampling rate.

For the signal to turn around once, its frequency must be half of the sampling frequency (or its period must be twice the sampling period). This looks exactly as if the signal didn't change at all. If the signal was of a lower frequency than half the sampling frequency, it would not have been able to turn around fast enough between samples. This indicates a fundamental limit of sampling called the Nyquist limit: when sampling, we can only properly handle signals that are composed of frequencies less than half of the sampling frequencies. If a signal has a component that has a frequency that is half the sampling rate or higher, it will appear the same as some lower frequency signal. This phenomenon is called aliasing.

Aliasing is a problem because there is no way to tell in the sampled signal if it is aliased or not. When we see something in a signal, it might have actually have been something else. For example, if we see a constant signal, it could be a periodic signal that has a period that is half, or twice, or seven times the sampling frequency.

This is true for any low frequency that we see. Anything that we think we see in the sampled signal might have really been caused by a high frequency that is aliasing as a lower frequency.

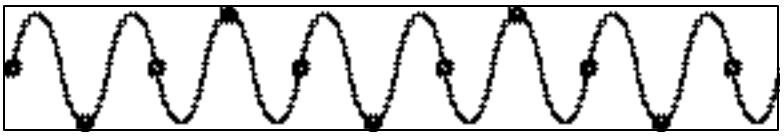


Figure 13: Sampling a signal below the Nyquist rate aliases the signal to a lower frequency.

In this example, when we sample a signal at 3/4ths of its frequency, we get an aliased signal that appears as if its 1/2 of the frequency of the original. By adjusting the ratio, we can get any low frequency to appear.

Unfortunately, once we have sampled a signal, there is little we can do about aliasing. When we see a low frequency (even a constant value) we have no way to know if the original signal had that frequency, or if what we are seeing is an alias of a frequency above the sampling rate.

On the other hand, the Nyquist limit gives us a method for preventing aliasing. If we know that the signal that we are sampling does not contain any frequencies as high as the Nyquist rate, then we know that aliasing is not occurring. Remember, before we said that without other information, we have no idea what happens in between the samples. If we know that the signal we are sampling obeys the Nyquist limit, this additional knowledge lets us be sure that the signal doesn't do much between samples.

To enforce the Nyquist limit, we must pre-filter the signal before sampling. That is, we must process the signal to remove frequencies higher than the Nyquist limit.

The Nyquist limit creates a tight connection between frequencies and sampling. If we want to capture signals with high frequencies, we must use a higher sampling rate.

Reconstruction

The opposite of sampling is reconstruction: the process of trying to figure out what the original signal was that created a set of samples. The issues are much the same as in sampling: we simply do not know what the signal did in between samples, without any additional information. The aim of resampling is to create one of the signals that could have been sampled to create the data. Ideally, we would choose a signal that most closely resembled the original signal, but without the original signal, what we can do is limited. Typically we use additional information about the signal or the sampling process to make a better guess at what the original signal was. One particularly useful piece of information is that the signal was properly sampled (that is, that the sampling rate is above the Nyquist limit for the original signal).

A simplistic view of reconstruction is that it is meant to answer questions of the form "what is the value of the signal at time t," where t is not the time of one of the samples. Effectively, we are connecting the dots. This process is also sometimes called interpolation.

The simplest way to connect the dots (or samples) is with straight lines. This is called linear interpolation. It is very easy and efficient to implement, but has the problem that the results it creates are not smooth, therefore can create high frequencies in the resulting reconstruction. Other types of interpolation have different smoothness properties.

If we know that a signal was sampled correctly, sampling theory tells us that we can reconstruct the original signal exactly from the samples. However, this "ideal" reconstruction requires a reconstruction process that is impossible to implement using time-domain operations. Ideal reconstruction is mathematically simple and elegant, but difficult to achieve in practice.

Frequency Domain Operators

To this point, we have considered the frequency domain as an analytical tool to help us understand signals. Now, we consider how to make operations that effect the frequency content of signals. Pre-filtering gives us an example of such an operation: we want to make sure that a signal has no frequency content above a certain frequency. This operation is called low-pass filtering because it allows the low frequencies to pass through the filter.

Frequency domain analysis tells us what kinds of things happen in a signals, without being specific about when these kinds of things happen. Frequency domain operators allow us to control what kinds of things happen in a signal. For example, a low pass filter would eliminate all high frequencies in the signal. This would require that the resulting signal have no sharp edges anywhere in its domain. This is clearly a problem: since such the filtering operation would have to affect the entire signal. In fact, changing even one

coefficient of the Fourier representation of a signal would require changing the entire time domain representation.

Just as a signal has a representation in both the frequency and time domain, an operation on a signal has corresponding meanings in both domains. Some operations are easy in one domain, but not the other. For example, to change the value of a signal at a specific time is easy in the time domain, but difficult in the frequency domain. Similarly, getting rid of high frequencies is easy in the frequency domain, but difficult in the time domain. One approach to dealing with this is to transform the operation.

To motivate how frequency operators work in the time domain, we begin by considering what we want to have happen in the time domain on discrete signals, and then relate this backwards to the signal theory. Since high frequencies correspond to rapid changes in the time domain, to reduce high frequencies, we might reduce the amount of rapid changes in the signal. We could do this by taking each point on the signal and changing it so it was closer to its neighboring points, effectively averaging each sample of the signal with the samples before and after. For example, if we used a uniform weighted average with the samples before and after we might get

Out[t] = 1/3 (In[t-1] + In[t] + In[t+1]).

Performing this running average will smooth out the input signal, effectively decreasing high frequencies.

For the simple example, we used an average in which each element was weighted identically. We could choose different weighting. Similarly, in the simple example we only used 3 samples to determine each one, we might use more. Another example might be

Out[t] = 1/10 In[t-2] + 1/5 In[t-1] + 2/5 In[t] + 1/5 In[t+1] + 1/10 In[t+1].

The effects of the weighting averaging process, or the filter that it implements, depends on the choice of what the weightings for the averaging are. We can describe the different averaging by giving the amounts of each scaling. In our examples, they would be [1/3, 1/3, 1/3] and [1/10, 1/5, 2/5, 1/5, 1/10]. This description is sometimes called a filter kernel or impulse response. A filter kernel is actual a signal itself. In these cases, the signal is zero at all times except for the 3 or 5 times that are specified.

The running weighted average process that we used to apply the kernel signal to the input signal is the discrete version of an operator called convolution. Convolution is an operation that takes two signals and computes a new signal that slides one signal along the other, and at each instant adds up the products of the signals. In summation notation, we would write this as

$$Out(t) = \sum_{i=-w}^{w} k_i In(t+i)$$

where k is the filter kernel ([1/10, 1/5, 2/5, 1/5, 1/10] is the example) and w is the width of the kernel (2 in the example). Notice that in order to compute the output at a given time we must look both forward and backwards in time. In signal processing terms, this means that the filter is not causal. A causal filter would look only at the current and previous times. We can change the filters given to a causal filter by introducing a delay.

The convolution process is easier to show in moving pictures than in words or diagrams (which makes it appropriate for animation). However, since this is a book, we have to use static images:

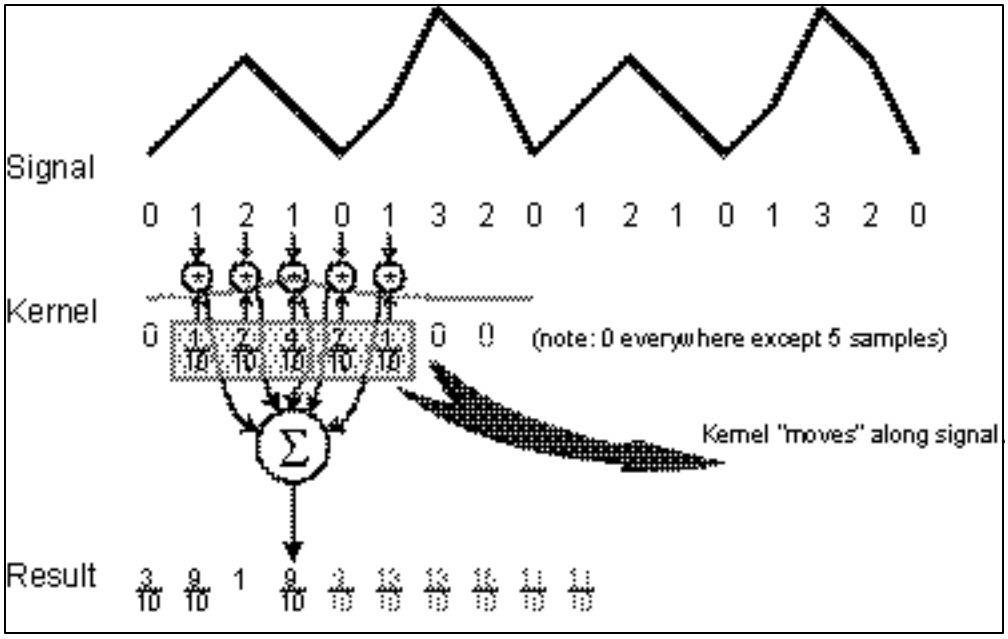


Figure 14: A visual display of convolution. Any sample in the result is computed as a weighted sum of the samples of the original signal.

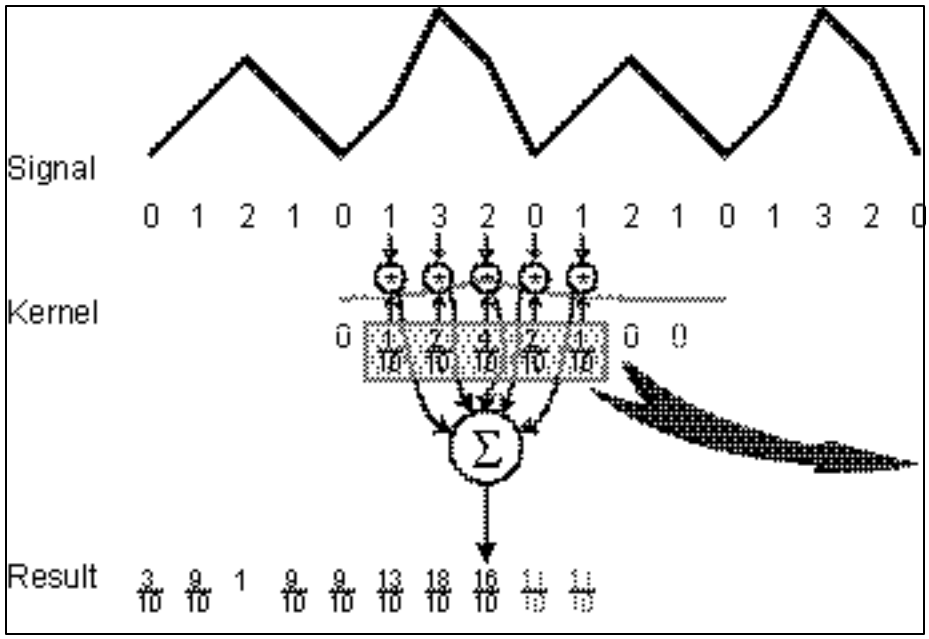


Figure 15: Later in the same convolution process. The kernel is shifted to produce each sample in the result.

The continuous version of the convolution operator computes an integral, rather than the sum, but the idea is the same. The connection between filtering and the running average process described above comes from the fact that the Fourier Transform of the convolution is multiplication. Inversely, the multiplication of two signals in the frequency domain is the convolution of those two signals in the time domain.

So we can now see how to implement a low pass filter (or other frequency space operation). An ideal low pass filter would multiply all of the low frequencies by 1, and all of the high frequencies by 0. We can express this as the multiplication of the original signal to be filtered by a special signal (a filter kernel) who

has frequency terms of 1s and 0s in the appropriate place. This is sometimes called a box or boxcar filter because a graph of its response (graphed in the frequency domain) looks like a square box. (Remember, we are only showing the positive side of the frequency graph.)

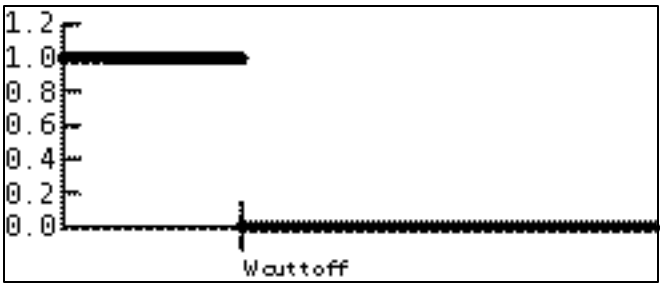


Figure 16: Frequency response of an ideal (or boxcar) filter.

The multiplication of the filter kernel and the original signal would need to happen in the frequency domain. One way to implement this would be to transform our input signal to the frequency domain, perform the multiplication with the filter signal, and then transform the result back to the time domain.

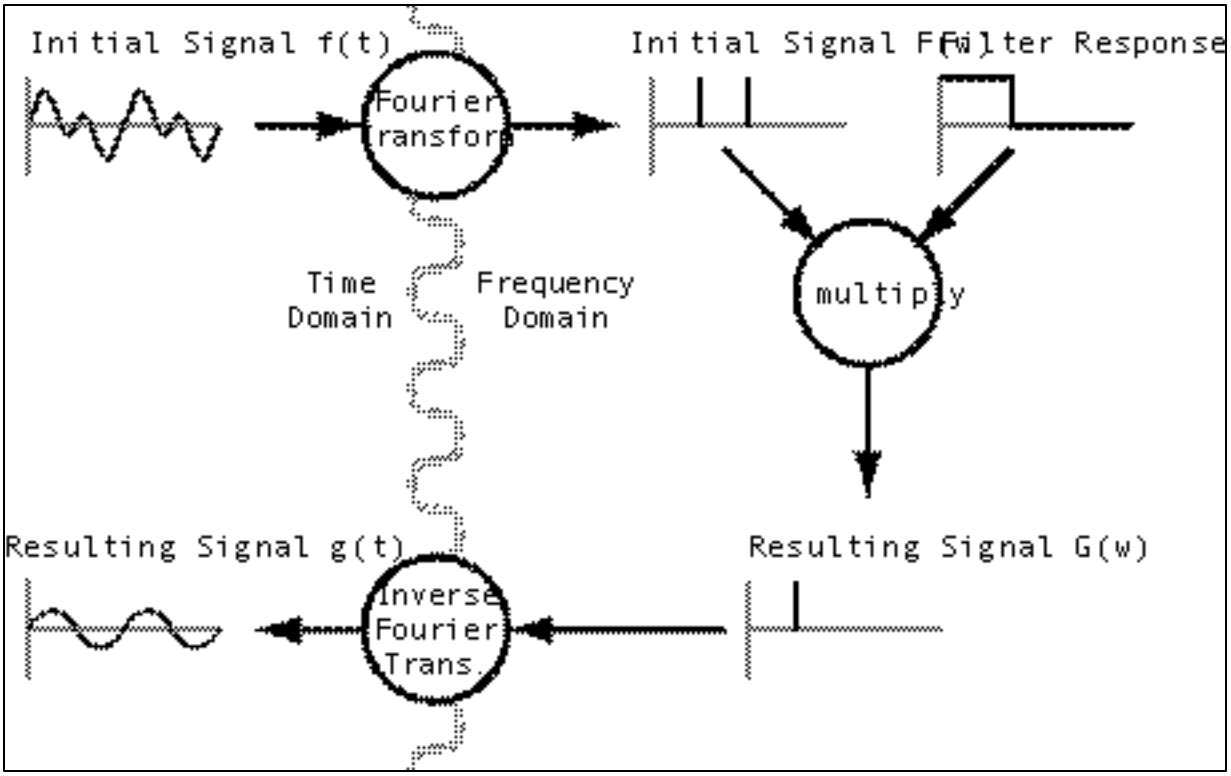


Figure 17: Schematic diagram of the process of performing a low-pass filtering of a signal.

Alternatively, we could transform the filter signal to the time domain and convolve this with the input signal.

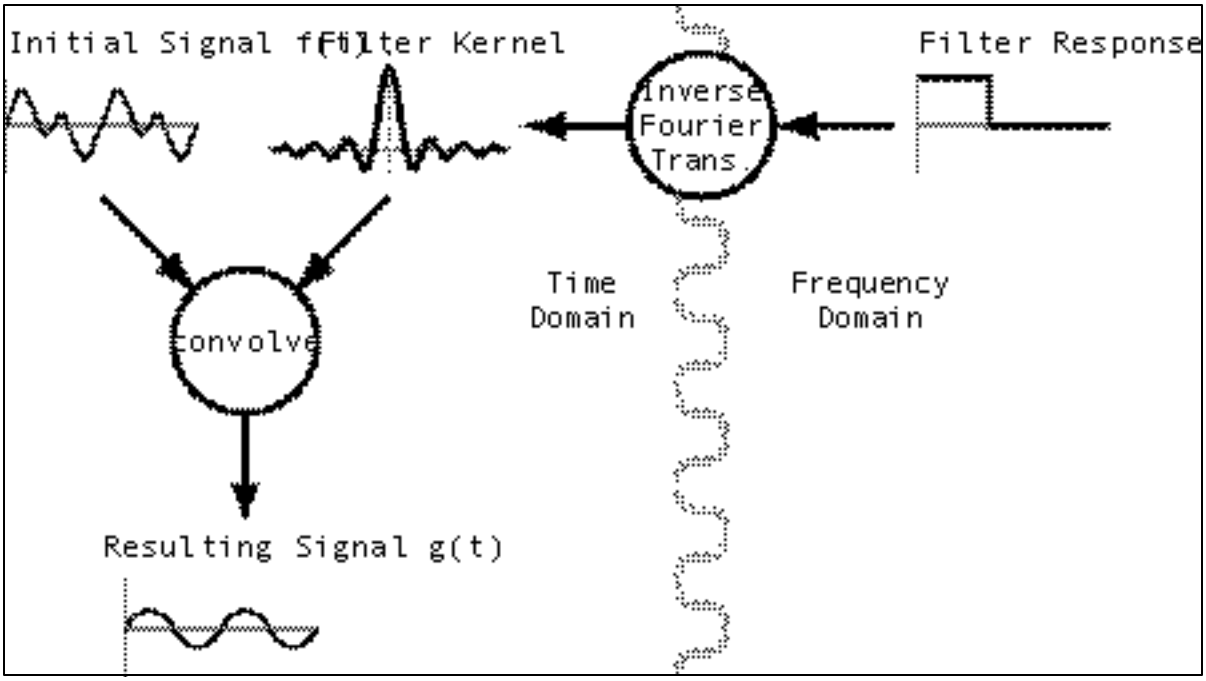


Figure 18: Schematic diagram of a more typical process for performing a low pass filtering operation.

This has the advantage of avoiding having to Fourier Transform the signal. Not only does this save performing a Fourier Transform, but it means that we can create the filter once and apply it to any signals we create.

The process that we described tells us how to determine what the filter kernels should be. The kernel is the inverse Fourier transform of the filter signal. Ideally, if we know what the frequency response we want is, we use that to create a filter signal that can be converted into a kernel. In practice, the task of creating filters is not so easy. Just as simple signals in the time domain (such as the square wave) do not have compact representations in the frequency domain, the converse is also true. The low pass signals do not have simple versions in the time domain. An ideal low-pass filter (e.g. one that cuts off all frequencies above a certain point), is just like a square wave. It has a time representation that extends infinitely. This is clearly impossible to implement a convolution with!

The Inverse Fourier Transform of the boxcar filter is the sinc function, sin(x)/x. We mention this not because it leads to practical filters for motion analysis, but because examining it gives us insights onto what other low pass filters would look like.

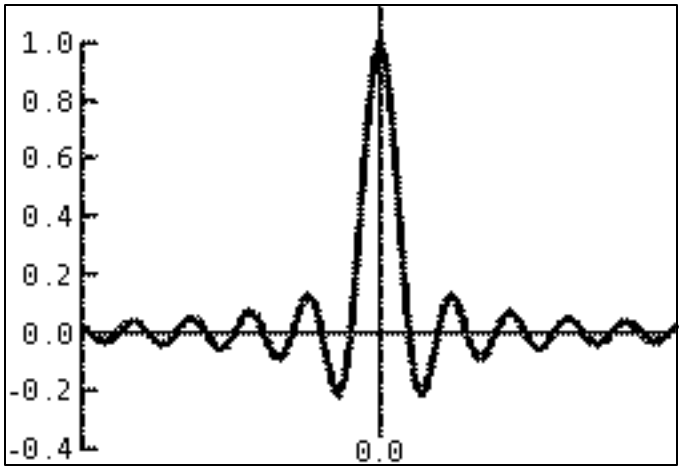


Figure 19: Plot of the sinc function, the filter kernel for an ideal low pass filter.

Notice that the sinc function's predominant feature is a large hump centered around zero. It also contains a number of smaller bumps, of decreasing size. These bumps continue indefinitely, but get smaller and smaller. We add that the width of the bumps (or the scaling of the x axis, depending on how you look at it), depends on the frequency limit (e.g. how wide the box that we transformed was).

A filter that has a finite sized kernel is called a finite impulse response (or FIR) filter. An FIR filter can only approximate an ideal low pass filter, because as we saw, the impulse response of an ideal filter is not finite. There is a large literature on how to best design these approximations and what the tradeoffs are in the use of FIR filters. Other issues are introduced because we must sample and quantize the filter.

Much of the complexity in designing filters comes from the fact that FIR filters do more than attenuate different frequencies. An FIR filter is also capable of delaying or shifting a signal. In fact, FIR filters tend to shift different frequencies by different amounts which tends to cause unwanted distortions.

In practice, when operating on many problems (including motions), true low pass filters are not even desirable. The example of approximating a square wave with a band-limited version (refer to FIGURE in Section "Approximations with Fourier Series") demonstrates some undesirable effects, such as the overshoot lobes and ringing at discontinuities. Often, we pick kernels that may not correspond to ideal low pass filters, but do not exhibit some of these effects.

The uniform average of a number of samples (as in our first example), is one common approximate low pass filter, often referred to as a box filter because its waveform has a square shape. The simplest box, sometimes called the unit box, consists of two samples of equal magnitude. Repeated application of the simple box filter gives a class of filters called Spline or Binomial filters. Because convolution is associative, we can create a family of Spline filters by convolving the unit box with other spline filters, and then applying one of these filters to our signal, rather than applying the unit box many times. The first few members of the family of Binomial filters are 1/2 * [1 1], 1/4 * [1 2 1], 1/8 * [1 3 3 1], 1/16 * [1 4 6 4 1], and 1/32 * [1 5 10 10 5 1]. Notice that each filter kernel's elements sum to one so that the filter does not attenuate a constant value.

Another important filter is given by the Gaussian function,

$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}.$$

This function has a number of important mathematical properties, including the fact that it is its own Fourier Transform. In practice, the Binomial filters are often used as an approximation to the Gaussian filters, except

in cases where the continuous filter parameter is needed (the binomial filters are only defined at constant intervals).

Understanding Convolution and Filtering

FIR filters and the discrete convolution that implements them are an important concept for operating on signals. By looking closely at some simple examples, we can better understand how they work, and see some of the tricks in their use.

The simplest FIR filter has a single non-zero value of 1. If this one value is at time zero, then the filter is the identity: the output of applying the filter is exactly what is input to the filter. If the output is not at time zero, the filter has the effect of shifting the timing of the signal. For example, if the non-zero element is at time 4, the effect of the filter would be to delay the signal 4 units of time (where each "unit" is the sampling interval). If the value of that non-zero element was other than one, the output signal would

This trivial example shows the 2 basic building blocks of FIR filters: the signal can be shifted (delayed) and scaled. In essence, we can think of an FIR filter as adding together several scaled and delayed copies of the input signal.

For a more interesting example, consider the common approximate low-pass filter which has coefficients [1/4, 1/2, 1/4]. This is the second of the Binomial or Spline filters.

Were the filter not zero centered, it would have the same effects as this filter, except that it would delay the output by a unit of time (which may be an undesirable effect). We can think of this filter as adding together three copies of the input signal, or as implementing the function

$$\text{Filter}(f(t)) = F * f = 1/4\, f(t-1) + 1/2\, f(t) + 1/4\, f(t+1).$$

To see what this filter does, we can try applying it to some example signals. For example, we could try applying it to a constant signal. Since the constant signal has no frequency content, we would expect it to be unaltered (which it is). Similarly, if we apply the filter to a signal that is a sine wave of frequency much lower than the sampling frequency, we would expect the filter to have little effect.

When the filter is applied to a square wave, it has more of an effect. Even the frequency of the square wave may be low, the square wave has high-frequency components (sharp changes) that are removed by the filter. This has the effect of making the result smoother, however, the low frequency content of the signal (the basic period of the square wave) is relatively unaltered.

If our input signal has a beginning and/or end, we will have a problem that this filter will "go off the end." This is related to the fact that the basic frequency concepts are defined for periodic signals. Depending on how we handle the samples "off the ends" will effect our result. For example, consider a signal that is a square wave from time 0 to 32. Some choices in handling "undefined times:"

- Assume out of bounds values are 0.
- Assume out of bounds values are the same as the last value.
- Copy the signal (repeat it).
- Reflect the signal about its endpoints.

The last two choices have the important property that the "added" signal has the same frequency characteristics as the signal does.

Part of what makes designing filters difficult is that filters potentially delay signals, as well as attenuate them. Filters can attenuate different frequencies in different ways, however, they also delay different frequencies differently. This can lead to distortions.

Filtering and Noise

One common problem with signals is that they become infected with noise. That is, we start out with a good signal that contains the information that we want to have, and through some process, this signal gets mixed up with an unwanted signal. We call this unwanted signal noise. Two examples of noise are is the interference that causes static when we transmit an audio signal by radio and the measurement errors when we perform motion capture. Often, our goal is to recover the original wanted signal given an infected signal. This process requires us to take a signal and effectively divide it into two pieces: the original signal and the noise.

If we knew exactly what the noise was, the problem of removing it would be easy, we could simply subtract it to recover our original signal. Unfortunately, this is rarely the case. Typically, noise is caused by some random process whose effects we cannot predict. The basic idea behind noise reduction is to try to characterize both our desired signals and the expected noise so that we can try to guess at what parts of a signal are likely or unlikely to be one of the two components. For example, suppose we know that in our original that the value never goes above 0 (for example, that the signal represents the angle of a knee joint in a motion capture session). If we ever encounter a value above 0 at any time, we know that there must be a contribution of noise at the instant. Unfortunately, this simple time domain example points to some of the difficulties in noise reduction: while it tells us that noise exists, it tells us little about what to do about it.

We often can make similar criteria on signals and noise in the frequency domain. We often know that a signal is band limited, or nearly band limited. For example, audio signals rarely contain significant amounts of frequencies above the threshold of human hearing. Therefore, content in a signal that is above this band is likely to be noise, not an important part of the audio information. So a strategy for doing noise reduction would be to remove the high frequencies from a received signal, as they are likely to be noise. The peril in this is that the original signal may have some high frequencies; if there is a low frequency square wave, for instance, there will be high frequencies present in the signal, and removing them will damage the integrity of the original signal. Also, there is no guarantee that the noise is exclusively high frequencies!

Multi-Resolution and Scale

Scale is a signal theory concept that is used by the computer vision community, but is not part of standard signal processing terminology. It is an extremely useful concept for thinking about motion, so we introduce the basic idea here.

When we look at something, what we see depends on how closely we look. For example, when we look at the ocean, if we look under a microscope we would see a completely different picture than if we looked from a satellite. Depending on how much of the thing we are looking at, the amount of detail we see changes. The satellite is unlikely to see microscopic organisms, and the microscope is unlikely to identify the direction of trans-oceanic currents. In each case, we are looking at the same object, we're just looking at a differently sized piece. The scale of the features that we are looking at is different.

The concept of scale is directly related to frequency content. If we are looking for fine details, we need to see the high frequencies. As we start to look at bigger and bigger pictures, we need to be able to ignore these fine details. To remove them, we filter out the high frequencies. A scale is therefore equivalent to a frequency limit: the lower the frequency limit, the larger the scale of features that we are looking at.

As an example, consider the following signal:

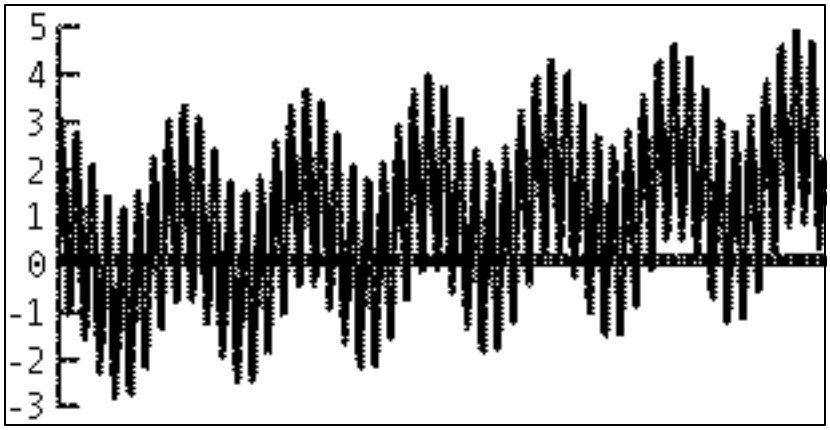


Figure 20: A signal with parts at different scales.

If we view this signal at a coarser scale (by setting a frequency limit), we get a very different view of what is going on:

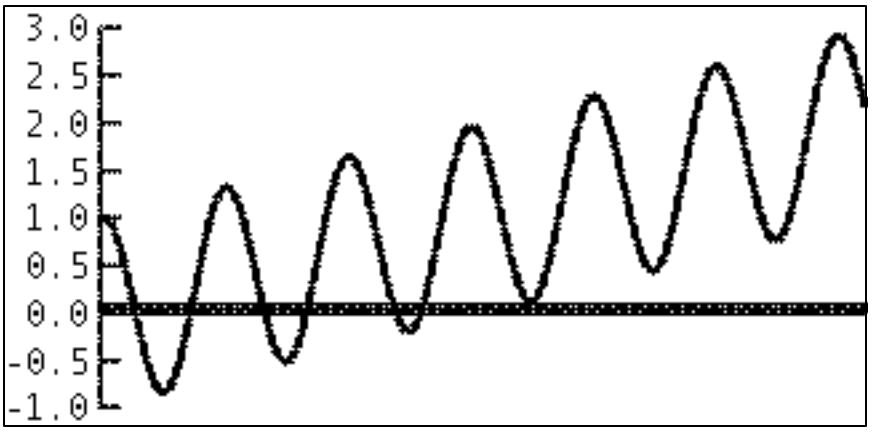


Figure 21: Filtering the signal of the previous example shows content at a different scale.

This second signal was created by filtering the first with a low-pass filter. If we look at an even coarser scale (by lowering the frequency limit) we get a still different view:

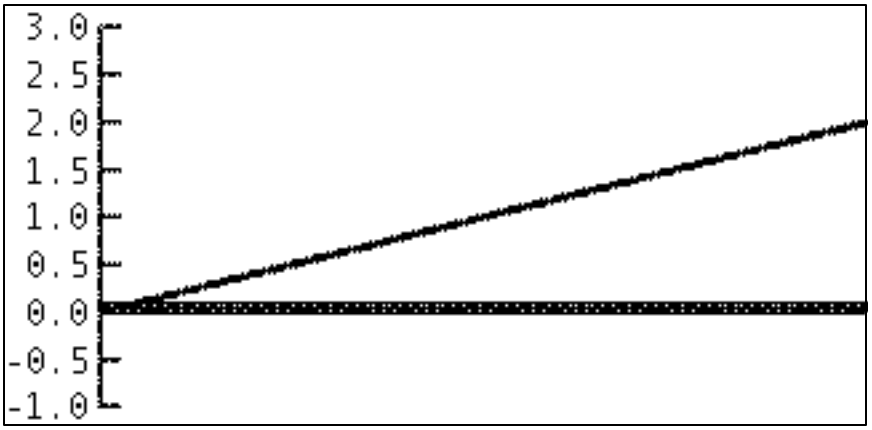


Figure 22: Further filtering of the example signal gives a view at a different scale.

To use a motion example, the first signal might be what we see when we look at the output of a motion capture system that creates a lot of high-frequency noise. Changing our view by looking at a different scale, we see a definite periodic motion (perhaps someone walking). At an even larger scale, we see that it is a person walking up a hill. In this case, multi-resolution or multi-frequency analysis has served to break the signal into component parts where each part has a distinct and different meaning. This actually turns out to be a common occurrence: signals often are created by mixing a set of distinct processes. It isn't always the case that they are distinct in frequency, as this contrived example was.

There are other methodologies of signal processing that are specifically suited to multi-resolution analysis. For example, a Wavelet is a signal representation that explicitly codes multi-resolution information. It can be thought of as another view of a signal, the same way that time and frequency domains serve as different signal representations.

Interpolation and Time Scaling

In this section we consider a specific, useful operation on a signal and look how the theory can be applied. We consider the problem of scaling the time that a signal takes. This requires us to perform a resampling operation.

To begin, let's consider the of a signal (call it f) that we would like to dialate (expand in time) by a factor of 2 (call the resulting signal g). This means that

$g(t) = f(t/2)$

We must consider the problem that these signals are uniformly sampled (at integer values of t). The issue arises that for some samples that we would like to have of g (namely the odd integers) do not correspond to a sample of f. As we know from our discussions of sampling theory, there is no way to know what the signal does in-between samples.

Theoretically, what we would like to do is construct a continuous representation for f, and then sample that. If we knew that the signal was sampled properly (e.g. the original signal had no frequencies higher than 1/2, which is the Nyquist rate for this sampling period), then we could do an "ideal" reconstruction.

The theoretical process is a good hint at what the "right" answer is: we should create g such that it has no newer high frequencies. Of course, that answer is only right if the original signal was properly sampled. In practice, the "right" answer may be a matter of artistic taste.

To look at a specific example, let's consider a simple f that is a triangle wave with sampled values [0 2 4 6 4 2 0 2 4 6 4 2 0]. This gives us a picture like:

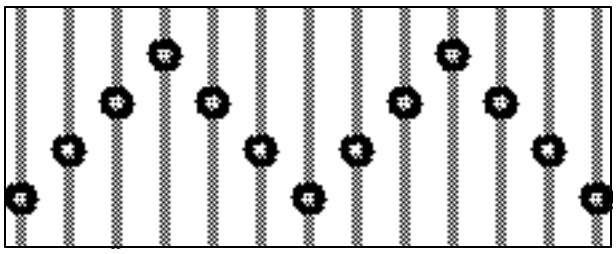


Figure 23: A simple triangle signal.

What we'd like to do is double the time, which means that we know the even samples.

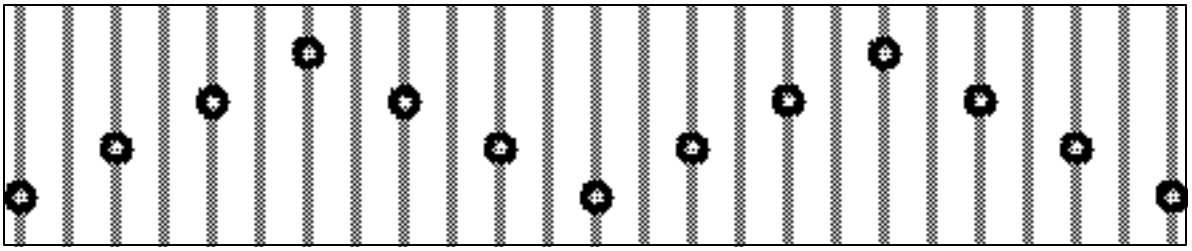


Figure 24: The triangle signal expanded in time.

So we need to know what happens "in between" these samples. If we had the original signal that f was sampled from, we could sample the signal at all of the desired points.

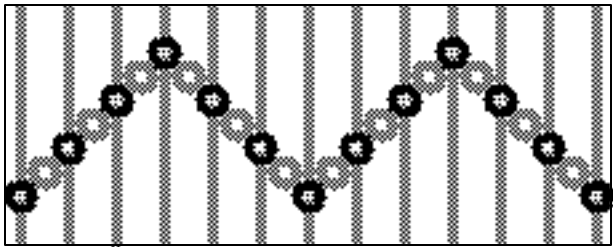


Figure 25: Knowing what the signal was (or reconstructing it) would allow for finding samples in between the original samples.

Which of course would have required us to have the original signal (or to at least reconstruct it).

With the numbers, there are two obvious choices:

- Simply double each sample (e.g. repeat it twice). So that our original signal will become [0 0 2 2 4 4 6 6 4 4 2 2 0 0 2 2 4 4 6 6 4 4 2 2 0 0].
- Make the new samples half way between the previous samples [0 1 2 3 4 5 6 5 4 3 2 1 0 1 2 3 4 5 6 5 4 3 2 1 0]. As it turns out, this is exactly the right answer for the picture drawn above. However, this is a fortunate coincident.

The first version we call value replication. The second is interpolation.

Value replication is sometimes called "nearest neighbor" because it picks the sample closest to the value we're looking for. In the case of doubling the signal size, we chose to pick the lower value in the case of a tie (e.g. when we look for a value for time = 1.5, we pick sample 1). For an example where this makes a difference consider tripling the time. In this case, we would want to look for time values of 1 1/3 and 1 2/3, which are closest to 1 and 2 respectively.

The halfway method is interpolation. More generally, we blend the nearby samples. The simple way to do this is to draw a line between the two samples and pick the value along the line. This is a simple form of reconstruction. In equation form, we might say

$$f(t) = s * f(\text{floor}(t)) + (1-s) * f(\text{ceil}(t))$$

where floor is the function that picks the largest integer smaller than t, ceil is the "ceiling" function that picks the smallest integer larger than t, and s is t-floor(t), or the distance between t and the sample. This is linear interpolation since we are fitting straight lines between the samples.

Now you might wonder which of the two methods described so far is "right". The answer is "it depends." Suppose we have the triangle wave, and we triple the time (rather than just doubling it). We get two very different looking results:

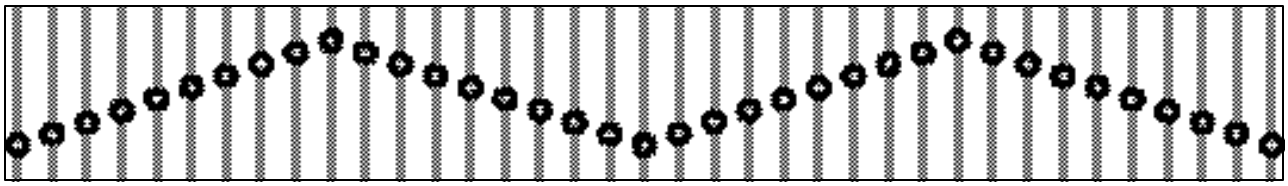


Figure 26: Performing linear interpolation on the sample triangle wave.

for linear interpolation, and

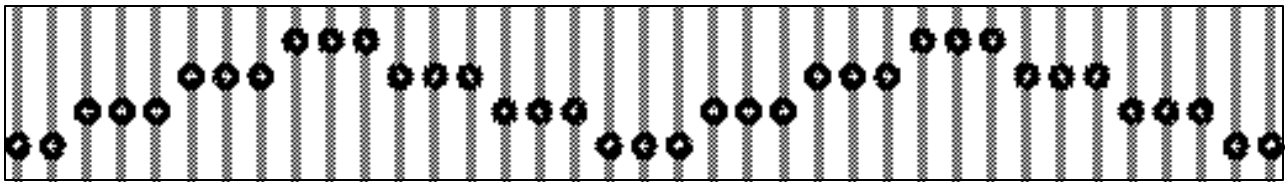


Figure 27: Perfoming nearest neighbor interpolation on the example triangle signal.

for nearest neighbor. Since in this example, we knew that we wanted a triangle wave, one is clearly better than the other. However, if we just had the sampled signal, it might really have been the "jaggy" stairstep. For example, suppose we have a square wave [0 0 1 1 0 0 1 1 0 0 1 1]. In this case, using linear gives an overly smooth result, while nearest neighbor gives a square wave. Nearest neighbor intepolation gives us a square wave

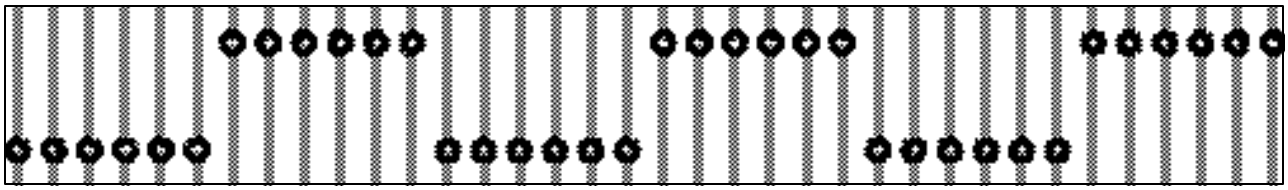


Figure 28: Dilating the sample square wave with nearest neighbor interpolation.

while linear interpolation gives us something that might be too smooth.

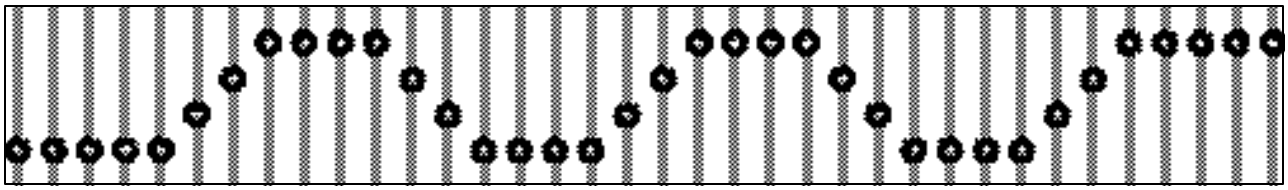


Figure 29: Dialating the example square wave using linear interpolation

Bi-Cubic interpolation achieves results between these two choices. It estimates how sharp and edge there should be by estimating the derivatives at each sample and then fitting a cubic curve between the samples.

Reconstruction Kernels in 1D

Let's consider a slightly non-intuitive way to implement these methods.

If we take our original samples and "space them out," we get a chain of spikes. Going back to our original example triangle wave [0 2 4 6 4 2 0 2 4 6 2 0], we would just put in zeros in the length doubled version, e.g. [0 0 2 0 4 0 6 0 4 0 2 0 0 0 2 0 4 0 6 0 4 0 2 0]. If we filter this "spike chain" we then get the reconstruction processes described above. By choosing the correct filters, we can different types of reconstruction. For example, nearest neighbor interpolation for size doubling can be implemented by the reconstruction kernel [0 1 1]. The linear interpolation can be implemented by the kernel [.5 1 .5].

For other spacings, we just use other kernels. For example, the nearest neighbor kernel for tripling is [1 1 1], and the linear interpolation kernel is 1/3 [1 2 3 2 1]. Other kernels give different reconstructions. For example, we might use the kernel 1/6 [1 5 6 5 1].

This implementation has several advantages. One, it gives a uniform way to implement lots of different interpolation types. By choosing the reconstruction kernels, we can get different types of results. Even the bicubic interpolation described above can be implemented this way. Second, it corresponds more closely with the theory of signal reconstruction, which makes design of the reconstruction kernels possible. Third, with a uniform method for kernel design, it is easier to extend this method to different scaling sizes and to 2D.