AMAT 415: Basic Signal Processing

Michael P. Lamoureux

April 10, 2012

Contents

1 Introduction

These notes follow the content of the course AMAT 415 at the University of Calgary. They focus on Digital Signal Processing and summarize some of the key concepts. For a more in-depth analysis, please read the course textbook (which this year is Lyon's "Understanding Digital Signal Processing.")

2 Signals and systems

In signal processing, we are mainly interested in *signals* and *systems.* A typical signal might be a sound, a bit of music, a photographic image, a seismic vibration, or just about any physical phenomena that can be measured over time and/or space. Usually, we represent a signal by a function $f(t)$, where t is time, and $f(t)$ is the value measured at time t. For an image, we would have a function of two variables $f(x, y)$, where f represents the intensity of the image at position x, y in the plane, say.

A *system* takes one signal in, and outputs another signal. It could be a physical system: an earthquake in India starts a signal (vibrations) on one side of the earth, the earth transmits the vibrations to the other side (the system), and a new signal is felt in Calgary (the received vibrations). It could be an electrical system: a sound is picked up by a microphone (the input signal), the signal is passed to a stereo amplifier (the system), and the resulting amplified signal is output to the speakers (the output signal). It could be a computational system: a string of numbers is input to a computer, the computer churns away on the numbers (adding, subtracting, multiplying, etc – the system), and a string of numbers is output by the computer. It could be a combination of such systems: a digital camera captures a real image through its lens, the intensities are converted to a function $f(x, y)$, and then the computer mucks around with the values of the function to compute a sharper image, represented by a new function $g(x, y)$. Here, the input is the real image, the system is the camera/computer, the output is the function $g(x, y)$.

The point is: signals are function, and systems operate on signals.

3 Sampling

The signal $f(t)$ is a function on the real line $\mathbb{R}$. When we compute with a computer, usually we can't know everything about the function, or store it all on the computer. So we just evaluate the signal at a sequence of times t_n, and define a vector $\mathbf{x} = (\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots)$ with components

$$x_n = f(t_n), \qquad \text{for all } n \in \mathbb{Z}. \tag{1}$$

For reasons that have mostly to do with engineering technology, we usually take the time samples t_n to be uniformly spaced. That is, we have a sequence of numbers separated by a uniform step Δt, and write $t_0 = 0, t_1 = \Delta t, t_2 = 2\Delta_t, \ldots$, and so the vector $\mathbf{x}$ has components

$$x_n = f(n\Delta t), \qquad \text{for all } n \in \mathbb{Z}. \tag{2}$$

Although the vector $\mathbf{x}$ is infinitely long, that is it has infinitely many components, it is important to realize that most of the component x_n are just zero. Why? Because we can't measure back to time minus infinity, or forward to plus infinity. So at some point we sop measuring, and can just assume everything else is zero.

The vector $\mathbf{x}$ is called a sampled signal. Δt is called the sampling interval. $1/\Delta t$ is called the sampling rate.

4 Aliasing

The problem with sampling is that you lose information in the process. Two different functions $f(t)$ and $g(t)$ might get sampled and produce the same vector $\mathbf{x}$. For instance, suppose f is a sine wave, $f(t) = \sin(t)$ and g is the zero function, $g(t) = 0$. These are two very different signals. But, with the sampling interval $\Delta t = \pi$, we see that

$$\begin{aligned} x_n &= f(n\pi) = \sin(n\pi) \\ &= 0 \\ &= g(n\pi), \end{aligned}$$

so f and g get sampled to appear as the same vector $\mathbf{x}$, which happens to be the zero vector. This is called *aliasing*: one signal appears the same as another.

5 Frequency aliasing

There is a special kind of aliasing, where two sinusoidal signals appear the same under sampling. A complex sinusoid is a function of the form

$$f(t) = e^{2\pi i\omega t} = \cos(2\pi\omega t) + i\sin(2\pi\omega t). \tag{3}$$

This signal is periodic, which repeats itself at a rate of ω cycles per unit time. Eg. $f(t) = e^{2\pi i 60t}$ represents a 60 Hertz signal, where t is measured in seconds. Two signals

$$f(t) = e^{2\pi i\omega_1 t} \qquad g(t) = e^{2\pi i\omega_2 t}, \tag{4}$$

will get aliased at a sample interval Δt if

$$f(n\Delta t) = g(n\Delta t) \text{ for all integers } n. \tag{5}$$

Equivalently,

$$(e^{2\pi i\omega_1\Delta t})^n = (e^{2\pi i\omega_2\Delta t})^n \text{ for all integers } n, \tag{6}$$

or more simply, if $e^{2\pi i\omega_1\Delta t} = e^{2\pi i\omega_2\Delta t}$. A bit of algebra shows this happens if $\omega_1 - \omega_2 = N/\Delta t$, for some integer N.

Thus, two sinusoids get aliased if the difference of their frequencies $\omega_1 - \omega_2$ is a multiple of the sampling rate $1/\Delta t$.

Usually, we are interested in measuring signals with frequencies in some interval $[-F, F]$. So, for instance, you might like to measure frequencies ω_1, ω_2 in $[-400Hz, 400Hz]$. The difference $\omega_1 - \omega_2$ could be as big as 800Hz. In order for this to not be a multiple of the sample rate, we have to choose a sample rate bigger than 800Hz. That is,

$$800Hz \leq \frac{1}{\Delta t}. \tag{7}$$

Being a little big lazy, we might choose the sample rate to be 1000Hz, and so the sampling interval is $\Delta t = .001$ second (a millisecond).

6 Sampled signals as a vector space

The sampled signal

$$\mathbf{x} = (\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots) \tag{8}$$

is supposed to look like a vector, just as you learned in linear algebra. It just happens to be infinitely long. You can still treat as you would regular vectors: add, subtract, pointwise multiply, scalar multiply, and take inner products. For instance, write

$$\begin{aligned}
\mathbf{x} &= (\ldots, 0, 0, 1, 2, 3, 0, 0, \ldots) \\
\mathbf{y} &= (\ldots, 0, 0, 4, 5, 6, 0, 0, \ldots) \\
\mathbf{x} + \mathbf{y} &= (\ldots, 0, 0, 5, 7, 9, 0, 0, \ldots) \\
\mathbf{y} - \mathbf{x} &= (\ldots, 0, 0, 3, 3, 3, 0, 0, \ldots) \\
\mathbf{x} \cdot \mathbf{y} &= (\ldots, 0, 0, 4, 10, 18, 0, 0, \ldots) \\
10\mathbf{x} &= (\ldots, 0, 0, 10, 20, 30, 0, 0, \ldots) \\
10\mathbf{y} &= (\ldots, 0, 0, 40, 50, 60, 0, 0, \ldots) \\
\langle \mathbf{x}, \mathbf{y} \rangle &= 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32.
\end{aligned}$$

It is a nuisance to keep writing all these zeros in the infinite vectors, so sometimes we shorten things to condensed vectors, like $\mathbf{x} = (1, 2, 3)$ and $\mathbf{y} = (4, 5, 6)$. In this form, we have the 0-th entry in $\mathbf{x}$ as the first number that appears in the short vector. So $x_0 = 1, x_1 = 2, x_2 = 3$, and the rest are zero.

7 Z transforms

The Z transform of the signal

$$\mathbf{x} = (\ldots, 0, 0, 1, 2, 3, 0, 0, \ldots) \tag{9}$$

is the polynomial

$$X(Z) = 1 + 2Z + 3Z^2. \tag{10}$$

The Z transform of the signal

$$\mathbf{y} = (\ldots, 0, 0, 4, 5, 6, 0, 0, \ldots) \tag{11}$$

is the polynomial

$$Y(Z) = 4 + 5Z + 6Z^2. \tag{12}$$

In general, for a signal $\mathbf{x} = (\ldots, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots)$, the Z transform is the polynomial

$$X(Z) = \sum_n x_n Z^n. \tag{13}$$

Notice that if there are non-zero x_n with $n < 0$, the polynomial could include negative powers of Z. So, for instance, if

$$x_{-1} = 3, x_0 = 5, x_1 = 7, x_2 = 9 \tag{14}$$

then

$$X(Z) = 3Z^{-1} + 5 + 7Z + 9Z^2. \tag{15}$$

NOTE: For historical reasons, engineers often replace Z with Z^{-1} or z^{-1} in the Z-transform (for example, our textbook by Lyons does this). This is a source of much confusion, but you might as well get used to it. The difference amounts to inverting the complex plane across the unit circle (whatever that means), but an example helps to explain. The finite sequence $x = (1, 2, 3)$ has Z-transform

$$X(Z) = 1 + 2Z + Z^2 \tag{16}$$

in our convention, which is a nice analytic function (of Z) on the complex plane. In the engineer's convention, the z-transform is

$$X(z) = 1 + 2z^{-1} + z^{-2}, \tag{17}$$

which is not even analytic at $z = 0$. (It has a pole of order 2 there.) It is, however, analytic at ∞, a vague concept for us, that could be made precise. (I won't bother.)

WHY: Why do engineers use the Z^{-1} convention? As far as I can tell, it is because they want it to represent a "delay by one" operator in signal space, where a sequence $x(n)$ is delayed by one to give output $y(n) = x(n-1)$. So the subtracting by one is remember with the -1 in Z^{-1}. IMHO, that's a fairly lame reason to do it, since you have to give up analyticity!

8 The Fourier transform

The Fourier transform of a sequence $\mathbf{x}$ is obtained by taking the Z-transform, and evaluating at points on the unit circle, $Z = e^{2\pi i f}$, where f is a number between -1/2 and 1/2 (i.e. we wrap all the way around the unit circle). For instance, the sequence

$$x_{-1} = 3, x_0 = 5, x_1 = 7, x_2 = 9 \tag{18}$$

has Z-transform

$$X(Z) = 3Z^{-1} + 5 + 7Z + 9Z^2 \tag{19}$$

and Fourier transform

$$X(e^{2\pi i f}) = 3e^{-2\pi i f} + 5 + 7e^{2\pi i f} + 9e^{4\pi i f}, \qquad -1/2 \le f \le 1/2. \tag{20}$$

Sometimes we "abuse notation" and write the Fourier transform like this

$$X(f) = 3e^{-2\pi i f} + 5 + 7e^{2\pi i f} + 9e^{4\pi i f}, \qquad -1/2 \le f \le 1/2. \tag{21}$$

Formally, we write the Fourier transform of a sequence $\mathbf{x}$ as the sum

$$X(f) = \sum_{n=-\infty}^{\infty} x_n e^{2\pi i n f}. \tag{22}$$

Remarkably, we can always recover the original sequence $\mathbf{x}$ from its Fourier transform, using the integral formula

$$x_n = \int_{-1/2}^{1/2} X(f) e^{-2\pi i n f} df. \tag{23}$$

You should check this with one of the examples above.

9 The Fast Fourier transform

The Fast Fourier Transform (FFT) just evaluate the FT on fractional values of frequency $f = 0, 1/N, 2/N, \ldots (N-1)/N$ in the form

$$X(\frac{k}{N}) = \sum_{n=0}^{N-1} x_n e^{2\pi i n k/N}, \qquad 0 \le k \le N-1. \tag{24}$$

Note the limits of the sum: the FFT applies to sequences of length N, starting at term x_0. The inversion formula reduces to a sum,

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X(\frac{k}{N}) e^{-2\pi i n k/N}, \qquad 0 \le n \le N-1. \tag{25}$$

Again, this only works for vectors of length N. (Of course, you can include some zeros in your vector.)

It's called the FAST Fourier transform because there is an algorithm that reduces the computation from about N^2 operations, to only $N \log_2 N$ operations. For a short seismic trace of $N = 1000$ samples, this means instead of one million operations on the computer, we only need about 10,000 – a speed-up of a factor of 100.

10 MATLAB's FFT

MATLAB indexes its vectors from 1 to N, and so the FFT is expressed with shifted indices. With a vector $\mathbf{x} = [x(1), x(2), \ldots, x(N)]$ and FFT $X = [X(1), X(2), \ldots, X(N)]$, MATLAB will compute

$$X(k) = \sum_{n=1}^{N} x(n) e^{2\pi i(n-1)(k-1)/N}, \qquad 1 \le k \le N. \tag{26}$$

The inverse ifft is computed as

$$x(n) = \frac{1}{N} \sum_{k=1}^{N} X(k) e^{-2\pi i(n-1)(k-1)/N}, \qquad 1 \le n \le N. \tag{27}$$

11 Symmetries of the Fourier transform

When the inputs sequence $\mathbf{x}$ is real (i.e. we use real data, not complex), the Fourier transform X is complex valued, but have the amazing symmetry:

$$X(-f) = \overline{X(f)}. \tag{28}$$

That is, the values at negative frequencies are the complex conjugate of the values at positive frequencies. (WHY? Check yourself!) It also means the absolute value of the values at negative frequencies are the same as those at the corresponding positive frequency:

$$|X(-f)| = |X(f)|. \tag{29}$$

Because of periodicity, we can also say

$$|X(1-f)| = |X(f)|, \text{ for } 0 \le f \le 1. \tag{30}$$

You will always see this symmetry in your MATLAB plots of Fourier transforms.

12 Convolution

Any two signals $\mathbf{x}, \mathbf{y}$ can be combined in a special operation called *convolution*. An easy way to define the convolution is using the Z transform. So, for example, if

$$\begin{aligned} \mathbf{x} &= (\ldots, 0, 0, 1, 2, 3, 0, 0, \ldots) \\ \mathbf{y} &= (\ldots, 0, 0, 4, 5, 6, 0, 0, \ldots), \end{aligned}$$

we have Z transforms $X(Z) = 1 + 2Z + 3Z^2, Y(Z) = 4 + 5Z + 6Z^2$. Take the product of the two polynomials,

$$(1 + 2Z + 3Z^2)(4 + 5Z + 6Z^2) = 4 + 13Z + 28Z^2 + 27Z^3 + 18Z^4, \tag{31}$$

which we can recognize as the Z transform of the vector

$$(\ldots, 0, 0, 4, 13, 28, 27, 18, 0, 0, \ldots) \tag{32}$$

which we define as the convolution of $\mathbf{x}$ with $\mathbf{y}$. That is,

$$\mathbf{x} * \mathbf{y} = (\ldots, 0, 0, 4, 13, 28, 27, 18, 0, 0, \ldots). \tag{33}$$

There is a general formula for convolution: if you think about how polynomial multiplication works, it is pretty easy to see that the n-th entry in the vector $\mathbf{x} * \mathbf{y}$ will be a sum of terms like $x_j y_k$, where $j + k = n$. In other words, we can write

$$(\mathbf{x} * \mathbf{y})_n = \sum_k x_{n-k} y_k. \tag{34}$$

Because we know how polynomial multiplication works, we can observe that convolution works in either order and gives the same answer: $\mathbf{x}*\mathbf{y} = \mathbf{y}*\mathbf{z}$. Also, the convolution operation distributes over addition: $\mathbf{x}*(\mathbf{y}+\mathbf{z}) = (\mathbf{x}*\mathbf{y})+(\mathbf{x}*\mathbf{z})$: because multiplication of polynomials distributes over addition. You can also move in scalar constants quite freely, so for instance, $3(\mathbf{x} * \mathbf{y}) = (3\mathbf{x}) * \mathbf{y} = \mathbf{x} * (3\mathbf{y})$, which is clear from the summation formula defining convolution.

13 Convolution as matrix-vector multiplication

Notice we can organize a matrix and a vector to get the same result as the convolution in the last section. We just do the simple $\mathbf{x}, \mathbf{y}$ example:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 \\ 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \\ 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 13 \\ 28 \\ 27 \\ 18 \end{bmatrix}. \tag{35}$$

The matrix representing $\mathbf{x}$ is constant along diagonals. That is called a Toeplitz matrix. $\mathbf{y}$ is organized as a column vector. The usual matrix-vector product that we learned in linear algebra gives the resulting column vector representing $\mathbf{x} * \mathbf{y}$.

Notice that although $\mathbf{x}, \mathbf{y}$ had only three non-zero entries, the matrix had to be 5 by 5, and the vector 5 by 1, in order for the arithmetic to work out. Similarly, for vectors with N non-zero entries (all in a row), we need matrices of size 2N + 1.

14 Convolution as matrix-matrix multiplication

We observe that we can also represent convolution as the product to two Toeplitz matrices:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 \\ 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 5 & 4 & 0 & 0 & 0 \\ 6 & 5 & 4 & 0 & 0 \\ 0 & 6 & 5 & 4 & 0 \\ 0 & 0 & 6 & 5 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 13 & 4 & 0 & 0 & 0 \\ 28 & 13 & 4 & 0 & 0 \\ 27 & 28 & 13 & 4 & 0 \\ 18 & 27 & 28 & 13 & 4 \end{bmatrix}. \tag{36}$$

Check the matrix multiplication; you will see this is correct. Again, the matrices have to be large enough to give room for the whole convolution to appear.

You could write out infinite matrices, but it's too hard to typeset at the moment!

15 Convolution by flipping and shifting

You can also obtain a convolution by flipping the order of one of the vectors, taking point-wise products, and sum. So, for instance, with the same $\mathbf{x}, \mathbf{y}$ as in the previous section, we flip around the $\mathbf{y}$ and write it underneath the $\mathbf{x}$ vector:

$$\begin{matrix} \ldots 0 & 0 & 1 & 2 & 3 & 0 \ldots \\ \ldots 6 & 5 & 4 & 0 & 0 & 0 \ldots \end{matrix}. \tag{37}$$

The pointwise product is

$$\ldots 0 \quad 0 \quad 4 \quad 0 \quad 0 \quad 0 \ldots \tag{38}$$

which sums up to 4, the first component of the convolution. To get the second component, we shift $\mathbf{y}$ once, and line up the vectors as

$$\begin{matrix} \ldots 0 & 0 & 1 & 2 & 3 & 0 \ldots \\ \ldots 0 & 6 & 5 & 4 & 0 & 0 \ldots \end{matrix}. \tag{39}$$

The pointwise product is

$$\ldots 0 \quad 0 \quad 5 \quad 8 \quad 0 \quad 0 \ldots \tag{40}$$

which sums up to 13, the second component of the convolution.

To get the third component, we shift $\mathbf{y}$ again, and line up the vectors as

$$\begin{matrix} \ldots 0 & 0 & 1 & 2 & 3 & 0 \ldots \\ \ldots 0 & 0 & 6 & 5 & 4 & 0 \ldots \end{matrix}. \tag{41}$$

The pointwise product is

$$\ldots 0 \quad 0 \quad 6 \quad 10 \quad 12 \quad 0 \ldots \tag{42}$$

which sums up to 28, the third component of the convolution. And so on. This works in general.

16 Convolution as a system

Fixing a vector $\mathbf{h}$, we define a system that acts on signals $\mathbf{x}$ as

$$\mathbf{x} \to A(\mathbf{x}) = \mathbf{h} * \mathbf{x} = \mathbf{y}. \tag{43}$$

That is, for input signal $\mathbf{x}$, our system outputs a signal $\mathbf{y}$ that is computed as $\mathbf{y} = \mathbf{h} * \mathbf{x}$.

From our description of convolution as matrices, just like in linear algebra, we certainly expect that this system is linear. That is,

$$\begin{aligned} A(\mathbf{x}_1 + \mathbf{x}_2) &= A(\mathbf{x}_1) + A(\mathbf{x}_2), \\ A(\alpha \mathbf{x}) &= \alpha A(\mathbf{x}). \end{aligned}$$

This is easy to verify from the formulas for convolution. Equivalently, the first equation follows since convolution distributes over addition, $\mathbf{h} * (\mathbf{x}_1 + \mathbf{x}_2) = \mathbf{h} * \mathbf{x}_1 + \mathbf{h} * \mathbf{x}_2$, and the second equation follows since multiplication by a scalar commutes with polynomial multiplication.

17 The shift operator

The shift operator, S, is an example of a system. It takes an input $\mathbf{x}$ and outputs the same vectors, except shifted to the right by one unit. So for instance, with

$$\begin{aligned} \mathbf{x} &= (\ldots, 0, 0, 1, 2, 3, 0, 0, \ldots) \\ \text{then } S\mathbf{x} &= (\ldots, 0, 0, 0, 1, 2, 3, 0, \ldots). \end{aligned}$$

In general, with

$$\begin{aligned} \mathbf{x} &= (\ldots, x_{-2}, x_{-1}, x_0, \ \ x_1, x_2, x_3, \ldots) \\ \text{then } S\mathbf{x} &= (\ldots, x_{-3}, x_{-2}, x_{-1}, x_0, x_1, x_2, \ldots). \end{aligned}$$

In terms of vector components, we can see that

$$(S\mathbf{x})_n = x_{n-1}. \tag{44}$$

It will be useful to notice that the operator S can be expressed as a convolution. We write δ^1 to be the special signal which is zero in all its components, except at the $n = 1$ place, where it takes the value 1. That is,

$$\delta^1 = (\ldots, 0, 0, 0, 1, 0, 0, \ldots), \tag{45}$$

where the 1 in the vector is placed exactly at the $n = 1$ place. Then

$$\delta^1 * \mathbf{x} = \sum_k x_{n-k} \delta^1_k = x_{n-1} \cdot 1 + \text{ a bunch of zeros.} \tag{46}$$

Thus $\delta^1 * \mathbf{x} = S\mathbf{x}$, which shows convolution by δ^1 is the same as shifting by 1.

It is easy to check that convolution by δ^n is the same as shifting by n steps (to the right, when n is bigger than zero, to the left when n is less than zero). This is the same as applying the operator S to the signal n times, which we denote by S^n.

You can think of δ^n as the vector

$$\delta^1 = (\ldots, 0, 0, 0, 0, 0, 1, 0, 0, \ldots), \tag{47}$$

where the 1 in the vector is placed exactly at the n-th spot.

You can think of the operator S as a matrix of the form

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \tag{48}$$

which is a Toeplitz matrix with ones just below the main diagonal. So, for instance, we apply this matrix to the vector $\mathbf{x} = (\ldots, 0, 0, 1, 2, 3, 0, \ldots)$ in a short form, to see

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 0 \end{bmatrix}. \tag{49}$$

That is, this matrix applied to $\mathbf{x}$ just shifts it.

Of course, for longer signals $\mathbf{x}$, we need a bigger matrix to represent the shift S. But the idea is the same. S will always have ones just below the main diagonal.

18 Signals as sums of δ^n's.

Look how we can write a 3-component vector as a linear combination of three basic vectors:

$$(1, 2, 3) = 1 \cdot (1, 0, 0) + 2 \cdot (0, 1, 0) + 3 \cdot (0, 0, 1). \tag{50}$$

For our infinite vectors, we can do something similar:

$$\begin{aligned} \mathbf{x} &= (\ldots, 0, 0, 1, 2, 3, 0, 0, \ldots) \\ &= 1 \cdot (\ldots, 0, 0, 1, 0, 0, 0, 0, \ldots) + 2 \cdot (\ldots, 0, 0, 0, 1, 0, 0, 0, \ldots) + 3 \cdot (\ldots, 0, 0, 0, 0, 1, 0, 0, \ldots) \\ &= 1 \cdot \delta^0 + 2 \cdot \delta^1 + 3 \cdot \delta^2. \end{aligned}$$

That is, the vector $\mathbf{x}$ is written as a linear combination of those basic vectors δ^n that we saw in the last section.

In general, any signal $\mathbf{x}$ can be written in the form

$$\mathbf{x} = \sum_n x_n \delta^n. \tag{51}$$

Notice in this formula, $\mathbf{x}$ is a vector, each δ^n is a vector, but the x_n are just numbers.

We need this for the proof in the next system.

19 Linear, shift-invariant systems

We want to find out what are the linear, shift-invariant systems. Linear means the system is relatively simple in that sums of input signals map to sums of output signals, while shift-invariant means if we delay the input signal, the resulting output signal is the same as before, only delayed. This is also a reasonable assumption for a system that does not change over time.

We express these conditions on a system A in the following equations:

$$\begin{aligned} A(\mathbf{x}_1 + \mathbf{x}_2) &= A(\mathbf{x}_1) + A(\mathbf{x}_2), \text{ for all signals } \mathbf{x}_1, \mathbf{x}_2, \\ A(\alpha \mathbf{x}) &= \alpha A(\mathbf{x}), \text{ for all signals } \mathbf{x}, \text{ scalars } \alpha, \\ A(S\mathbf{x}) &= SA(\mathbf{x}), \text{ for all signals } \mathbf{x}, \end{aligned}$$

where S is the shift operator.

Theorem 1 *Suppose A is a linear, shift-invariant system. Then there is a vector* $\mathbf{h}$ *so that A is just convolution by* $\mathbf{h}$. *That is,*

$$A(\mathbf{x}) = \mathbf{h} * \mathbf{x}, \textit{ for all signals } \mathbf{x}. \tag{52}$$

The proof will go like this. $\mathbf{h}$ is just A applied to the delta vector δ^0, the vector with zeros everywhere except at the $n = 0$ spot. We then use shift invariance to find that A acting on any δ^n is just that delta function, convolved with $\mathbf{h}$. We then use linearity to conclude A acts on any vector by convolving with $\mathbf{h}$.

The details are like this. Let $\mathbf{h} = A(\delta^0)$, which is a vector, since A acts on the given input vector δ^0 to produce some output vector, which we call $\mathbf{h}$. We note the shift operator S takes the vector δ^0 to δ^1, so by shift-invariance, we see

$$A(\delta^1) = A(S\delta^0) = S(A(\delta^0)) = S\mathbf{h} = \mathbf{h} * \delta^1, \tag{53}$$

since by definition, $A(\delta^0)$ is $\mathbf{h}$, and applying S to $\mathbf{h}$ is the same as convolving by δ^1.

Repeating this argument, we see that $A(\delta^n) = A(S^n\delta^0) = S^n A(\delta_0) = S^n\mathbf{h} = \mathbf{h} * \delta^n$. So now we know that A, applied to any of the delta vectors, just gives $\mathbf{h}$ convolved with the vector.

Now, any vector **x** is a linear combination of the δ^n, so by linearity of A, we have

$$A(\mathbf{x}) = A(\sum x_n \delta^n) = \sum x_n A(\delta^n) = \sum x_n \mathbf{h} * \delta^n = \mathbf{h} * (\sum x_n \delta^n) = \mathbf{h} * \mathbf{x}, \tag{54}$$

where at the second last equality, we use the fact that convolution by **h** is linear.

And that's it. If you want to worry about mathematical details, you should worry about whether these infinite sums converge. For our purposes, we can just assume the **x** is always given by a finite sum. (All but finitely many of the x_n are zero.)

20 Impulse response of a LSI system, Z transform

The vector **h** that appeared in the last section is call the *impulse response* of the system A. From an engineering point of view, it is the response of the system to getting a whack at time $t = 0$. Knowing the impulse response basically tells you everything you need to know about the linear, shift invariant system.

The Z transform of the system A is given as the Z transform of the impulse response **h**, which is the polynomial

$$H(Z) = \sum_k h_k Z^k, \tag{55}$$

which we discussed before.

21 FIR systems, minimum phase, maximum phase

A Finite Impulse Response (FIR) system is a linear, shift-invariant system where the impulse response function **h** only has finitely many non-zero coefficients h_n. These are particularly useful in computations, since only finite sums are needed to compute them.

As an example, let's take $\mathbf{h} = (6, 1, -1)$ (in our short vector notation). The LSI system $\mathbf{x} \to \mathbf{h} * \mathbf{x} = \mathbf{y}$ is given by the formula

$$y_n = 6x_n + x_{n-1} - x_{n-2}, \text{ for all } n. \tag{56}$$

See how there is only 3 terms in the sum given by the convolution with **h**.

This is an example of a *causal* system: the value of y_n depends only on the value of x_n and earlier coefficients in **x**. In general, a LSI system with be causal if the impulse response **h** is zero on the negative integers.

For the example above, the Z transform is the polynomial

$$H(Z) = 6 + Z - Z^2. \tag{57}$$

Notice this polynomial factors, as $(6 + Z - Z^2) = (3 - Z)(2 + Z)$. The linear terms $(3 - Z), (2 + Z)$ are called couplets. From the couplets, we can see the zeros of this polynomial are $Z = 3$ and

$Z = -2$. These zeros have magnitude bigger than one, so they live outside the unit circle in the complex plane. Because of this, we say the system is minimum phase.

By the Fundamental Theorem of Algebra, any polynomial can be factored into couplets. The zeros can be identified as points on the complex plane. If the zeros are all outside the unit circle, we say the system is minimum phase. If the zeros are inside the unit circle, we say the system is maximum phase. If some zeros are inside the circle, and some are outside, we say the system is mixed phase.

Minimum and maximum phase will have something to do with delays in our systems, which we will see later.

22 Recursive (IIR) systems

When the impulse response $\mathbf{h}$ of a LSI system has infinitely many non-zero coefficients, we say the system is an Infinite Impulse Response (IIR) system. From a computational point of view, this is a nuisance since the direct convolution formula

$$y_n = \sum_k x_{n-k} h_k \tag{58}$$

has an infinite sum in it, which our computer will choke on.

However, there is a nice little structure that allows us to compute some IIR systems with only finite sums. This structure feeds back the earlier y values into the computation, and hence is called a *recursive* system.

It is probably best understood with a simple example.

Suppose we have a formula to compute output $\mathbf{y}$ in terms of $\mathbf{x}$, in the form:

$$y_n = x_n + \frac{1}{2} y_{n-1}. \tag{59}$$

This is a finite sum (only two terms), and assuming we know the earlier output coefficients $\ldots, y_{n-3}, y_{n-2}, y_{n-1}$, we can always compute the next y_n.

Let's see what the impulse response is. With $\mathbf{x} = \delta^0$, we have that all the x_n equal zero, except for $x_0 = 1$. We can assume then that all the earlier y_n are zero, for all $n < 0$. Then, we find

$$\begin{aligned}
y_0 &= x_0 + \frac{1}{2} y_{-1} = 1 + \frac{1}{2} \cdot 0 = 1 \\
y_1 &= x_1 + \frac{1}{2} y_0 = 0 + \frac{1}{2} \cdot 1 = \frac{1}{2} \\
y_2 &= x_2 + \frac{1}{2} y_1 = 0 + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \\
\ldots & \\
y_n &= \frac{1}{2^n}, \text{ for each } n \geq 0
\end{aligned}$$

Thus, the impulse response is the signal $\mathbf{h} = (\ldots, 0, 0, 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots)$, which is an infinite impulse response. The Z transform of the system is thus

$$H(Z) = 1 + \frac{Z}{2} + \frac{Z^2}{4} + \frac{Z^3}{8} + \cdots + \frac{Z^n}{2^n} + \cdots, \tag{60}$$

which we recognize as a geometric series, so we can sum it to

$$H(Z) = \frac{1}{1 - Z/2}. \tag{61}$$

Notice the linear polynomial in the denominator as a root at $Z = 2$. This point is called a pole of the filter. Because this root is outside the unit circle, we would call this a minimum phase system.

23 Rational functions and IIR systems

Here is another way to compute the Z transform of an IIR system. Using the same example as in the last section, we write a simple recursive system as

$$y_n = x_n + \frac{1}{2} y_{n-1}. \tag{62}$$

Pulling the $y's$ onto the lefthand side, we have

$$y_n - \frac{1}{2} y_{n-1} = x_n, \tag{63}$$

which we can express as a convolution,

$$\mathbf{g} * \mathbf{y} = \mathbf{x}, \tag{64}$$

where $\mathbf{g} = (1, -1/2)$ is the vector of coefficients for the convolution on the y side of the equation. Taking Z transforms of everything in the last equation, we have

$$G(Z)Y(Z) = X(Z), \tag{65}$$

which we rewrite as

$$Y(Z) = \frac{1}{G(Z)} X(Z). \tag{66}$$

Notice that since $\mathbf{g}$ is a short vector, its Z transform is easily computed, as $G(Z) = 1 - Z/2$. The fraction $\frac{1}{G(Z)} = \frac{1}{1 - Z/2}$ is the system response of the LSI with impulse response $\mathbf{h}$ as in the last section. Notice it agrees with the geometric series summation we did in the last section.

A general recursive system is written like this:

$$y_n = \sum_{k=0}^{N} f_k x_{n-k} - \sum_{k=1}^{M} g_k y_{n-k}, \tag{67}$$

where $\mathbf{f} = (f_0, f_1, \ldots, f_N)$ and $\mathbf{g} = (1, g_1, g_2, \ldots, g_M)$ are some fixed coefficients that will determine our LSI system. By carefully choosing $g_0 = 1$, we can rewrite the last equation as two convolutions, so

$$\sum_{k=0}^{M} g_k y_{n-k} = \sum_{k=0}^{N} f_k x_{n-k}, \tag{68}$$

or more succinctly in vector form as

$$\mathbf{g} * \mathbf{y} = \mathbf{f} * \mathbf{x}. \tag{69}$$

Taking Z transforms, we have

$$G(Z)Y(Z) = F(Z)X(Z), \tag{70}$$

which we write in input-output form as

$$Y(Z) = \frac{F(Z)}{G(Z)} X(Z). \tag{71}$$

The function

$$H(Z) = \frac{F(Z)}{G(Z)} = \frac{f_0 + f_1 Z + f_2 Z^2 + \cdots + f_N Z^N}{g_0 + g_1 Z + g_2 Z^2 + \cdots g_M Z^M} \tag{72}$$

is the Z transform of the LSI systems. Notice it is the ratio of two polynomials in Z, which is an example of a rational function in complex analysis.

24 Zeros and poles

By the fundamental theorem of algebra, the polynomials in a rational function can always be factored into a product of linear terms, so

$$\begin{aligned} H(Z) &= \frac{F(Z)}{G(Z)} = \frac{f_0 + f_1 Z + f_2 Z^2 + \cdots + f_N Z^N}{g_0 + g_1 Z + g_2 Z^2 + \cdots g_M Z^M} \\ &= \frac{f_N}{g_M} \frac{(Z - z_1)(Z - z_2) \cdots (Z - z_N)}{(Z - p_1)(Z - p_2) \cdots (Z - p_M)} \end{aligned}$$

where the complex numbers $z_1, \ldots, z_N$ are the roots of the polynomial on top, and $p_1, \ldots, p_M$ are the roots of the polynomial on the bottom. The z_k are called the *zeros* of the function $H(Z)$, and the p_k are called the *poles* of the function. This is because, as a complex analytic function, $H(Z)$ is equal to zero at the z_k, and blows up (division by zero) at the p_k.

25 Stability of recursive LSI systems

Fix a complex number α and define a recursive LSI system as

$$y_n = x_n + \alpha y_{n-1}. \tag{73}$$

Computing the impulse response as we did above for the simple example, we see that the impulse response is just

$$\mathbf{h} = (\ldots, 0, 0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^n, \ldots). \tag{74}$$

This sequence will go to zero if $|\alpha| < 1$, but will blow up with $|\alpha| > 1$. The first case is called *stable*, while the second case is called *unstable*. Notice that the reciprocal $1/\alpha$ is the single pole in thus LSI system, since $G(Z) = 1 - \alpha Z$ has a root at $Z = 1/\alpha$. Thus the stability of this simple system depends on the location of the pole $1/\alpha$: outside the unit circle, the system is stable. Inside the unit circle, it is unstable.

This result generalizes. Given a rational function

$$H(Z) = \frac{F(Z)}{G(Z)}, \tag{75}$$

the corresponding LSI system is stable if all the poles are outside the unit circle. It is unstable if the poles are inside the unit circle.

Try not to worry about the case when the poles are exactly on the unit circle. Generally speaking, they tend to be unstable due to small numerical errors that build up on the computer.

26 Example: a smoothing system. FIR and IIR

Let's imagine you want to design a system that smooths out any given input signal. For instance, with a photo, you might want to blur out some details so that certain faces cannot be identified. Or, if you are building an electrical system, you might realize sudden jumps in current are bad for the electronic devices, so you want to smooth out any jumps. Or you are building a car, and sudden jerks and shocks in the motion will annoy the passengers, so you want to smooth those out.

Without explaining why, here are two smoothers. The first is an FIR, non-recursive system. We choose $\mathbf{h}$ to be the vector

$$\mathbf{h} = (\ldots, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, \ldots). \tag{76}$$

That is, $\mathbf{h}$ has 10 ones in it, the rest are zero. (This is sometimes called a boxcar, because its graph looks like a chunky boxcar on a train.) The system $\mathbf{x} \to \mathbf{h} * \mathbf{x} = \mathbf{y}$ is given from the convolution as the term-by-term sum

$$y_n = x_n + x_{n-1} + x_{n-2} + \cdots + x_{n-9}. \tag{77}$$

If we take a simple boxcar $\mathbf{x} = (\ldots, 0, 0, 1, 1, 1, \ldots, 1, 1, 1, 0, 0, \ldots)$ as an input, it is easy to compute by hand what the output is: the ones will sum up as an increasing sequence, gets as big as 10, stays there for a while, and comes down. (You should try this yourself.) The output is thus

$$\mathbf{y} = (\ldots, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, \ldots, 10, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 0, \ldots). \tag{78}$$

This ramp $\mathbf{y}$ is a bit smoother than the original boxcar input $\mathbf{x}$. Figure 1 shows the $\mathbf{x}$ and the $\mathbf{y}$.

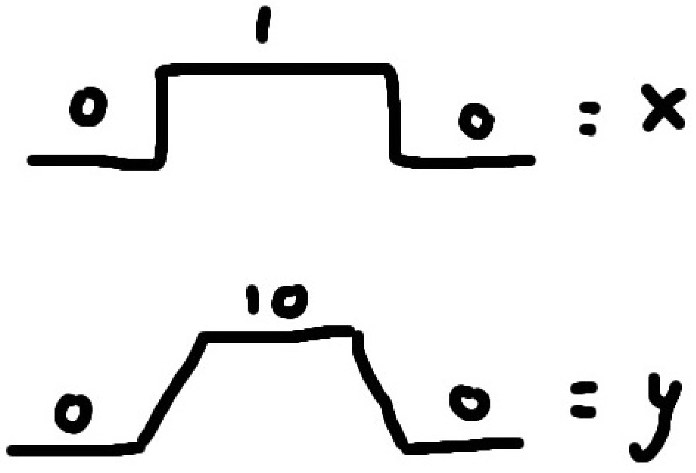

Figure 1: FIR smoother: A boxcar input $\mathbf{x}$ and its smoothed output $\mathbf{y}$.

Now, we could choose a different $\mathbf{h}$ to get a different kind of smoothing operation. But, let's now do something completely different. Let's build a recursive system of the form

$$y_n = x_n + 0.9y_{n-1}. \tag{79}$$

Now, with the same input $\mathbf{x} = (\ldots, 0, 0, 1, 1, 1, \ldots, 1, 1, 1, 0, 0, \ldots)$, we can compute the output by hand, as

$$\begin{aligned}
y_0 &= x_0 + 0.9y_{-1} = 1 + 0 = 1 \\
y_1 &= x_1 + 0.9y_0 = 1 + 0.9 * 1 = 1.9 \\
y_2 &= x_2 + 0.9y_1 = 1 + 0.9 * 1.9 = 2.71 \\
y_3 &= x_3 + 0.9y_2 = 1 + 0.9 * 2.71 = 3.439 \\
y_4 &= 4.0951 \\
y_5 &= 5.2170 \\
y_6 &= 5.6953 \\
y_7 &= 6.1258 \\
&etc.
\end{aligned}$$

As you can see, the numbers aren't pretty. But as you work it out, you will see they go up for a while, level off, and then go down. We can plot this in MATLAB, and the input/output looks something like Figure 2.

Again, we see that the output $\mathbf{y}$ is a smoothed out version of the input $\mathbf{x}$. However, the details are different from the FIR case in Figure 1. For instance, we don't have a linear ramp going up in the output, instead it is more of a smooth concave taper that creeps up to the value 10. And on the way down, the curve is a smooth, convex curve leveling off at zero.

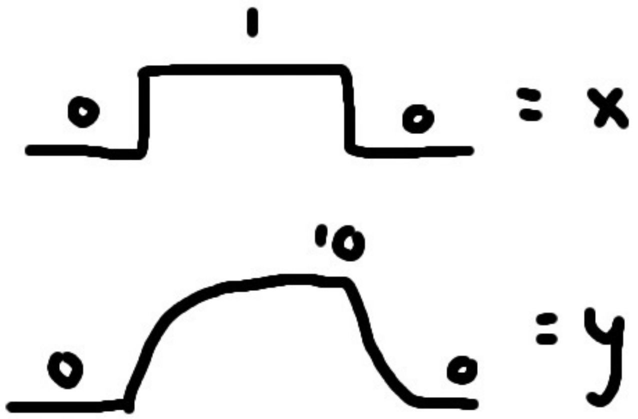

Figure 2: IIR smoother: A boxcar input $\mathbf{x}$ and its smoothed output $\mathbf{y}$.

Which is better? Well, it depends on what you are using these things for. Notice that the IIR system only uses one multiplication and one addition, to compute each y_n. This can be an advantage when speed is required. The FIR requires 10 additions, per sample y_n, which is a lot.

Much of the interest in signal processing is in designing good systems, FIR or IIR, which are fast to compute and do the required modifications on signals. We will get to this later.

27 Frequency response of a LSI system

Given a linear, shift invariant system A, we know there is an impulse response vector $\mathbf{h}$ that completely describe the system as a convolution,

$$x \mapsto A(\mathbf{x}) = \mathbf{h} * \mathbf{x}. \tag{80}$$

From this impulse response, we define the *frequency response* of the system as the function

$$H(\omega) = \sum_k h_k e^{-i\omega k}. \tag{81}$$

Now, remembering that the Z transform is given by $H(Z) = \sum_k h_k Z^k$, we see immediately that the frequency response is obtained from the Z transform by setting $Z = e^{-i\omega}$. That is, we have that the frequency response is given as

$$H(\omega) = H(Z), \text{ where } Z = e^{-i\omega}. \tag{82}$$

Now, there might be a little confusion here because $H(Z)$ can be thought of as a function of a complete variable, but $H(\omega)$ is a function of the real parameter ω. But this notation is standard in DSP, so we are kind of stuck with it.

You should notice that $H(\omega)$ is periodic in variable ω, since the exponential $Z = e^{-i\omega}$ is periodic. So we have that $H(\omega + 2\pi) = H(\omega)$, and thus $H(\omega)$ is completely determined by its values on the interval $[-\pi, \pi]$.

You should also be aware that some people (including me) like to use the definition

$$H(\omega) = \sum_k h_k e^{-2\pi i \omega k}, \tag{83}$$

in which case $H(\omega)$ is a periodic function, with period one. It is completely determined by its values on the interval $[-1/2, 1/2]$. Our textbook author Karl uses the other convention.

The function $H(\omega)$ tells us the frequency response of the system. Specifically, if we input a sinusoid $\mathbf{x}$ of the form

$$x_n = e^{i\omega n} = \cos(\omega n) + i \sin(\omega n), \tag{84}$$

where ω is a fixed number, then the output from our system A is just a multiple of the original,

$$A(\mathbf{x}) = H(\omega)\mathbf{x}. \tag{85}$$

That is, the output $A(\mathbf{x})$ is just the original input $\mathbf{x}$, multiplied by the fixed number $H(\omega)$. Roughly speaking, this says "sine wave in, yields sine wave out."

Except we are using complex sinusoids here.

To see why this happens, let's compute the convolution of $x_n = e^{i\omega n}$ with the impulse response $\mathbf{h}$, to see the result of the system $A(\mathbf{x}) = \mathbf{h} * \mathbf{x} = \mathbf{y}$. We have

$$\begin{aligned} y_n &= \sum_k x_{n-k} h_k \\ &= \sum_k e^{i\omega(n-k)} h_k \\ &= e^{i\omega n} \sum_k e^{i\omega k} h_k \\ &= x_n H(\omega). \end{aligned}$$

That is, in vector notation,

$$\mathbf{y} = H(\omega)\mathbf{x}, \tag{86}$$

so the output $\mathbf{y}$ is some multiple of the input $\mathbf{x}$.

This is another way of saying that the complex sinusoids are eigenvectors for the LSI system A, with eigenvalues given as $H(\omega)$.

28 Frequency response of a simple smoother

Here is a simple example. Define an FIR system by the formula

$$y_n = \frac{1}{3}(x_{n-1} + x_n + x_{n+1}). \tag{87}$$

That is, we just average the sample x_n with its nearest neighbours. This will tend to smooth out a given input signal, like we saw in the example in Section 21.

The impulse response $\mathbf{h}$ has $h_{-1} = h_0 = h_1 = 1/3$, with all the other coefficients zero. The Z transform of the system is

$$H(Z) = \frac{Z^{-1} + 1 + Z}{3}, \tag{88}$$

and the frequency response is obtained by setting $Z = e^{-i\omega}$, so

$$H(\omega) = \frac{e^{i\omega} + 1 + e^{-i\omega}}{3} = \frac{1}{3}(1 + 2\cos\omega). \tag{89}$$

A plot of this function is in Figure 3.

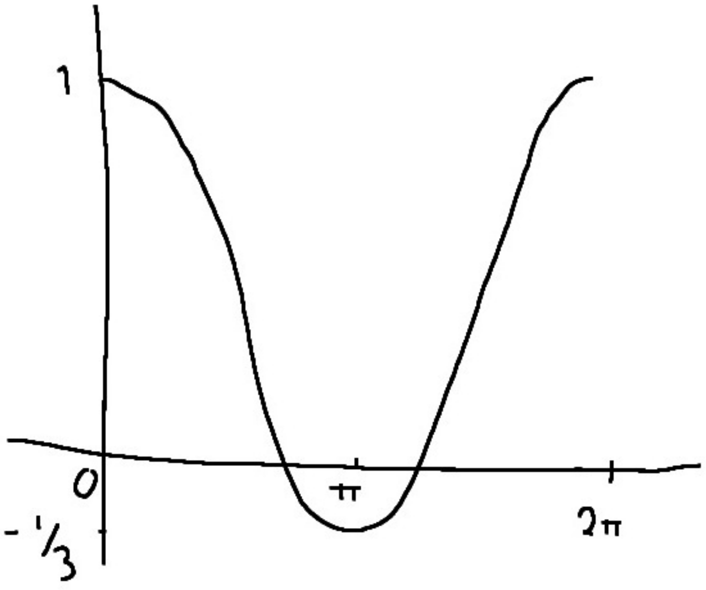

Figure 3: Plot of the frequency response $H(\omega)$ for a three point smoother.

What should you notice. Well, first, the function $H(\omega)$ is real valued, which is nice, and perhaps unexpected since we started with complex exponentials. Second, the function is symmetric about $\omega = \pi$, which also may seem unexpected, but happens a lot in filter design. Third, the function is zero for certain values of ω, which means for certain sinusoid inputs, we can get a zero output. For instance, with $\omega = 2\pi/3$, we have $H(\omega) = 0$. This means if we set input

$$x_n = e^{2\pi i n/3} = \left(-\frac{1}{2} + \frac{\sqrt{3}}{2}i\right)^n \tag{90}$$

we get output $\mathbf{y} = 0$. (Think about your complex numbers. Any three consecutive x_n are the three different cube roots of unity, so the three of them sum to zero. Hence their three point average is zero. Hence the total output is zero.)

You also should notice that $H(\omega)$ is negative for some values of ω, which indicates the sign of the input signal can flip. This is easy enough to see, by taking $\mathbf{x}$ to be an alternating series of ± 1, and compute the output as an alternating series of $\pm 1/3$, but with the opposite signs. That is, using $\mathbf{x}$ like this, we compute the three point moving average to find $\mathbf{y}$ as:

$$\begin{array}{rcl} \mathbf{x} & = & (\ldots, \quad 1, \quad -1, \quad 1, \quad -1, \quad 1, \quad -1, \quad \ldots) \\ \mathbf{y} & = & (\ldots, \quad -\frac{1}{3}, \quad \frac{1}{3}, \quad -\frac{1}{3}, \quad \frac{1}{3}, \quad -\frac{1}{3}, \quad \frac{1}{3}, \quad \ldots) \end{array} \tag{91}$$

See how the sign flips? See also how the magnitude is -1/3, just like the value of $H(\omega)$ on the graph, at its lowest point.

The point is, the function $H(\omega)$ tells us a lot about how the system behaves. In fact, from $H(\omega)$, we can learn everything about the system.

29 A better three point smoother

Frankly, in the last example, it is weird to have negative values for $H(\omega)$, and weird for it to be zero at a frequency $\omega = 2\pi/3$. It seems it would be better for a smoother to pass the low frequencies, and attenuate the high frequencies, only hitting zero at the point $\omega = \pi$. That is, we might want a filter response like the curve in Figure 4. With some inspired thinking, we can get this using the

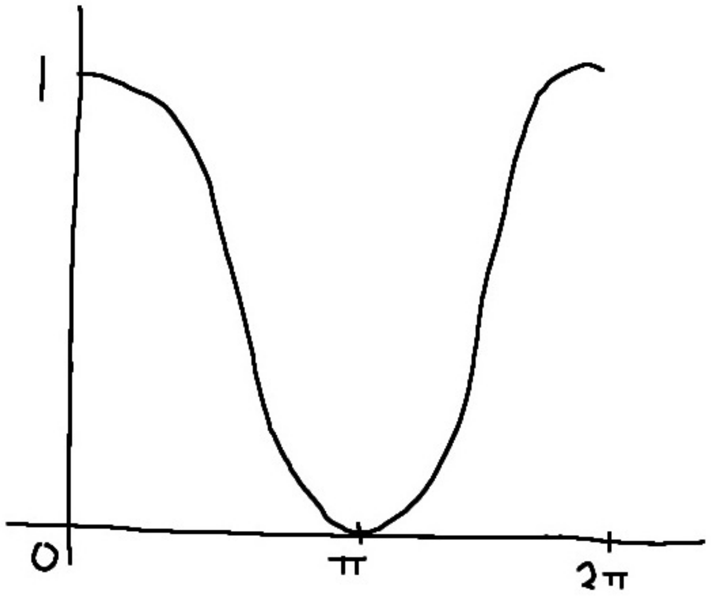

Figure 4: Plot of the frequency response $H(\omega)$ for a better three point smoother.

cosine function, shifted up by one, so we write

$$H(\omega) = \frac{1}{2}(1 + \cos\omega) = \frac{1}{4}(2 + e^{-i\omega} + e^{i\omega}), \tag{92}$$

as our desired frequency response. Examining this carefully, we see that this is the frequency response of a system with Z transform

$$H(Z) = \frac{1}{4}(Z^{-1} + 2 + Z). \tag{93}$$

From this, we see the impulse response is

$$\mathbf{h} = (\ldots, 0, 0, \frac{1}{4}, \frac{2}{4}, \frac{1}{4}, 0, 0, \ldots). \tag{94}$$

Thus, we can define our better three point smoother using this $\mathbf{h}$, and find

$$y_n = \frac{x_{n-1} + 2x_n + x_{n+1}}{4}. \tag{95}$$

This way, the output y_n is a weighted average of input x_n and its nearest neighbours. It turns out, for many reasons, that this is a better smoother for our system.

30 A rant on radians

The book (and many practitioners) insist on parameterizing frequencies ω in measures of radians per unit time. So, for instance, with a sinusoidal signal in the form

$$f(t) = e^{i\omega t} \tag{96}$$

where t is measured in seconds, $\omega = 1$ corresponds to oscillations of one radian per second, while $\omega = 10$ corresponds to oscillations of ten radians per second. Figure 8 shows plots of these two sinusoids (the imaginary part), and it is hard to see what 1 rad, or 10 rad, means.

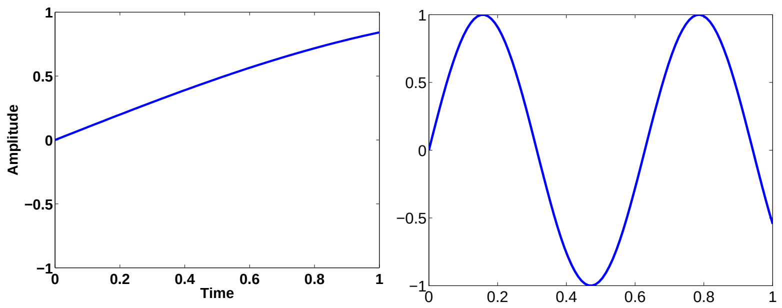

Figure 5: Sine waves at 1 rad per second (left) and 10 rad per second (right). These are not natural units!

Compare this to the more natural system of parameterizing frequencies ω in measures of cycles per second. With this parameterization, we write a sinusoidal signal in the form

$$f(t) = e^{2\pi i\omega t}. \tag{97}$$

Again with time measured in seconds, then $\omega = 1$ corresponds to oscillations of one cycle per second, while $\omega = 10$ corresponds to oscillations of ten cylces per second. Figure 9 shows plots of

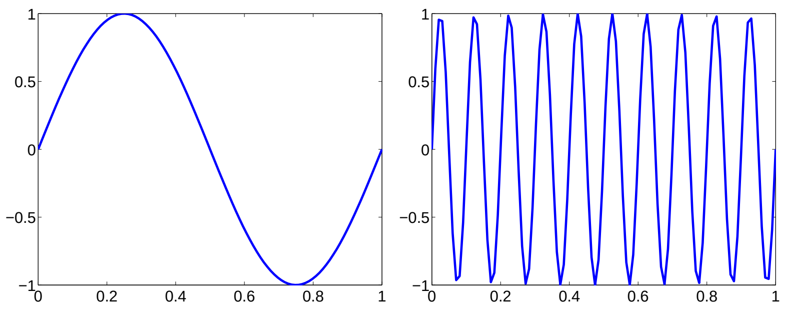

Figure 6: Sine waves at 1 cycle per second (left) and 10 cycles per second (right). This is natural.

these two sinusoids (the imaginary part), and we clearly see in the plots the one cycle, and ten cycles, respectively.

With this in mind, from now on in the course we will use natural parameterization of ω in cycles per unit of time, even though this conflicts with the book. (It turns out to have some mathematical benefits as well, in that the natural Fourier transform is automatically normalized, which we will see later.)

31 Symmetries of frequency response $H(\omega)$ in natural units.

With these natural units in mind, we define the frequency response of a system with impulse response $\mathbf{h} = (\ldots, h_{-1}, h_0, h_1, \ldots)$ as

$$H(\omega) = \sum_k h_k e^{-2\pi i k\omega}, \tag{98}$$

which is the eigenvalue for the (sampled signal) eigenvector $\mathbf{x}$ at the corresponding frequency ω in cycles per unit time given by

$$\mathbf{x}_n = e^{2\pi i n\omega}. \tag{99}$$

You might like to check the algebra in Section 22 for this new parameterization. Everything works out.

What's different is that this new function $H(\omega)$ is 1-periodic; that is, it repeats itself as ω is incremented by units of one. Thus,

$$\begin{aligned} H(0) &= H(1) = H(2) = H(3) = \ldots \\ H(0.2) &= H(1.2) = H(2.2) = H(3.2) = \ldots \\ H(0.4) &= H(1.4) = H(2.4) = H(3.4) = \ldots \\ &\text{etc.,} \end{aligned}$$

and in general, $H(\omega) = H(\omega + N)$ for all integers N, and all real numbers ω.

This implies that it is enough just to look at the values of $H(\omega)$ on the interval $[0,1]$, since it just repeats itself after that. So for instance, with the three point smoother considered earlier, where $\mathbf{h} = (1/4, 1/2, 1/4)$, we have that

$$H(\omega) = \frac{1}{2}(1 + \cos 2\pi\omega), \tag{100}$$

which we see plotted in Figure 10. Observe how the values on the interval $[0,1]$ (plotted in the solid

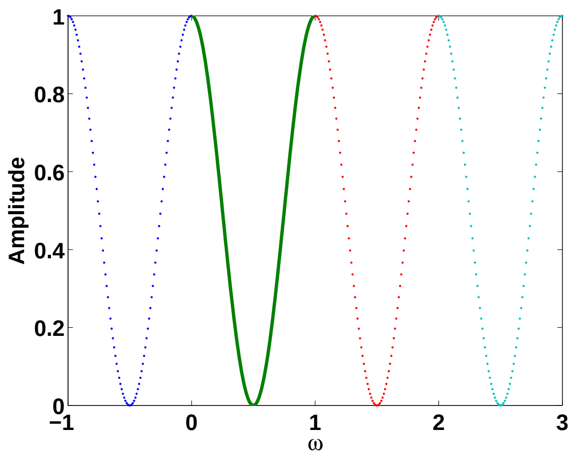

Figure 7: 1-periodicity of the frequency response curve $H(\omega)$ for a 3 point smoother.

line) just get repeated in each subsequent interval. This is the periodicity of $H(\omega)$, with period one.

It is common to plot only the values of $H(\omega)$ in the interval $[0,1]$, as shown in Figure 11 (left). Also common is to plot it in the interval $[-0.5, 0.5]$, Figure 11 (center). And since these frequency response functions are often symmetric about 0, it is good enough to plot just the values in $[0, 0.5]$, Figure 11 (right).

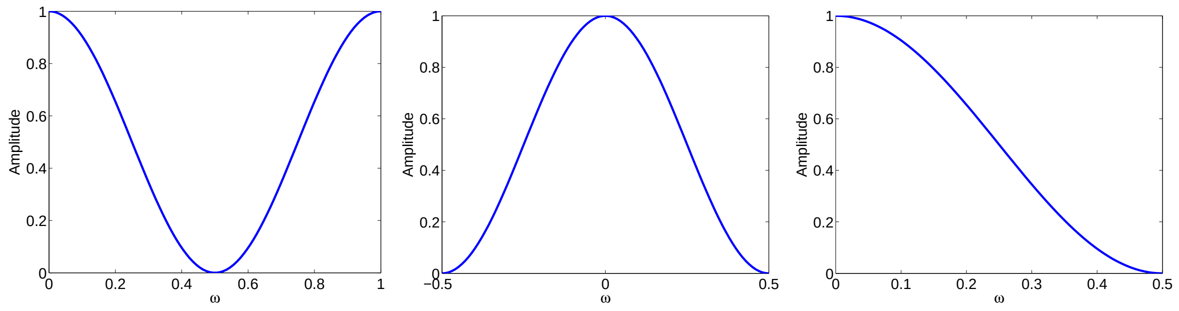

Figure 8: The frequency response curve of a 3 point smoother. Three standard plots.

You should get used to these three plots, as you will see all three in in different contexts. MATLAB often displays the one on $[0,1]$, but also sometimes on $[-0.5, 0.5]$. Make sure you know what you are looking at.

And what do these plots mean? Well, $\omega = 0$ corresponds to a sinusoid with zero frequency – one that does not oscillate at all. $H(0)$ gives the DC response of the system – what it does to constant signals. In our example $H(0) = 1$, so a DC signal is passed unchanged. The case $\omega = 0.5$ corresponds to a sinusoidal signal that is oscillating at 1/2 of the sampling rate. For a system that samples at 10kHz, this would be a 5kHz signal. In our example, $H(1/2) = 0$, so our 3 point smoother will kill off this signals (a 5kHz signal is attenuated to zero amplitude). At an intermediate point, say $\omega = 0.25$, we find $H(0.25) = .5 * (1 + \cos(\pi/2)) = .5$, so a signal at 1/4 the sampling rate will be attenuated by a factor of 1/2. Eg. at 10kHz sampling, the 2.5kHz signal is attenuated by one half.

It is important to remember that ω is a measure of frequency relative to the sampling rate. So, for instance with a sampling rate of 10kHz, the $\omega = 0.5$ corresponds to a signal at frequency 5kHz. For a CD system, the sampling rate is 44.1kHz. In this case, the same $\omega = 0.5$ corresponds to a signal at 22.05kHz. Just remember that ω is relative to the sampling rate (NOT the Nyquist rate).

In our example of the three point smoother, we saw the symmetry $H(-\omega) = H(\omega)$; equivalently, H is even. In the assignment, we saw that for symmetric impulse response, the frequency response is a sum of cosines,

$$H(\omega) = h_0 + 2\sum_k h_k \cos(2\pi k\omega), \tag{101}$$

and thus this $H(\omega)$ is also even. That is, $H(-\omega) = H(\omega)$.

This even property is not true in general. However, we have the following:

Theorem 2 *Suppose the impulse response $\mathbf{h} = (\ldots, h_{-1}, h_0, h_1, \ldots)$ is real-valued. Then the frequency response function $H(\omega)$ satisfies a skew-symmetry,*

$$H(-\omega) = \overline{H(\omega)}, \text{ for all } \omega, \tag{102}$$

where the bar above means complex conjugate.

This is an important theorem since many systems we design must have real coefficients in them, so we get the skew symmetry for free. It also implies the absolute value of the frequency response is an even function, so

$$|H(-\omega)| = |H(\omega)| \text{ for all } \omega, \tag{103}$$

since the absolute value is the same on any complex number and its conjugate.

The proof of the theorem is easy. We just observe the sum at $-\omega$ is

$$\begin{aligned}
H(\omega) &= \sum_k h_k e^{-2\pi i k(-\omega)} \\
&= \sum_k h_k \overline{e^{-2\pi i k\omega}}, \text{ the complex conjuate} \\
&= \sum_k \overline{h_k e^{-2\pi i k\omega}}, \text{ since } h_k \text{ is real}
\end{aligned}$$

$$
\begin{aligned}
&= \overline{\sum_k h_k e^{-2\pi i k\omega}}, \text{ by linearity} \\
&= \overline{H(\omega)},
\end{aligned}
$$

as desired.

32 Magnitude and phase response

Since $H(\omega)$ is a complex number for each value of ω, it can be expanded in the form of a real amplitude $|H| = |H(\omega)| \geq 0$ and a phase $\phi = \phi(\omega)$, with

$$
H(\omega) = |H(\omega)| e^{2\pi i \phi(\omega)}. \tag{104}
$$

Both the amplitude and the phase have a specific physical meaning for action of the system on the sinusoids of frequency ω.

Figure 12 shows this. Here we see a sine wave of a particular fixed frequency ω input into a system, and we plot right next to it the output. We see the output is still basically a sine wave, but with a small height, and shifted to the right. The change in amplitude is given by $A = |H(\omega)|$, the amplitude spectrum, and the amount of shift is given by $\phi = \phi(\omega)$, the phase spectrum. We have to be careful, though: for a positive ϕ, the shift is to the left; for negative ϕ, it is to the right.

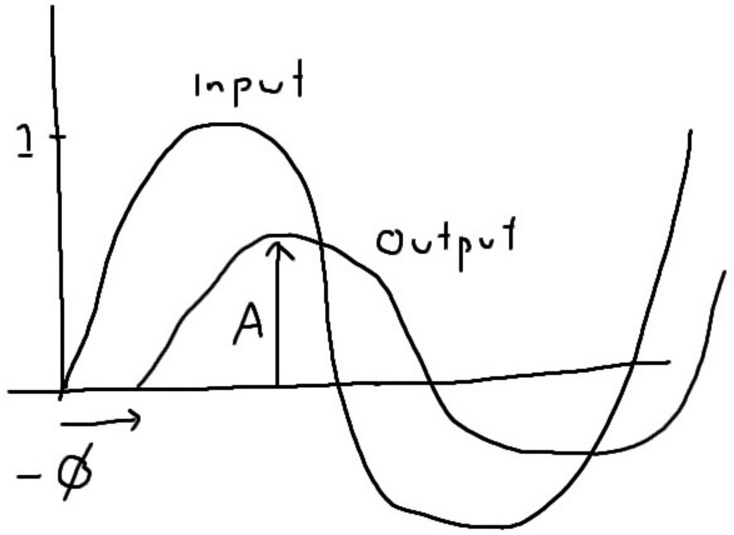

Figure 9: Input/output response with the amplitude and phase effect.

To see this, observe that if we start with a sampled sinusoid $x_n = e^{2\pi i \omega n}$, the

$$
\begin{aligned}
y_n &= H(\omega) x_n \\
&= |H(\omega)| e^{2\pi i \phi(\omega)} x_n \\
&= |H(\omega)| e^{2\pi i (\omega n + \phi(\omega))}
\end{aligned}
$$

which is just a complex sinusoid of the same frequency, but with amplitude $|H(\omega)|$, and shifted (to the left) by phase $\phi(\omega)$. Maybe it is easier to look at the, say, imaginary parts, with

$$\begin{aligned} Im(x_n) &= \sin(2\pi\omega n) \\ Im(y_n) &= |H(\omega)| \sin(2\pi(\omega n + \phi(\omega)) \end{aligned}$$

So we see that the output is just a sine wave, but with amplitude $|H(\omega)|$ and phase shifted by $-\phi(\omega)$.

Think about what ϕ really measures. It is the shift, in units of a cycle at that frequency. So $\phi = 0.5$ corresponds to a shift by exactly half a cycle, at that frequency.

A system that delays by exacly n samples (eg $\mathbf{h} = \delta^n$) , has a linear phase delay, $\phi(\omega) = -n\omega$. Waves at different frequencies get shifted by different number of cycles – which makes since, since for higher frequencies, we have shorter cycles, so we must shift by more of them to get the delay of n samples. (Explain.)

We will be interested in designing systems with specified amplitude and phase characteristics.

33 Designing filters

We can run this process in reverse. Specify the filter response $H(\omega)$ that we want, find the corresponding filter coefficients $\mathbf{h}$, shorten if necessary to a finite impulse response, and we are done.

33.1 FIR - Zeros of polynomials

A FIR filter with coefficients $\mathbf{h} = (h_0, h_1, h_2, \ldots, h_n)$ has z-transform which is a polynomial,

$$H(z) = h_0 + h_1 z + h_2 z^2 + \ldots h_n z^n. \tag{105}$$

Specifying the zeros of the polynomials determines the polynomial (up to a constant). We can use these zeros to control the performance of the filter.

Suppose we want a filter that kills a signal at the Nyquist frequency. This corresponds to a normalized frequency $\omega = 1/2$ and thus a z-value of $z = e^{2\pi i 1/2} = -1$. So, we choose a polynomial with zero at $z = -1$. The simplest such polynomial is

$$H(z) = 1 + z. \tag{106}$$

However, at zero frequency $\omega = 0$, this filter takes value $H(e^{2\pi i 0}) = 1 + 1 = 2$, which means the zero frequency signal (DC) is boosted by a factor of 2. A better choice of filter is the normalized version

$$H(z) = 0.5 + 0.5z. \tag{107}$$

This filter has coefficients

$$\mathbf{h} = (0.5, 0.5), \tag{108}$$

which is the smoothing filter discussed earlier. This is a simple version of a lowpass filter (it passes the low frequencies, and tries to block the high frequencies).

A higher order lowpass filter can be created by repeating the zeros of the polynomial. For instance, take

$$H(z) = (1/8) * (1+z)^3 = 1/8 + (3/8)z + (3/8)z^2 + (1/8)z^3. \tag{109}$$

This filter has coeffcients

$$\mathbf{h} = (1/8, 3/8, 3/8, 1/8), \tag{110}$$

which is also a smoothing filter.

A high-pass filter is created by putting a zero at DC frequency, $\omega = 0$,so $z = 1$. We should also normalize. A good choice is

$$H(z) = .5 - .5z. \tag{111}$$

The filter coefficients for this differencing filter are

$$\mathbf{h} = (.5, -.5). \tag{112}$$

A higher order high-pass filter is obtained by taking powers of this polynomial, so with $H(z) = (.5 - .5z)^3$ we get coefficients

$$\mathbf{h} = (1/8, -3/8, 3/8, -1/8), \tag{113}$$

which is also a differencing filter.

An Example: Build a simple filter that blocks noise at 120Hz, on a signal with a sample rate of 1000Hz.

The plan is to set a zero at the point on the unit circle corresponding to normalized frequency $\omega = 120/1000 = 0.12$. This give the zero at

$$z_0 = \exp(2\pi i * 0.12) = 0.7290 + 0.6845i. \tag{114}$$

However, we want a polynomial that has real coefficients, so we throw in a second zero at the complex conjugate

$$\overline{z_0} = 0.7290 - 0.6845i. \tag{115}$$

The corresponding polynomial is

$$H(z) = (z - z_0)(z - \overline{z_0}) = z^2 - 1.458z + 1. \tag{116}$$

The corresponding filter coefficients are

$$\mathbf{h} = (1, -1.458, 1). \tag{117}$$

It might be nice to normalize this filter so that $H(1) = 1$. That is, DC gets passed with no amplitude change. So we take

$$\mathbf{h} = (1, -1.458, 1)/(.542) = (1.845, -2.69, 1.845). \tag{118}$$

Another choice for normalization is to try for $\max |H(e^{2\pi i\omega})| = 1$, in which case we choose

$$\mathbf{h} = (1, -1, 458, 1)/(3.458) = (.2892, -0.4216, 0.2892). \tag{119}$$

33.2 FIR - Fourier coefficients

While it is true that a polynomial is determined by its zeros, it is not a great way to design more complicated filters. A better way is to specify the amplitude (and phase) response that we want, and find a smart way to determine the corresponding filter coefficients.

Here is one way.

1. specify the filter response $H(\omega)$ as a function of normalized frequency ω;
2. compute the coefficients **h** using the inverse Fourier transform;
3. truncate the list of coefficients so we get a finite sequence of non-zero coefficients. This will approximate the filter response we want.

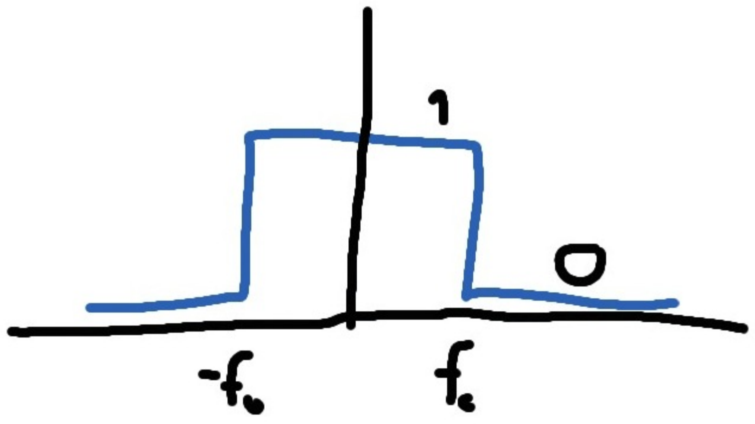

Figure 10: An ideal brick-wall style low pass filter response.

Here is an example. Suppose we want a low-pass filter, that passes frequencies up to some F_0, for a sampling rate Fs. In normalized frequencies, we pass everything up to some $f_0 = F_0/Fs$, and then cut off. Using symmetry, we require our filter response to be

$$H(\omega) = 1, \text{ for } -f_0 \leq \omega \leq f_0 \ , \tag{120}$$

and zero everywhere else. This is illustrated in Fig 10.

The inverse Fourier transform gives us coefficients

$$\begin{aligned} h_n &= \int_{-0.5}^{0.5} H(\omega) e^{-2\pi i n \omega} \, d\omega & (121) \\ &= \int_{-f_0}^{f_0} e^{-2\pi i n \omega} \, d\omega & (122) \end{aligned}$$

$$\begin{aligned} &= 2\int_0^{f_0} \cos(2\pi n\omega)\, d\omega \text{ by symmetry} \quad &(123)\\ &= 2\,\frac{\sin(2\pi n f_0)}{2\pi n} &(124)\\ &= 2f_0 \operatorname{sinc}(2\pi n f_0), &(125)\end{aligned}$$

where $\operatorname{sinc}(x) = \sin(x)/x$.[1] However, the problem is that this vector $\mathbf{h}$ has infinitely many non-zero values. What we can do is select a subset of them. For instance, let's take the centre 7 elements, with cutoff frequency $f_0 = 0.2$. This give filter coefficients

$$\begin{aligned} \mathbf{h} &= (h_{-3}h_{-2}, h_{-1}, h_0, h_1, h_2, h_3) &(126)\\ &= (-0.0624, 0.0935, 0.3027, 0.4000, 0.3027, 0.0935, -0.0624) &(127)\end{aligned}$$

.

The truncated filter response is shown in Figure 11, on the left side. We notice there is a ramp down at about 0.2 in normalized frequency, but it is not very much like a brick wall response. A better response is obtained by using more filter coefficients. Using the centre 31 elements, we get a longer filter (31 taps) with the filter response shown on the right side of Figure 11. Notice the steep cutoff at frequency $f_0 = 0.2$, as desired.

The ripples in the curve are not desired. This is an example of Gibbs phenomena, and in the next section we discuss how to reduce these ripples.

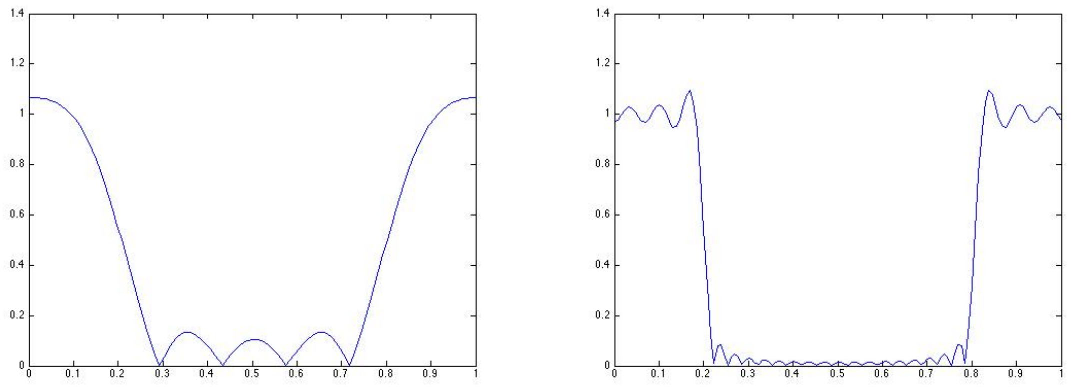

Figure 11: Low pass filter response with 7 taps (left) and 31 taps (right).

33.3 Better filters - windowing

The key to getting rid of the ripples (Gibbs phenomena) is rather than chopping off to zero that sequence of filter coefficients $h_n = 2f_0 \operatorname{sinc}(2\pi f_0 n)$, we should reduce them slowly to zero. Figure 12 shows this, with the original coefficients on the left, and the slowly reduced coefficients on the right.

[1]Warning: MATLAB defines sinc(x) as $\sin(\pi x)/\pi x$. Get used to it!

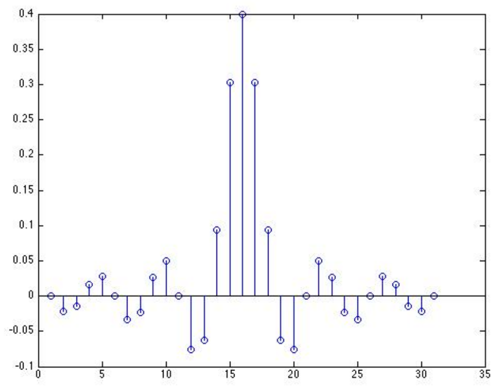
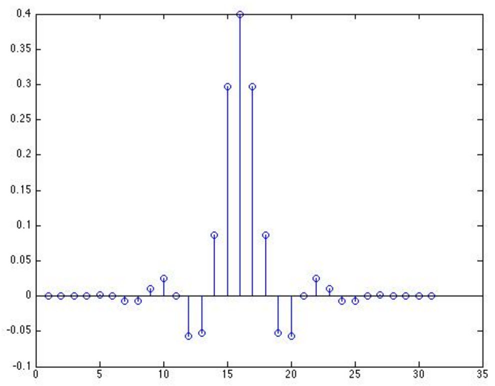

Figure 12: 31 filter coefficients, suddenly cut to zero (left) and gradually reduced to zero (right).

The resulting improvement is shown in Figure 13, where on the left we have the original filter response for the 31 tap filter, and on the right, we have the improved filter response. Notice in the improved version, the ripples are gone. We also see the nice drop off at normalized feequency $f_0 = 0.2$, show the transition from the lowpass band to the high frequency cutoff. However the cutoff on the right is not so steep as on the left – this is the price you pay for getting rid of the ripples.

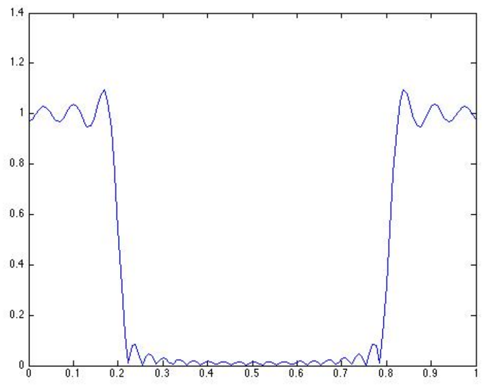
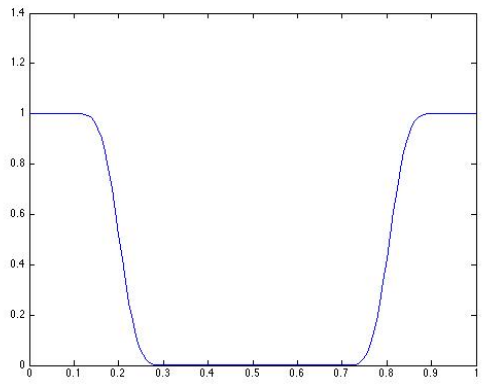

Figure 13: Low pass filter response, original (left) and modified (right).

The details of how to do this gradual reduction is called windowing. The idea is to replace h_n with a modified version

$$h'_n = w_n \cdot h_n, \tag{128}$$

where the sequence w_n (called the window function) is selected so that it gradually reduces to zero. Figure 14 shows a 31 tap Blackman window. The formula for an N-tap Blackman filter is

$$w_n = .42 - .5\cos(2\pi(k-1)/(N-1)) + .08\cos(4\pi(k-1)/(N-1)), \text{ for } n = 1, 2, 3, \ldots, N. \tag{129}$$

(I know, my indices are inconsistent. Just centre everything!.)

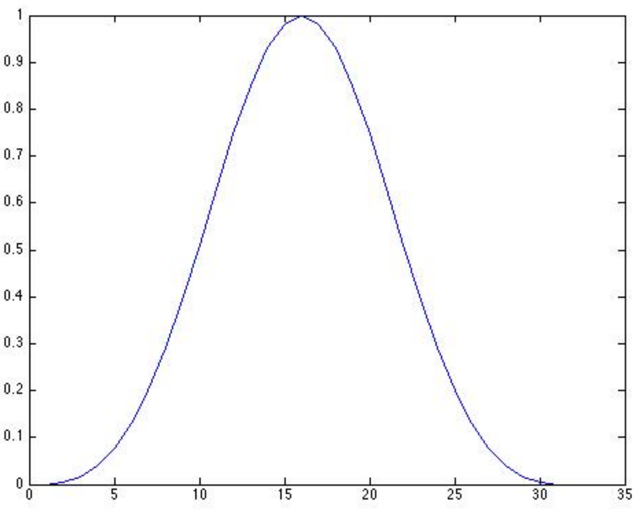

Figure 14: The shape of a Blackman window with 31 taps.

Other windows practitioners typically use are Hann, Hamming, Kaiser, and Chebyshev windows. There are whole books written on just the windows alone!

33.4 IIR - zeros and poles

Infinite Impulse Response (IIR) filters are similar to FIR filters, except they use both input and output values in computing new output values. Two simple examples clarify the difference:

$$\begin{aligned} y_n &= x_n + x_{n-1}, \text{ a simple FIR filter,} \\ y_n &= x_n + y_{n-1}, \text{ a simple IIR filter.} \end{aligned}$$

IIR filters are also called recursive filters because of the way old output values (like y_{n-1}) feed back into the new input value.

To clarify, think of the x_n as entries in an input data stream, and the y_n as output values that are computed on the fly as the data comes in. From the formula $y_n = x_n + y_{n-1}$, we can compute with a sample input

$$\mathbf{x} = (1, 2, 3, 4, 0, 0, 0, \ldots) \tag{130}$$

the output values

$$\begin{aligned} y_0 &= x_0 + y_{-1} = 1 + 0 = 1, \\ y_1 &= x_1 + y_0 = 2 + 1 = 3, \\ y_2 &= x_2 + y_1 = 3 + 3 = 6, \\ y_3 &= x_3 + y_2 = 4 + 6 = 10, \\ y_4 &= x_4 + y_3 = 0 + 10 = 10, \end{aligned}$$

$$
\begin{aligned}
y_5 &= x_3 + y_2 = 0 + 10 = 10, \\
\dots
\end{aligned}
$$

giving output vector $\mathbf{y} = (1, 3, 6, 10, 10, 10, \ldots)$. Incidentally, in this example if we chose $\mathbf{x}$ as the delta spike $\mathbf{x} = (1, 0, 0, 0, \ldots)$, we compute an output vector $\mathbf{h} = (1, 1, 1, 1, \ldots)$ which is the corresponding impulse response of the filter. The correspond z-transform for the filter is computed from the impulse response as

$$H(z) = 1 + z + z^2 + z_3 + \cdots = \frac{1}{1-z}, \tag{131}$$

where we have summed the geometric series to get the rational function $1/(1-z)$.

A more general IIR filter looks something like this:

$$
\begin{aligned}
y_n &= 2x_n + 3x_{n-1} + 4x_{n-2} && (132) \\
&\quad +5y_{n-1} + 6y_{n-2} + 7y_{n-3} + 8y_{n-4} && (133)
\end{aligned}
$$

Note both x and y values appear on the right hand side; but the$y's$ only contained "earlier" values y_{n-1}, y_{n-2}, etc. The z-transform for this filter is given as

$$H(z) = \frac{2 + 3z + 4z^2}{1 - 5z - 6z^2 - 7z^3 - 8z^4}. \tag{134}$$

Notice that the coefficients from the x_n's appear in the numerator, while the coefficients of the y_{n-k} appear in the denominator, with reversed signs. It's not hard to see why; the filter equation 133 can be rewritten as

$$y_n - 5y_{n-1} - 6y_{n-2} - 7y_{n-3} - 8y_{n-4} = 2x_n + 3x_{n-1} + 4x_{n-2} \tag{135}$$

which can be expressed as a convolution

$$\mathbf{y} * [1, -5, -6, -7, -8] = \mathbf{x} * [2, 3, 4]. \tag{136}$$

Taking z-transforms (and using the convolution theorem that says a convolution converts to a product under the z-transform), we get

$$Y(z)(1 - 5z - 6z^2 - 7z^3 - 8z^4) = X(z)(2 + 3z + 4z^2). \tag{137}$$

By division, we have

$$Y(z) = \frac{2 + 3z + 4z^2}{1 - 5z - 6z^2 - 7z^3 - 8z^4} X(z), \tag{138}$$

so the rational function $H(z)$ appears as the transfer function between X and Y.

Computing the impulse response involves a bit of arithmetic, and doesn't usually lead to a nice pattern. With $\mathbf{x} = (1, 0, 0, 0, 0, \ldots)$ we find in this example that the impulse response output is

$$
\begin{aligned}
y_0 &= 2x_0 + 3x_{-1} + 4x_{-2} + 5y_{-1} + 6y_{-2} + 7y_{-3} + 8y_{-4} = 2 + 0 = 2 \\
y_1 &= 2x_1 + 3x_0 + 4x_{-1} + 5y_0 + 6y_{-1} + 7y_{-2} + 8y_{-3} = 3 * 1 + 5 * 2 = 13 \\
y_2 &= 2x_2 + 3x_1 + 4x_0 + 5y_1 + 6y_0 + 7y_{-1} + 8y_{-2} = 4 * 1 + 5 * 13 + 6 * 2 = 81 \\
&\mathit{etc.}
\end{aligned}
$$

MATLAB's filter command can do this faster for you.

The point is, while an FIR filter is described by a polynomial (using the z-transform), an IIR filter is described by a rational function

$$H(z) = \frac{p(z)}{q(z)}, \tag{139}$$

which is just a ratio of polynomials. The zeros of polynomial $p(z)$ are the zeros of $H(z)$, while the zeros of denominator polynomial $q(z)$ are the poles of $H(z)$, the places where H "blows up." The number of zeros and poles depends on the order of the numerator and denominator polynomials, respectively.

The poles and zeros are located in the complex plane. To get a stable filter (one that gives a bounded output, for any bounded input), you need to put the poles outside the unit circle. (This is a theorem.)[2] The location of the zeros does not affect stability.

Since FIR filters have no poles, they are always stable.

Since polynomials with real coefficients have zeros occurring in complex conjugate pairs, filters with real coefficients will have both zeros and poles in conjugate pairs.

Here's an interesting fact. The amplitude response of a filter $|H(e^{2\pi i\omega})|$ is not changed by replacing a pole or zero with the complex conjugate of its reciprocal. For instance, the two z-transforms

$$H(z) = 1 - 2z, \qquad K(z) = 2 - z \tag{140}$$

have zeros at $z = 1/2$ and $z = 2$, respectively (these are reciprocals). They have the same amplitude response, since

$$\begin{aligned}
|H(e^{2\pi i\omega})| &= |1 - 2e^{2\pi i\omega}| \\
&= |(e^{2\pi i\omega})(e^{-2\pi i\omega} - 2)| \\
&= |(e^{2\pi i\omega})| \cdot |\overline{(e^{2\pi i\omega} - 2)}| \\
&= 1 \cdot |(e^{2\pi i\omega} - 2)| \\
&= |K(e^{2\pi i\omega})|.
\end{aligned}$$

The same thing works with poles. Why conjugate reciprocals? Think about this:

$$|z - z_o| = |z(1 - \overline{z}z_o)| = |z||\overline{1 - \overline{z}z_o}| = |1 - z\overline{z_o}| \tag{141}$$

for points z on the unit circle. The monomial $(1 - z\overline{z_o})$ has a zero at $1/\overline{z_o}$, which is the complex conjugate of the reciprocal of z_o.

You can swap zeros (or poles) in and out of the the unit circle by replacing with conjugate reciprocals; this does not change the magnitude response of the filter. It does, however, change the phase response.

[2]Note,in electrical engineering, the z-transform convention is reversed. So they talk about putting the zeros and poles inside the unit circle. Sorry for the confusion, just be aware of it.

IIR filters can have really weird phase response, mainly because the poles can really mess things up. Often, we don't care. Sometimes, it is really important. In particular, an IIR filter can really mess up the shape of a pulse. So be careful.

To find the phase response of an IIR filter, usually you have to compute it numerically. The MATLAB command freqz does this for you. Be careful, though, which is the numerator, which is the numerator polynomial.

So why do people use IIR filters? In general, because you can get steep cutoffs, using only a few coefficients. And to get interesting phase response. In principle, they are faster to implement as well.

33.5 IIR - specialty filters

Butterworth, Elliptic, Chebyshev, etc. These are specially designed lowpass, high pass, or bandpass filters. They are IIR filters, and have special low-ripple characteristics in the pass band, or the reject band. Probably too much to know for this class.

But very useful. Read about them in the course textbook, or MATLAB notes.

33.6 IIR - inverse filters

Recall this IIR filter

$$\begin{aligned} y_n &= 2x_n + 3x_{n-1} + 4x_{n-2} \\ &\quad +5y_{n-1} + 6y_{n-2} + 7y_{n-3} + 8y_{n-4}, \end{aligned}$$

which has z-transform

$$H(z) = \frac{2 + 3z + 4z^2}{1 - 5z - 6z^2 - 7z^3 - 8z^4}. \tag{142}$$

The inverse filter will have z-transform which is the reciprocal

$$K(z) = \frac{1 - 5z - 6z^2 - 7z^3 - 8z^4}{2 + 3z + 4z^2}. \tag{143}$$

We normalize, dividing top and bottom by 2 to get a "1" on the bottom, so

$$K(z) = \frac{.5 - 2.5z - 3z^2 - 3.5z^3 - 4z^4}{1 + 1.5z + 2z^2}. \tag{144}$$

The recursion formula for this filter is thus

$$\begin{aligned} y_n &= .5x_n - 2.5x_{n-1} - 3x_{n-2} - 3.5x_{n-3} - 4x_{n-4} \\ &\quad -1.5y_{n-1} - 2y_{n-2}. \end{aligned}$$

This was a complicated example, yet it is pretty clear how to find the inverse filter just by flipping the z-transform.

A simpler example is to take the filter $y_n = x_n + y_{n-1}$ which has impulse response $\mathbf{h} = (1, 1, 1, 1, 1, \ldots)$. The inverse filter is an FIR filter, with impulse reponse $\mathbf{k} = (1, -1)$. It is a simple matter to check that the convolution gives

$$h * k = (1, 0, 0, 0, \ldots), \text{ the delta spike,} \tag{145}$$

which in z-transform terms just says

$$H(z)K(z) = 1. \tag{146}$$

That is

$$\frac{1}{1-z}(1-z) = 1. \tag{147}$$

In general, any FIR or IIR filter has an inverse, which is typically another IIR filter. Computing them is easy – you just work from the z-transform and take reciprocals. This raises the interesting possibility of undoing any linear filter operations.

For instance, in seismic, the reflectivity of a layered earth model can be represented by a time series r. The seismic data recorded, d, is givens a convolution of reflectivity with the source waveform s (e.g. a dynamite blast), with

$$d = r * s. \tag{148}$$

Reflectivity is recovered by convolving with the inverse of s, so

$$d * s^{-1} = r * s * s^{-1} = r * \delta_o = r, \tag{149}$$

where δ_o is the delta spike which acts as the convolutional identity.

33.7 IIR - minimum phase delay filters

An IIR filter which is causal, stable, with stable inverse, is called minimum phase delay. To get a minimum phase delay filter, you need to put both the poles and the zeros outside the unit circle.

Minimum phase is a physical property – it means most of the energy of a signal stays concentrated at the front (start) of the signal.

34 Filtering in the Fourier transform domain

Recall that convolution in the time domain gets changed into multiplication in the frequency domain. This suggests we can do filtering in the Fourier transform domain by simple multiplication. More precise, instead of doing a filter by convolution, in the form

$$\mathbf{y} = \mathbf{h} * \mathbf{x}, \tag{150}$$

where $\mathbf{x}, \mathbf{y}$ are the input and output, in the Fourier transform domain we write

$$Y(\omega) = H(\omega)X(\omega), \tag{151}$$

with H the frequency response of the filter operation. This leads to a 3 step procedure for filtering:

1. Take FT of sequence $\mathbf{x}$ to get its transform $X(\omega)$.

2. Multiply X by the filter response H to get the result $Y(\omega) = H(\omega)X(\omega)$

3. Take the inverse FT of Y to obtain the output sequence $\mathbf{y} = ifft(Y)$.

There are a number of practical considerations to keep in mind, however:

1. DFT: On the computer, we use the discrete Fourier transform (DFT);

2. Speed: The DFT takes about $N \cdot \log_2 N$ operations to complete, where N is the length of the signal. For long signals, this might be slow. It is fastest when N is a power of 2. The multiplication by $H(\omega)$ takes only N operations, so it is the DFT that slows things down.

3. Wrap-around: The DFT suffers from "‘wrap-around" effects, where energy near the end of the signal gets "leaked" into the front end. Essentially, the DFT treats the signal as if it were on a circle! (Wrap-around can be eliminated by using zero-padding.)

4. Symmetry: the freq response $H(\omega)$ should be carefully designed to preserve symmetry of real signals. (In particular, we want $H(k/N) = \overline{H(1 - k/N)}$ for all indices k in the freq domain.

5. Freq bins: Notice that the k-th bin in the frequence domain corresponds to frequency $(k/N) * F_s$, N is the length of the signal, and F_s is the sampling rate. $k = 0$ is the DC component. Often, people mess this up!

6. Transients: Sharp edges in the graph of $H(\omega)$ cause nasty transients in the filtered output (Gibbs phenomena). You might want to use a smoother function for H.

7. Zero-phase: Real-valued $H(\omega)$ lead to zero-phase filters. These have the unusual property of sending energy both forwards and backwards in time, which may not be the effect you want.

In Figure 15 we show the results of a straightforward low pass filter, used to remove the high frequency component in an input signal. The MATLAB code to implement this filter is simply y = ifft(h.*fft(x)).

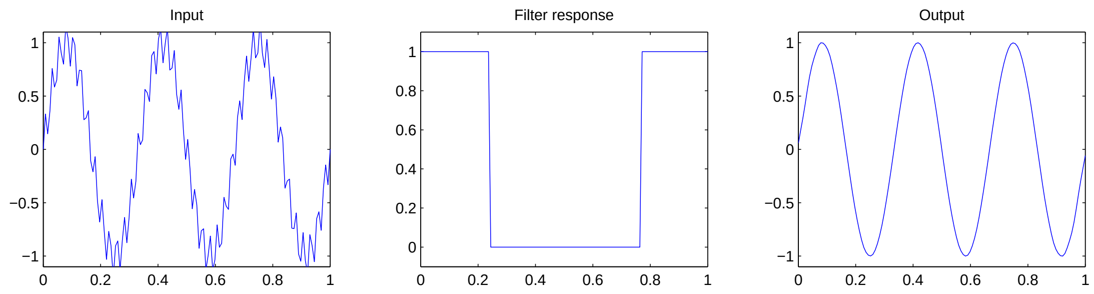

Figure 15: An input signal, lowpass response, and the resulting output.

The vector h is the frequency response of the filter, set up here as a symmetry vector:

h = [1,ones(1,30),zeros(1,Fs-61),ones(1,30)];

The symmetry is important – notice the two strings of ones match, and there is one extra one at DC component. Without this careful matching of symmetry, the output will not necessarily be real-valued.

In FIgure 16, we see the result of lowpass filtering a delta spike. The output is a bit of a spike, but with many wiggles (the transients) on either side of the central peak. These transients are another illustration of Gibbs phenomena. The fact that the central peak is surrounded on both sides by transients shows this result is NOT causal – energy is produced at times BEFORE the input spike. This doesn't happen with IIR and FIR filters; it is very common in the FT domain filtering.

Just be aware that some applications of filtering require causal filters (e.g. real time filtering of audio), while others require zero phase (e.g. deconvolution of seismic data).

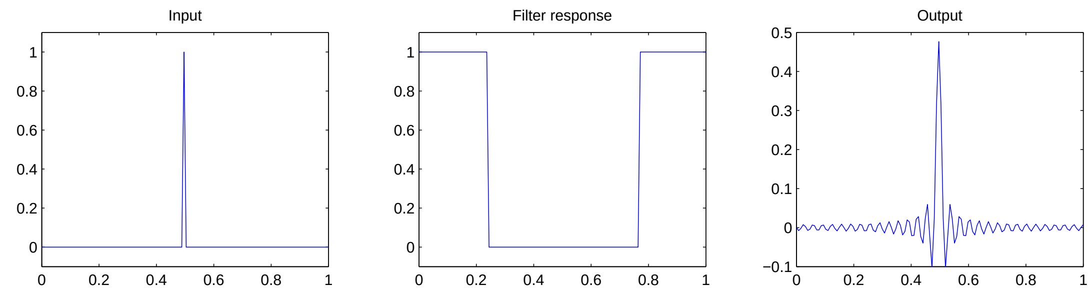

Figure 16: An input spike, lowpass response, and the output showing transients.

To reduce transients, we can smooth out the lowpass filter response. We insert a linear ramp in the filter response h, to smoothly connect the amplitude values of 1 to amplitude values of zero. As a MATLAB vector, we can write, for instance

h = [1,ones(1,20),linspace(1,0,10),zeros(1,Fs-61),linspace(0,1,10),ones(1,20)];

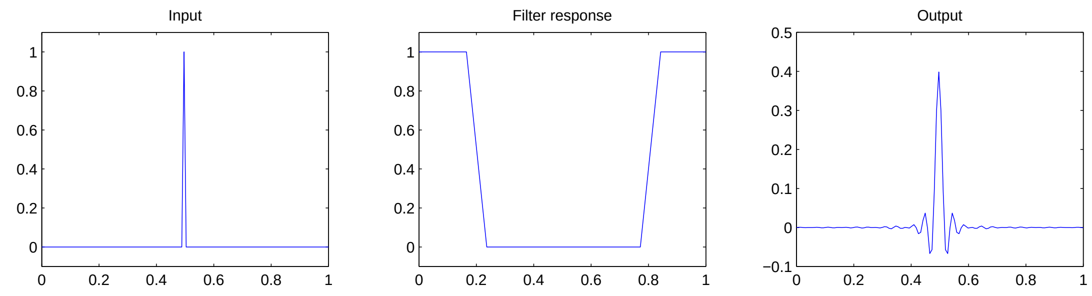

Figure 17: An input spike, lowpass response, and the output showing transients.

The last problem to observe is the wrap-around effect. The problem is that the FFT treats the data on an interval as if the end of the signal is connected to the front of the signal – visualize wrapping an interval into a circle. To observe this effect, consider Figure 18. Here, the input signal has a delta spike near the end of the data. The output is a filtered spike, with transients. Some of these transients get leaked to the front of the output signal.

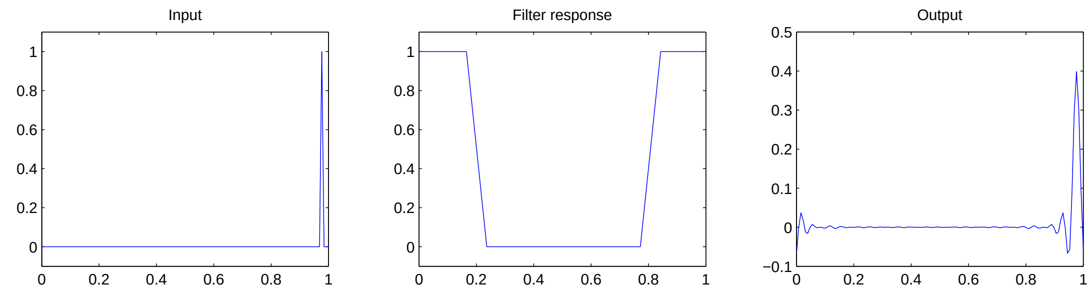

Figure 18: An input spike, filtered result wraps around from end to start.

35 Speed of the Fourier transform

The formula for the discrete Fourier transform

$$y_k = \sum_{n=0}^{N-1} e^{2\pi ikn/N} x_n \tag{152}$$

shows the transform is an $N \times N$ matrix transform, which in principle takes about N^2 operations to complete. For a moderately size signal of length 1,000, this corresponds to one million operations to complete.

Some speedup is possible simply by noticing that there are only 1000 unique exponential values that appear in the summation, so these can be pre-computed, leading to faster operations.

There is a divide-and-conquer algorithm (also called a decimation method) that computes the DFT in about $N \log_2 N$ operations. WIth $N = 1000$, this amounts to 10,000 operations, a big speed up over the direct method. The key to the method is to break up the input of length N into two shorter vectors of length $N/2$. The DFT applied to these two takes about $(N/2)^2$ operations; some nifty algebra shows the full DFT takes only $(N/2)^2 + N$ operations. This is then repeated over and over. As an example, we can see the operations count decrease as follows (say with $N = 64 = 2^6$).

$$\begin{aligned}
N^2 = 64^2 \quad &\rightarrow \quad (64/2)^2 + 64 = 32^2 + 64 \\
&\rightarrow \quad (32/2)^2 + 64 + 64 = 16^2 + 2 * 64 \\
&\rightarrow \quad (16/2)^2 + 64 + 64 + 64 = 8^2 + 3 * 64 \\
&\rightarrow \quad (8/2)^2 + 64 + 64 + 64 + 64 = 4^2 + 4 * 64 \\
&\rightarrow \quad (4/2)^2 + 64 + 64 + 64 + 64 + 64 = 2^2 + 5 * 64 \\
&\rightarrow \quad (2/2)^2 + 64 + 64 + 64 + 64 + 64 + 64 = 1 + 6 * 64 = 1 + 64 * \log_2(64).
\end{aligned}$$

J.W. Cooley and John Tukey made this algorithm famous in 1965. It works best when N is a power of 2, like 64, 126, 256, 1024, etc.

No one in their right mind would code this themselves anymore, as really fast implementations have already been worked out by the experts. See the webpage FFTW.org for the fastest code out there!

36 Hilbert transforms

The Hilbert transform is a sweet little filter that shifts a sine wave, or cosine wave by exactly 90 degrees – independent of the frequency. Why it works is not as interesting as the fact that it does work.

As an example, look at Figure 19. The input, we see a cosine wave, the output is the same wave, but shifted by 90 degrees – it appears as a sine. I, Figure 20, we see a sine wave as input, and the output is shifted by 90 degrees, giving a negative cosine of the same frequency.

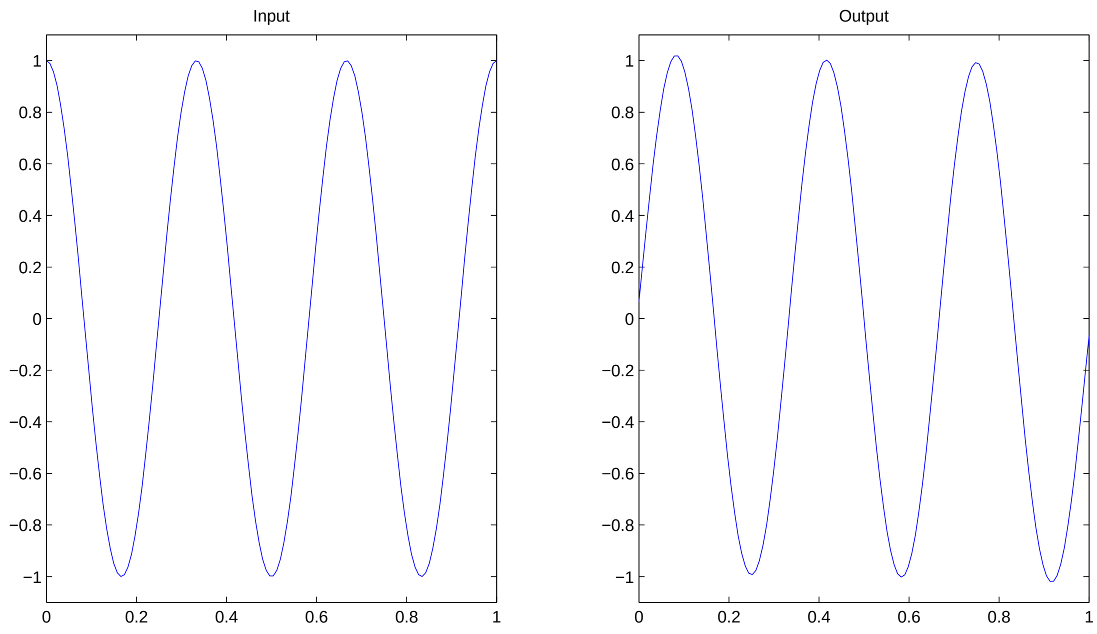

Figure 19: Hilbert transform of a cosine – shifts it by 90 degrees.

Implementing the Hilbert transform in the Fourier domain is quite easy. The hint is that a cosine $\cos(\omega) = e^{\omega} + e^{-\omega}$ must become a sine $\sin(\omega) = (e^{\omega} - e^{-\omega})/i$. So we multiply the positive frequency components by $1/i$, and the negative frequency components by $-1/i$.

As an example, in MAtLAB we set up the response vector

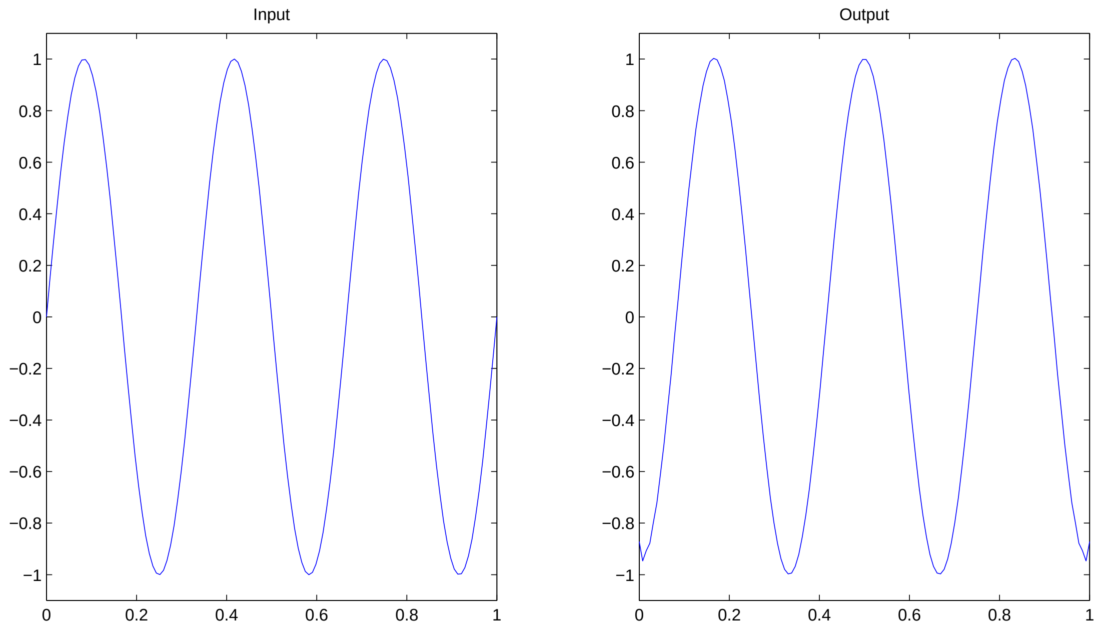

Figure 20: Hilbert transform of a sine – shifts it by 90 degrees.

h = [0,-i*ones(1,(Fs/2)-1),0,i*ones(1,(Fs/2)-1)]

and implement the Hilbert transform as y = iffy(h.*fft(x)). What could be simpler?

The applications are many. For instance, the Hilbert transform can be used to create minimum phase filters with a specified amplitude response. They are also used to correct for improper phase rotations in seismic deconvolution. They are used in envelop detection, production of quadrature signals, among other things.

The impulse response of the Hilbert transform is shown in Figure 21. It is an approximation to the function 1/n on the range $[-N/2, N/2]$. It is an interesting exercise in complex integrals to compute the coefficient values for the impulse response.

Here is a typical application of the Hilbert transform. Recall in Figures 16 and 17, there were these unusual wiggles around both sides of the filtered output of a delta spike. These wiggles are not physical – they have energy the appears before the time of the spike, which is not what we expect in the real world. A correction to this is to use a minimum phase delay filter – one that concentrates energy at the start time of the input signal, not before (and not after). Figure 22 shows the result of using a minimum phase delay filter. Note in the output, there are wiggles after the peak – but always to the right of the peak, corresponding to a time AFTER the input spike. Which is more physical.

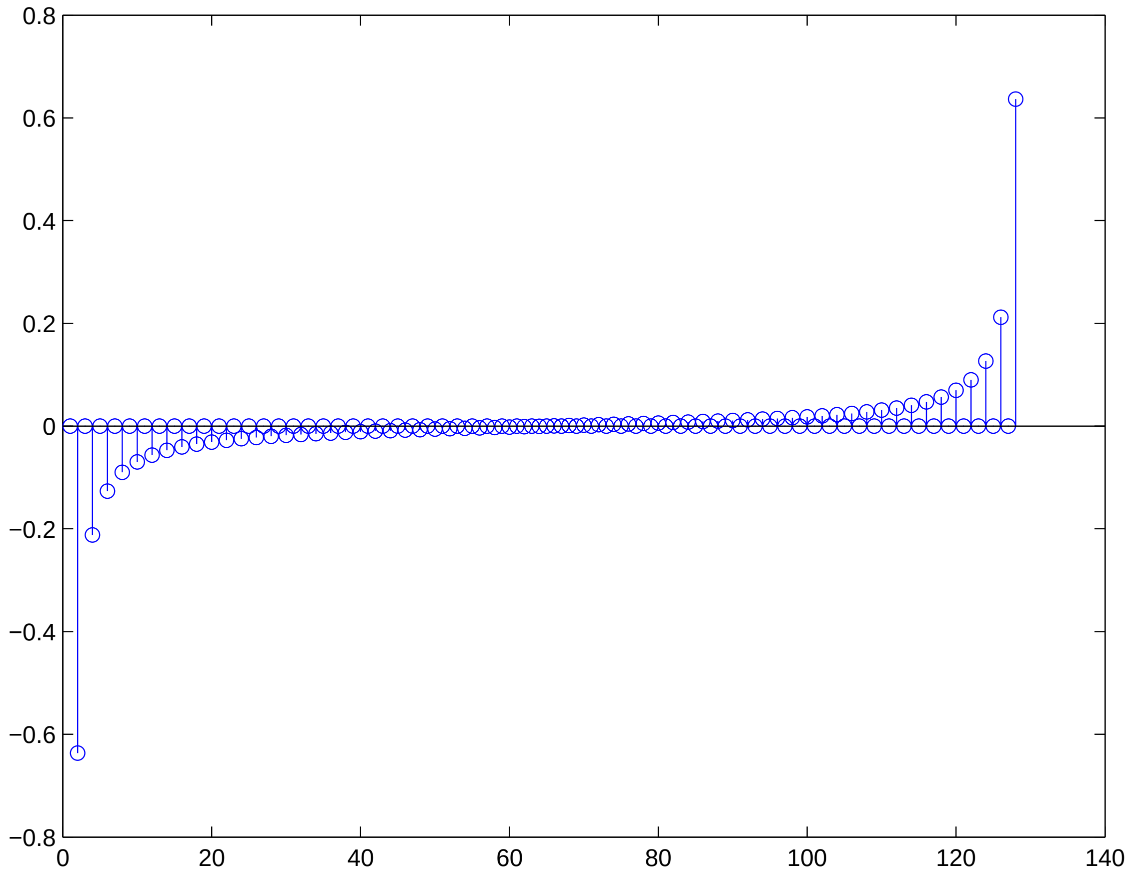

Figure 21: The impulse response of a Hilbert transform.

How is this minimum phase delay achieve? With $H(\omega)$ the desired amplitude response of the filter, we set

$$\begin{aligned} U(\omega) &= \log|H(\omega)| \\ V &= \text{Hilbert(U)},\ \text{the Hilbert transform of } U \\ K(\omega) &= \exp(U(\omega) + i \cdot V(\omega)). \end{aligned}$$

Then $K(\omega)$ is the frequency response of the minimum phase filter that implements the amplitude response we want.

Some MATLAB code might help to make this example clear:

```
Fs = 128;
t = linspace(0,1,Fs);
h = [1,ones(1,20),linspace(1,0.05,10),.01*ones(1,Fs-61),linspace(.05,1,10),ones(1,20)];
```

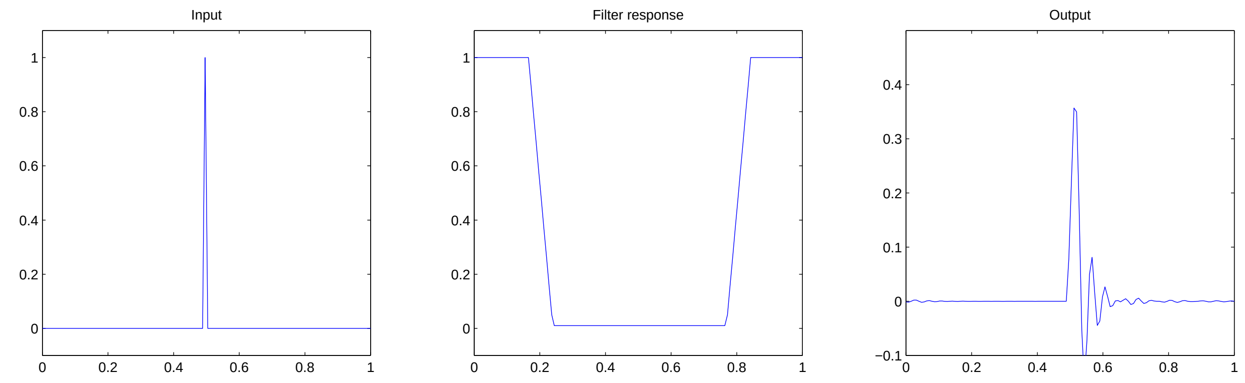

Figure 22: The minimum phase delay filtering of a spike. Wiggles on right only.

```
u = log(abs(h));
v = ifft(fft(u).*[0,-ones(1,(Fs/2)-1),0,ones(1,(Fs/2)-1)]*1i);
k = (exp(u-1i*v));
x=zeros(1,Fs); x(Fs/2) = 1;
y = ifft(k.*fft(x));
subplot(1,3,1); plot(t,x); ylim([-.1,1.1]);title('Input')
subplot(1,3,2); plot(t,abs(k)); ylim([-.1,1.1]);title('Filter response')
subplot(1,3,3); plot(t,y); ylim([-.1,.5]);title('Output')
```

Notice we have to ensure that $H(\omega)$ is non-zero, otherwise the logarithm is undefined. In the MATLAB code, we switched signs to $\exp(U - i * V)$ because of the different sign conventions used in the fft.

Minimum phase delay filters come up a lot in physical modelling, because nature is often minimum phase – dynamite blasts, seismic wave attenuation, all involve minimum phase processes.

37 Circular convolution

Recall that the Fourier transform converts a convolution of sequences into a product of functions (on the interval), and vice versa. The discrete Fourier transform almost does this – it converts a "circular" convolution of vectors (of length n) into a product of vectors (of length n). Symbolically, we write

$$DFT(x *_c y) = DFT(x) \cdot DFT(y), \tag{153}$$

where $*_c$ stands for circular convolution.

The formula for circular convolution is the same as regular convolution, except the indices are computed modulo n, the length of the vectors. So, for vectors

$$\mathbf{x} = (x_0, x_1, x_2, \ldots x_{n-1}), \quad \mathbf{y} = (y_0, y_1, y_2, \ldots y_{n-1}), \tag{154}$$

the circular convolution result $\mathbf{z} = x *_c y$ is given by the formula

$$z_k = \sum_{j=0}^{n-1} x_j y_{k-j \text{ (mod } n)}. \tag{155}$$

As an example, with $n = 4$, we have vectors

$$\mathbf{x} = (x_0, x_1, x_2, x_3), \quad \mathbf{y} = (y_0, y_1, y_{2,3}), \tag{156}$$

and circular convolution given by

$$\begin{aligned} z_0 &= x_0y_0 + x_1y_3 + x_2y_2 + x_3y_1 \\ z_1 &= x_0y_1 + x_1y_0 + x_2y_3 + x_3y_2 \\ z_2 &= x_0y_2 + x_1y_1 + x_2y_0 + x_3y_3 \\ z_3 &= x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0 \end{aligned}$$

Circular convolution is not very physical – and it is the source of the wrap-around error we saw in the section on Filtering in the Fourier transform domain. In particular, circular convolution shows this unusual wraparound effect of data at the end leaking into data at the beginning of the vectors. There is an easy fix – zero padding. Make the vectors longer, with the extra entries all zero. This will fix the problem. (Why? Test it yourself.)

38 Speed of convolution

The convolution formula

$$z_k = \sum_{j=0}^{n-1} x_j y_{k-j \text{ (mod } n)} \tag{157}$$

shows it takes about $2n$ operations to compute each value z_k, for a total of about $2n^2$ operations to compute the full convolution. For moderate values of n, this still can be a lot of operations. (For instance $n = 1000$ leads to two millions operations.)

The Fast Fourier transform works in about $n \log_2 n$ operations, so we can compute a convolution much more quickly using the formula

$$\mathbf{x} *_c \mathbf{y} = \text{ifft}(\text{fft}(\mathbf{x}) \cdot \text{fft}(\mathbf{y})). \tag{158}$$

Note there are 3 calls to the fft/ifft routines which takes about $3n \log_n$ operations, and another n operations to do the multiplications of the two fft results.

Correlations can be computed by flipping the $\mathbf{y}$ vector, or by taking complex conjugates of the fft result. This gives a fast way of computing autocorrelations as well.

Underlying the circular convolution is the fact that the integers modulo n form a group, and any such group has convolution defined on it. Groups also have Fourier transforms, and the connection between Fourier transforms, convolution and products hold much more generally. This is the basis for the study of Harmonic Analysis, a very deep and rich area of mathematics.

39 Fourier transforms in 2 dimensions and higher

We have been talking mainly about one-dimension signals like audio, music, seismic recordings. Images (pictures) are examples of two dimensional signals, which can be represented as an array of values (x_{mn}).

For these arrays, we can define a Fourier transform, a function of two variables $X(\omega, \tau)$, by the formula

$$X(\omega, \tau) = \sum_{m,n=-\infty}^{\infty} \mathbf{x}_{mn} e^{2\pi i(m\omega + n\tau)}. \tag{159}$$

The array is recovered using the inverse Fourier transform, which in this case is given by a double integral:

$$\mathbf{x}_{mn} = \int_0^1 \int_0^1 X(\omega, \tau) e^{-2\pi i(m\omega + n\tau)}\, d\omega\, d\tau. \tag{160}$$

Notice the negative sign in the exponential here; it is important.

The 2D discrete Fourier transform is defined in a similar way. We start with a finite array $\mathbf{x}$ of data of size M by N, with entries

$$\mathbf{x}_{mn}, \qquad \text{for } 0 \leq m < M, 0 \leq n < N. \tag{161}$$

Its Fourier transform is also an array, let's call it X, whose entries are given by

$$X_{jk} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathbf{x}_{mn} e^{2\pi i(mj/M + nk/N)}. \tag{162}$$

This discrete Fourier transform has an inverse as well. The formula to recover the $\mathbf{x}_{mn}$ from the X_{jk} looks almost the same, except we flip a sign in the exponential, and normalize by dividing by MN. So, we write the inverse as

$$\mathbf{x}_{mn} = \frac{1}{MN} \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} X_{jk} e^{-2\pi i(mj/M + nk/N)}. \tag{163}$$

We can also define convolution in 2D, again in the obvious way:

$$z_{jk} = \sum_{mn} x_{mn} y_{j-m,k-n}. \tag{164}$$

The usual theorems hold. So, for instance the 2D FT transforms convolutions into products, and vice versa. The discrete FT transforms (doubly) circular convolutions into products, and vice versa.

Similarly, filtering can be done either by convolution, or by Fourier domain techniques.

MATLAB has built in command for fft2, ifft2, conv2 to deal with 2D data. The reader should investigate these examples on their own.

As a quick example, consider the oval drawn on the left half of Figure 23. This was created using a meshgrid in MATLAB, to build an array of zeros and ones, with the image displayed using command 'imagesc' which scales the magnitudes so the images look nice. The 2D Fourier transform was computed using 'fft2' and the absolute values are displayed on the right half of Figure 23, again using 'imagesc.' Note that most of the energy is concentrated near the corners of the Fourier transform – this corresponds to the low frequency energy in the original image.

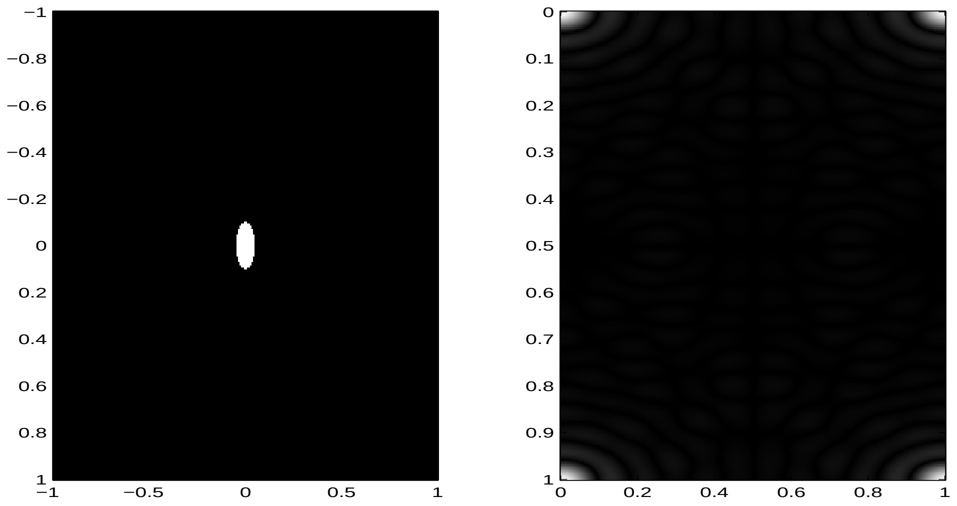

Figure 23: The 2D Fourier transform of the image of an oval (fft2).

A better FFT image is created by shifting the corners into the centre of the FFT. Essentially, we are remapping the normalized frequencies from the interval $[0, 1]$ to the interval $[-.5, .5]$ in both the horizontal and vertical axes. The result is shown in Figure 24, where on the right hand side, we seen the FFT result with the low frequency energy displayed near the centre. The MATLAB command uses 'imagesc(abs(fftshift(fft2(z)))),' where z is the image to be transformed.

As we saw in the 1D transforms, it is often useful to convert to decibels in order to see the low level detail in the Fourier transform. The command 'imagesc(pow2db(abs(fftshift(fft2(z)))))' does this for us, resulting in the detailed Fourier transform of the oval, shown in Figure 25. Notice all the intricate curves in the FFT image. You should also notice that while the original oval was vertical, the FFT image has a horizontal oval in the centre. These ovals can be seen in both Figure 24 and 25. This hints at some of the duality that happens in the transformation of geometric objects in the Fourier domain.

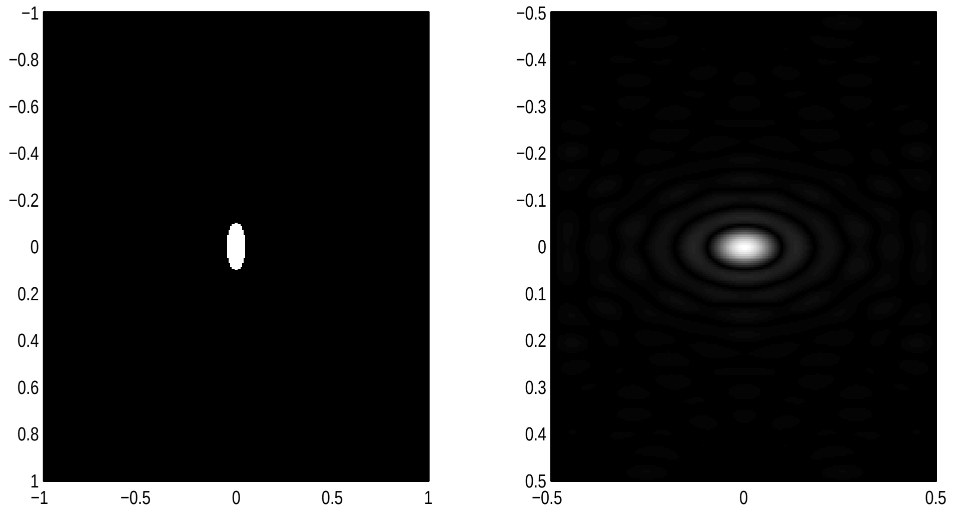

Figure 24: The shifted 2D Fourier transform of the image of an oval (fftshift, fft2).

40 Fourier transform on the real line

We can also take the Fourier transform of any function defined on the real line. The definition is given as follows: if $x(t)$ is a function of parameter $t \in \mathbb{R}$, its Fourier transform is a new function $X(\omega)$ on the real line, defined by the integral

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{2\pi i t\omega}\, dt. \tag{165}$$

The inverse transform is defined similarly, and we can recover the original function $x(t)$ from its FT $X(\omega)$ by the integral formula

$$x(t) = \int_{-\infty}^{\infty} X(\omega) e^{-2\pi i t\omega}\, d\omega. \tag{166}$$

As an example, take $x(t)$ as the boxcar function , $x(t) = 1$ for t in the interval $[-1, 1]$, and zero everywhere else. Its Fourier transform is computed as

$$\begin{aligned} X(\omega) &= \int_{-\infty}^{\infty} x(t) e^{2\pi i t\omega}\, dt \\ &= \int_{-1}^{1} 1 e^{2\pi i t\omega}\, dt \\ &= \frac{e^{2\pi i t\omega}}{2\pi i\omega}\Big|_{t=-1}^{t=1} \\ &= \frac{e^{2\pi i\omega} - e^{-2\pi i\omega}}{2\pi i\omega} \\ &= \frac{2\sin(2\pi\omega)}{2\pi\omega} = 2\mathrm{sinc}(2\pi\omega). \end{aligned}$$

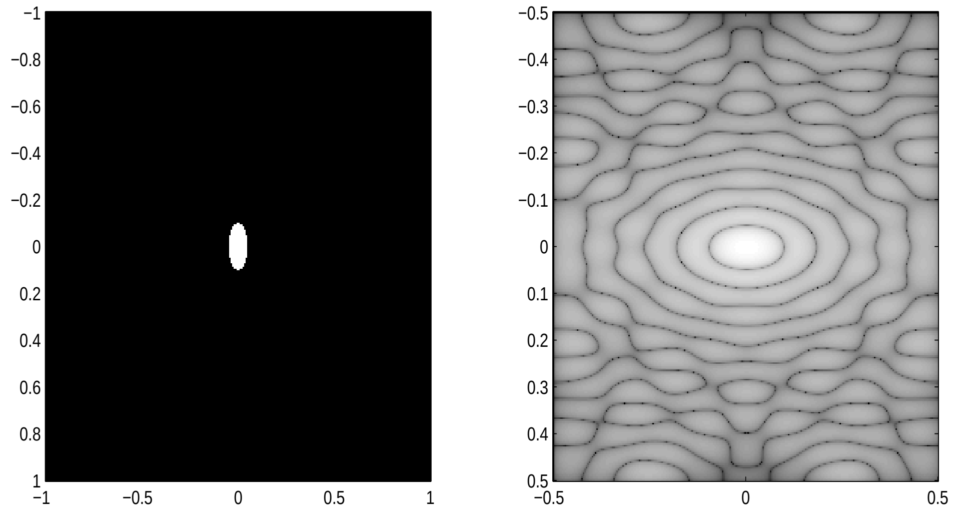
Figure 25: The 2D Fourier transform of an oval, rescaled to decibels (pow2db, fftshift, fft2).

So, the Fourier transform of a boxcar function is a sinc function. You might recall we saw a similar result in the discrete time situation.

As a second example, take $x(t) = e^{-t^2}$ as a Gaussian. Its Fourier transform is computed as

$$\begin{aligned}
X(\omega) &= \int_{-\infty}^{\infty} x(t) e^{2\pi i t\omega}\, dt \\
&= \int_{-\infty}^{\infty} e^{-t^2} e^{2\pi i t\omega}\, dt \\
&= \int_{-\infty}^{\infty} e^{-t^2 + 2\pi i t\omega}\, dt, \text{ now complete the square,} \\
&= \int_{-\infty}^{\infty} e^{-(t-\pi i\omega)^2 - \pi^2\omega^2}\, dt, \text{ pull out the part independent of } t, \\
&= e^{-\pi^2\omega^2} \int_{-\infty}^{\infty} e^{-(t-\pi i\omega)^2}\, dt, \text{ change variables, } s = t - \pi i\omega, \\
&= e^{-\pi^2\omega^2} \int_{-\infty}^{\infty} e^{-s^2}\, ds, \text{ and note the integral collapses to a constant,} \\
&= \sqrt{\pi} e^{-\pi^2\omega^2}.
\end{aligned}$$

That is to say, the Fourier transform of a Gaussian is a rescaled Gaussian.

As a exercise, compute the Fourier transform of a wave packet

$$x(t) = e^{-t^2} \cos(2\pi\omega_0 t) \tag{167}$$

with centre frequency ω_0. You should find the Fourier transform is the sum of two shifted Gaussians, centered at $\pm\omega_0$.

The physical meaning of these functions on the real line is straightforward. If $x(t)$ is a function of time, with t measured in seconds, then $X(\omega)$ is a function of frequency, with ω measured in Hertz (i.e. cycles per second). The function $x(t)$ represents the signal as it evolves in time, while $X(\omega)$ represents the energy content of the signal at each frequency.

Similarly, if $x(t)$ represents a signal in space, with t measured in metres, then $X(\omega)$ measure the signal in spatial frequencies, with ω measured in cycles per metre – that is, the inverse of wavelength.

As you might expect, the Fourier transform on the real line takes convolutions to products, and products to convolutions. This begs the question: what is the convolution of two functions on the real line? It is defined by the integral formula

$$(x * y)(t) = \int_{-\infty}^{\infty} x(t-s)y(s)\,ds, \tag{168}$$

where x, y are each functions on the real line, and their convolution $x * y$ is a new function on the real line.

It is a simple matter to verify that the convolution of two boxcar function is a continuous, piecewise linear function (the graph looks like a little triangle on the real line). Its Fourier transform is the square of a sinc function.

The Shannon Sampling Theorem gives a precise formula to connect the signal $x(t)$ on the real line, and the discrete time, sampled version $\mathbf{x}_n = x(n\Delta_t), n = 0, \pm 1, \pm 2, \ldots$. The connection is through the two Fourier transforms – the real line version of $x(t)$, and the transform of the sequence $\mathbf{x}_n$. Basically, the theorem says if you sample fast enough (that is, if the time step Δ_t is small enough), then you can recover the function $x(t)$ exactly from the samples $\mathbf{x}_n$. The formula is given as

$$x(t) = \sum_{n=-\infty}^{\infty} \mathbf{x}_n \operatorname{sinc}(\pi(t/\Delta_t - n)). \tag{169}$$

The formula itself is less important than deciding how fast the sampling rate has to be, for the formula to be valid. Recall the sampling rate is given as the reciprocal of the sampling interval (or time step), $F_s = 1/\Delta_t$. Half this value is called the Nyquist frequency, $F_{Nyq} = \frac{1}{2}F_s$. Provided the signal has no energy beyond Nyquist, that is, if $X(\omega)$ is zero outside the interval $[-F_{Nyq}, F_{Nyq}]$, then the sampling formula is exact.

WARNING: Pay attention to units. In these notes, we use cycles per second (or cycles per unit distance) in the transforms. Other texts will use radians per second, so the formulas differ by a factor of 2π. Some redefine the sinc function to include an extra factor of π, so the sampling formula can look quite different. Try not to let it bother you too much.

41 General Fourier transforms

We have seen several examples of Fourier transforms now: the Fourier transform of a sequence $\mathbf{x} - (\ldots, x_{-1}, x_0, x_1, x_2, \ldots)$, the (discrete) Fourier transform of a finite sequence $\mathbf{x} = (x_0, x_1, \ldots, x_{N-1})$, the Fourier transform of a 2D array $\mathbf{x}_{mn}$, and the Fourier transforms of a function on a real line $x(t)$. It turns out the notion of a Fourier transform is a very general idea. The main point is we start with a function on a set, and transform it to a different function on another set. These sets have to have a bit of structure, but once that structure is in place, the Fourier transform is automatic.

What what are these sets with structure? They are *groups*: that is, sets of numbers that we can add, that include zero, and include the negatives of each number. So, for instance the set of real numbers $\mathbb{R}$ with usual addition is such a group. The set of three-vectors $(x, y, z) \in \mathbb{R}^3$ is a bigger group, using vector addition for the group operation; here, the origin $(0, 0, 0)$ is the zero element in the set. The set of positive and negative integers $\mathbb{Z}$ with usual addition is a group, as is the 3D version of triples of integers $(m, n, l) \in \mathbb{Z}^3$. A slightly more complicated example is a finite set of integers $\mathbb{Z}_4 = \{0, 1, 2, 3\}$ with addition modulo 4, or more generally

$$\mathbb{Z}_N = \{0, 1, 2, 3, \ldots, N-1\}, \qquad \text{with addition modulo N.} \tag{170}$$

The interval of real numbers $[0, 1]$ with addition modulo 1 is also a group, although we should think of 0 and 1 as representing the same number in this set.

There are some additional mathematical requirements that we won't get into here, but think of these groups as just collections of "numbers" that you can add.[3] We usually denote the group as a set G.

Once you have a group G, you can find its *characters.* These are functions complex valued functions on G that have absolute value one, and respect the group addition. So, we are looking for functions $\alpha : G \to \mathbb{C}$ with

$$\begin{aligned} |\alpha(g)| &= 1, \text{ for all numbers } g \text{ in the group ,} \\ \alpha(g+h) &= \alpha(g)\alpha(h), \text{ for all numbers } g, h \text{ in the group.} \end{aligned}$$

It's easy to check that these two conditions also imply $\alpha(0) = 1$.

How do you find characters? Well, we've seen a bunch of them before. For instance, with the integers $G = \mathbb{Z}$, the characters are functions of the form

$$\alpha(n) = e^{2\pi i n\omega}, \tag{171}$$

where ω is any fixed real number between zero and one. It is easy to check that this α is a character – the complex exponential gives numbers of absolute value one, and the group addition is respected since

$$\alpha(n+m) = e^{2\pi i(n+m)\omega} = e^{2\pi i n\omega}e^{2\pi i m\omega} = \alpha(n)\alpha(m), \tag{172}$$

as desired. To see that these are the ONLY characters on the group of integers, notice that if α is an arbitrary character on $G = \mathbb{Z}$, its value at integer 1 has to be a complex number on the unit circle, so

$$\alpha(1) = e^{2\pi i\omega}, \tag{173}$$

[3]The technical assumptions are that the group has a locally compact topology, and that group is abelian, which means the order in which we add things does not matter. These assumptions include all the examples we have seen in this course.

for some real number ω between zero and one. By the group property, we find

$$\alpha(2) = \alpha(1+1) = \alpha(1)\alpha(1) = e^{2\pi i\omega}e^{2\pi i\omega} = e^{2\pi i 2\omega}, \tag{174}$$

and by repeating, we find

$$\alpha(n) = \alpha(1+1+\cdots+1) = \alpha(1)\alpha(1)\cdots\alpha(1) = \left(e^{2\pi i\omega}\right)^n = e^{2\pi i n\omega}, \tag{175}$$

which is exactly the form we saw above.

For a finite group, like $G = \mathbb{Z}_3 = \{0, 1, 2\}$, there are only finitely many characters. To find them, we start off again by observing $\alpha(1)$ is a complex number on the unit circle, so we can find a number ω between zero and one with

$$\alpha(1) = e^{2\pi i\omega}. \tag{176}$$

Of course, we then know that $\alpha(2) = e^{2\pi i 2\omega}$ and then

$$1 = \alpha(0) = \alpha(1+1+1) = \alpha(1)\alpha(1)\alpha(1) = e^{2\pi i 3\omega}. \tag{177}$$

Well, this only happens if 3ω is a multiple of one. So, we have three choices, $\omega = 0, 1/3, 2/3$. And that's it for the characters of $G = \mathbb{Z}_3$. It is convenient to write the three characters as $\alpha_0, \alpha_1, \alpha_2$ and give their formulas as

$$\alpha_j(n) = e^{2\pi i jn/3}, \text{ for } j, n = 0, 1, 2. \tag{178}$$

Notice in this example, the product of any two characters is again a character. In fact, it is easy to check that

$$\alpha_j(n)\alpha_k(n) = \alpha_{j+k}(n), \text{ for } j+k \text{ modulo } 3. \tag{179}$$

This happens in general. The set of characters, a set of complex-valued functions, is itself a group, with the group 'addition' just being simple multiplication of functions. The 'zero' in the characters is the constant function one. This set of characters is called the dual group of G, and is usually denoted by $\widehat{G}$.

Now we can define the Fourier transform. Given a function x on the group G, we define a function $\widehat{x}$ on the dual group $\widehat{G}$ as follows: given any character α, we write

$$\widehat{x}(\alpha) = \sum_n x(n)\alpha(n), \tag{180}$$

where we sum over all n in the group G. (Sometimes, we replace the sum with an integral, such as the case where the group is a interval, or the real line.)

And that's it. Now we can just compute a bunch of dual groups and the corresponding Fourier transforms. It is sometime convenient to keep track of the characters as a parameterized family of functions, but other than that, the computation is very straightforward.

Here is a short table of groups, dual groups, characters, and Fourier transforms:

G	$\widehat{G}$	Characters	Fourier transform
$\mathbb{Z}$	$[0,1]$	$n,\omega \mapsto e^{2\pi i n\omega}$	$\widehat{x}(\omega) = \sum_n x(n)e^{2\pi i n\omega}$
$[0,1]$	$\mathbb{Z}$	$\omega, n \mapsto e^{2\pi i n\omega}$	$\widehat{x}(n) = \int_0^1 x(\omega)e^{2\pi i n\omega}\,d\omega$
$\mathbb{Z}_N$	$\mathbb{Z}_N$	$n,k \mapsto e^{2\pi i n k/N}$	$\widehat{x}(k) = \frac{1}{\sqrt{N}}\sum_n x(n)e^{2\pi i n k/N}$
$\mathbb{R}$	$\mathbb{R}$	$t,\omega \mapsto e^{2\pi i t\omega}$	$\widehat{x}(\omega) = \int_{-\infty}^{\infty} x(t)e^{2\pi i t\omega}\,dt$
$\mathbb{Z}^2$	$[0,1]^2$	$(n_1,n_2),(\omega_1,\omega_2) \mapsto e^{2\pi i(n_1\omega_1+n_2\omega_2)}$	$\widehat{x}(\omega_1,\omega_2) = \sum_{n_1,n_2} x(n_1,n_2)e^{2\pi i(n_1\omega_1+n_2\omega_2)}$
$\mathbb{Z}_M \times \mathbb{Z}_N$	$\mathbb{Z}_M \times Z_N$	$(m,n),(j,k) \mapsto e^{2\pi i(mj/M+nk/N)}$	$\widehat{x}(j,k) = \frac{1}{\sqrt{MN}}\sum_n x(m,n)e^{2\pi i(mj/M+nk/N)}$
$\mathbb{R}^2$	$\mathbb{R}^2$	$(s,t),(\rho,\omega) \mapsto e^{2\pi i(s\rho+t\omega)}$	$\widehat{x}(\rho,\omega) = \int\int x(s,t)e^{2\pi i(s\rho+t\omega)}\,ds\,dt$

Some important results on the general Fourier transform:

1. Every locally compact abelian group has a Fourier transform;
2. The Fourier transform preserves energy: $||\widehat{x}||^2 = ||x||^2$;
3. The Fourier transform preserves inner products $\langle \widehat{x}, \widehat{y}\rangle = \langle x, y\rangle$;
4. The Fourier transform changes convolution to products: $\widehat{x * y} = \widehat{x}\cdot\widehat{y}$;
5. The Fourier transform changes products to convolution: $\widehat{x\cdot y} = \widehat{x} * \widehat{y}$;
6. The dual of the dual group is the original group: $\widehat{\widehat{G}} = G$;
7. The Fourier transform of the Fourier transform is the original function, up to sign: $\widehat{\widehat{x}}(n) = x(-n)$.

These all require proofs, and you have to know how to define energy, and convolution. One must also be careful about choosing normalization constants (factors of $\pi, 2\pi, \sqrt{N}$, etc). But, trust me, it all works out.

Incidently, convolution on the finite group $\mathbb{Z}_N$ is just the circular convolution discussed earlier in these notes. Energy in this finite group is just the length square of the finite vector.

For 2D, 3D transforms, etc, the pattern is kind of clear in the table. Basically, the dual group of a product of groups, is the product of the duals. e.g.

$$\widehat{\mathbb{Z}^2} = \widehat{\mathbb{Z}} \times \widehat{\mathbb{Z}} = [0,1]\times[0,1], \tag{181}$$

and the character of the product is a product of independent characters. The Fourier transform of a function of several variables can be obtained by doing independent transforms in each variable separately, one by one.

42 Time-frequency transforms

A problem with the Fourier transform is the all-or-nothing nature of the change into the frequency domain: you either see the signal in the time domain, or in the frequency domain (via the FT), but nothing else. An example of the problem is shown in Figure 26. In the left panel, we see a chirp signal, $x(t) = \sin(2\pi 200t^2)$ which gives an increasing sweep of frequencies from 0 Hz to 400 Hz; on the right we see the Fourier transform indicating the presence of energies in the full range, from 0 to 400 Hz, and the mirror image of energy from 600 to 1000Hz, which is the sampling rate. But, in this FFT display, there is no indication whether the ramp is increasing in frequency, decreasing in frequency, or some other mix of high and low frequencies.

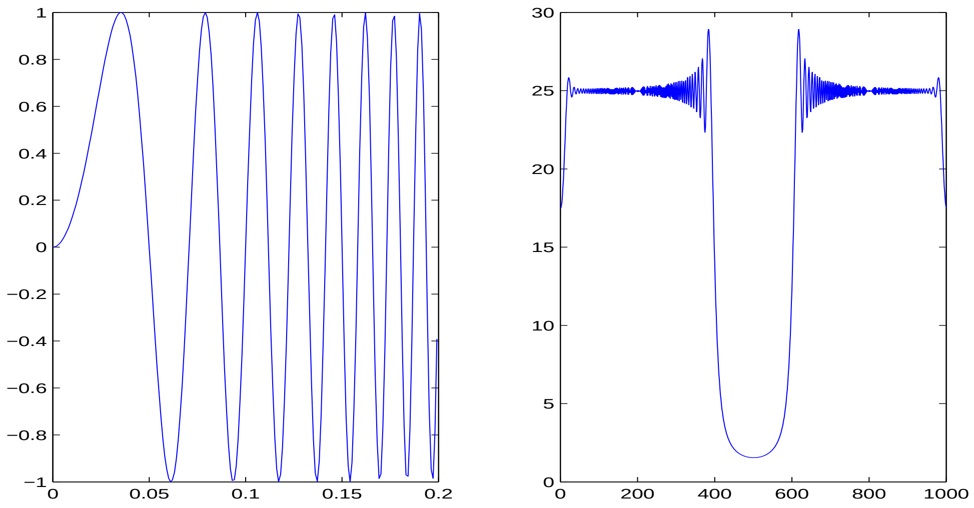

Figure 26: A linear chirp and its Fourier transform (freqs from 0 to 400 Hz).

One solution to this problem is to break up the signal $x(t)$ into short, windowed segments, and apply the Fourier transform to each segment. By piecing together the transforms of these segments, we obtain a two dimensional representation of the signal, as shown in Figure 27. (This was computed in MATLAB using the spectrogram command.) In this image, we clearly see the linear ramp of frequencies. At time $t = 0$ there is lots of energy concentrated near the low frequencies; as t increases, so does the main frequency, until we hit time $t = 1$ with the energy concentrated at frequency 400Hz. We would hear this sound as a slowly rising tone.

The basic form for the time-frequency transform is called the Short Time Fourier Transform (STFT). The idea is to fix a time t, examine the values of the signal x in an interval $[t - L, t + L]$ centred at that time t, and take the Fourier transform. We would write the transform as

$$X(t, \omega) = \int_{-L}^{L} x(t + s)e^{2\pi i s\omega}\, ds, \text{ (STFT)} \tag{182}$$

where t is the centre time, and ω is the frequency.

Chopping a signal to an interval $[t - L, t + L]$ is never a good idea in signal processing, as you artificially introduce discontinuities and jumps in the result – making the Fourier transform show high frequency effects that aren't really there. Denis Gabor proposed replace this "boxcar" window with a Gaussian – the Gabor transform takes this form

$$X(t, \omega) = \int_{-\infty}^{\infty} x(t + s)e^{-s^2} e^{2\pi i s\omega}\, ds. \text{ (Gabor)} \tag{183}$$

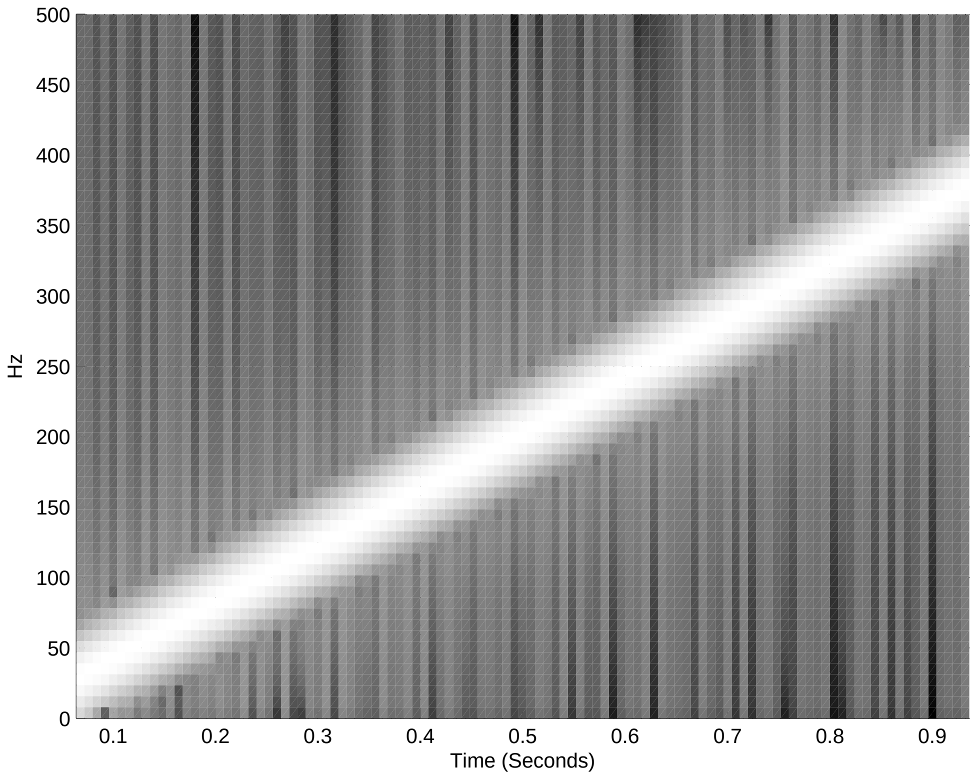

Figure 27: The Gabor transform of a linear chirp – a time-frequency display

Of course, one may choose wider or narrower Gaussians - a narrow Gaussian gives more precise time information, while a wider Gaussian gives more precise frequency information. This tradeoff between accuracy in time and frequency is Heisenberg's uncertainty principle.

For the record, here is a Gabor transform with Gaussian width controlled by parameter σ:

$$X(t,\omega,\sigma) = \frac{1}{\sigma}\int_{-\infty}^{\infty} x(t+s)e^{-s^2/\sigma^2}e^{2\pi i s\omega}\,ds. \text{ (Gabor, adjustable window)} \tag{184}$$

The parameter $1/\sigma$ in front is a useful normalization constant. There is nothing too special about the Gaussian window; other windows could be used as well. In MATLAB's spectrogram you do have a choice of windows.

The S-transform (or Stockwell transform) is closely related to the Gabor transform, except the window with σ is chosen to be inversely proportional to frequency. It can be expressed in the form

$$X(t,\omega) = |\omega|\int_{-\infty}^{\infty} x(t+s)e^{-\pi s^2\omega^2}e^{2\pi i s\omega}\,ds. \text{ (S-transform)} \tag{185}$$

Note the S-transform is sometimes defined with an additional phase factor of $e^{-2\pi i t}$ in from of the integral. When looking at absolute values (or power), it doesn't matter.

The S-transform is a special case of the continuous waveform. We observe the function in the integral above, $e^{-s^2/\sigma^2}e^{2\pi i s\omega}$ is really just a function of $s\omega$. So, given ANY function on the real line, say $\phi(t)$, we can define the continuous wavelet transform of signal $x(t)$ as

$$X(t,\omega) = |\omega|^{1/2}\int_{-\infty}^{\infty} x(t+s)\phi(s\omega)\,ds.\text{ (Continuous wavelet transform)} \tag{186}$$

Here again, t corresponds to time, and ω is something like frequency – more properly, ω is inversely proportional to a wavelength. The factor $|\omega|^{1/2}$ in front is to ensure that this transform preserves energy, like the Fourier transform does.

Traditionally, researchers in wavelet theory work in SCALE rather than FREQUENCY, and define scale $a = 1/\omega$ as the reciprocal of the "frequency" parameter ω. Why? It's because the general wavelet $\phi(t)$ isn't a pure sinusoid, so it doesn't really have a single frequency associated to it. But is does has a scale, the scaling factor we get when we go from $\phi(t)$ to $\phi(t/a)$.

In terms of time and scale, the continuous wavelet transform is given by the formula

$$X(t,a) = \frac{1}{|a|^{1/2}}\int_{-\infty}^{\infty} x(t+s)\phi(s/a)\,ds.\text{ (Continuous wavelet transform, in scale)} \tag{187}$$

In MATLAB, the command 'cwt' computes the continuous wavelet transform of any signal. Figure 28 shows the wavelet transform of a linear chirp, the same signal used in the examples above. The image shows a curve in the form of a hyperbola, which is from the linear ramp $y = \omega$ converting to its reciprocal (scale) with $y = 1/a$. In this example, we used the Morlet wavelet ('morl' in MATLAB) to get something close to the S-transform.[4] At the initial time (near $t = 0$) there is lots of energy at large scale (scale =31); large scale equals large wavelength, which corresponds to low frequency, so our ramp starts out at a low frequency. As time passes, the scales drop, indicating the wavelengths are getting shorter – that is, higher frequency, which is exactly what our linear ramp chirp is doing. We can debate just how useful this transform really is. It certainly is general.

43 Fast time-frequency transforms

Just like the Fourier transform, all these time-frequency transforms have fast versions. In particular, the fast discrete wavelet transform is ridiculously fast, at a speed of order N for vectors of length N. This is even faster than the FFT. The price you pay, though, is limited choice of the wavelets you can work with. Haar wavelets are a place to start (first used in the 1920s), more modern version are the Daubechies wavelets, invented by Ingrid Daubechies of Princeton. Biorthogonal wavelets (whatever that is) gives a very general method of defining these fast transforms.

The fast wavelet transform is the basis for many industry standards, including the FBI's use of compression for fingerprints, and the JPEG2000 image compression method for computer image files. MATLAB implements many of these fast transforms.

There are whole books written on this, well beyond what we can cover in this class.

[4] In MATLAB, I used the command cwt(x,1:32,'morl','plot').

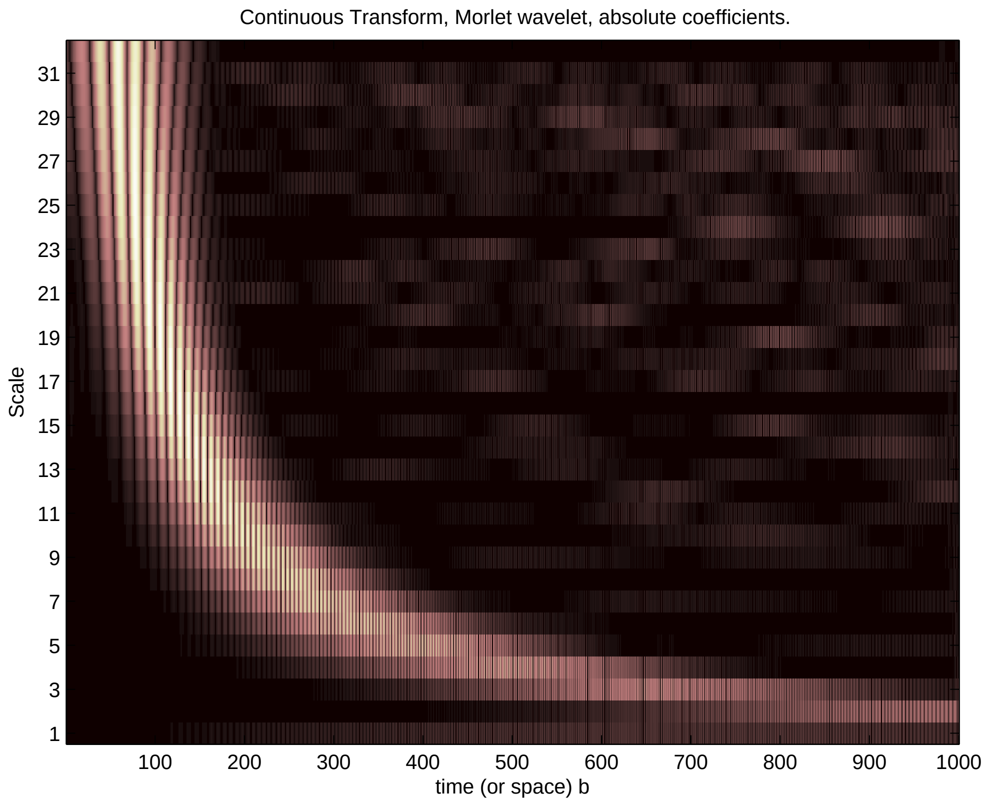

Figure 28: The wavelet transform of a linear chirp – a time-scale display

44 Differential equations and the Fourier transform

It is a remarkable property of the Fourier transform that it changes derivatives into simple multiplication, and thus linear differential equations become simple algebraic equations.

To see this, observe from the inverse Fourier transform of a function on the real line, we can write $x(t)$ in terms of its Fourier transform $X(\omega)$ by the integral

$$x(t) = \int_{-\infty}^{\infty} X(\omega) e^{-2\pi i t\omega}\, d\omega. \tag{188}$$

Computing the derivative, we have

$$\begin{aligned} x'(t) &= \frac{d}{dt}\int_{-\infty}^{\infty} X(\omega) e^{-2\pi i t\omega}\, d\omega \\ &= \int_{-\infty}^{\infty} X(\omega) \frac{d}{dt} e^{-2\pi i t\omega}\, d\omega \\ &= \int_{-\infty}^{\infty} X(\omega)(-2\pi i\omega) e^{-2\pi i t\omega}\, d\omega, \end{aligned}$$

which says that the Fourier transform of the derivative $x'(t)$ is just the Fourier transform of x, times the function $-2\pi i\omega$.

A simple differential equation

$$x''(t) + 2x'(t) + 3x(t) = g(t) \tag{189}$$

transforms in the Fourier domain to

$$(-2\pi i\omega)^2 X(\omega) + 2(-2\pi i\omega)X(\omega) + 3X(\omega) = G(\omega), \tag{190}$$

so applying a bit of algebra, we see

$$X(\omega) = \frac{G(\omega)}{-4\pi^2\omega^2 - 4\pi i\omega + 3}. \tag{191}$$

While we may not be able to compute this exactly, we can use numerical methods to take an inverse Fourier transform of the right hand side, to recover $x(t)$.

The wave equation involves a function $u(x, y, z, t)$ of four variables (three for space, one for time) and is expressed in the form

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} = g(x, y, z, t), \tag{192}$$

where u measure the amplitude of the wave displacement, c is the speed of sound in the medium, and g is a forcing term (like a dynamite blast). Taking Fourier transforms in all 4 variables, we obtain the algebraic equation

$$4\pi^2\left(-\frac{\omega^2}{c^2} + k_x^2 + k_y^2 + k_z^2\right)U(k_x, k_y, k_z, \omega) = G(k_x, k_y, k_z, \omega), \tag{193}$$

where U, G are the Fourier transforms of u, g respectively, k_x, k_y, k_z are the spatial wave numbers, and ω is the frequency in time. Again, notice this new equation has no derivatives, and we can use numerical methods to solve it.

Such methods are the basis for many numerical algorithms for solving the wave equation in physics, geophysics, and engineering.

45 Laplace transforms

Unfortunately, we will have to skip this one.

The Laplace transform of a function $x(t)$ on the positive real line $[0, \infty)$ is given by the integral

$$L_x(s) = \int_0^\infty x(t)e^{-st}\,dt. \tag{194}$$

It looks a bit like the Fourier transform, except there are no complex exponentials, and we only integrate on half the real line.

Like the Fourier transform, it turns derivatives into multiplication. By starting the integration at $t = 0$, we can easily handle initial conditions for our differential equations. The main utility of the Laplace transform is for exact (symbolic) solutions to linear, constant coefficient differential equations.

Numerical methods based on the Laplace transform are fraught with problems, mainly because the inverse Laplace transform behaves very badly in the presence of noise (noise in the data, or noise in the computer arithmetic).

APPENDIX 1: Collected wisdom

In case we are missing the forest for the trees, here are some basic facts about signal processing that you should always keep in mind. (I will add to this as the course goes along.)

Signals:

- Signals are functions (of time, space, etc).
- Many useful signals are sums of sines and cosines.
- A sine wave or cosine wave is specified by a frequency and an amplitude. Negative frequencies give basically the same signal as a positive frequency.
- Sines, cosines are the same function, just shifted in time.
- Unless we know the start time, sine waves and cosine waves at the same frequency are basically the same thing. Same for any shifts of these waves.
- The complex sinusoid $e^{2\pi i\omega t}$ makes the algebra of sines and cosines easy. But in real life, we never really see complex valued waves.

Physical signals:

- We hear sounds in the range 20Hz to 20,000 Hz.
- Seismic waves are measured in the range 4Hz to 150 Hz (approx.) As technology improves, we expand this range.
- Radio waves are in the range of kilohertz (AM), megahertz (FM), gigahertz (cellphones).
- Sound waves are acoustic waves (variations in air pressure). Seismic waves are elastic waves (solid motion). Radio waves are electromagnetic waves.

Sampled signals:

- For practical reasons, we sample signals.
- Sample rate determines the highest frequency we can represent
$$\frac{1}{2}(\text{sample rate}) = \text{Nyquist rate}. \tag{195}$$
- Frequencies higher than that are aliases, and cause errors.
- In real systems, we use electronics to eliminate those higher frequencies, before sampling.

Systems:

- Basic mode is "Signal in $\rightarrow$ Signal out."
- Convolution gives a LSI system, and vice versa.
- Specify a LSI system by its impulse response $\mathbf{h}$.
- Practical LSI system given by finite $\mathbf{h}$ or ratio of two such finite ones.

APPENDIX 2: Eigenvalues and eigenvectors

Recall a matrix A has an eigenvector $\mathbf{x} \neq 0$ and an eigenvalue λ if they satisfy

$$A\mathbf{x} = \lambda \mathbf{x}. \tag{196}$$

That is, the matrix applied to vector $\mathbf{x}$ just returns the same vector $\mathbf{x}$, multiplied by the number λ.

As an example, with the matrix $A = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix}$, we find two eigenvectors $[1,1]$ and $[1,-1]$ by observing

$$\begin{aligned} \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} &= 6 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} &= -2 \begin{bmatrix} 1 \\ -1 \end{bmatrix}. \end{aligned}$$

So 6 and -2 are the eigenvalues of the matrix. Notice that $6 - 2 = 4$, the trace of the matrix (sum of the diagonal elements), while $6 * (-2) = -12$, the determinate of the matrix. This is a general property of the eigenvalues: they tell us a lot about the matrix.

In the frequency response, the only difference is that we are working in infinite dimensions. The eigenvectors are signals $\mathbf{x}$ that are complex exponentials at frequency ω. The corresponding eigenvalues are the complex numbers $H(\omega)$. That is, when $\mathbf{x}$ is a complex exponential, we have the eigenvector equation

$$\mathbf{h} * \mathbf{x} = \lambda \mathbf{x}, \tag{197}$$

where the eigenvalue λ is the (complex) number $\lambda = H(\omega)$, given by the frequency response evaluated at ω.

APPENDIX 3: Music and frequencies

Humans can hear sounds in the range of 20Hz to 20,000 Hz. (Hz = Hertz = cycles per second). Basically any periodic wave that repeats itself that many times a second will be heard by the human ear as a musical tone. The frequency (cycles per second) corresponds to pitch of the note. The low (deep) sounds have the low frequency (eg 25) and the high (shrill) sounds have the high frequency

A piano plays notes in the range of about 25 Hz to 4200 Hz. The note "A above middle C" is defined as a pitch of 440 Hz. All the other pitches are defined relative to that frequency. The equal tempered scale uses powers of the number $\alpha = \sqrt[12]{2}$ to define other frequencies as

$$freq = 440 * \alpha^n, \tag{198}$$

where n is the number of semitones above "A", or below it for n negative.

Doubling the frequency ($n = 12$ semitones up) corresponds to moving up an octave in the musical scale. Halving the frequency ($n = -12$ semitones down) moves down an octave.

Bach was involved in the notion of an equal tempered scale, which uses the power of the 12th root of 2 to determine frequencies. You may be familiar with his keyboard composition call"The Well-tempered Klavier." This was motivated by the desire to get all instruments in an orchestra to sound in tune, not matter what key they were playing in.

Before equal tempering, notes were based on the idea that frequencies that were related as the ratio of simple fractions often sound good together. This is deeply connected to the notion of harmonics in musical tones, but it seems rather mathematical.

For instance, the frequencies 440Hz, 550Hz, 660Hz all sound good together, and their ratios are

$$\frac{550}{440} = \frac{5}{4} \qquad \frac{660}{440} = \frac{3}{2}, \tag{199}$$

which are simple fraction with small integers in the fractions. These three notes correspond to the three notes in a major triad (A, C#, E). It is interesting to note that the middle note, 550Hz, would actually be as high as 554Hz in the equal tempered scale. To those of you with good ears, the 554Hz tone sounds a little sharp. We get this 554Hz value by going up four semitones in the equal tempered scale, so $440 * 2^{4/12} = 554.37$.

You can experiment with these ideas in MATLAB by setting up some simple signals and playing them out.

```
Fs = 10000; % the sampling rate
dt = 1/Fs; % the time step
T1 = 0:dt:1; % one second of time, in steps of dt
A = sin(2*pi*440*T1);  % the note A above middle C
Cs = sin(2*pi*550*T1); % the note C#
E = sin(2*pi*660*T1); % the note E
sound([A,Cs,E],Fs);  % play the notes one after the other
sound(.3*(A+Cs+E),Fs); % play the notes all together as one
```

If you are getting tired of hearing pretty sine waves, try raising it to some (odd) power, to get some richer harmonics in your music.

```
Fs = 10000;
```

```
dt = 1/Fs;
T1 = 0:dt:1;
A = sin(2*pi*440*T1).^5;  % the note A above middle C, with harmonics
Cs = sin(2*pi*550*T1).^5; % the note C#
E = sin(2*pi*660*T1).^5; % the note E
sound([A,Cs,E],Fs);
sound(.3*(A+Cs+E),Fs);
```

To get the major scale, using fractions instead of powers of two, you can use the ratios

$$\frac{1}{1}, \frac{9}{8}, \frac{5}{4}, \frac{4}{3}, \frac{3}{2}, \frac{27}{16}, \frac{15}{8}, \frac{2}{1}. \tag{200}$$

In the A major scale, this corresponds to frequencies

$$440, 495, 550, 586.7, 660, 742.5, 825, 880\, Hz. \tag{201}$$

Some might argue that other fractions are better. For instance, we might replace the 5/4 with $81/64 = (1/4) * (3/2)^4$. If you know something about music theory, you will see this is connected to the circle of fifths idea.

Now, just because its fun, try this. It is a sweep through the frequency range

```
Fs = 10000;
dt = 1/Fs;
T1 = 0:dt:1;
sweep = sin(2*pi*440*T1.^3);  % a sweep
sound(sweep,Fs);
```