6.801/6.866: Machine Vision, Lecture 16

Professor Berthold Horn, Ryan Sander, Tadayuki Yoshitake
MIT Department of Electrical Engineering and Computer Science
Fall 2020

These lecture summaries are designed to be a review of the lecture. Though I do my best to include all main topics from the lecture, the lectures will have more elaborated explanations than these notes. Therefore, if you're looking for the most rigorous review and treatment of these topics, we encourage you to rewatch the lecture videos. With that said, we hope these summaries are beneficial for your learning. If you have any feedback for these lecture summaries, please submit it **here**.

1 Lecture 16: Fast Convolution, Low Pass Filter Approximations, Integral Images, (US 6,457,032)

1.1 Sampling and Aliasing

Sampling is a ubiquitous operation for machine vision and general signal processing. Recall that PatMax, for instance, uses sampling in its methodology. PatMax also performs an interesting operation: low-pass filtering before sampling. Why is this performed? To answer this question, let us revisit the **Nyquist Sampling Theorem**.

1.1.1 Nyquist Sampling Theorem

Before we dive into the mathematics behind this theorem, let us first build some intuition surrounding this theory.

- If we can sample a signal at a high enough frequency, we can recover the signal exactly through reconstruction.
- How is this reconstruction performed? We will convolve samples from the signal with sinc functions, and then superimpose these convolved results with one another.
- It is hard to sample from a signal with infinite support.
- What frequency do we need for this? Intuitively, to pick out how fast the signal needs to be moving, we certainly need to sample as quickly as the signal's fastest-varying component itself. But do we need to sample even faster? It turns out the answer is yes. As we will see below:

$$f_{\max} < \frac{f_{\text{sample}}}{2} \implies f_{\text{sample}} > 2f_{\max}$$

  I.e. we will need to sample at more than twice the frequency of the highest-varying component of the signal.

Let us look at this graphically. What happens if we sample at the frequency of the signal?

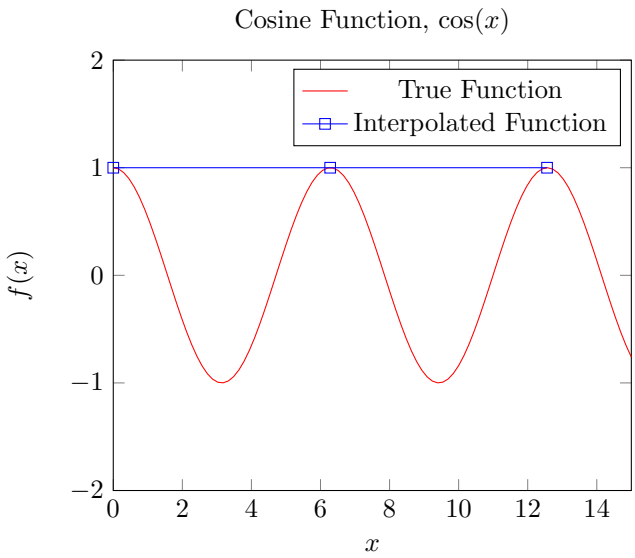


Figure 1: Sampling only once per period provides us with a constant interpolated function, from which we cannot recover the original. Therefore, we must sample at a higher frequency.

Note that this holds at points not on the peaks as well:

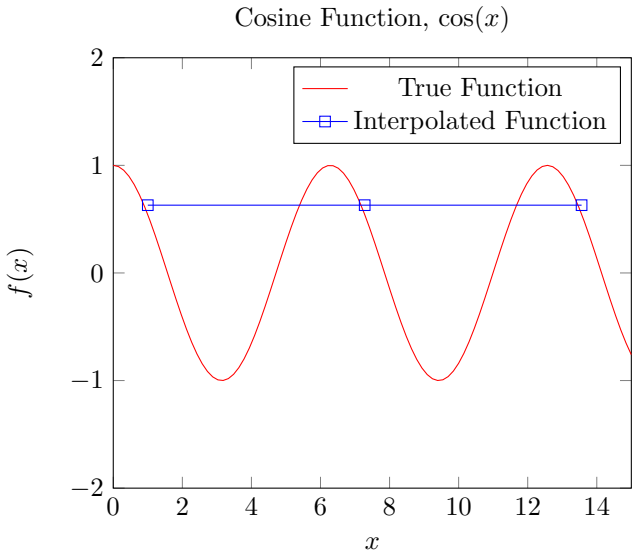


Figure 2: Sampling only once per period provides us with a constant interpolated function, from which we cannot recover the original. Therefore, we must sample at a higher frequency.

What if we sample at twice the frequency? I.e. peaks and troughs:

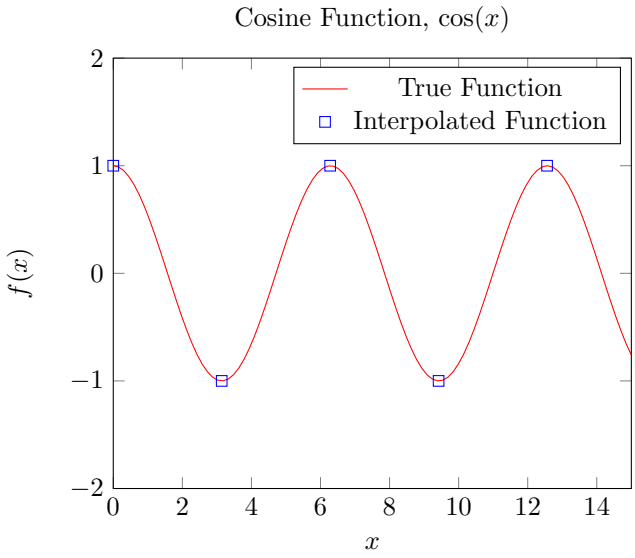


Figure 3: Sampling at twice the rate as the highest-varying component almost gets us there! This is known as the **Nyquist Rate**. It turns out we need to sample at frequencies that are strictly greater than this frequency to guarantee no aliasing - we will see why in the example below.

Is this good enough? As it turns out, the inequality for Nyquist's Sampling Theorem is there for a reason: we need to sample at **greater than** twice the frequency of the original signal in order to uniquely recover it:

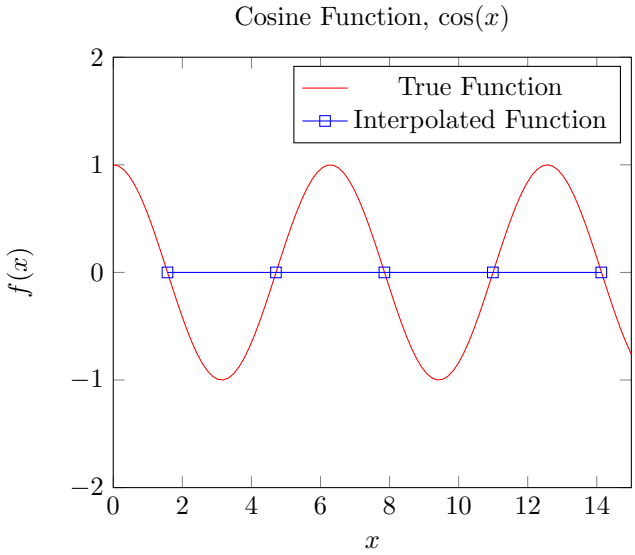


Figure 4: It turns out we need to sample at frequencies that are strictly greater than this frequency to guarantee no aliasing - we will see why in the example below.

Therefore, any rate above 2 times the highest-varying frequency component of the signal will be sufficient to completely avoid aliasing. As a review, let us next discuss aliasing.

1.1.2 Aliasing

Aliasing occurs when higher frequencies become indistinguishable from lower frequencies, and as a result they add interference and artifacts to the signal that are caused by sampling at too low of a frequency.

Suppose we have a signal given by:

$$f(x) = \cos(2\pi f_0 x)$$

And suppose we sample this signal with frequency given by $f_s = \frac{1}{\delta}$. Then our sampled signal is given by:

$$s_k = \cos(2\pi f_0 k \frac{1}{\delta}) = \cos(2\pi \frac{f_0}{f_s} k) \; \forall \; k \in \{1, 2, ...\}$$

Now let us consider what happens when we add multiples of 2π to this:

$$\begin{aligned} s_{k-2\pi} &= \cos\Big(2\pi\frac{f_0}{f_s}k - 2\pi k\Big) \\ &= \cos\Big(2\pi\Big(\frac{f_0}{f_s} - 1\Big)k\Big) \\ &= \cos\Big(2\pi\Big(\frac{f_0 - f_s}{f_s}\Big)k\Big) \\ &= \cos\Big(2\pi\Big(\frac{f_s - f_0}{f_s}\Big)k\Big), \ \text{ since } \cos(x) = \cos(-x)\ \forall\ x \in \mathbb{R} \end{aligned}$$

Another way to put this - you cannot distinguish multiples of base frequencies with the base frequencies themselves if you sample at too low a frequency, i.e. below the **Nyquist Rate**.

1.1.3 How Can We Mitigate Aliasing?

There are several strategies to mitigate aliasing effects:

- (Note) Anti-aliasing measures must be taken before sampling. After sampling occurs, information is "lost", so to speak, and the original signal cannot be recovered.
- High frequency noise suppression with (approximate) low-pass filtering. As it turns out, exact lowpass filtering is impossible due to convergence properties of Fourier Series at cutoff frequencies (a phenomenon known as the **Gibbs Phenomenon** [1].

We will touch more on these anti-aliasing techniques and strategies throughout this lecture.

1.2 Integral Image

Next, we will shift gears somewhat to discuss the notion of an **integral image**, and the critical role this technique plays in improving computational efficiency in image processing and machine vision. We will disucss this concept in both 1D and 2D.

1.2.1 Integral Images in 1D

Block averaging is a common operation in computer vision in which we take the average over a set of values across an entire vector (1D) or matrix (2D), such as an image. This involves summing and then dividing by the total number of elements, which can become prohibitively computationally-expensive, particularly if this operation is being called many times and the averages that we need to compute are very large. Is there a more computationally-efficient way to do this?

It turns out this computationally-simpler solution is through integral images. An integral image is essentially the sum of values from the first value to the i$^{\text{th}}$ value, i.e if g_i defines the i$^{\text{th}}$ value in 1D, then:

$$G_i \triangleq \sum_{k=1}^{i} g_k\ \forall\ i \in \{1, \cdots, K\}$$

Why is this useful? Well, rather than compute averages (normalized sums) by adding up all the pixels and then dividing, we simply need to perform a single subtraction between the integral image values (followed by a division by the number of elements we are averaging). For instance, if we wanted to calculate the average of values between i and j, then:

$$\bar{g}_{[i,j]} = \frac{1}{j-i}\sum_{k=i}^{j} g_k = \frac{1}{j-i}(G_j - G_i)$$

This greatly reduces the amortized amount of computation, because these sums only need to be computed once, when we calculate the initial values for the integral image.

1.2.2 Integral Images in 2D

Now, we can extend the notion of an integral image to 2D! Note further that integral "images" can extend beyond images! E.g. these can be done with gradients, Jacobians, Hessians, etc. One example in particular is calculating a Histogram of Gradients (HoG), which is quite useful for feature matching algorithms such as Scale-Invariant Feature Transform (SIFT). These approaches also map nicely onto GPUs, enabling for even faster computation times.

Let us now see how block averaging looks in 2D - in the diagram below, we can obtain a block average for a group of pixels in the 2D range (i, j) in x and (k, l) in y using the following formula:

$$\bar{g}_{([i,j],[k,l])} = \frac{1}{(j-i)(l-k)} \sum_{x=i}^{j} \sum_{y=k}^{l} g_{x,y}$$

But can we implement this more efficiently? We can use integral images again:

$$G_{i,j} = \sum_{k=1}^{i} \sum_{l=1}^{j} g_{k,l}$$

Referencing the figure below, this becomes:

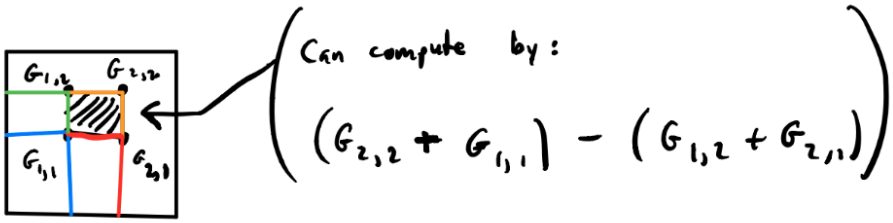


Figure 5: Block averaging using integral images in 2D. As pointed out above, block averaging also extends beyond pixels! This can be computed for other measures such as gradients (e.g. Histogram of Gradients).

Using the integral image values, the block average in the 2D range (i, j) in x and (k, l) in y becomes:

$$\bar{g}_{([i,j],[k,l])} = \frac{1}{(j-i)(l-k)} \Big((G_{j,l} + G_{i,k}) - (G_{i,l} + G_{j,k}) \Big)$$

Some comments about this:

- This analysis can be extended to higher dimensions as well! Though the integral image will take longer to compute, and the equations for computing these block averages become less intuitive, this approach generalizes to arbitrary dimensions.
- As we saw in the one-dimensional case, here we can also observe that after computing the integral image (a one-time operation that can be amortized), the computational cost for averaging each of these 2D blocks becomes **independent of the size of the block being averaged**. This stands in stark contrast to the naive implementation - here, the computational cost scales quadratically with the size of the block being averaged (or linearly in each dimension, if we take rectangular block averages).
- Why is this relevant? Recall that block averaging implements approximate lowpass filtering, which can be used as a frequency suppression mechanism to avoid aliasing when filtering.
- In other domains outside of image processing, the integral image is known as the "Summed-area Table" [2].

Since we intend to use this for approximate lowpass filtering, let us now change topics toward fourier analysis of this averaging mechanism to see how efficacious it is.

1.3 Fourier Analysis of Block Averaging

Let us consider a one-dimensional "filter" that implements an approximate lowpass filtering for mechanism through block averaging. Let us consider a function such that it takes 0 value outside of the range $(-\frac{-\delta}{2}, \frac{\delta}{2})$, and value $\frac{1}{\delta}$ inside this range:

$$h(x) = \begin{cases} \frac{1}{\delta} & x \in (-\frac{-\delta}{2}, \frac{\delta}{2}) \\ 0 & \text{o.w.} \end{cases}$$

Visually:

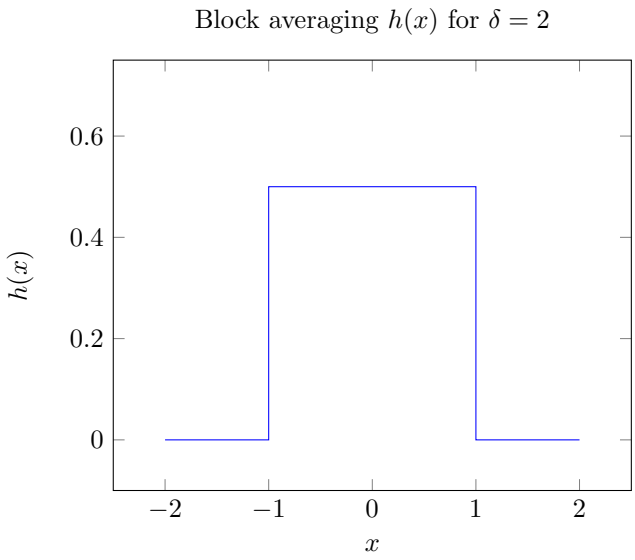


Figure 6: Example $h(x)$ for $\delta = 2$.

Let's see what this Fourier Transform looks like. Recall that the Fourier Transform (up to a constant scale factor, which varies by domain) is given by:

$$F(j\omega) = \int_{-\infty}^{\infty} f(x)e^{-j\omega x}dx$$

Where $j\omega$ corresponds to complex frequency. Substituting our expression into this transform:

$$H(j\omega) = \int_{-\infty}^{\infty} h(x)e^{-j\omega x}dx$$

$$= \int_{-\frac{\delta}{2}}^{\frac{\delta}{2}} \frac{1}{\delta}e^{-j\omega x}dx$$

$$= \frac{1}{\delta}\frac{1}{j\omega}[e^{-j\omega x}]_{x=-\frac{\delta}{2}}^{x=\frac{\delta}{2}}$$

$$= \frac{e^{-\frac{j\omega\delta}{2}} - e^{\frac{j\omega\delta}{2}}}{-j\omega\delta}$$

$$= \frac{\sin\left(\frac{\delta\omega}{2}\right)}{\frac{\delta\omega}{2}} \quad \textbf{(Sinc function)}$$

Where in the last equality statement we use the identity given by:

$$\sin(x) = \frac{e^{jx} - e^{-jx}}{-2j}$$

Graphically, this sinc function appears (for $\delta = 2$):

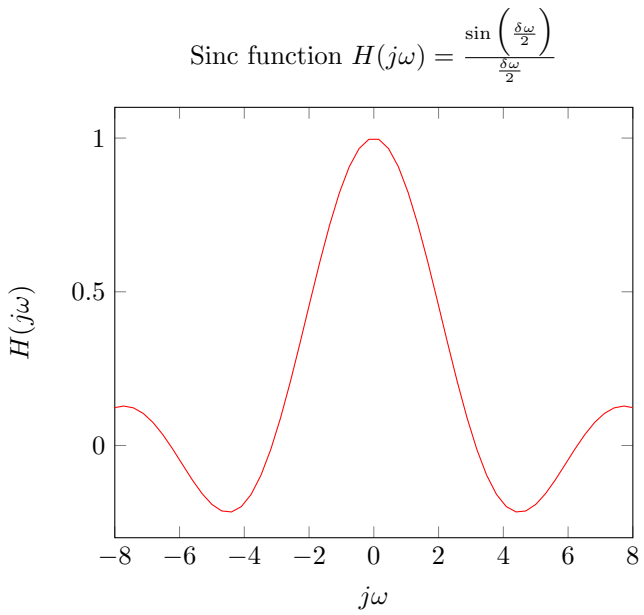


Figure 7: Example $H(j\omega)$ for $\delta = 2$. This is the Fourier Transform of our block averaging "filter".

Although sinc functions in the frequency domain help to attenuate higher frequencies, they do not make the best lowpass filters. This is the case because:

- Higher frequencies are not completely attenuated.
- The first zero is not reached quickly enough. The first zero is given by:

$$\frac{\omega_0 \delta}{2} = \frac{\pi}{2} \implies \omega_0 = \frac{\pi}{\delta}$$

Intuitively, the best lowpass filters perfectly preserve all frequencies up to the cutoff frequencies, and perfectly attenuate everything outside of the passband. Visually:

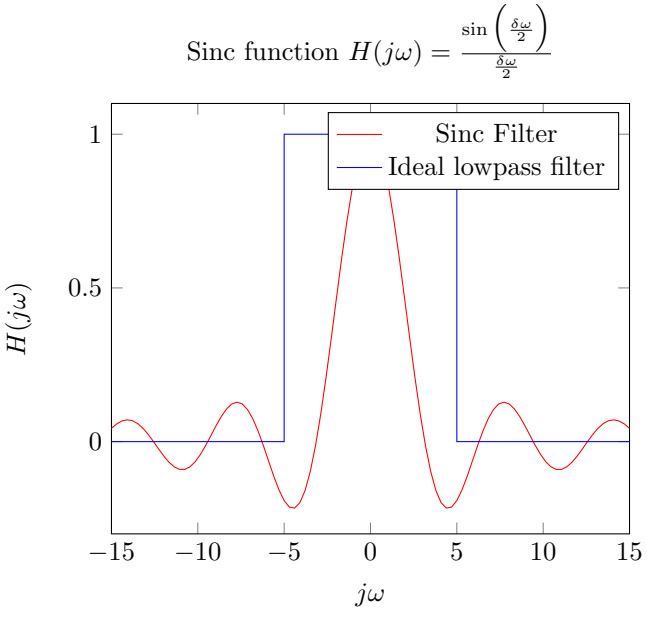


Figure 8: Frequency response comparison between our block averaging filter and an ideal lowpass filter. We also note that the "boxcar" function and the sinc function are Fourier Transform pairs!

Although sinc functions in the frequency domain help to attenuate higher frequencies, they do not make the best lowpass filters. This is the case because:

- Higher frequencies are not completely attenuated.

- The first zero is not reached quickly enough. The first zero is given by:

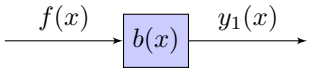
$$\frac{\omega_0 \delta}{2} = \frac{\pi}{2} \implies \omega_0 = \frac{\pi}{\delta}$$

Where else might we see this? It turns out cameras perform block average filtering becuase pixels have finite width over which to detect incident photons. But is this a sufficient approximate lowpass filtering technique? Unfortunately, oftentimes it is not. We will see below that we can improve with **repeated block averaging**.

1.4 Repeated Block Averaging

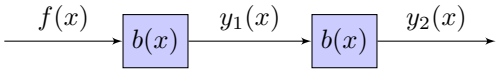
One way we can improve our ability to attenuate higher frequencies - repeated block averaging! If our "boxcar" filter given above is given by $b(x)$ (note that this was $h(x)$ above), then our previous result $y(x)$ can be written as:

$$y_1(x) = f(x) \otimes b(x)$$



What happens if we add another filter? Then, we simply add another element to our convolution:

$$y_2(x) = (f(x) \otimes b(x)) \otimes b(x) = y_1(x) \otimes b(x)$$



Adding this second filter is equivalent to convolving our signal with the convolution of two "boxcar" filters, which is a triangular filter:

Triangular filter for $\delta = 2$.

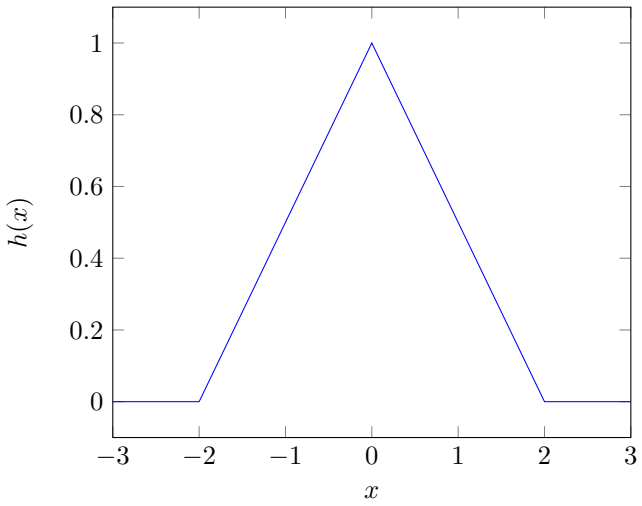


Figure 9: Example of a triangular filter resulting from the convolution of two "boxcar" filters.

Additionally, note that since convolution is associative, for the "two-stage" approximate lowpass filtering approach above, we do not need to convolve our input $f(x)$ with two "boxcar" filters - rather, we can convolve it directly with our trinagular filter $b_2(x) = b(x) \otimes b(x)$:

$$\begin{aligned} y_2(x) &= (f(x) \otimes b(x)) \otimes b(x) \\ &= f(x) \otimes (b(x) \otimes b(x)) \\ &= f(x) \otimes b_2(x) \end{aligned}$$

Let us now take a brief aside to list out how discontinuities affect Fourier Transforms in the frequency domains:

- **Delta Function**: $\delta(x) \overset{\mathcal{F}}{\leftrightarrow} 1$

  Intuition: Convolving a function with a delta function does not affect the transform, since this convolution simply produces the function.

- **Unit Step Function**: $u(x) \overset{\mathcal{F}}{\leftrightarrow} \frac{1}{j\omega}$

  Intuition: Convolving a function with a step function produces a degree of averaging, reducing the high frequency components and therefore weighting them less heavily in the transform domain.

- **Ramp Function**: $r(x) \overset{\mathcal{F}}{\leftrightarrow} -\frac{1}{\omega^2}$

  Intuition: Convolving a function with a ramp function produces a degree of averaging, reducing the high frequency components and therefore weighting them less heavily in the transform domain. **Derivative**: $\frac{d}{dx} f(x) \overset{\mathcal{F}}{\leftrightarrow} j\omega F(j\omega)$

  Intuition: Since taking derivatives will increase the sharpness of our functions, and perhaps even create discontinuities, a derivative in the spatial domain corresponds to multiplying by $j\omega$ in the frequency domain.

As we can see from above, the more "averaging" effects we have, the more the high-frequency components of the signal will be filtered out. Conversely, when we take derivatives and create discontinuities in our spatial domain signal, this increases high frequency components of the signal because it introduces more variation.

To understand how we can use repeated block averaging in the Fourier domain, please recall the following special properties of Fourier Transforms:

1. **Convolution in the spatial domain corresponds to multiplication in the frequency domain**, i.e. for all $f(x), g(x), h(x)$ with corresponding Fourier Transforms $F(j\omega), G(j\omega), H(j\omega)$, we have:

$$h(x) = f(x) \otimes g(x) \overset{\mathcal{F}}{\leftrightarrow} H(j\omega) = F(j\omega)G(j\omega)$$

2. **Multiplication in the spatial domain corresponds to convolution in the frequency domain**, i.e. for all $f(x), g(x), h(x)$ with corresponding Fourier Transforms $F(j\omega), G(j\omega), H(j\omega)$, we have:

$$h(x) = f(x)g(x) \overset{\mathcal{F}}{\leftrightarrow} H(j\omega) = F(j\omega) \otimes G(j\omega)$$

For block averaging, we can use the first of these properties to understand what is happening in the frequency domain:

$$y_2(x) = f(x) \otimes (b(x) \otimes b(x)) \overset{\mathcal{F}}{\leftrightarrow} Y(j\omega) = F(j\omega)(B(j\omega)^2)$$

This operation is equivalent to a sinc^2 function in the spatial domain:

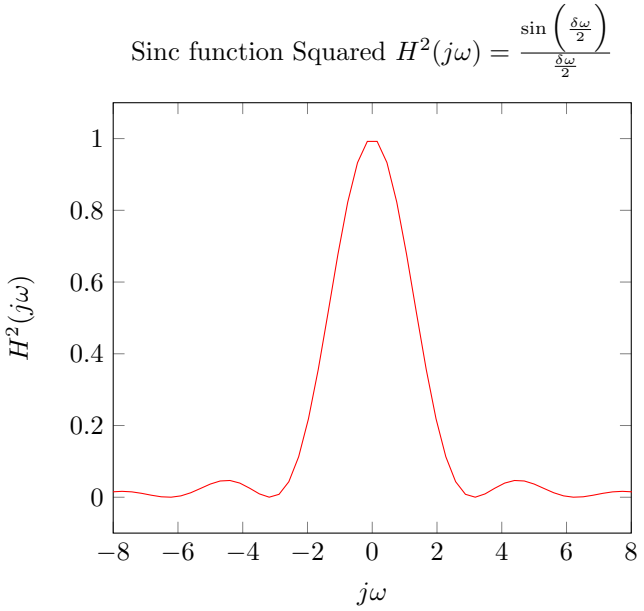


Figure 10: Example $H^2(j\omega)$ for $\delta = 2$. This is the Fourier Transform of our block averaging "filter" convolved with itself in the spatial domain.

.

This is not perfect, but it is an improvement. In fact, the frequencies with this filter drop off with magnitude $\left(\frac{1}{\omega}\right)^2$. What happens if we continue to repeat this process with more block averaging filters? It turns out that for N "boxcar" filters that we use, the magnitude will drop off as $\left(\frac{1}{\omega}\right)^N$. Note too, that we do not want to go "too far" in this direction, because this repeated block averaging process will also begin to attenuate frequencies in the passband of the signal.

1.4.1 Warping Effects and Numerical Fourier Transforms: FFT and DFT

Two main types of numerical transforms we briefly discuss are the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT). The FFT is an extension of the DFT that relies on using a "divide and conquer" approach to reduce the computational runtime from $f(N) \in O(N^2)$ to $f(N) \in O(N \log N)$ [3].

Mathematically, the DFT is given as a transform that transforms a sequence of N complex numbers $\{x_n\}_{n=1}^N$ into another sequence of N complex numbers $\{X_k\}_{k=1}^N$ [4]. The transform for the K^{th} value of this output sequence is given in closed-form as:

$$X_k = \sum_{i=1}^{N} x_n e^{-j\frac{2\pi}{N}kn}$$

And the inverse transform for the n^{th} value of this input sequence is given as:

$$x_n = \sum_{k=1}^{N} X_k e^{j\frac{2\pi}{N}kn}$$

One aspect of these transforms to be especially mindful of is that they introduce a wrapping effect, since transform values are spread out over 2π intervals. This means that the waveforms produced by these transforms, in both the spatial (if we take the inverse transform) and frequency domains may be repeated - this repeating can introduce undesirable discontinuities, such as those seen in the graph below:

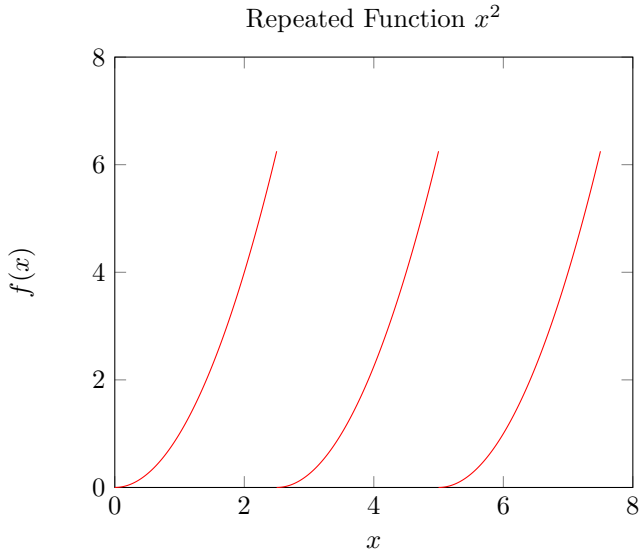


Figure 11: Example of a repeated waveform that we encounter when looking at DFTs and FFTs.

Fun fact: It used to be thought that natural images had a power spectrum (power in the frequency domain) that falls off as $\frac{1}{\omega}$. It turns out that this was actually caused by warping effects introduced by discrete transforms.

This begs the question - how can we mitigate these warping effects? Some methods include:

- **Apodizing**: This corresponds to multiplying your signal by a waveform, e.g. Hamming's Window, which takes the form akin to a Gaussian, or an inverted cosine.
- **Mirroring**: Another method to mitigate these warping effects is through waveform mirroring - this ensures continuity at points where discontinuties occurred:

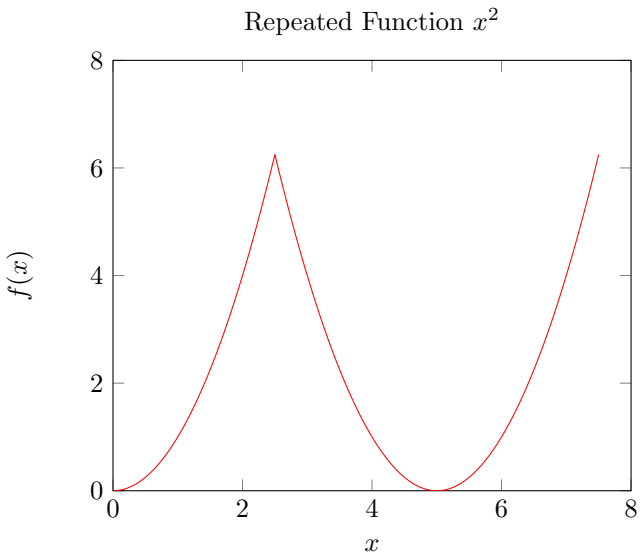


Figure 12: Example of a mirrored waveform that we can use to counter and mitigate the discontinuity effects of warping from transforms such as the DFT and FFT.

With this approach, the power spectrum of these signals falls off at $\frac{1}{\omega^2}$, rather than $\frac{1}{\omega}$.

- **Infinitely Wide Signal**: Finally, a less practical, but conceptual helpful method is simply to take an "infinitely wide signal".

Let us now switch gears to talk more about the unit impulse and convolution.

1.5 Impulses and Convolution

In this section, we will review impulse functions, and discuss how they relate to many properties with convolution. We will begin by reviewing delta functions and their properties.

1.5.1 Properties of Delta Functions

Recall that the Dirac delta function $\delta(x)$, or impulse function, is defined according to the two properties:

1. **Unit Area**: $\int_{-\infty}^{\infty} \delta(x)dx = 1$
2. **Sifting Property**: $f(x_0) = \int_{-\infty}^{\infty} f(x)\delta(x - x_0)dx$

Another way to conceptualize delta functions is through **probabilistic distributions**. We can use Gaussians (one of the only distributions to have a Fourier Transform). Recall that the (zero-mean) Gaussian Fourier Transform pair is given by:

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x}{2\sigma^2}} \overset{\mathcal{F}}{\longleftrightarrow} e^{-\frac{\omega^2\sigma^2}{2}}$$

An impulse can be conceptualized as the limit in which the variance of this Gaussian distribution σ^2 goes to 0, which corresponds to a Fourier Transform of 1 for all frequencies (which is the Fourier Transform of a delta function).

Another way to consider impulses is that they are the limit of "boxcar" functions as their width goes to zero.

Let us next generalize from a single impulse function to combinations of these functions.

1.5.2 Combinations of Impulses

When are combinations of impulses helpful? it turns out that one particular combination can be used for approximating the derivative using our prior work on finite differences:

$$h(x) = \frac{1}{\epsilon}\Big(\delta(x + \frac{\epsilon}{2}) - \delta(x - \frac{\epsilon}{2})\Big) \text{ for some } \epsilon > 0$$

Correlating (*note that this is not convolution - if we were to use convolution, this derivative would be flipped) this combination of impulse "filter" with an arbitrary function $f(x)$, we compute a first-order approximation of the derivative:

$$\begin{aligned} f'(x) &\approx \int_{-\infty}^{\infty} f(x)h(x)dx \\ &= \int_{-\infty}^{\infty} \Big(\frac{1}{\epsilon}\Big(\delta(x+\frac{\epsilon}{2}) - \delta(x-\frac{\epsilon}{2})\Big)\Big)dx \end{aligned}$$

Therefore, combinations of impulses can be used to represent the same behavior as the "computational molecules" we identified before. It turns out that there is a close connection between **linear, shift-invariant** operators and **derivative operators**.

With impulses motivated, let us now formally review convolution.

1.5.3 Convolution Review

Recall from the example above that the convolution operation is simply the result of the correlation operation flipped. Mathematically, the convolution of functions $f(x)$ and $h(x)$ is given by the following commutative operation:

$$\begin{aligned} g(x) &= f(x) \otimes h(x) \\ &= \int_{-\infty}^{\infty} f(\xi)h(x-\xi)d\xi \\ &= h(x) \otimes g(x) \\ &= \int_{-\infty}^{\infty} h(\xi)f(x-\xi)d\xi \end{aligned}$$

1.5.4 Analog Filtering with Birefringent Lenses

Why filter in the analog domain? Returning to our original problem of mitigating aliasing artifacts through high-frequency suppression, unfortunately, if we wait to digitally filter out high frequencies after the image has already been taken by a camera or sensor, then we have already caused aliasing.

One way to achieve this analog filtering is through **Birefringent Lenses**. Here, we essentially take two "shifted" images by convolving the image with a symmetric combination of offset delta functions, given mathematically by:

$$h(x) = \frac{1}{2}\delta(x+\frac{\epsilon}{2}) + \frac{1}{2}\delta(x-\frac{\epsilon}{2}) \text{ for some } \epsilon > 0$$

Let us look at the Fourier Transform of this filter, noting the following Fourier Transform pair:

$$\delta(x-x_0) \overset{\mathcal{F}}{\leftrightarrow} e^{-j\omega x_0}$$

With this we can then express the Fourier Transform of this filter as:

$$\begin{aligned} F(j\omega) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} h(x)e^{-j\omega x}dx \\ &= \frac{1}{2}\Big(e^{-\frac{j\omega\epsilon}{2}} + e^{\frac{j\omega\epsilon}{2}}\Big) \\ &= \cos\Big(\frac{\omega\epsilon}{2}\Big) \end{aligned}$$

With this framework, the first zero to appear here occurs at $\omega_0 = \frac{\pi}{\epsilon}$. A few notes about these filters, and how they relate to high-frequency noise suppression.

- When these birefringent lenses are cascaded with a **block averaging** filter, this results in a combined filtering scheme in which the zeros of the frequency responses of these filters cancel out most of the high-frequency noise.
- In the 2D case, we will have 2 birefringent filters, one for the x-direction and one for the y-direction. Physically, these are rotated 90 degrees off from one another, just as they are for a 2D cartesian coordinate system.

- High-performance lowpass filtering requires a large **support** (see definition of this below if needed) - the computational costs grow linearly with the size of the support in 1D, and quadratically with the size of the support in 2D. The support of a function is defined as the set where $f(\cdot)$ is nonzero [5]:

$$\text{supp}(f) = \{x : f(x) \neq 0, x \in \mathbb{R}\}$$

- Therefore, one way to reduce the computatonal costs of a filtering system is to reduce the size/cardinality of the support $|\text{supp}(f)|$ - in some sense to encourage sparsity. Fortunately, this does not necessarily mean looking over a narrower range, but instead just considering less points overall.

1.5.5 Derivatives and Integrals as Convolution Operators and FT Pairs

As we have seen before, convolving a function with a unit step function results in integrating the given function. Let us verify this below:

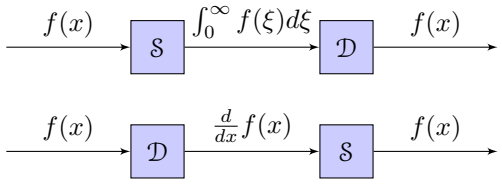
$$\begin{aligned} f(x) \otimes u(x) &= \int_{-\infty}^{\infty} f(\xi)u(x-\xi)d\xi \\ &= \int_{-\infty}^{\infty} f(x-\xi)u(\xi)d\xi \\ &= \int_{0}^{\infty} f(\xi)d\xi \end{aligned}$$

Therefore, we can represent integral and derivative operators as Fourier Transform pairs too, denoted $\mathcal{S}$ for integration and $\mathcal{D}$ for derivative:

- $\mathcal{S} \overset{\mathcal{F}}{\longleftrightarrow} \frac{1}{j\omega}$
- $\mathcal{D} \overset{\mathcal{F}}{\longleftrightarrow} j\omega$

Niote that we can verify this by showing that convolving these filter operators corresponds to multiplying these transforms in frequency space, which results in no effect when cascaded together:

$$(f(x) \otimes \mathcal{D}) \otimes \mathcal{S} = f(x) \otimes (\mathcal{D} \otimes \mathcal{S}) \overset{\mathcal{F}}{\longleftrightarrow} F(j\omega)\Big(j\omega\frac{1}{j\omega}\Big) = F(j\omega) \overset{\mathcal{F}}{\longleftrightarrow} f(x)$$



Can we extend this to higher-order derivatives? It turns out we can. One example is the convolution of two derivative operators, which becomes:

$$h(x) = \delta(x+\frac{\epsilon}{2}) - 2\delta(x) + \delta(x-\frac{\epsilon}{2}) = \mathcal{D} \otimes \mathcal{D} \overset{\mathcal{F}}{\longleftrightarrow} H(j\omega) = D(j\omega)^2 = (j\omega)^2 = -\omega^2 \text{ (Recall that } j^2 = -1)$$

In general, this holds. Note that the number of integral operators $\mathcal{S}$ must be equal to the number of derivative operators $\mathcal{D}$, e.g. for K order:

$$\Big(\otimes_{i=1}^{K} \mathcal{S}\Big) \otimes \Big(\Big(\otimes_{i=1}^{K} \mathcal{D}\Big) \otimes f(x)\Big)$$

1.5.6 Interpolation and Convolution

A few notes about interpolation methods:

- These methods work well computationally when there is sparsity in the operators we with - for instance, cubic or linear interpolation. Photoshop and other photo-editing software frameworks iuse this interpolation techniques. Performance deteriorates when we switch from **cubic interpolation** to **Nearest Neighbor interpolation**.
- The **bicubic spline** is a popular interpolation technique. It was invented by IBM and was used in early satellite image processing. However, it requires many neighboring pixels, as it is composed of 7 points to interpolate over, so it is less computationally-efficient.

- Recall that one key element of computational efficiency we pursue is to use integral images for block averaging, which is much more efficient than computing naive sums, especially if (1) This block averaging procedure is repeated many times (the amortized cost of computing the integral image is lessened) and (2) This process is used in higher dimensions.
- **Linear interpolation** can be conceptualized as connecting points together using straight lines between points. This corresponds to piecewise-linear segments, or, convolution with a triangle filter, which is simply the convolution of two "boxcar filters":

  $$f(x) = f(1)x + f(0)(1-x)$$

  Unfortunately, one "not-so-great" property of convolving with triangular filters for interpolation is that the noise in the interpolated result varies depending on how far away we are from the sampled noise.
- **Nearest Neighbor** techniques can also be viewed through a convolutional lens - since this method produces piecewise-constant interpolation, this is equivalent to convolving our sampled points with a "boxcar" filter!

1.5.7 Rotationally-Symmetric Lowpass Filter in 2D

Where u and v are the spatial frequencies of x and y, i.e. the 2D Fourier Transform and Inverse Fourier Transform then take the forms of:

$$F(u,v) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j(ux+vy)} dxdy$$
$$f(x,y) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v) e^{j(ux+vy)} dudv$$

The inverse transform of this can be thought of as a **sinc** function in polar coordinates:

$$f(\rho,\theta) = \frac{B^2}{2\pi} \frac{J_1(\rho B)}{\rho B}$$

A few notes about this inverse transform function:

- This is the point spread function of a microscope.
- $J_1(\cdot)$ is a 1st-order Bessel function.
- This relates to our **defocusing** problem that we encountered before.
- In the case of defocusing, we can use the "symmetry" property of the Fourier Transform to deduce that if we have a circular point spread function resulting from defocusing of the lens, then we will have a Bessel function in the frequency/Fourier domain.
- Though a pointspread function is a "pillbox" in the ideal case, in practice this is not perfect due to artifacts such as lens aberrations.

1.6 References

1. Gibbs Phenomenon, https://en.wikipedia.org/wiki/Gibbs_phenomenon
2. Summed-area Table, https://en.wikipedia.org/wiki/Summed-area_table
3. Fast Fourier Transform, https://en.wikipedia.org/wiki/Fast_Fourier_transform
4. Discrete Fourier Transform, https://en.wikipedia.org/wiki/Discrete_Fourier_transform
5. Support, https://en.wikipedia.org/wiki/Support_(mathematics)