Lecture 5: Multidimensional Signal Processing

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2021

Reading: Multidimensional Signal, Image, and Video Processing and Coding, John Woods, Chapter 1

Outline

What is a Multidimensional Signal?

A multidimensional signal is one that can be indexed in many directions. For example, a typical video that you would play on your laptop is a 4-dimensional signal, $x[k, t, r, c]$:

- k indexes color ($k = 0$ for red, $k = 1$ for green, $k = 2$ for blue)
- t is the frame index
- r is the row index
- c is the column index

If there are 3 colors, 30 frames/second, 480 rows and 640 columns, with one byte per pixel, then that's
$3 \times 30 \times 480 \times 640 = 27684000$ bytes/sec.

Generic Indexing of Multidimensional Signals

When we don't care about the meaning of the indices, we'll often talk about $x[n_1, n_2]$, where

- n_1 is the index along the first dimension
- n_2 is the index along the second dimension
- Anything we say about two dimensions can usually be extended to 3 or 4 dimensions, unless specified otherwise

Vector Indexing of Multidimensional Signals

For convenience, we'll sometimes use a vector index.

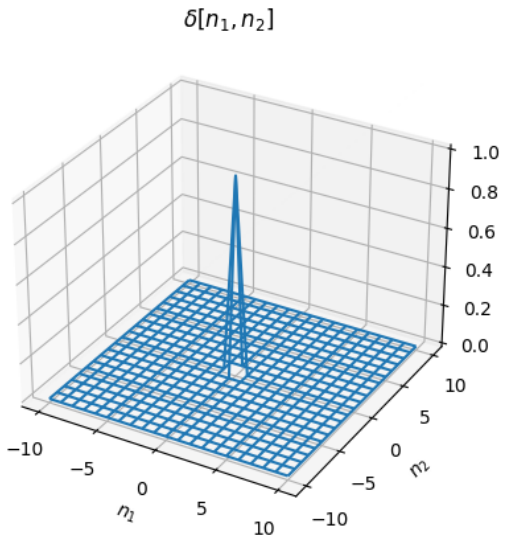
$$x[\vec{n}] = x[n_1, n_2], \quad \vec{n} = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}$$

The use of vector indices is a way of abstracting away from the exact dimension of the problem. In general, $\vec{n}$ might be 2d, 3d, 4d, or whatever dimension is necessary for the problem at hand.

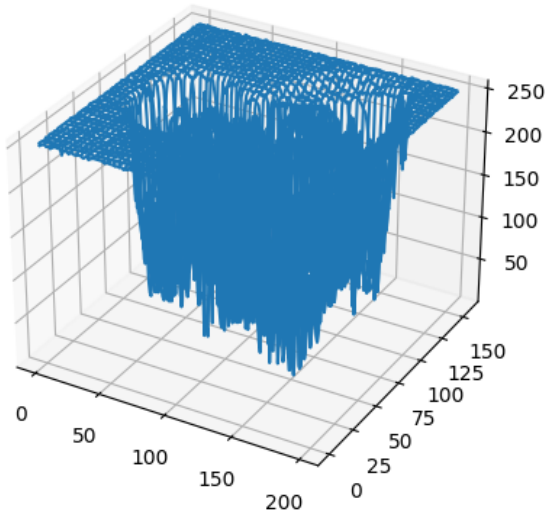
2D Delta Function

Example: the two-dimensional delta function is

$$\delta[n_1, n_2] = \begin{cases} 1 & n_1 = 0, n_2 = 0 \\ 0 & \text{otherwise} \end{cases}$$



Example: this signal, $f[n_1, n_2]$, is an image of Joseph Fourier, simplified from a public domain image available on Wikipedia.
Here, n_1 is the row index, n_2 is the column index.
Notice that, as in most images, white is the highest possible amplitude. Thus the background regions (in the image on the left) show up as very high amplitude regions (in the 3D mesh plot on the right).



Outline

Multidimensional Fourier Transform

The 2d Fourier transform is

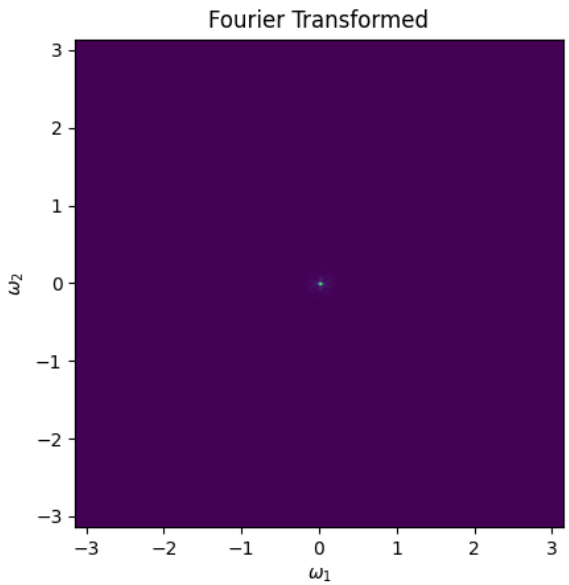
$$X(\omega_1, \omega_2) = \sum_{n_1} \sum_{n_2} x[n_1, n_2] e^{-j(\omega_1 n_1 + \omega_2 n_2)}$$

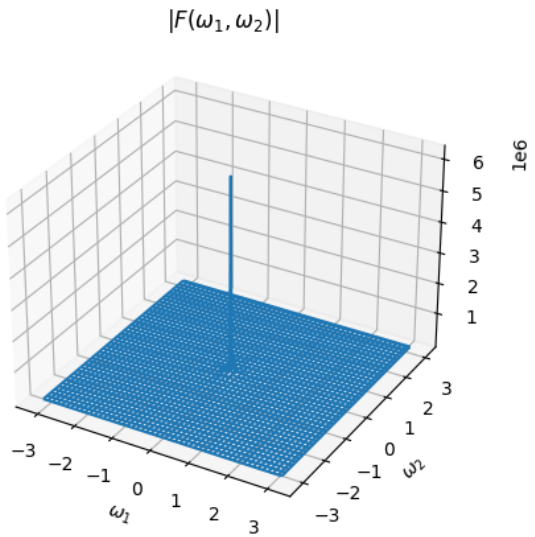
Vector indexing of the image can be matched by vector indexing of the frequency domain, e.g., $\vec{\omega} = [\omega_1, \omega_2]^T$. In that case,

$$X(\vec{\omega}) = \sum_{\vec{n}} x[\vec{n}] e^{-j\vec{\omega}^T \vec{n}}$$

Example: Fourier Transformed

For example, here is the magnitude $|F(\omega_1, \omega_2)|$ of the 2D Fourier transform of the image of Joseph Fourier. Notice that, like most images, it has almost all of its energy at very low frequencies (near $\omega_1 \approx 0, \omega_2 \approx 0$).

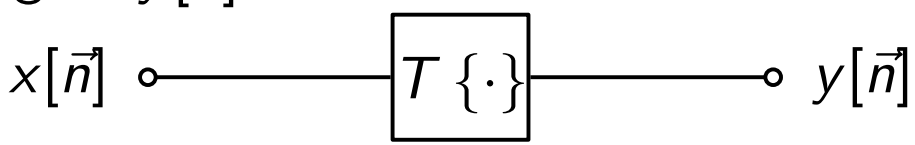




Outline

Multidimensional Systems

A multidimensional system $y[\vec{n}] = T\{x[\vec{n}]\}$ takes in a signal $x[\vec{n}]$, and outputs a signal $y[\vec{n}]$.



Linear Systems

A system is **linear** iff, when you add two signals at the input, the corresponding output is the sum of their two component outputs. Thus, if:

$$T\{x_1[\vec{n}]\} = y_1[\vec{n}]$$
$$T\{x_2[\vec{n}]\} = y_2[\vec{n}]$$

then $T\{\cdot\}$ is linear if and only if:

$$T\{x_1[\vec{n}] + x_2[\vec{n}]\} = y_1[\vec{n}] + y_2[\vec{n}]$$

As a special case of the above, scaling the input by any constant a causes the output to scale by the same constant, i.e., $T\{ax[\vec{n}]\} = ay[\vec{n}]$.

Shift-Invariant Systems

A system is **shift-invariant** iff, when you shift the input signal by any offset, the corresponding output is shifted by the same offset vector. Thus if

$$T\{x[n_1, n_2]\} = y[n_1, n_2]$$

then $T\{\cdot\}$ is shift-invariant if and only if:

$$T\{x[n_1 - m_1, n_2 - m_2]\} = y[n_1 - m_1, n_2 - m_2]$$

Outline

Multidimensional Convolution

Any linear, shift-invariant system can be implemented as a convolution. 2D convolution is defined as

$$\begin{aligned} y[n_1, n_2] &= x[n_1, n_2] * h[n_1, n_2] \\ &= \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} x[m_1, m_2] h[n_1 - m_1, n_2 - m_2] \end{aligned}$$

We can generalize to vector indices like this:

$$\begin{aligned} y[\vec{n}] &= x[\vec{n}] * h[\vec{n}] \\ &= \sum_{\vec{m}} x[\vec{m}] h[\vec{n} - \vec{m}] \end{aligned}$$

Notice that this is expensive! If both $x[\vec{n}]$ and $h[\vec{n}]$ are 100×100 signals, then the convolution requires $(100)^4$ multiplications.

Impulse Response

The function $h[n_1, n_2]$ has many names. We sometimes call it the **filter** or **kernel**. We also call it the **impulse response** or **point spread function** (PSF) because it is the response of the system to an input impulse:

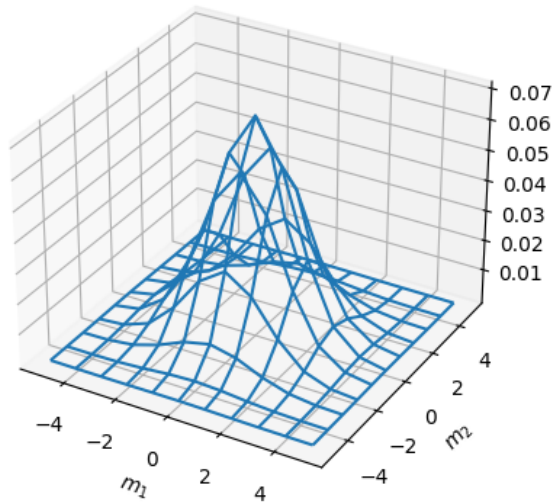
$$\begin{aligned}\delta[n_1, n_2] * h[n_1, n_2] &= \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} \delta[m_1, m_2] h[n_1 - m_1, n_2 - m_2] \\ &= h[n_1, n_2]\end{aligned}$$

Example: Gaussian Blur

For example, consider the impulse response shown below. This is a Gaussian blur kernel of size $(2M+1) \times (2M+1)$, with variance $\sigma^2 = 1.5$, meaning that

$$h[n_1, n_2] = \begin{cases} \frac{1}{2\pi\sigma^2} e^{-\left(\frac{n_1^2+n_2^2}{2\sigma^2}\right)} & -M \leq n_1, n_2 \leq M \\ 0 & \text{otherwise} \end{cases}$$

Gaussian Blur kernel $g[m_1, m_2]$, with $\sigma = 1.5$



Example: Gaussian Blur

Filtering an image through any lowpass filter results in a smoothed version of the same image. Here's what we get when we convolve the image of Fourier with the Gaussian blur filter:

→

Frequency Response

The frequency response of an LSI system is the Fourier transform of its impulse response:

$$H(\vec{\omega}) = \sum_{\vec{n}} h[\vec{n}] e^{-j\vec{\omega}^T \vec{n}}$$

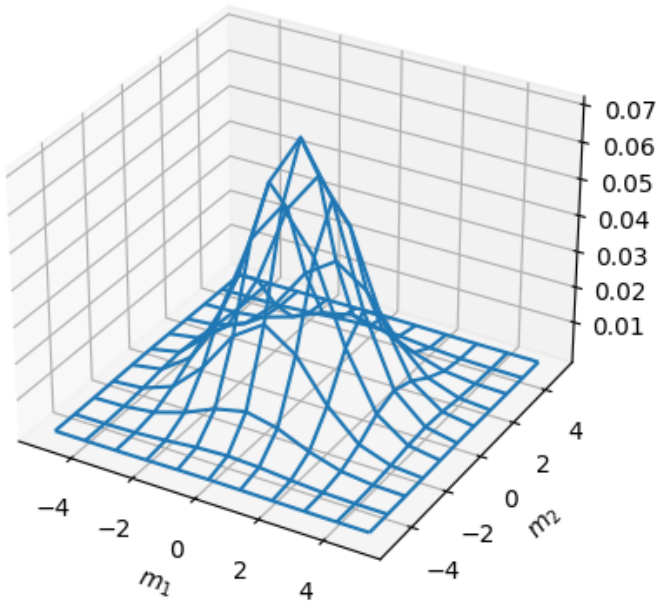
The Fourier transform of convoluton is multiplication:

$$y[\vec{n}] = x[\vec{n}] * h[\vec{n}] \Leftrightarrow Y(\vec{\omega}) = H(\vec{\omega})X(\vec{\omega})$$

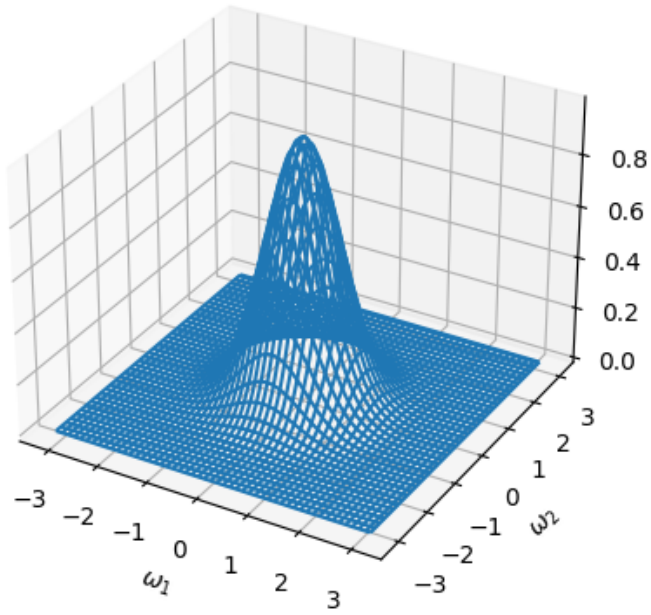
Example: Gaussian Blur

A Gaussian blur kernel is a lowpass-filter, meaning that it has higher energy at low frequencies than at high frequencies. Here is the Gaussian kernel, and its Fourier transform.

Gaussian Blur kernel $g[m_1, m_2]$, with $\sigma = 1.5$



Gaussian Blur $|G(\omega_1, \omega_2)|$



$\Leftrightarrow$

Outline

Computational Complexity of Regular Convolution

Recall the formula for regular convolution:

$$\begin{aligned} y[n_1, n_2] &= x[n_1, n_2] * h[n_1, n_2] \\ &= \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} x[m_1, m_2] h[n_1 - m_1, n_2 - m_2] \end{aligned}$$

This is an extremely expensive operation. If $h[\vec{n}]$ and $x[\vec{n}]$ are both $N \times N$ signals, then convolution requires N^4 multiplications.

Reduced Computation

There are two ways computation can be reduced:

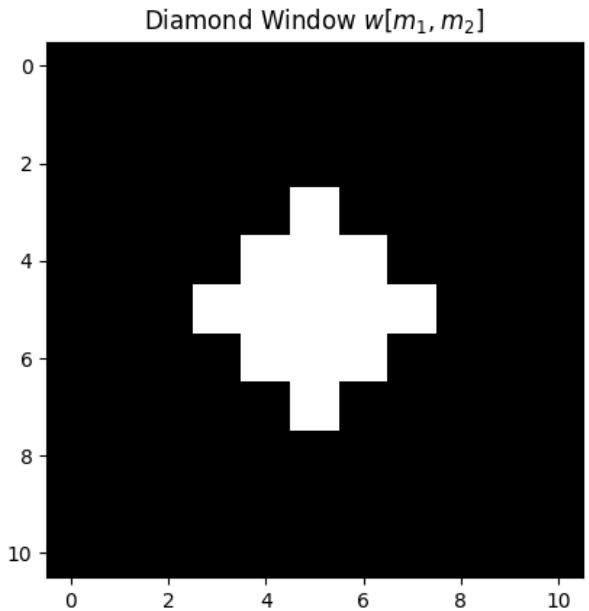
- Keep the kernel $h[\vec{n}]$ very small. This is the method usually used in neural nets, e.g., a 3×3 kernel is common.
- Use separable convolution. This is the method usually used for filters that are designed by hand.

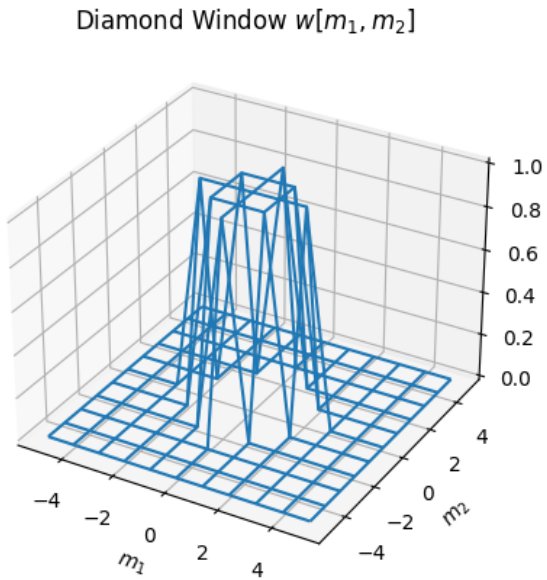
Separable Filters

A filter $h[n_1, n_2]$ is called "separable" if it can be written as

$$h[n_1, n_2] = h_1[n_1]h_2[n_2]$$

Not all filters are separable. For example, the diamond window shown below is not separable.





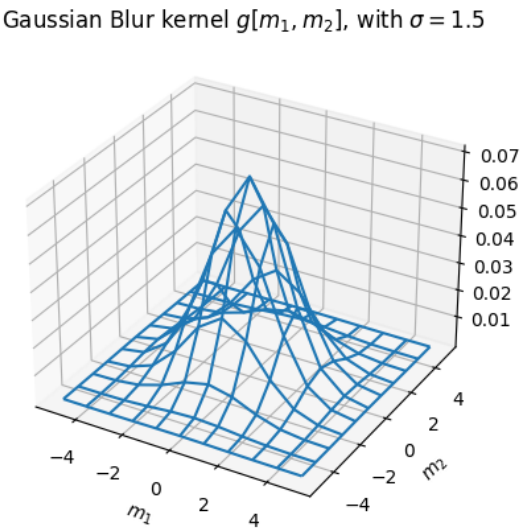
Example: Gaussian Blur

An important example of a separable filter is the Gaussian blur filter, with variance σ^2:

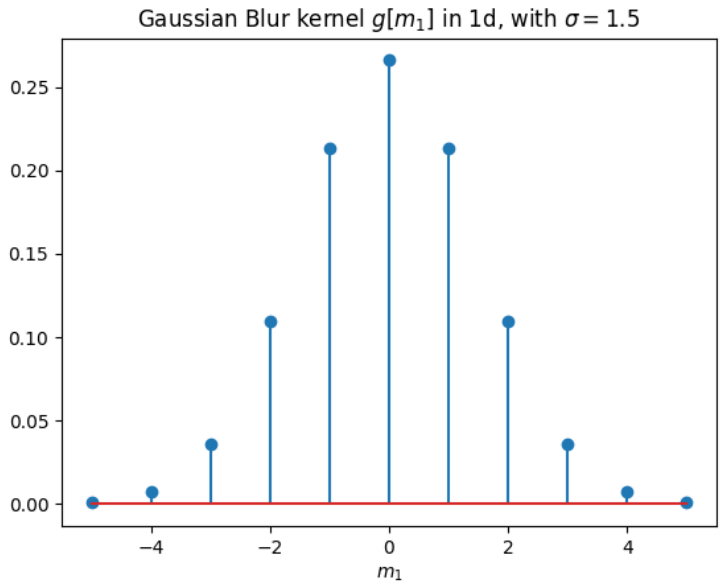
$$\begin{aligned} h[n_1, n_2] &= \frac{1}{2\pi\sigma^2} e^{-\left(\frac{n_1^2+n_2^2}{2\sigma^2}\right)}, \quad -M \leq n_1, n_2 \leq M \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{n_1^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{n_2^2}{2\sigma^2}}\right), \quad -M \leq n_1, n_2 \leq M \\ &= h_1[n_1] h_2[n_2] \end{aligned}$$

Example: Gaussian Blur

This 2D filter:



. . . is the product of two 1D filters, each with this form:



Separable Convolution

If a filter is separable, then the computational cost of convolution can be reduced by the following trick:

$$
\begin{aligned}
y[n_1, n_2] &= x[n_1, n_2] * h[n_1, n_2] \\
&= \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} h[m_1, m_2] x[n_1 - m_1, n_2 - m_2] \\
&= \sum_{m_1=-\infty}^{\infty} h_1[m_1] \left(\sum_{m_2=-\infty}^{\infty} h_2[m_2] x[n_1 - m_1, n_2 - m_2] \right)
\end{aligned}
$$

Separable Convolution: Computational Complexity

$$x[\vec{n}] * h[\vec{n}] = \sum_{m_1=-\infty}^{\infty} h_1[m_1] \left(\sum_{m_2=-\infty}^{\infty} h_2[m_2] x[n_1 - m_1, n_2 - m_2] \right)$$

- The part inside the parentheses computes a 1D convolution, which is an N^2 operation, for each of the N rows. Cost: N^3.
- The part outside the parentheses then computes a 1D convolution for each of the N columns. Cost: N^3.
- Total cost: $2N^3$ computations.

If N is large (e.g., 1000), then often, $2N^3$ is a quite reasonable number (e.g., two billion computations), while N^4 is quite unreasonable (e.g., a trillion).

Separable Convolution: Notation

It is sometimes useful to have a special notation for separable convolution. For example, we could define the **row convolution** operator $*_2$, and the **column convolution** operator $*_1$:

$$h_2[n_2] *_2 x[n_1, n_2] \equiv \sum_{m_2=-\infty}^{\infty} h_2[m_2] x[n_1, n_2 - m_2]$$

$$h_1[n_1] *_1 x[n_1, n_2] \equiv \sum_{m_1=-\infty}^{\infty} h_1[m_1] x[n_1 - m_1, n_2]$$

Then:

$$x[n_1, n_2] * h[n_1, n_2] = h_1[n_1] *_1 h_2[n_2] *_2 x[n_1, n_2]$$

Separable Convolution: How to Do It

Using `numpy`, you will almost always want to use separable convolutions. To do that, you need to define some variable $v[\vec{n}]$ which is intermediate between $x[\vec{n}]$ and $y[\vec{n}]$. Then you do the following:

- Compute $v[\vec{n}] = h_2[n_2] *_2 x[\vec{n}]$, i.e., convolve h_2 with each of the rows of the image.
- Compute $y[\vec{n}] = h_1[n_1] *_1 v[\vec{n}]$, i.e., convolve h_1 with each of the columns of the image.

In fact, that's how I computed the Gaussian-blurred image of Fourier.

Other facts about separable filters

- The Fourier transform is a separable operation, so it can be done most efficiently by first transforming each row, then transforming each column. This is what `np.fft.fft2` does for you.

$$X(\omega_1, \omega_2) = \sum_{n_1} e^{-j\omega_1 n_1} \left(\sum_{n_2} e^{-j\omega_2 n_2} x[n_1, n_2] \right)$$

- If a filter is separable, then its Fourier transform is also separable:

$$h[n_1, n_2] = h_1[n_1] h_2[n_2] \Leftrightarrow H(\omega_1, \omega_2) = H_1(\omega_1) H_2(\omega_2)$$

Outline

Pencil and Paper Examples

Consider the $M \times M$ rectangular window signal:

$$h[n_1, n_2] = \begin{cases} 1 & 0 \leq n_1, n_2 \leq M-1 \\ 0 & \text{otherwise} \end{cases}$$

Is this a separable filter? Is it a lowpass or a highpass filter? Write $x[\vec{n}] * h[\vec{n}]$ — what is another name for the output of this filter? What is the frequency response $H(\vec{\omega})$?

Jupyter Examples

Create a rectangular window with this form:

$$h[n_1, n_2] = \begin{cases} 1 & 0 \leq n_1, n_2 \leq M-1 \\ 0 & \text{otherwise} \end{cases}$$

Convolve it with some image, and show the result. Take its Fourier transform using `np.fft.fft2` (making sure to zero-pad it to a large N, maybe $N = 7M$ or so), then fftshift it using `np.fft.fftshift`, and show the absolute value of the result using `matplotlib.pyplot.plot_wireframe`.

Outline

Summary

- **Fourier Transform:**

$$H(\vec{\omega}) = \sum_{\vec{n}} h[\vec{n}] e^{-j\vec{\omega}^T \vec{n}}$$

- **Convolution:**

$$h[\vec{n}] * x[\vec{n}] = \sum_{\vec{m}} h[\vec{m}] x[\vec{n} - \vec{m}]$$

- **Separable Filtering:** If $h[n_1, n_2] = h_1[n_1] h_2[n_2]$, then

$$h[\vec{n}] * x[\vec{n}] = h_1[n_1] *_1 (h_2[n_2] *_2 x[\vec{n}])$$