Parseval's Theorem

Once an orthobasis is introduced into any Hilbert space, it can be mapped to standard Euclidean space. As we will see, inner products between vectors become standard dot products between their expansion coefficients, and (induced) norms become standard sum-of-squares norms. The key identity is called Parseval's theorem[1], and it is relatively easy to establish.

Let $\mathcal{S}$ be a Hilbert space with inner product $\langle \cdot, \cdot \rangle_S$ which induces the norm $\| \cdot \|_S$. Let $\{v_k\}_{k\in\Gamma}$ be an orthobasis[2] for $\mathcal{S}$. Then for every $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}$,

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_S = \sum_{k\in\Gamma} \alpha_k \overline{\beta_k},$$

where

$$\alpha_k = \langle \boldsymbol{x}, \boldsymbol{v}_k \rangle_S, \qquad \beta_k = \langle \boldsymbol{y}, \boldsymbol{v}_k \rangle_S.$$

You can think of the $\{\alpha_k\}$ as the transform coefficients of $\boldsymbol{x}$ and the $\{\beta_k\}$ as the transform coefficients of $\boldsymbol{y}$. So we have

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_S = \langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle_{\ell_2},$$
$$\|\boldsymbol{x}\|_S^2 = \|\boldsymbol{\alpha}\|_2^2.$$

$\Rightarrow$ An orthobasis makes every Hilbert space **equivalent** to ℓ_2.

[1] In its most common usage, the name Parseval is usually associated with the preservation of energy (to within a constant) between a function $f(t)$ and its Fourier transform, as we discussed in the last set of notes. Here, we will use this terminology more broadly.

[2] We are using Γ to be an arbitrary index set here; it can be either finite, e.g. $\Gamma = 1, 2, \ldots, N$, or infinite, e.g. $\Gamma = \mathbb{Z}$.

All of the geometry (lengths, angles) maps into standard Euclidean geometry in coefficient space. As you can imagine, this is a pretty useful fact.

Proof of Parseval. With $\alpha_k = \langle \boldsymbol{x}, \boldsymbol{v}_k \rangle$ and $\beta_k = \langle \boldsymbol{y}, \boldsymbol{v}_k \rangle$, we can write

$$\boldsymbol{x} = \sum_{k\in\Gamma} \alpha_k\, \boldsymbol{v}_k, \quad \text{and} \quad \boldsymbol{y} = \sum_{k\in\Gamma} \beta_k\, \boldsymbol{v}_k,$$

and so

$$\begin{aligned}
\langle \boldsymbol{x}, \boldsymbol{y} \rangle_S &= \left\langle \sum_{k\in\Gamma} \alpha_k \boldsymbol{v}_k, \sum_{\ell\in\Gamma} \beta_\ell \boldsymbol{v}_\ell \right\rangle_S \\
&= \sum_{k\in\Gamma} \alpha_k \left\langle \boldsymbol{v}_k, \sum_{\ell\in\Gamma} \beta_\ell \boldsymbol{v}_\ell \right\rangle_S \\
&= \sum_{k\in\Gamma} \sum_{\ell\in\Gamma} \alpha_k \overline{\beta_\ell} \langle \boldsymbol{v}_k, \boldsymbol{v}_\ell \rangle_S.
\end{aligned}$$

For a fixed value of k, only one term in the inner sum above will be nonzero, as $\langle \boldsymbol{v}_k, \boldsymbol{v}_\ell \rangle = 0$ unless $\ell = k$. Thus

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_S = \sum_{k\in\Gamma} \alpha_k \overline{\beta_k}.$$

A straightforward consequence of the result above is that distances in $\mathcal{S}$ under the induced norm are equivalent to Euclidean (ℓ_2) distances in coefficient space:

$$\|\boldsymbol{x}-\boldsymbol{y}\|_S = \|\boldsymbol{\alpha}-\boldsymbol{\beta}\|_2 = \left(\sum_{k\in\Gamma}(\alpha_k-\beta_k)^2\right)^{1/2}.$$

Thus changing the value of an orthobasis expansion coefficient by an amount ϵ will change the signal by an amount (as measured in $\|\cdot\|_S$) ϵ.

To be more precise about this, suppose $\boldsymbol{x}$ has transform coefficients $\{\alpha_k = \langle \boldsymbol{x}, \boldsymbol{v}_k\rangle_S\}$. If I perturb one of them, say at location k_0, by setting

$$\tilde{\alpha}_k = \begin{cases}\alpha_{k_0}+\epsilon & k=k_0\\ \alpha_k & k\neq k_0\end{cases},$$

and then synthesizing

$$\tilde{\boldsymbol{x}} = \sum_{k\in\Gamma}\tilde{\alpha}_k\boldsymbol{v}_k,$$

we will have

$$\|\boldsymbol{x}-\tilde{\boldsymbol{x}}\|_S = \epsilon.$$

Notice that while the error is localized to one expansion coefficient, it could effect the entire reconstruction, but its net effect will still be ϵ.

More generally, suppose we add an error to every coefficient:

$$\tilde{\alpha}_k = \alpha_k + \epsilon_k,$$

then re-synthesize the vector

$$\tilde{\boldsymbol{x}} = \sum_{k\in\Gamma} \tilde{\alpha}_k \boldsymbol{v}_k.$$

The total error (as measured by the norm $\|\cdots\|_{\mathcal{S}}$) in the reconstructed vector will be the same as the total error (as measured by the Euclidean norm $\|\cdot\|_2$):

$$\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_{\mathcal{S}} = \|\boldsymbol{\alpha} - \tilde{\boldsymbol{\alpha}}\|_2 = \left(\sum_{k\in\Gamma} |\epsilon_k|^2\right)^{1/2}.$$

The upshot of this is that as we manipulate the expansion coefficients $\{\alpha_k\}$, we know what the net effect will be in the reconstructed function.

Truncating ortho expansions and linear approximation

Say $\{\boldsymbol{v}_k\}_{k=0}^{\infty}$ in and orthobasis for a Hilbert space $\mathcal{S}$. Let $\mathcal{T}$ be the subspace spanned by the first 10 elements of $\{\boldsymbol{v}_k\}$:

$$\mathcal{T} = \operatorname{span}\left(\{\boldsymbol{v}_0, \ldots, \boldsymbol{v}_9\}\right).$$

1. Given $\boldsymbol{x} \in \mathcal{S}$, what is the closest point in $\mathcal{T}$ (call it $\hat{\boldsymbol{x}}$) to $\boldsymbol{x}$? We have seen that it is

$$\hat{\boldsymbol{x}} = \sum_{k=0}^{9} \langle \boldsymbol{x}, \boldsymbol{v}_k \rangle \boldsymbol{v}_k.$$

2. How good an approximation is $\hat{\boldsymbol{x}}$ to $\boldsymbol{x}$? If we measure this in the induced norm $\|\cdot\|_S$, then

$$\begin{aligned}\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_S^2 &= \left\| \sum_{k=0}^{\infty} \langle \boldsymbol{x}, \boldsymbol{v}_k \rangle \boldsymbol{v}_k - \sum_{k=0}^{9} \langle \boldsymbol{x}, \boldsymbol{v}_k \rangle \boldsymbol{v}_k \right\|_S^2 \\ &= \left\| \sum_{k=10}^{\infty} \langle \boldsymbol{x}, \boldsymbol{v}_k \rangle \boldsymbol{v}_k \right\|_S^2 \\ &= \sum_{k=10}^{\infty} |\langle \boldsymbol{x}, \boldsymbol{v}_k \rangle|^2.\end{aligned}$$

Since also

$$\|\boldsymbol{x}\|_S^2 = \sum_{k=0}^{\infty} |\langle \boldsymbol{x}, \boldsymbol{v}_k \rangle|^2$$

the approximation error for $\hat{x}$ will be small if the first 10 transform coefficients

$$\langle \boldsymbol{x}, \boldsymbol{v}_0 \rangle,\ \langle \boldsymbol{x}, \boldsymbol{v}_1 \rangle,\ \ldots,\ \langle \boldsymbol{x}, \boldsymbol{v}_9 \rangle,$$

contain "most" of the total energy.

Of course, there is nothing special about taking the first 10 coefficients. We can just as easily form a K term approximation using

$$\hat{\boldsymbol{x}}_K = \sum_{k=0}^{K-1} \langle \boldsymbol{x}, \boldsymbol{v}_k \rangle \boldsymbol{v}_k$$

which has error

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}_K\|_S^2 = \sum_{k=K}^{\infty} |\langle \boldsymbol{x}, \boldsymbol{v}_k \rangle|^2.$$

If the sum above is small for moderately large K, we can "compress" $\boldsymbol{x}$ by using just the first K terms in the expansion.

This is precisely what is done in image and video compression — more details on this in a couple of lectures.

Example:
Any real-valued function on $[-1/2, 1/2]$ with even symmetry can be built up out of harmonic cosines:

$$x(t) = \alpha_0 + \sum_{k=1}^{\infty} \alpha_k \sqrt{2} \cos(2\pi k t).$$

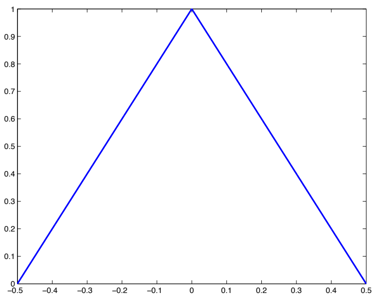
(That this is true follows directly from the observation that every signal on $[-1/2, 1/2]$ that is real-valued and even has a Fourier series which is real-valued and even.) This is an orthobasis expansion in the standard inner product with

$$v_0(t) = 1,\ v_1(t) = \sqrt{2}\cos(2\pi t),\ \ldots,\ v_k(t) = \sqrt{2}\cos(2\pi k t),\ \ldots$$

It is easy to check that $\langle \boldsymbol{v}_k, \boldsymbol{v}_\ell \rangle = 0,\ k \neq \ell$ and $\langle \boldsymbol{v}_k, \boldsymbol{v}_k \rangle = 1$.

For the triangle function below

$$x(t) = \begin{cases} 1+2t, & -1/2 \le t \le 0 \\ 1-2t, & 0 \le t \le 1/2 \end{cases}$$



the expansion coefficients are

$$\begin{aligned} \alpha_0 &= 1/2, \\ \alpha_k &= \int_{-1/2}^{1/2} x(t)\sqrt{2}\cos(2\pi k t)\,\mathrm{d}t \\ &= 2\sqrt{2}\int_0^{1/2}(1-2t)\cos(2\pi k t)\,\mathrm{d}t \\ &= \begin{cases} 0 & k \text{ even, } k \neq 0 \\ \frac{2\sqrt{2}}{\pi^2 k^2} & k \text{ odd} \end{cases}. \end{aligned}$$

First, let's compute the norm in time and coefficient space just to make sure they agree:

$$\|\boldsymbol{x}\|_2^2 = \int_{-1/2}^{1/2}|x(t)|^2\,\mathrm{d}t = 2\int_0^{1/2}(1-2t)^2\,\mathrm{d}t = 1/3,$$

and

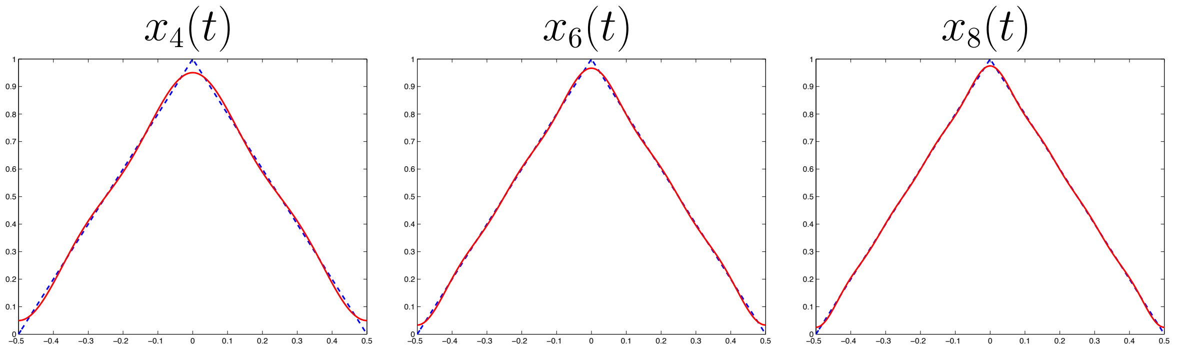
$$\begin{aligned} \sum_{k=0}^{\infty}|\alpha_k|^2 &= \frac{1}{4} + \frac{8}{\pi^4}\sum_{k'=0}^{\infty}\frac{1}{(1+2k')^4} \\ &= \frac{1}{4} + \frac{8}{\pi^4}\left(\frac{\pi^4}{96}\right) \\ &= \frac{1}{3}. \end{aligned}$$

When we truncate the expansion at K terms,

$$x_K(t) = \frac{1}{2} + \sum_{k=1}^{K-1} \alpha_k \sqrt{2}\cos(2\pi kt),$$

we can interpret the result as an **approximation** of $x(t)$ that is a member of the K-dimensional subspace span$(\{\sqrt{2}\cos(2\pi kt\}_{k=0}^{K-1})$, and we know that it is the best approximation in that subspace.

Here are the approximation for $K = 4, 6, 8$:



We can compute the error in each of these approximations explicitly, as

$$\begin{aligned} x(t) - x_K(t) &= \sum_{k=0}^{\infty} \alpha_k\sqrt{2}\cos(2\pi kt) - \sum_{k=0}^{K-1} \alpha_k\sqrt{2}\cos(2\pi kt) \\ &= \sum_{k=K}^{\infty} \alpha_k\sqrt{2}\cos(2\pi kt), \end{aligned}$$

and so

$$\|x(t) - x_K(t)\|_2^2 = \sum_{k=K}^{\infty} |\alpha_k|^2,$$

or, since $x_K(t) \perp x(t) - x_K(t)$,

$$\|x(t) - x_K(t)\|_2^2 = \|x(t)\|_2^2 - \|x_K(t)\|_2^2.$$

In the three examples above, we have

$$\|x(t) - x_4(t)\|_2^2 \approx 1.92 \cdot 10^{-4}, \quad \|x(t) - x_6(t)\|_2^2 \approx 6.01 \cdot 10^{-5},$$
$$\|x(t) - x_8(t)\|_2^2 \approx 2.59 \cdot 10^{-5}.$$

The DCT and JPEG

A great example of how manipulating orthobasis expansion coefficients can lead to something useful is given by the JPEG image compression standard. This is a complicated compression standard, but it is based on a simple idea: break the image into pieces, represent each piece using a cosine basis, then achieve compression by truncating/quantizing the expansion.

Lets start by recalling the discrete Fourier transform for vectors in $\mathbb{C}^N$. This is basically the same as Fourier series, except that the sinusoidal basis vectors are discrete (instead of functions of a continuous variable). Any $\boldsymbol{x} \in \mathbb{C}^N$ can be written as

$$\boldsymbol{x} = \sum_{k=0}^{N-1} \alpha_k \boldsymbol{\psi}_k, \quad \psi_k[n] = \frac{1}{\sqrt{N}} e^{j2\pi nk/N}, \quad n = 0, \ldots, N-1.$$

Computing $\boldsymbol{\alpha}$ for a given $\boldsymbol{x}$ can be done in $O(N \log_2 N)$ time using the *fast Fourier transform* (FFT), one of the most important algorithms in engineering and applied mathematics.

Closely related is the *discrete cosine transform* (DCT). This is a basis for $\mathbb{R}^N$, and has slightly more favorable symmetry properties than the standard DFT. The DCT basis functions for $\mathbb{R}^N$ are

$$\psi_k[n] = \begin{cases} \sqrt{\frac{1}{N}} & k = 0 \\ \sqrt{\frac{2}{N}} \cos\left(\frac{\pi k}{N}(n + \frac{1}{2})\right) & k = 1, \ldots, N-1 \end{cases} \tag{1}$$

for sample indices $n = 0, 1, \ldots, N-1$. Showing that

$$\sum_{n=0}^{N-1} \psi_k[n]\psi_\ell[n] = \begin{cases} 1 & k = \ell \\ 0 & k \neq \ell \end{cases}$$

is an exercise you can do at home. Notice that the samples of the cosines are on the half-sample points (we see $(n+1/2)$ in the expression above instead of n).

The DCT can be quickly computed from the DFT of a symmetric extension of $\boldsymbol{x}$. That means we have a **fast algorithm** for computing the DCT — the cost is essentially the same as for an FFT, $O(N \log N)$.

An *image* is an array of pixels arranged on a grid indexed by n_1, n_2. We can think of $x[n_1, n_2]$ as signifying the intensity of an image at location (n_1, n_2). If we want more than black and white images, we can use three intensity arrays (three "channels") to represent the color at every point.

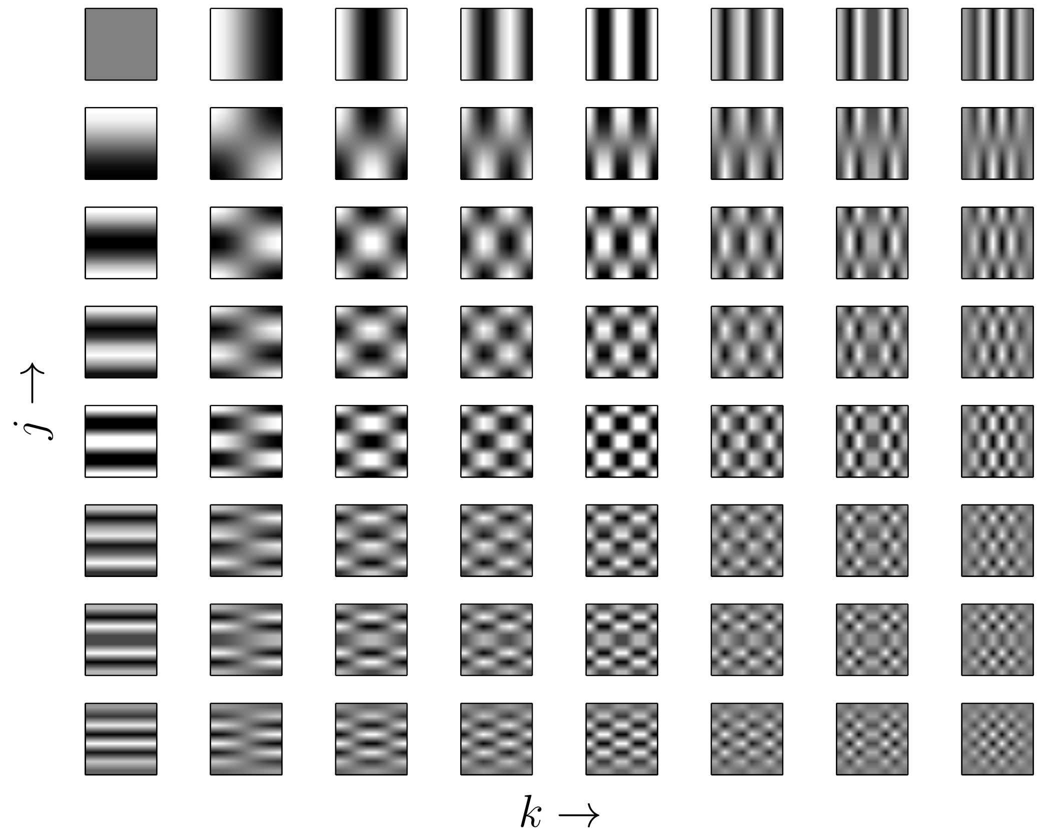
Here, we will assume that these arrays are square, $N \times N$, although the discussion below is easily adapted for rectangular arrays. We will still call $\{x[n_1, n_2],\ 0 \leq n_1, n_2 \leq N-1\}$ a vector, even though it is naturally indexed by two variables . (It should be clear anyway that arrays of numbers like this obey our requirements for an abstract vector space.) We will call this space of 2D arrays $\mathbb{R}^N \times \mathbb{R}^N$.

There is a straightforward way to create an orthobasis for $\mathbb{R}^N \times \mathbb{R}^N$ from an orthobasis for $\mathbb{R}^N$. We simply create *separable 2D arrays* from the 1D basis vectors. More precisely, if $\{\boldsymbol{\psi}_k,\ k = 0, \ldots, N-1\}$ is an orthobasis for $\mathbb{R}^N$, then $\{\boldsymbol{\psi}^{\text{2D}}_{j,k},\ 0 \leq j, k \leq N\}$ is an orthbobasis for $\mathbb{R}^N \times \mathbb{R}^N$, where

$$\psi^{\text{2D}}_{j,k}[n_1, n_2] = \psi_j[n_1]\psi_k[n_2].$$

You will prove this (and actually something much more general than this) on next week's homework.

Let's take a look at the 2D DCT orthobasis for 8×8 image patches. The 64 2D DCT basis functions for $N = 8$ are shown below:



$\psi_{j,k}[m, n]$ for $j, k = 0, \ldots, 7$

2D DCT coefficients are indexed by two integers, and so are naturally arranged on a grid as well:

$$\begin{matrix} \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,N-1} \\ \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,N-1} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{N-1,0} & \alpha_{N-1,1} & \cdots & \alpha_{N-1,N-1} \end{matrix}$$

The DCT in image and video compression

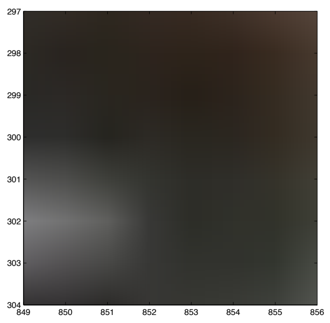
The DCT is basis of the popular JPEG image compression standard. The central idea is that while energy in a picture is distributed more or less evenly throughout, in the DCT transform domain it tends to be *concentrated* at low frequencies.

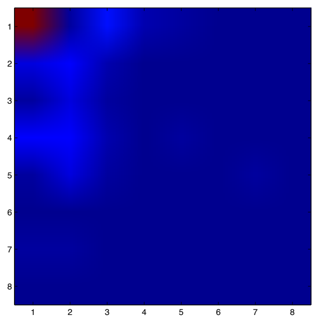
JPEG compression work roughly as follows:

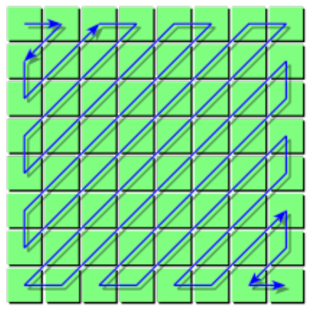
1. Divide the image into 8×8 blocks of pixels
2. Take a DCT within each block
3. Quantize the coefficients — the rough effect of this is to keep the larger coefficients and remove the samller ones
4. Bitstream (losslessly) encode the result.

There are some details we are leaving out here, probably the most important of which is how the three different color bands are dealt with, but the above outlines the essential ideas.

The basic idea is that while the energy within an 8×8 block of pixels tends to be more or less evenly distributed, the DCT concentrates this energy onto a relatively small number of transform coefficients. Moreover, the significant coefficients tend to be at the same place in the transform domain (low spatial frequencies).

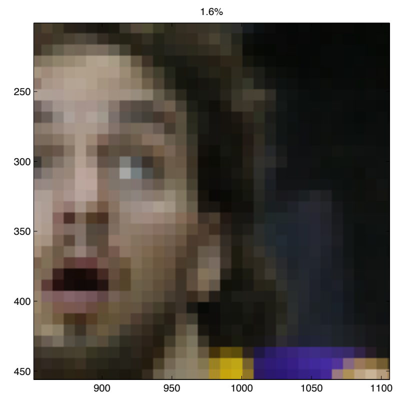

8×8 block


2D DCT coeffs

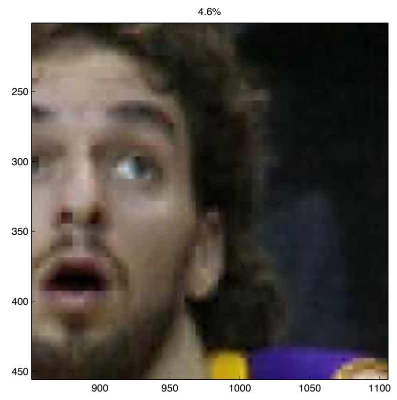

ordering

To get a rough feel for how closely this model matches reality, let's look at a simple example. Here we have an original image 2048×2048, and a zoom into a 256×256 piece of the image:

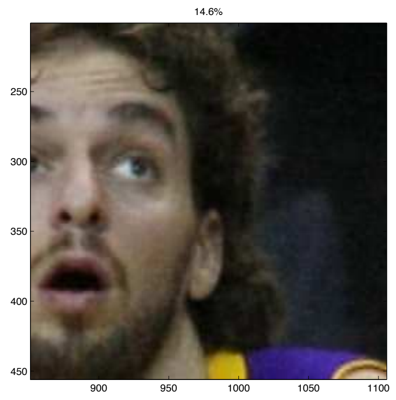
Here is the same piece after using 1 of the 64 coefficients per block ($1/64 \approx 1.6\%$), $3/64 \approx 4.6\%$ of the coefficients, and $10/64 \approx 15.62\%$:



1/64



3/64



10/64

So the "low frequency" heuristic appears to be a good one.

JPEG does not just "keep or kill" coefficients in this manner, it quantizes them using a fixed quantization mask. Here is a common example:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

The quantization simply maps $\alpha_{j,k} \rightarrow \tilde{\alpha}_{j,k}$ using

$$\tilde{\alpha}_{j,k} = Q_{j,k} \cdot \text{round}\left(\frac{\alpha_{j,k}}{Q_{j,k}}\right)$$

You can see that the coefficients at low frequencies (upper left) are being treated much more gently than those at higher frequencies (lower right).

The **decoder** simply reconstructs each 8×8 block $\boldsymbol{x}_b$ using the synthesis formula

$$\tilde{\boldsymbol{x}}_b[m,n] = \sum_{k=0}^{7}\sum_{\ell=0}^{7} \tilde{\alpha}_{k,\ell}\, \phi_{k,\ell}[m,n]$$

By the Parseval theorem, we know exactly what the effect of quantizing each coefficient is going to be on the total error, as

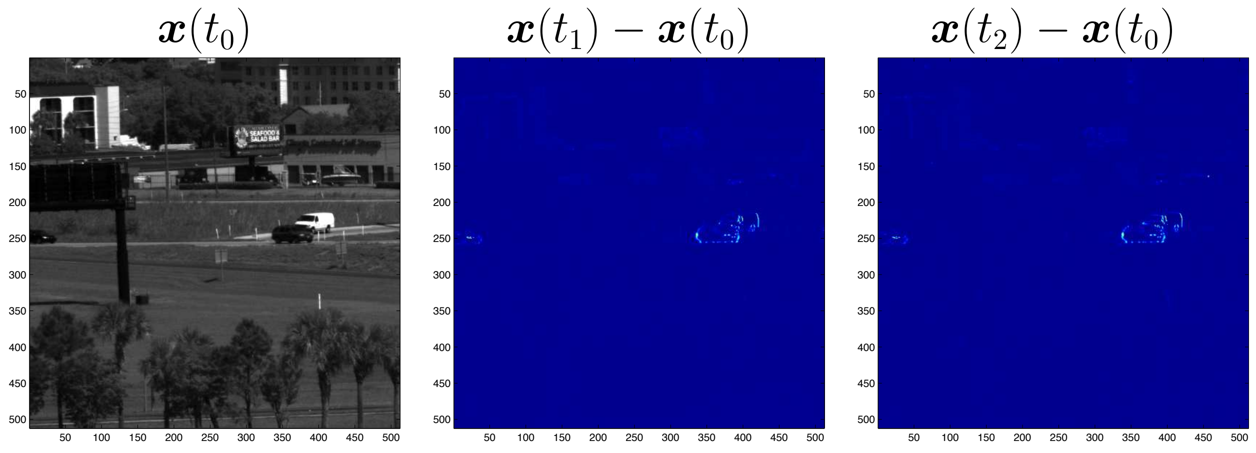
$$\|\boldsymbol{x}_b - \tilde{\boldsymbol{x}}_b\|_2^2 = \|\boldsymbol{\alpha} - \tilde{\boldsymbol{\alpha}}\|_2^2 = \sum_{k=0}^{7}\sum_{\ell=0}^{7} |\alpha_{k,\ell} - \tilde{\alpha}_{k,\ell}|^2.$$

Video compression

The DCT also plays a fundamental role in video compression (e.g. MPEG, H.264, etc.), but in a slightly different way. Video codecs are complicated, but here is essentially what they do:

1. Estimate, describe, and quantize the motion in between frames.
2. Use the motion estimate to "predict" the next frame.
3. Use the (block-based) DCT to code the residual.

Here is an example video frame, along with the differences between this frame and the next two frames (in false color):


$\boldsymbol{x}(t_0)$
$\boldsymbol{x}(t_1) - \boldsymbol{x}(t_0)$
$\boldsymbol{x}(t_2) - \boldsymbol{x}(t_0)$

The only activity is where the car is moving from left to right.