Picture Perfect: The Mathematics of JPEG Compression

Kurt Bryan

May 19, 2011

Images

- A typical color image might be 600 by 800 pixels.

Images

- A typical color image might be 600 by 800 pixels.
- Each pixel has a red (R), green (G) and blue (B) value associated to it.

Images

- A typical color image might be 600 by 800 pixels.
- Each pixel has a red (R), green (G) and blue (B) value associated to it.
- Each pixel needs 3 bytes of storage (one for each color).

Images

- A typical color image might be 600 by 800 pixels.
- Each pixel has a red (R), green (G) and blue (B) value associated to it.
- Each pixel needs 3 bytes of storage (one for each color).
- That's $600 \times 800 \times 3 = 1.44$ megabytes.

Images

- A typical color image might be 600 by 800 pixels.
- Each pixel has a red (R), green (G) and blue (B) value associated to it.
- Each pixel needs 3 bytes of storage (one for each color).
- That's $600 \times 800 \times 3 = 1.44$ megabytes.
- But a typical 600 by 800 JPEG image takes only about 120 kilobytes, less than $1/10$ the expected amount!

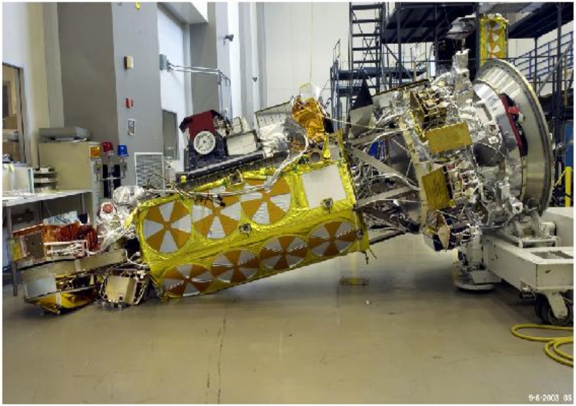
Images

- A typical color image might be 600 by 800 pixels.
- Each pixel has a red (R), green (G) and blue (B) value associated to it.
- Each pixel needs 3 bytes of storage (one for each color).
- That's $600 \times 800 \times 3 = 1.44$ megabytes.
- But a typical 600 by 800 JPEG image takes only about 120 kilobytes, less than $1/10$ the expected amount!

How can we eliminate 90 percent of the required storage?

Compression

An interesting image:



Bad Compression

Image compressed 10-fold:

The Basis for Compression

JPEG compression (and a good chunk of applied mathematics) is based on one of the great ideas of 19th century mathematics:

> *Functions can be decomposed into sums of sines and cosines of various frequencies.*

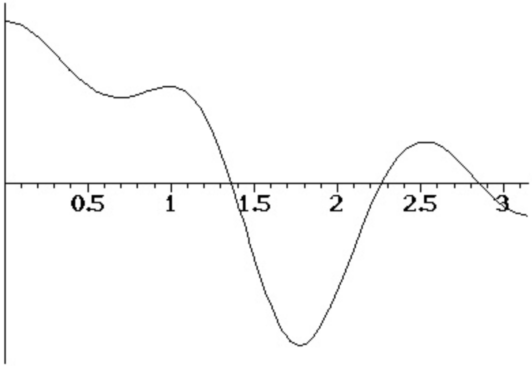
Fourier Cosine Series

Fourier series come in many flavors. We'll be interested in *Fourier Cosine Series*: Any reasonable function $f(t)$ defined on $[0, \pi]$ can be well-approximated as a sum of cosines,

$$\begin{aligned} f(t) \approx\ & a_0 \\ & + a_1 \cos(t) \\ & + a_2 \cos(2t) \\ & + \cdots \\ & + a_N \cos(Nt) \end{aligned}$$

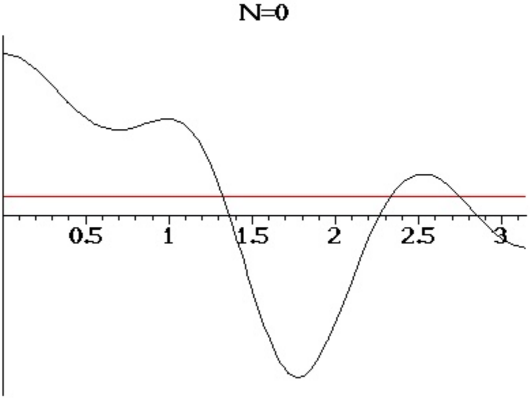
if we pick the a_k correctly (and take N large enough).

A Function to Approximate



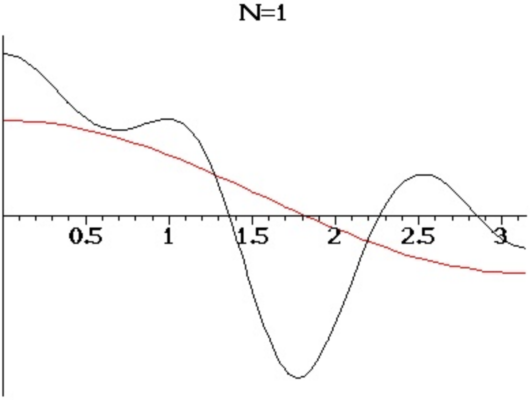
Cosine Series Example

$f(t) \approx 4.70$



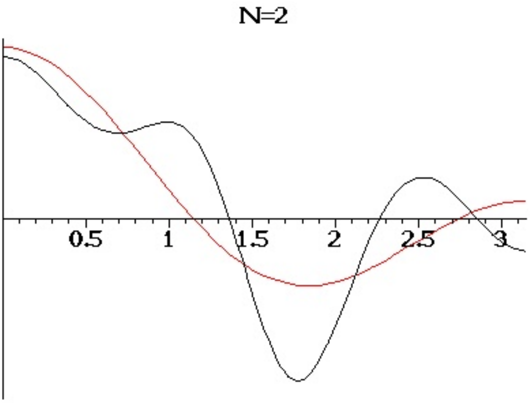
Cosine Series Example

$$f(t) \approx 4.70 + 19.1\cos(t)$$



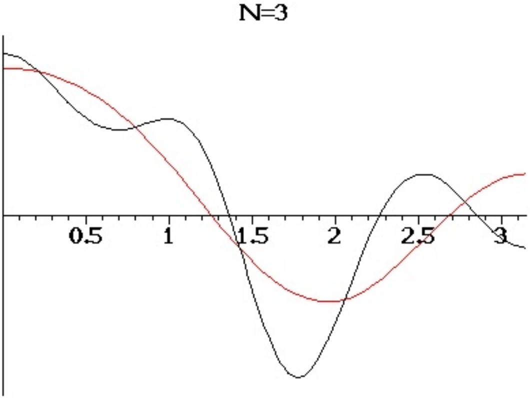
Cosine Series Example

$$f(t) \approx 4.70 + 19.1\cos(t) + 19.0\cos(2t)$$



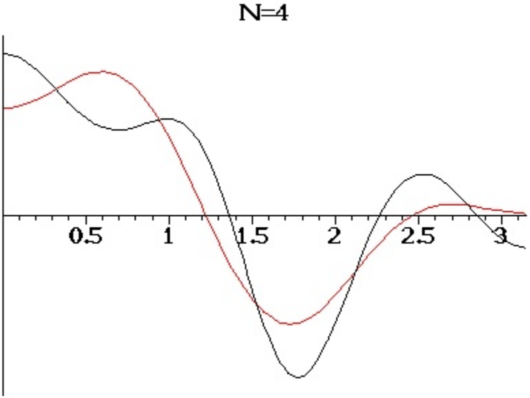
The Cosine Series

$$f(t) \approx 5.97 + 19.1\cos(t) + 19.0\cos(2t) - 5.88\cos(3t)$$



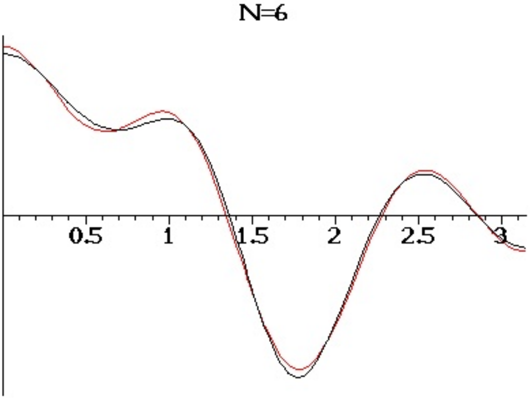
The Cosine Series

$$f(t) \approx 5.97 + 19.1\cos(t) + 19.0\cos(2t) - 5.88\cos(3t) - 9.92\cos(4t)$$



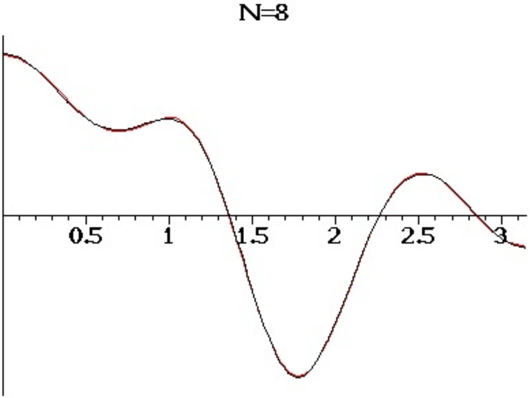
The Cosine Series

$$+\cdots+12.4\cos(5t)+2.97\cos(6t)$$



The Cosine Series

$$+\cdots - 1.70\cos(7t) - 0.53\cos(8t)$$



The Cosine Coefficients

A piecewise continuously-differentiable function $f(t)$ defined on $[0, \pi]$ can be approximated with a Fourier sum

$$f(t) \approx \frac{a_0}{2} + a_1 \cos(t) + a_2 \cos(2t) + \cdots + a_N \cos(Nt)$$

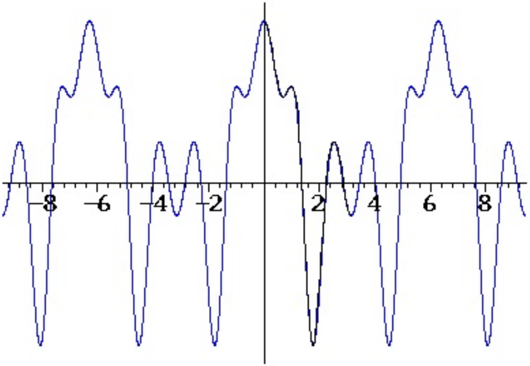
where

$$a_k = \frac{2}{\pi} \int_0^\pi f(t) \cos(kt)\, dt$$

As $N \to \infty$ the sum converges to $f(t)$ for any t at which f is continuous.

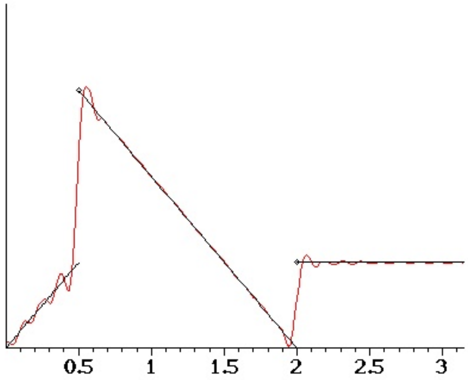
Periodicity

Outside the interval $[0, \pi]$ the cosine series extends f as an even period 2π function to the real line:



Discontinuities

Discontinuities are difficult to represent, for they take a lot of cosine terms to synthesize accurately. Here $N = 50$:



"Compressing" a Function

So a function $f(t)$ on $[0, \pi]$ is determined by its Fourier cosine coefficients $a_0, a_1, a_2 \ldots$, an infinite amount of information.

We could compress this information by

"Compressing" a Function

So a function $f(t)$ on $[0, \pi]$ is determined by its Fourier cosine coefficients $a_0, a_1, a_2 \ldots$, an infinite amount of information.

We could compress this information by

- Throwing out all "small" a_k, e.g., $|a_k| < \delta$ for some δ, and

"Compressing" a Function

So a function $f(t)$ on $[0, \pi]$ is determined by its Fourier cosine coefficients $a_0, a_1, a_2 \ldots$, an infinite amount of information.

We could compress this information by

- Throwing out all "small" a_k, e.g., $|a_k| < \delta$ for some δ, and
- "Quantizing" the remaining a_k by, for example, rounding them to the nearest integer, or more generally,

“Compressing” a Function

So a function $f(t)$ on $[0, \pi]$ is determined by its Fourier cosine coefficients $a_0, a_1, a_2 \ldots$, an infinite amount of information.

We could compress this information by

- Throwing out all “small” a_k, e.g., $|a_k| < \delta$ for some δ, and
- “Quantizing” the remaining a_k by, for example, rounding them to the nearest integer, or more generally,
- Rounding them to the nearest multiple of “r” for some fixed r.

Function Compression Example

The function $f(t)$ above has as its first 12 Fourier cosine coefficients

$$9.403, 19.09, 18.97, -5.883, -9.919, 12.38, 2.967, -1.705, -0.5301$$
$$0.4059, 0.04522, -0.05778$$

Function Compression Example

The function $f(t)$ above has as its first 12 Fourier cosine coefficients

$$9.403, 19.09, 18.97, -5.883, -9.919, 12.38, 2.967, -1.705, -0.5301$$
$$0.4059, 0.04522, -0.05778$$

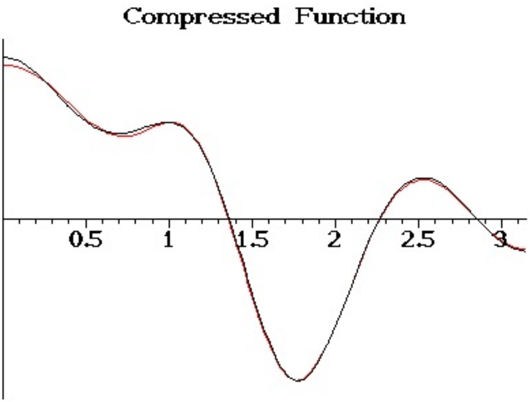
If we round each to the nearest integer we obtain

$$9, 19, 19, -6, -10, 12, 3, -2, -1, 0, 0, 0$$

We can reconstruct f from these approximate cosine coefficients.

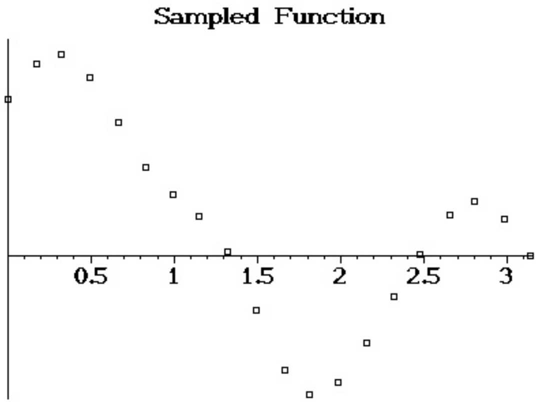
Function Compression Example

The original function (black) and compressed/reconstructed version (red)



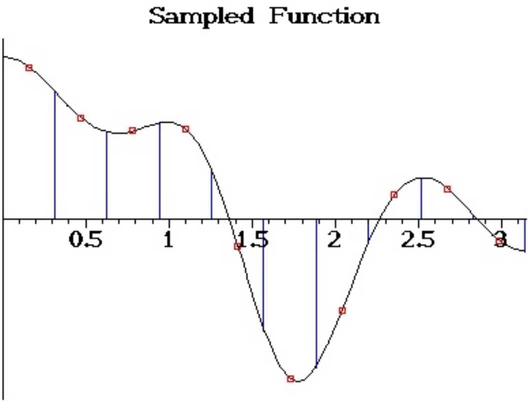
Sampling and Discretization

Signals and images aren't presented as functions, but as sampled function values:



Sampling and Discretization

We might break $[0, \pi]$ into N subintervals, sample f at each midpoint $t_0, t_1, \ldots, t_{N-1}$:



Sampling and Discretization

We replace $f(t)$ with the N-vector $\mathbf{f} = (f(t_0), f(t_1), \ldots, f(t_{N-1}))$.

Each basis function $\cos(kt)$ also gets replaced with a vector

$$\mathbf{v}_k = \sqrt{\frac{2}{N}}(\cos(kt_0), \cos(kt_1), \ldots, \cos(kt_{N-1}))$$

except

$$\mathbf{v}_0 = \sqrt{\frac{1}{N}}(1, 1, \ldots, 1)$$

(The factor in front makes the vectors have length one.)

Symmetries and Discretization

The vectors $\mathbf{v}_0, \ldots, \mathbf{v}_{N-1}$ are orthonormal, and so form a basis for $\mathbb{R}^N$! We can thus write any signal vector $\mathbf{f}$ as

$$\mathbf{f} = c_0\mathbf{v}_0 + c_1\mathbf{v}_1 + \cdots + c_{N-1}\mathbf{v}_{N-1}$$

for certain constants c_k, analogous to the cosine series

$$f(t) = \frac{a_0}{2} + a_1 \cos(t) + a_2 \cos(2t) + \cdots$$

How do we compute the c_k?

Sampling and Discretization

Dot each side of

$$\mathbf{f} = c_0\mathbf{v}_0 + c_1\mathbf{v}_1 + \cdots + c_{N-1}\mathbf{v}_{N-1}$$

with $\mathbf{v}_k$ and use $\mathbf{v}_j \cdot \mathbf{v}_k = 0$ for $j \neq k$ (while $\mathbf{v}_k \cdot \mathbf{v}_k = 1$) to find

$$c_k = \mathbf{f} \cdot \mathbf{v}_k.$$

Sampling and Discretization

The formula

$$c_k = \mathbf{f} \cdot \mathbf{v}_k$$

is analogous to

$$a_k = \frac{2}{\pi} \int_0^{\pi} f(t) \cos(kt)\, dt.$$

In fact, the former is just the midpoint rule for evaluating the latter integral!

The Discrete Cosine Transform

Then any signal vector $\mathbf{f} = c_0\mathbf{v}_0 + c_1\mathbf{v}_1 + \cdots + c_{N-1}\mathbf{v}_{N-1}$ where

$$c_k = \mathbf{f} \cdot \mathbf{v}_k.$$

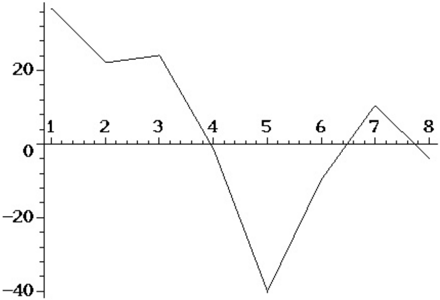
The map $\mathbf{f} \rightarrow \mathbf{c} = (c_0, \ldots, c_{N-1})$ is the Discrete Cosine Transform (DCT).

DCT Example

Consider a signal vector

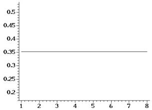
$$\mathbf{f} = < 36.0, 22.3, 24.2, -1.55, -40.4, -9.90, 10.3, -3.99 >$$

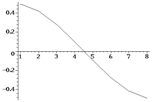
in $\mathbb{R}^8$:



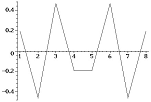
DCT Example

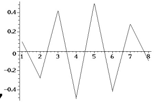
We can decompose $\mathbf{f} = c_0\mathbf{v}_0 + c_1\mathbf{v}_1 + \cdots + c_6\mathbf{v}_6 + c_7\mathbf{v}_7$,

$\mathbf{f} =$ 19.0 

$+$ 42.5 

$\vdots$

$+$ 11.7 

$+$ 4.7 

Here $\mathbf{c} = < 19.0, 42.5, 31.7, -14.0, -14.9, 23.0, 11.7, 4.70 >$.

Matrix Formulation of the DCT

The DCT maps a vector $\mathbf{f} = (f_0, \ldots, f_{N-1})$ to a vector $\mathbf{c} = (c_0, \ldots, c_{N-1})$ as $c_k = \mathbf{v}_k \cdot \mathbf{f}$, or in matrix terms,

$$\mathbf{c} = \mathbf{C}_N \mathbf{f}$$

where $\mathbf{C}_N$ is the $N \times N$ matrix with the $\mathbf{v}_k$ as rows. The inverse DCT is just

$$\mathbf{f} = \mathbf{C}_N^T \mathbf{c}$$

since $\mathbf{C}_N$ turns out to be orthogonal.

Discrete Compression

To compress a discretized signal (vector) $\mathbf{f} \in \mathbb{R}^N$:

Discrete Compression

To compress a discretized signal (vector) $\mathbf{f} \in \mathbb{R}^N$:

1. Perform a DCT $\mathbf{c} = \mathbf{C}_N \mathbf{f}$.

Discrete Compression

To compress a discretized signal (vector) $\mathbf{f} \in \mathbb{R}^N$:

1. Perform a DCT $\mathbf{c} = \mathbf{C}_N \mathbf{f}$.
2. Zero out all c_k below a certain threshold, quantize the rest (round to nearest integer, near tenth, etc.) This is where we get compression.

Discrete Compression

To compress a discretized signal (vector) $\mathbf{f} \in \mathbb{R}^N$:

1. Perform a DCT $\mathbf{c} = \mathbf{C}_N \mathbf{f}$.
2. Zero out all c_k below a certain threshold, quantize the rest (round to nearest integer, near tenth, etc.) This is where we get compression.
3. The signal can be (approximately) reconstituted using the thresholded quantized c_k and an inverse DCT.

Audio Compression Example

- The file "gong1.wav" consists of 50,000 sampled audio values, so $\mathbf{f} \in \mathbb{R}^{50000}$.

Audio Compression Example

- The file "gong1.wav" consists of 50,000 sampled audio values, so $\mathbf{f} \in \mathbb{R}^{50000}$.
- With Matlab, we can perform a DCT, round each coefficient to the nearest integer (or multiple of r). This is where compression happens (most c_k will be zero).

Audio Compression Example

- The file "gong1.wav" consists of 50,000 sampled audio values, so $\mathbf{f} \in \mathbb{R}^{50000}$.
- With Matlab, we can perform a DCT, round each coefficient to the nearest integer (or multiple of r). This is where compression happens (most c_k will be zero).
- We can approximately reconstitute the original audio signal with an inverse DCT.

Blocking

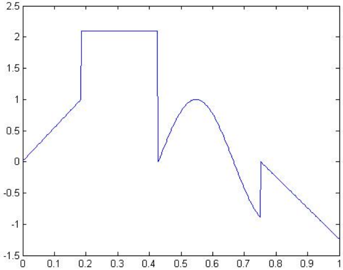
1. Discontinuities cause many DCT coefficients to be large—they don't decay to zero very fast—and so the file is hard to compress.

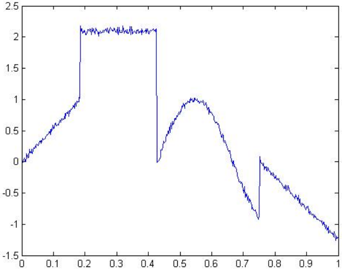
Blocking

1. Discontinuities cause many DCT coefficients to be large—they don't decay to zero very fast—and so the file is hard to compress.
2. It can help to break the signal into blocks and compress each individually, to limit the effect of any discontinuity to one block.

Compression Example 2

A signal with discontinuities, and its compressed version (eight-bit)

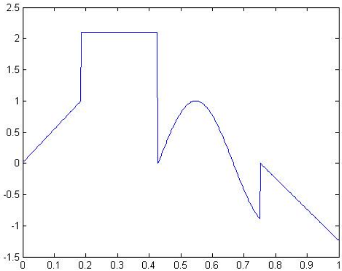


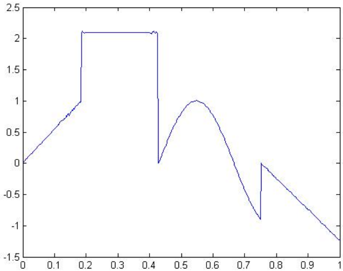


62 percent of the DCT coefficients remain non-zero.

Compression Example 2

A signal with discontinuities, compressed in blocks of 50 samples:





38 percent of the DCT coefficients remain non-zero.

Grayscale Images

1. A grayscale image on a region $(x, y) \in [0, A] \times [0, B]$ is modeled by a function $f(x, y)$, with (for example) $f = 0$ as black, $f = 255$ as white.

Grayscale Images

1. A grayscale image on a region $(x, y) \in [0, A] \times [0, B]$ is modeled by a function $f(x, y)$, with (for example) $f = 0$ as black, $f = 255$ as white.
2. A color image might be modeled by three functions, $R(x, y), G(x, y)$, and $B(x, y)$, assigning a red, green, and blue value to each point.

Grayscale Images

1. A grayscale image on a region $(x, y) \in [0, A] \times [0, B]$ is modeled by a function $f(x, y)$, with (for example) $f = 0$ as black, $f = 255$ as white.
2. A color image might be modeled by three functions, $R(x, y), G(x, y)$, and $B(x, y)$, assigning a red, green, and blue value to each point.
3. We'll only be concerned with grayscale images.

Fourier Series in 2D

A function $f(x,y)$ defined on $(x,y) \in [0,\pi] \times [0,\pi]$ can be expanded into a Fourier cosine series

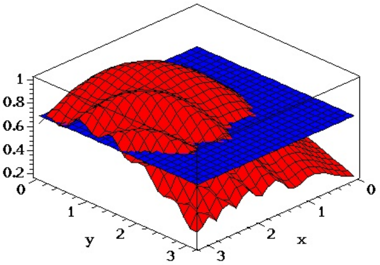
$$\begin{aligned} f(x,y) &= \frac{a_{0,0}}{4} + \frac{1}{2}\sum_{j=1}^{\infty}(a_{j,0}\cos(jx) + a_{0,j}\cos(jy)) \\ &+ \sum_{j=1}^{\infty}\sum_{k=1}^{\infty} a_{jk}\cos(jx)\cos(ky) \end{aligned}$$

where

$$a_{jk} = \frac{4}{\pi^2}\int_0^{\pi}\int_0^{\pi} f(x,y)\cos(jx)\cos(ky)\,dx\,dy.$$

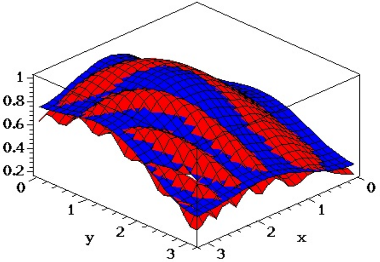
2D Example

$f(x,y) = e^{-((x-2)^2+(y-1)^2)/5} + \sin(x^2 y)/20$, $a_{0,0}$ term only:



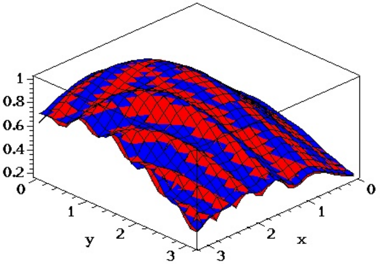
2D Example

All terms up to $j = 2, k = 2$:



2D Example

All terms up to $j = 10, k = 10$:



Sampling in 2D

A grayscale image on a square $0 \leq x, y \leq \pi$ can be considered as a function $f(x, y)$.

The sampled version is the $M \times N$ matrix $\mathbf{q}$ with entries

$$q_{jk} = f(x_k, y_j),$$

sampled on a rectangular grid. Each sample value q_{jk} is the gray value of the corresponding pixel.

The 2D DCT

The 2D DCT

- Replace image function f with sampled version $\mathbf{q}$, an $M \times N$ matrix.

The 2D DCT

- Replace image function f with sampled version $\mathbf{q}$, an $M \times N$ matrix.
- Replace $\cos(jx)\cos(ky)$ basis function with sampled version, $M \times N$ basis matrix $\mathbf{E}_{jk}$, $0 \leq j \leq N-1, 0 \leq k \leq M-1$.

The 2D DCT

- Replace image function f with sampled version $\mathbf{q}$, an $M \times N$ matrix.
- Replace $\cos(jx)\cos(ky)$ basis function with sampled version, $M \times N$ basis matrix $\mathbf{E}_{jk}$, $0 \leq j \leq N-1, 0 \leq k \leq M-1$.
- We can write

$$\mathbf{q} = \sum_{j=0}^{M} \sum_{k=0}^{N} \tilde{q}_{jk} \mathbf{E}_{jk}$$

for certain $\tilde{q}_{jk}$, components of an $M \times N$ matrix $\tilde{\mathbf{q}}$.

The 2D DCT

- Replace image function f with sampled version $\mathbf{q}$, an $M \times N$ matrix.
- Replace $\cos(jx)\cos(ky)$ basis function with sampled version, $M \times N$ basis matrix $\mathbf{E}_{jk}$, $0 \leq j \leq N-1, 0 \leq k \leq M-1$.
- We can write

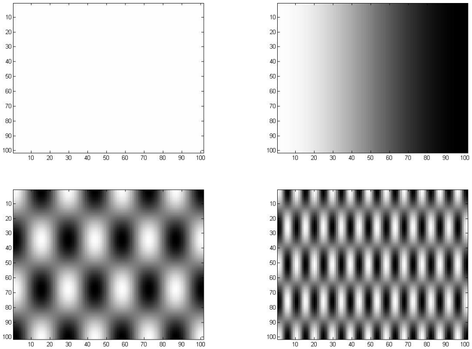
$$\mathbf{q} = \sum_{j=0}^{M} \sum_{k=0}^{N} \tilde{q}_{jk} \mathbf{E}_{jk}$$

for certain $\tilde{q}_{jk}$, components of an $M \times N$ matrix $\tilde{\mathbf{q}}$.
- It turns out that

$$\tilde{\mathbf{q}} = \mathbf{C}_M \mathbf{q} \mathbf{C}_N^T.$$

The 2D DCT

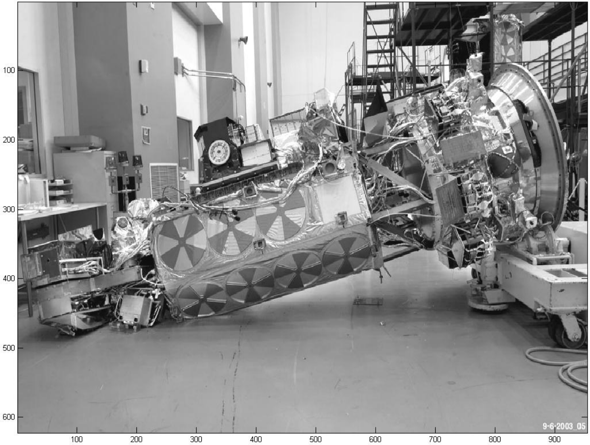
As grayscale images, the matrices $\mathbf{E}_{jk}$ look like (cases $(j, k) = (0, 0), (0, 1), (3, 7), (4, 21)$):



The 2D DCT expresses any image as a "sum" of these basic images.

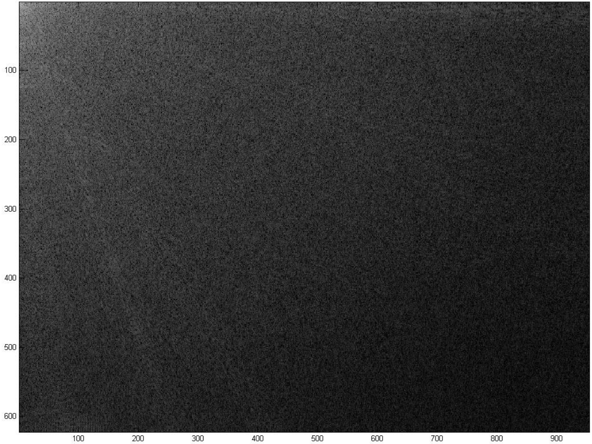
2D Compression Example

A grayscale image to compress:



2D Compression Example

The DCT, displayed as a grayscale image (log rescaling):



2D Compression Example

The strategy:

1. Perform a DCT on the image, to produce $M \times N$ array $\tilde{\mathbf{q}}$ of DCT coefficients.

2D Compression Example

The strategy:

1. Perform a DCT on the image, to produce $M \times N$ array $\tilde{\mathbf{q}}$ of DCT coefficients.
2. Quantize the array by rounding each $\tilde{q}_{jk}$, zeroing out small $\tilde{q}_{jk}$ (this is where compression occurs).

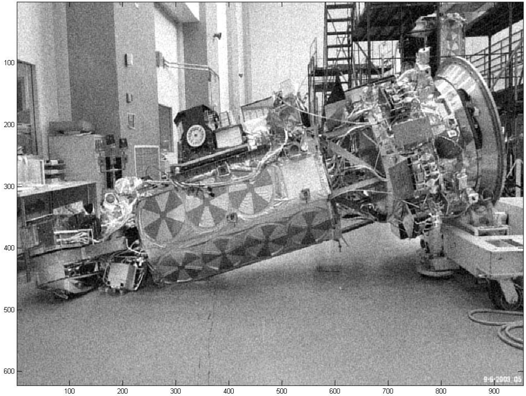
2D Compression Example

The strategy:

1. Perform a DCT on the image, to produce $M \times N$ array $\tilde{\mathbf{q}}$ of DCT coefficients.
2. Quantize the array by rounding each $\tilde{q}_{jk}$, zeroing out small $\tilde{q}_{jk}$ (this is where compression occurs).
3. The image can be reconstructed by using the quantized array and an inverse DCT (2D).

2D Compression Example

Round DCT coefficients to near multiple of 100 (yields 90 percent compression):



Block Compression

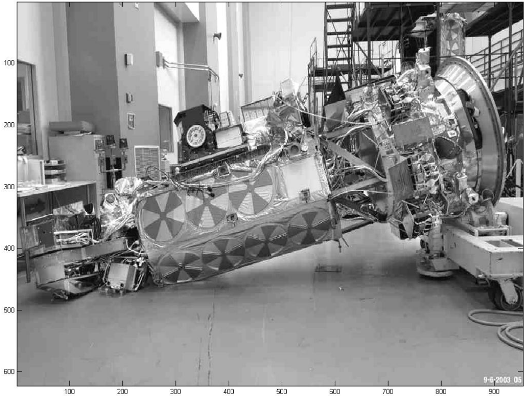
- As in 1D, discontinuities (edges) cause trouble. It can help to divide the image into blocks and compress each individually.

Block Compression

- As in 1D, discontinuities (edges) cause trouble. It can help to divide the image into blocks and compress each individually.
- The JPEG standard uses 8×8 pixel blocks.

2D JPEG Example

The satellite image compressed in 8×8 blocks, 90 percent DCT coefficients zeroed out:



Final Remarks

- The ideas of Fourier analysis appears in many other areas of image processing, e.g., noise removal, edge detection.

Final Remarks

- The ideas of Fourier analysis appears in many other areas of image processing, e.g., noise removal, edge detection.
- The new JPEG 2000 standard replaces the cosine basis functions with "wavelet" basis functions.