Numerical Analysis and Computing

Lecture Notes #15
— Approximation Theory —
The Fast Fourier Transform, with Applications

Joe Mahaffy,
⟨mahaffy@math.sdsu.edu⟩

Department of Mathematics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://www-rohan.sdsu.edu/∼jmahaffy

Spring 2010

Outline

Trigonometric Polynomials: Least Squares ⇒ Interpolation.

Last Time: We used trigonometric polynomials, *i.e.* linear combinations of the functions:

$$\begin{cases} \Phi_0(x) &= \dfrac{1}{2} \\ \Phi_k(x) &= \cos(kx), \quad k = 1, \dots, n \\ \Phi_{n+k}(x) &= \sin(kx), \quad k = 1, \dots, n-1 \end{cases}$$

to find **least squares** approximations (where $n < m$) to equally spaced data ($2m$ points) in the interval $[-\pi, \pi]$, at the node points $x_j = -\pi + (j\pi/m)$, $j = 0, 1, \dots, (2m-1)$.

This Time: We will find the **interpolatory** ($n = m$) trigonometric polynomials... and we will figure out how to do it **fast!**

Why use Interpolatory Trigonometric Polynomials?

Interpolation of large amounts of **equally spaced** data by trigonometric polynomials produces very good results (close to optimal, *c.f.* Chebyshev interpolation).

Some Applications

- Digital Filters (Lowpass, Bandpass, Highpass)
- Signal processing/analysis
- Antenna design and analysis
- Quantum mechanics
- Optics
- Spectral methods numerical solutions of equations.
- Image processing/analysis

Interpolatory Trigonometric Polynomials

Let x_i be $2m$ equally spaced node points in $[-\pi, \pi]$, and $f_i = f(x_i)$ the function values at these nodes. We can find a trigonometric polynomial of degree m: $P(x) \in \mathcal{T}_m$ which interpolates the data:

$$S_m(x) = \frac{a_0}{2} + \frac{a_m}{2}\cos(mx) + \sum_{k=1}^{m-1}\left[a_k\cos(kx) + b_k\sin(kx)\right],$$

where

$$a_k = \frac{1}{m}\sum_{j=0}^{2m-1} f_j\cos(kx_j) \quad b_k = \frac{1}{m}\sum_{j=0}^{2m-1} f_j\sin(kx_j).$$

The only difference in this formula compared with the one corresponding to the least squares approximation, $S_n(x)$, $n < m$ is the division by two of the a_m coefficient.

Where does the factor of 2 come from???

Finding the Factor of 2: Breakdown of the Lemma

The factor of two comes from the failure of the second part of the lemma (which we showed last time):

Lemma

If the integer r is not a multiple of $2m$, then

$$\sum_{j=0}^{2m-1}\cos(rx_j) = \sum_{j=0}^{2m-1}\sin(rx_j) = 0.$$

Moreover, if r is **not** *a multiple of m, then*

$$\sum_{j=0}^{2m-1}[\cos(rx_j)]^2 = \sum_{j=0}^{2m-1}[\sin(rx_j)]^2 = m.$$

Now, since $n = m$, we end up with one instance (the $\cos(mx_j)$-sum) where $r = m$, and the lemma fails.

Finding the Factor of 2: Computing the Sum

Now, since we are interpolating, the basis function $\Phi_m(x) = \cos(mx)$ is part of the set. When we compute $\langle \Phi_m, \Phi_m \rangle$ we need

$$\begin{aligned}
\sum_{j=0}^{2m-1}[\cos(mx_j)]^2 &= \sum_{j=0}^{2m-1}[\cos(-\pi m + m\frac{j\pi}{m})]^2 \\
&= \sum_{j=0}^{2m-1}[\cos((j-m)\pi)]^2 \\
&= \sum_{j=0}^{2m-1}(-1)^{2(j-m)} \\
&= \sum_{j=0}^{2m-1} 1 = 2\,m.
\end{aligned}$$

Historical Perspective

Much of the analysis was done by **Jean Baptiste Joseph Fourier** in the early 1800s, but the use of the Fourier series representation was not practical until 1965.

Why? The straight-forward implementation requires about $\mathbf{4m^2}$ operations in order to compute the coefficients $\tilde{\mathbf{a}}$, and $\tilde{\mathbf{b}}$.

In 1965 **Cooley and Tukey** published a 4-page paper titled *"An algorithm for the machine calculation of complex Fourier series"* in the journal **Mathematics of Computation.** The paper describes an algorithm which computes the coefficients using only $\mathcal{O}(\mathbf{m}\log_2\mathbf{m})$ operations.

It is hard to overstate the importance of this paper!!!

The algorithm is now known as the **"Fast Fourier Transform"** or just the **"FFT"**.

Computing the FFT

Instead of generating the coefficients $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ separately, we define the complex coefficient $c_k = m(-1)^k(a_k + ib_k)$ and consider the Fourier transforms with complex coefficients

$$S_m(x) = \frac{1}{m}\sum_{k=0}^{2m-1} c_k e^{ikx}, \quad \text{where} \quad c_k = \sum_{j=0}^{2m-1} f_j e^{ik\pi j/m}$$

The reduction of the number of required operations come from the fact that for any $n \in \mathbb{Z}$,

$$e^{in\pi} = \cos(n\pi) + i\sin(n\pi) = (-1)^n.$$

Computing the FFT, Reduction of Operations

Suppose $\mathbf{m = 2^p}$ for some $p \in \mathbb{Z}^+$, then for $k = 0, 1, \dots, (m-1)$:

$$\begin{aligned} c_k + c_{m+k} &= \sum_{j=0}^{2m-1} f_j \left[e^{ik\pi j/m} + e^{i(m+k)\pi j/m}\right] \\ &= \sum_{j=0}^{2m-1} f_j e^{ik\pi j/m}(1 + e^{i\pi j}). \end{aligned}$$

Using the fact that

$$1 + e^{i\pi j} = \begin{cases} 2, & \text{if } j \text{ is even} \\ 0, & \text{if } j \text{ is odd} \end{cases}$$

only half the terms in the sum need to be computed, *i.e*

$$\mathbf{c_k + c_{m+k}} = \mathbf{2}\sum_{j=0}^{\mathbf{m-1}} f_{\mathbf{2j}} e^{ik\pi \mathbf{2j}/m}.$$

Computing the FFT, Reduction of Operations

We have

$$\mathbf{c_k + c_{m+k}} = \mathbf{2}\sum_{j=0}^{\mathbf{m-1}} f_{\mathbf{2j}} e^{ik\pi \mathbf{2j}/m}.$$

In a similar way we can get

$$\mathbf{c_k - c_{m+k}} = \mathbf{2e^{ik\pi m}}\sum_{j=0}^{\mathbf{m-1}} f_{\mathbf{2j+1}} e^{ik\pi \mathbf{2j}/m}.$$

We now need $m + (m+1)$ complex multiplications for each $k = 0, 1, \dots, (m-1)$ to compute these sums, that is

$$m(2m+1) = 2m^2 + m \quad \text{operations.}$$

Big Whoop™: We reduced the number of operations from $4m^2$ to $2m^2 + m$.

Computing the FFT, Reduction of Operations

Observation: The new sums have **the same structure** as the initial sum, so we can apply the same operations-reducing scheme again, which reduces the $2m^2$ part of the operation count:

$$2\left[\frac{m}{2}\frac{m}{2} + \frac{m}{2}\left(\frac{m}{2} + 1\right)\right] = m^2 + m.$$

Our total operation count is down to $m^2 + 2m$. After repeating the same procedure r times we are down to

$$\frac{m^2}{2^{r-2}} + mr \quad \text{operations.}$$

Since $m = 2^p$ we can keep going until $r = p + 1$, and we have

$$\frac{2^{2p}}{2^{p-1}} + m(p+1) = 2m + pm + m = 3m + m\log_2 m = \mathcal{O}(\mathbf{m}\log_2 \mathbf{m})$$

operations.

Comparing Operation Counts

m	$4m^2$	$3m + m\log_2 m$	Speedup
16	1,024	112	9
64	16,384	576	28
256	262,144	2,816	93
1,024	4,194,304	13,312	315
4,096	67,108,864	61,440	1,092
16,384	1,073,741,824	278,528	3,855
65,536	17,179,869,184	1,245,184	13,797
262,144	274,877,906,944	5,505,024	49,932
1,048,576	4,398,046,511,104	24,117,248	182,361
4,194,304	70,368,744,177,664	104,857,600	671,088
8,388,608	**281,474,976,710,656**	**218,103,808**	**1,290,555**
16,777,216	1,125,899,906,842,624	452,984,832	2,485,513

Comparing Operation Counts

m	$4m^2$	$3m + m\log_2 m$	Speedup
1,048,576	4,398,046,511,104	24,117,248	182,361
8,388,608	**281,474,976,710,656**	**218,103,808**	**1,290,555**

$m = 8,388,608$ roughly corresponds to an 8-Megapixel camera

If a 3.8 GHz Pentium chip could perform one addition or multiplication per clock-cycle (which it can't), we could compute the Fourier coefficients for the 8-Megapixel image in

FFT	Slow FT
0.057 seconds	20.576 hours

Each "FFT second" translates roughly to 15 "Slow-FT days."

Implementing the FFT

Assigning implementation of the FFT falls in the category of *cruel and unusual punishment*.

Matlab implements

`fft` the Fourier transform (data → coefficients)

`ifft` the inverse Fourier transform (coefficients → data)

and the helper functions

`fftshift` `ifftshift` Shifting (and unshifting the coefficient (mostly for display purposes)

The 2-dimensional version `[i]fft2`, and *n*-dimensional `[i]fftn` version of the FFT are also implemented.

The Fastest Fourier Transform in the West (FFTW)

[`http://www.fftw.org/`]

FFTW is a C / Fortran subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions, of both real and complex data, and of arbitrary input size.

FFTW is free software, distributed under the GNU General Public License.

Benchmarks, performed on on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software. Moreover, FFTW's performance is portable: the program will perform well on most architectures without modification.

The Fastest Fourier Transform in the West (FFTW)

It is difficult to summarize in a few words all the complexities that arise when testing many programs, and there is no "best" or "fastest" program.

However, FFTW appears to be the fastest program most of the time for in-order transforms, especially in the multi-dimensional and real-complex cases. Hence the name, "FFTW," which stands for the somewhat whimsical title of "Fastest Fourier Transform in the West." Please visit the benchFFT [http://www.fftw.org/benchfft/] home page for a more extensive survey of the results.

The FFTW package was developed at MIT by Matteo Frigo and Steven G. Johnson

Matlab uses FFTW these days... see `help fftw`.

Homework #9, Not Due Fall 2009

Read the matlab `help` for `fft`, `ifft`, `fftshift`, and `ifftshift`.

[1] Let `x=(-pi:(pi/8):(pi-0.1))'`.
Let `f1=cos(x)`, `f2=cos(2*x)`, `f3=cos(7*x)`, `f4=cos(8*x)`, `f4=cos(9*x)`, `g1=sin(x)`, `g2=sin(2*x)`.
Compute the `fft` of these functions.

[2] Let `f=1+cos(2*x)`. Compute `fftshift(fft(f))`.
Let `g=1+sin(3*x)`. Compute `fftshift(fft(g))`.

[3] Use your observations from [1] and [2] to construct a low-pass filter: Given a function `f`, compute the `fft(f)`. For a given N, keep only the coefficients corresponding to the N lowest frequencies (set all the others to zero). `ifft` the result.
Let `f=1+cos(x)+sin(2*x)+5*cos(7*x)` and apply the above with N=4. Plot both the initial and the filtered f.

[4] Use the FFT to determine the trigonometric interpolating polynomial of degree 8 for $f(x) = x^2 \cos x$ on $[-\pi, \pi]$.

FFT Applications

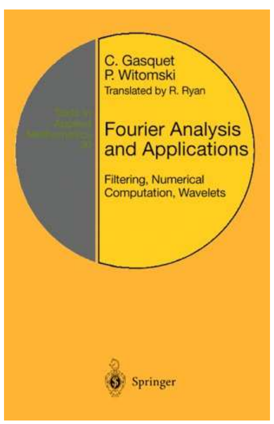


Figure: Some additional reading?

The Fast Fourier Transform, Recap

The Fast Fourier Transform (FFT) is an $\mathcal{O}(m \log_2(m))$ algorithm for computing the $2m$ complex coefficients c_k in the discrete Fourier transform:

$$S_m(x) = \frac{1}{m}\sum_{k=0}^{2m-1} c_k e^{ikx}, \quad \text{where} \quad c_k = \sum_{j=0}^{2m-1} f_j e^{ik\pi j/m}$$

whereas the straight-forward implementation of the sum would require $\mathcal{O}(m^2)$ operations.

We noted [last time] that for a problem of size $2^{23} = 8,388,608$ this reduced the computation time by a factor of a 2 weeks — *i.e.* a computation can be done in t seconds using the FFTs would require $\sim 2t$ weeks to complete using the "Slow" FT.

Notes from: mathworld.wolfram.com 1 of 2

FFTs were first discussed by Cooley and Tukey (1965), although Gauss had actually described the critical factorization step as early as 1805.

A discrete Fourier transform can be computed using an FFT if the number of points N is a power of two.

If the number of points N is not a power of two, a transform can be performed on sets of points corresponding to the prime factors of N which is slightly degraded in speed.

An efficient *real* Fourier transform algorithm or a *fast Hartley transform* gives a further increase in speed by approximately a factor of two.

Notes from: mathworld.wolfram.com 2 of 2

Base-4 and base-8 fast Fourier transforms use optimized code, and can be 20-30% faster than base-2 fast Fourier transforms.

Prime factorization is slow when the factors are large, but discrete Fourier transforms can be made fast for $N = 2, 3, 4, 5, 7, 8, 11, 13, 16$ using the Winograd transform algorithm*.

* this fact, among others, are used in the Fastest Fourier Transform in the West (FFTW).

Example: 1080p High-Def Video

A full 1080p HD-frame has 1920x1080 (2,073,600) pixels. At a bit-depth of 24 bits/pixel, 30 frames/second, 3600 seconds/hour, and 2 hours/movie, that puts us at

$$1{,}343{,}692{,}800{,}000 \text{ bytes/movie} = 1{,}3 \text{ TB/movie}$$

The Blu-ray format has the following storage capacity

	Blu-ray
Capacity/layer	25 GB
#Layers	2
Total capacity	50 GB
Minimum compression	1/26

"Some" processing is required.

Communications From: www.spd.eee.strath.ac.uk 1 of 4

In communications theory the signal is usually a voltage or a current, and Fourier theory is essential to understanding how the signal will behave when it passes through filters, amplifiers and communications channels.

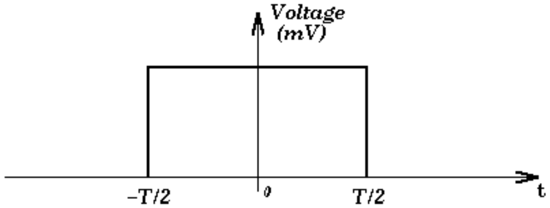
Even discrete digital communications which use 0's or 1's to send information still have frequency contents. This is perhaps easiest to grasp in the case of trying to send a single square pulse down a channel.

The field of communications over a vast range of applications from high level network management down to sending individual bits down a channel. The Fourier transform is usually associated with these low level aspects of communications.

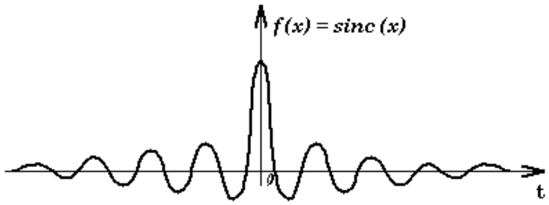
Communications

From: www.spd.eee.strath.ac.uk

If we take simple digital pulse that is to be sent down a telephone line, it will ideally look like this:



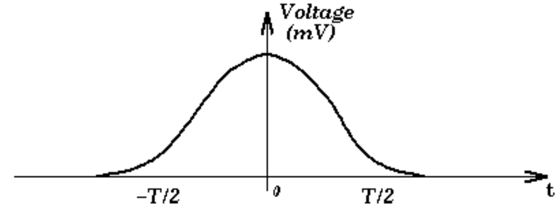
If we take the Fourier transform of this to show what frequencies make up this signal we get something like:



Communications

From: www.spd.eee.strath.ac.uk

This means that the square pulse is a sum of infinite frequencies. However if the telephone line only has a bandwidth of 10 MHz then only the frequencies below 10 MHz will get through the channel. This will cause the digital pulse to be distorted *e.g.*



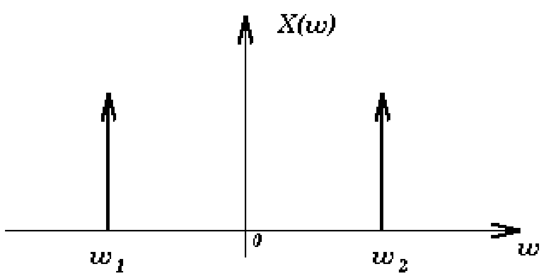
This fact has to be considered when trying to send large amounts of data down a channel, because if too much is sent then the data will be corrupted by the channel and will be unusable.

Communications

From: www.spd.eee.strath.ac.uk

Extending the example of the telephone line, whenever you dial a number on a " touch-tone" phone you hear a series of different tones. Each of these tones is composed of two different frequencies that add together to produce the sound you hear.

The Fourier transform is an ideal method to illustrate this, as it shows these two frequencies *e.g.*



Side Scan Sonar

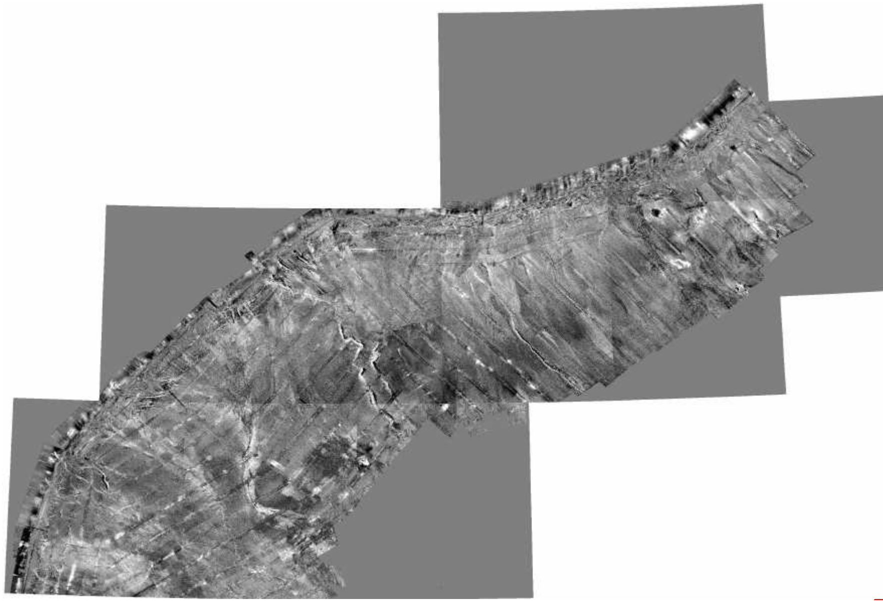
From: www.spd.eee.strath.ac.uk

Another use of DSP techniques (including DFT/FFT) is in sonar. The example given here is side-scan sonar which is a little different from the normal idea of sonar.

With this method a 6.5-kHz sound pulse is transmitted into the ocean toward the sea floor at an oblique angle. Because the signal is transmitted at an oblique angle rather than straight down, the reflected signal provides information about the inclination of the sea floor, surface roughness, and acoustic impedance of the sea floor. It also highlights any small structures on the sea floor, folds, and any fault lines that may be present.

Side Scan Sonar

From: www.spd.eee.strath.ac.uk



Side Scan Sonar

From: www.spd.eee.strath.ac.uk

The image on the previous slide came from the United States Geological Survey (USGS) which is using geophysical Long Range ASDIC (GLORIA) sidescan sonar system to obtain a plan view of the sea floor of the Exclusive Economic Zone (EEZ). The picture element (pixel) resolution is approximately 50 meters. The data are digitally mosaiced into image maps which are at a scale of 1:500,000. These mosaics provide the equivalent of "aerial photographs" that reveal many physiographic and geologic features of the sea floor.

To date the project has covered approximately 2 million square nautical miles of sea floor seaward of the shelf edge around the 30 coastal States. Mapping is continuing around the American Flag Islands of the central and western Pacific Ocean.

Astronomy

From: www.spd.eee.strath.ac.uk

Venus is Earth's closest companion, and is comparable in it's size and diameter and yet scientists knew virtually nothing about it, as it is perpetually covered with a cloud layer which normal optical telescopes can't penetrate, so Magellan (a satellite) had radar and advanced Digital Signal Processing that was designed to "see" through this cloud layer. It's mission was map 70% of the planet with radar and to reveal surface features as small as 250 meters across.

The black and white pictures that it sent back were strips of the planets surface, about 20km wide, from the north pole to the south pole.

Astronomy

From: www.spd.eee.strath.ac.uk

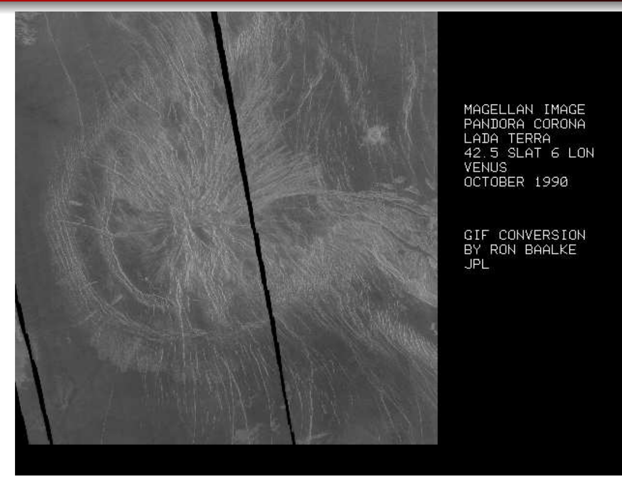


Figure: One example image of a surface feature called "Pandora Corona." If you look at the image, you will see a two black lines through the picture. This is just a mismatch between the strips sent back by Magellan. It also gives you an idea of the scale of the image as each strip is 20km wide.

Astronomy

From: www.spd.eee.strath.ac.uk

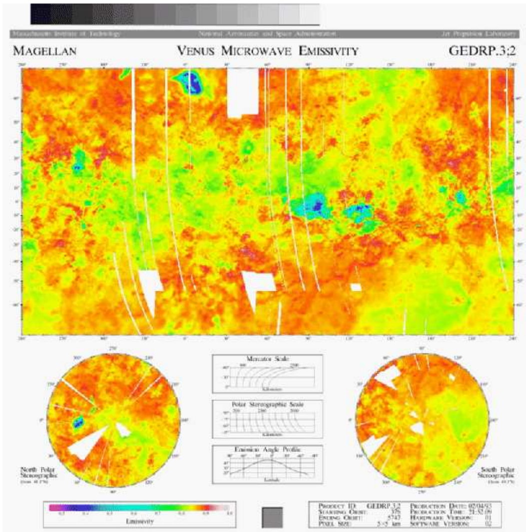


Figure: A global map showing emissivity of the Venus's surface.

Example: Image Processing

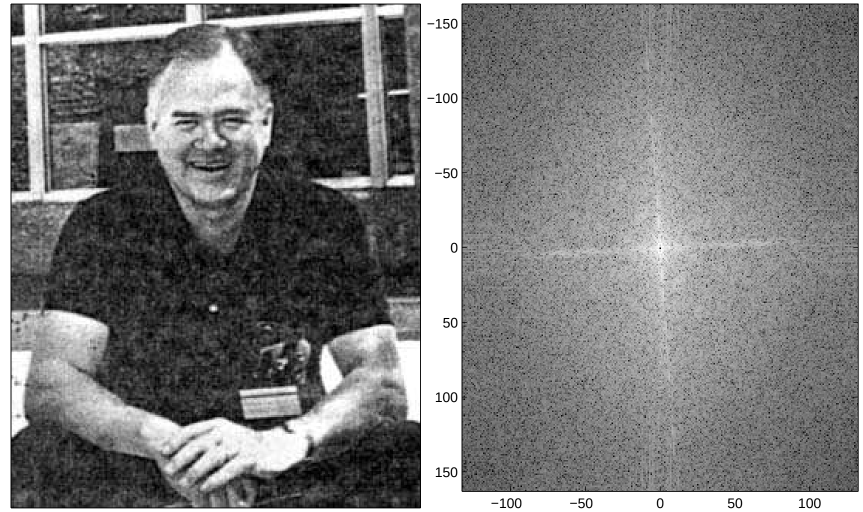


Figure: John Tukey, and his Fourier transform. The coefficients corresponding to the low frequencies are in the center of the plot, and the ones corresponding to the high frequencies are toward the edges.

Example: Image Processing — 25% Compression

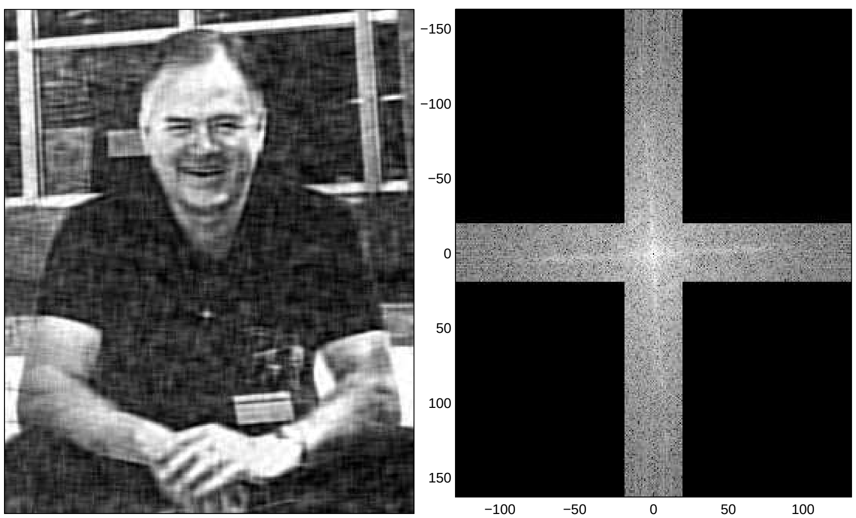


Figure: Here, we filter out $\sim 75\%$ of the Fourier coefficients, and reconstruct the image.

Example: Image Processing — 25% Compression

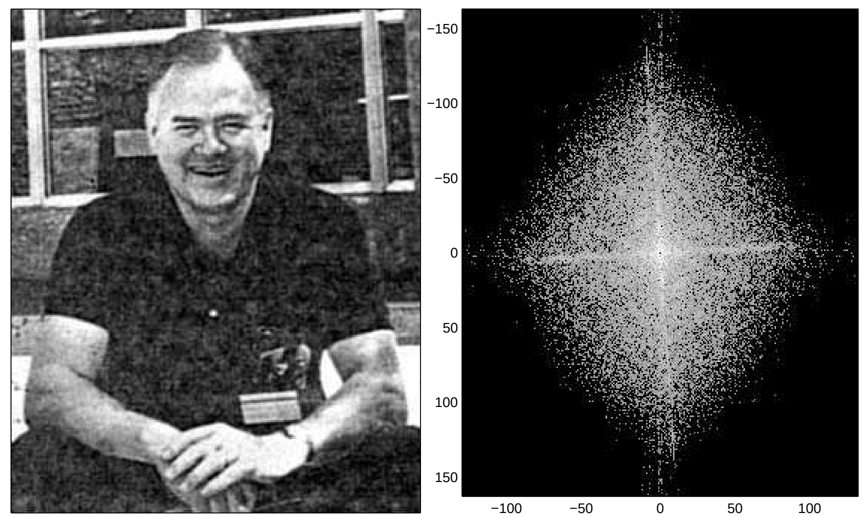


Figure: Here, we filter the $\sim 25\%$ of the Fourier coefficients with highest energy (largest value in the absolute sense), and reconstruct the image.

Example: Image Processing — Highpass Filtering

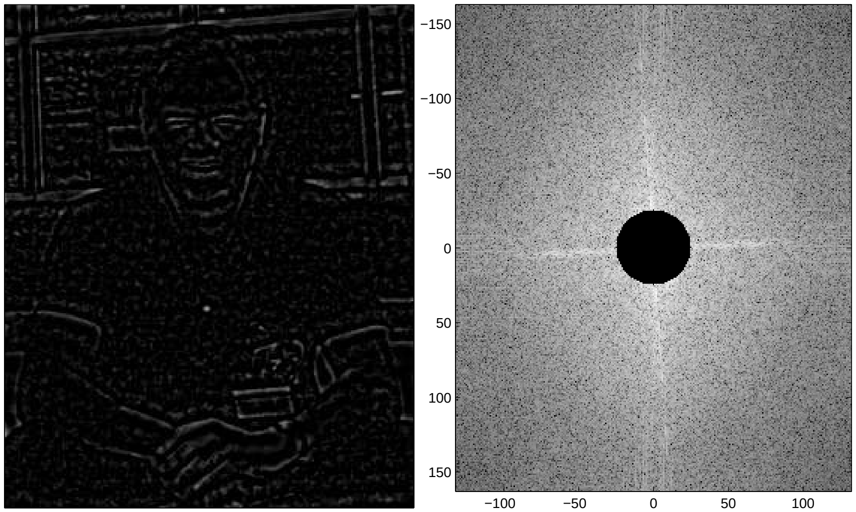


Figure: Example of high-pass filtering the image. We have filtered out the Fourier coefficients corresponding to the lowest frequencies.

Example: Image Processing — Lowpass Filtering

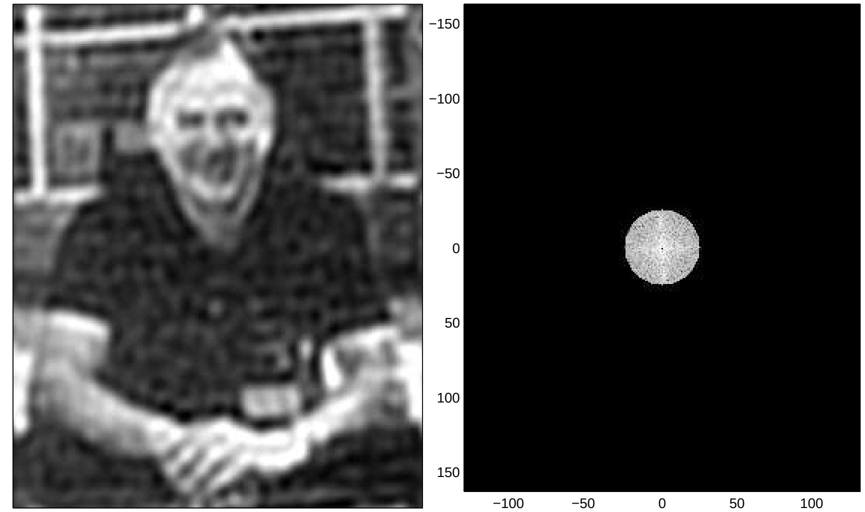


Figure: Example of low-pass filtering the image. We have filtered out the Fourier coefficients corresponding to the highest frequencies.

[See movies]