Chapter 6

FFT Algorithms

Contents

Efficient computation of the DFT 6.2
Applications of FFT 6.6
Computing DFT of 2 real sequences 6.6
Efficient computation of the DFT of a $2N$-point real sequence 6.6
Use of the FFT in linear filtering 6.6
Linear Filtering Approach to Computing the DFT 6.6
Quantization Effects in Computing the DFT 6.6
Summary 6.6
The discrete cosine transform 6.7
1D DCT 6.7

We have seen already a few applications of the DFT, and there are *many* more. Because the DFT is so ubiquitous, fast methods for computing the $X[k]$'s have been studied extensively, and continues to be an active research topic.

Many software packages for the FFT are available, so many DSP users will never need to write their own FFT routines. But it is important to understand how FFTs work, just like understanding arithmetic is essential for effective use of a calculator.

6.1 Efficient computation of the DFT

The problem:
Given signal samples: $x[0], \ldots, x[N-1]$ (some of which may be zero), develop a procedure to compute

$$X[k] = \sum_{n=0}^{N-1} x[n] \, W_N^{kn}$$

for $k = 0, \ldots, N-1$ where

$$W_N = \mathrm{e}^{-\jmath \frac{2\pi}{N}} .$$

We would like the procedure to be
- fast,
- accurate,
- simple.

Fast is the most important, so we will sacrifice simplicity for speed, hopefully with minimal loss of accuracy.

6.1.1 Direct computation of the DFT

Let us start with the "simple" way.

Assume that W_N^{kn} has been precomputed and stored in a table for the N of interest.
How big should the table be? W_N^m is periodic in m with period N, so we just need to tabulate the N values:

$$W_N^m = \cos\left(\frac{2\pi}{N} m\right) - \jmath \sin\left(\frac{2\pi}{N} m\right)$$

for $m = 0, \ldots, N-1$.
(Possibly even less since $\sin$ is just $\cos$ shifted by a quarter period, so we could save just $\cos$ when N is a multiple of 4.)

Why tabulate? To avoid repeated function calls to $\cos$ and $\sin$ when computing the DFT.

Now we can compute each $X[k]$ directly form the formula as follows

$$X[k] = \sum_{n=0}^{N-1} x[n] \, W_N^{kn} = x[0] \, W_N^0 + x[1] \, W_N^k + x[2] \, W_N^{2k} + \cdots + x[N-1] \, W_N^{(N-1)k}.$$

For each value of k, there are N complex multiplications, and $N-1$ complex additions. There are N values of k, so the total number of complex operations is

$$N \cdot N + N(N-1) = 2N^2 - N \equiv O(N^2).$$

Complex multiplies require 4 real multiplies and 2 real additions, whereas complex additions require just 2 real additions.
So the N^2 complex multiplies are the primary concern.

N^2 increases *rapidly* with N, so how can we reduce the amount of computation? By exploiting the following properties of W:
- Symmetry property: $W_N^{k+N/2} = -W_N^k = \mathrm{e}^{\jmath\pi} \, W_N^k$
- Periodicity property: $W_N^{k+N} = W_N^k$
- Recursion property: $W_N^2 = W_{N/2}$

The first and third properties hold for even N, *i.e.*, when 2 is one of the prime factors of N. There are related properties for other prime factors of N.

6.1.2 Divide and conquer approach

Idea: break N-point DFT, where $N = N_1 N_2$, into two (or more) shorter DFTs and combine the results appropriately.

Skim

6.1.3 Radix-2 FFT

Useful when N is a power of 2:

$$N = r^\nu$$

for integers r and ν. r is called the **radix**, which comes from the Latin word meaning "a root," and has the same origins as the word radish.

When N is a power of $r = 2$, this is called **radix-2**, and the natural "divide and conquer approach" is to split the sequence into two sequences of length $N/2$.

This is a very clever trick that goes back many years.

Decimation in time

Split the sequence $x[n]$ into two sequences of length $N/2$,
- $s_1[n] = x[2n]$, even samples
- $s_2[n] = x[2n+1]$, odd samples

for $n = 0, \dots, N/2 - 1$.

$$\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n]\, W_N^{kn} \\
&= \sum_{\substack{n=0 \\ n \text{ even} \\ \text{so } n = 2m}}^{N-1} x[n]\, W_N^{kn} + \sum_{\substack{n=0 \\ n \text{ odd} \\ \text{so } n = 2m+1}}^{N-1} x[n]\, W_N^{kn} \\
&= \sum_{m=0}^{N/2-1} x[2m]\, W_N^{k2m} + \sum_{m=0}^{N/2-1} x[2m+1]\, W_N^{k(2m+1)} \\
&= \sum_{m=0}^{N/2-1} s_1[m]\, W_{N/2}^{km} + W_N^k \sum_{m=0}^{N/2-1} s_2[m]\, W_{N/2}^{km} \\
&\qquad \text{by def'n of } s_l[n] \text{ and because of the recursion property } W_N^2 = W_{N/2} \\
&= S_1[k] + W_N^k\, S_2[k]
\end{aligned}$$

where

$$S_1[k] = \sum_{n=0}^{N/2-1} s_1[n]\, W_{N/2}^{kn} \text{ and } S_2[k] = \sum_{n=0}^{N/2-1} s_2[n]\, W_{N/2}^{kn}.$$

Key trick above was the recursion property: $W_N^2 = W_{N/2}$.

What are $S_1[k]$ and $S_2[k]$? $S_1[k]$ and $S_2[k]$ are just the $N/2$-point DFTs of $s_1[n]$ and $s_2[n]$ respectively, or more precisely, the N-point circular extension thereof.

So we have shown

$$\boxed{X[k] = S_1[k \bmod N/2] + W_N^k\, S_2[k \bmod N/2], \qquad k = 0, \dots, N-1.}$$

Recipe:
- Compute $N/2$-point DFT $S_1[k]$ of $s_1[n]$ = even samples of $x[n]$.
- Compute $N/2$-point DFT $S_2[k]$ of $s_2[n]$ = odd samples of $x[n]$.
- Combine: $X[k] = S_1[k \bmod N/2] + W_N^k\, S_2[k \bmod N/2], \quad k = 0, \dots, N-1$

How much computation? Each DFT requires $(N/2)^2$ complex multiplies, and there are N complex multiplies needed to combine. So $2(N/2)^2 + N \approx N^2/2$. Compared with N^2 complex multiplies before.
So *just by rearranging the formula*, we have saved a factor of 2 in complex multiplies!

We can save a little more computation by exploiting periodicity.

What is the period of $S_1[k]$? Since $S_1[k]$ and $S_2[k]$ are $N/2$-point DFT's, they are periodic with period $N/2$.

For $k = 0, \ldots, N/2-1$, using symmetry property of W_N's:

$$X[k+N/2] = S_1[k+N/2 \bmod N/2] + W_N^{k+N/2} S_2[k+N/2 \bmod N/2] = S_1[k] - W_N^k S_2[k] \,.$$

So

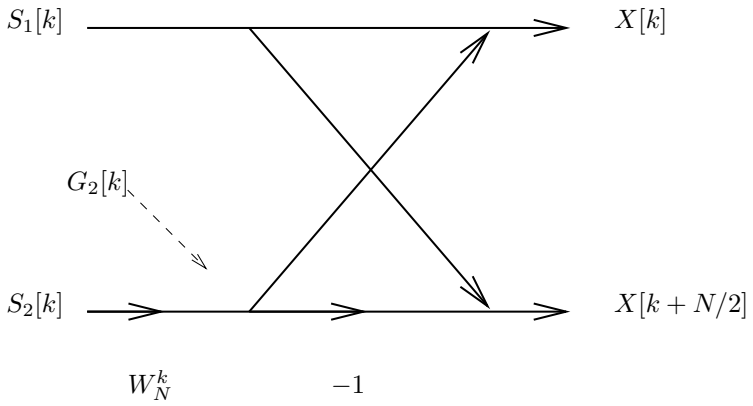
$$\begin{aligned} X[k] &= S_1[k] + W_N^k S_2[k] \\ X[k+N/2] &= S_1[k] - W_N^k S_2[k], \qquad k = 0, \ldots, N/2-1. \end{aligned}$$

New recipe:
- Compute $N/2$-point DFT $S_1[k]$ of $s_1[n]$ = even samples of $x[n]$.
- Compute $N/2$-point DFT $S_2[k]$ of $s_2[n]$ = odd samples of $x[n]$.
- Compute $G_2[k] = W_N^k \, S_2[k], \; k = 0, \ldots, N/2-1$
- Combine:

$$\begin{aligned} X[k] &= S_1[k] + G_2[k] \\ X[k+N/2] &= S_1[k] - G_2[k], \qquad k = 0, \ldots, N/2-1. \end{aligned}$$

$2(N/2)^2 + N/2$ complex multiplies, saving just a little more. Perhaps more importantly, it allows for in-place computation.

Butterfly:



Summary of key idea: splitting into two sequences (even/odd) each of half the length, take DFT of each, then combine, saves a factor of 2 in complex multiplies.

Recursion

Apply the same idea to the two $N/2$-point DFTs!

Can do it $\log_2$ times, and the total number of complex multiplies is approximately

$$\frac{N}{2} \log_2 N$$

which is much smaller than N^2.

e.g., for $N = 32$, $N^2 = 1024$, whereas $N/2 \log_2 N = 16 \cdot 5 = 80$.

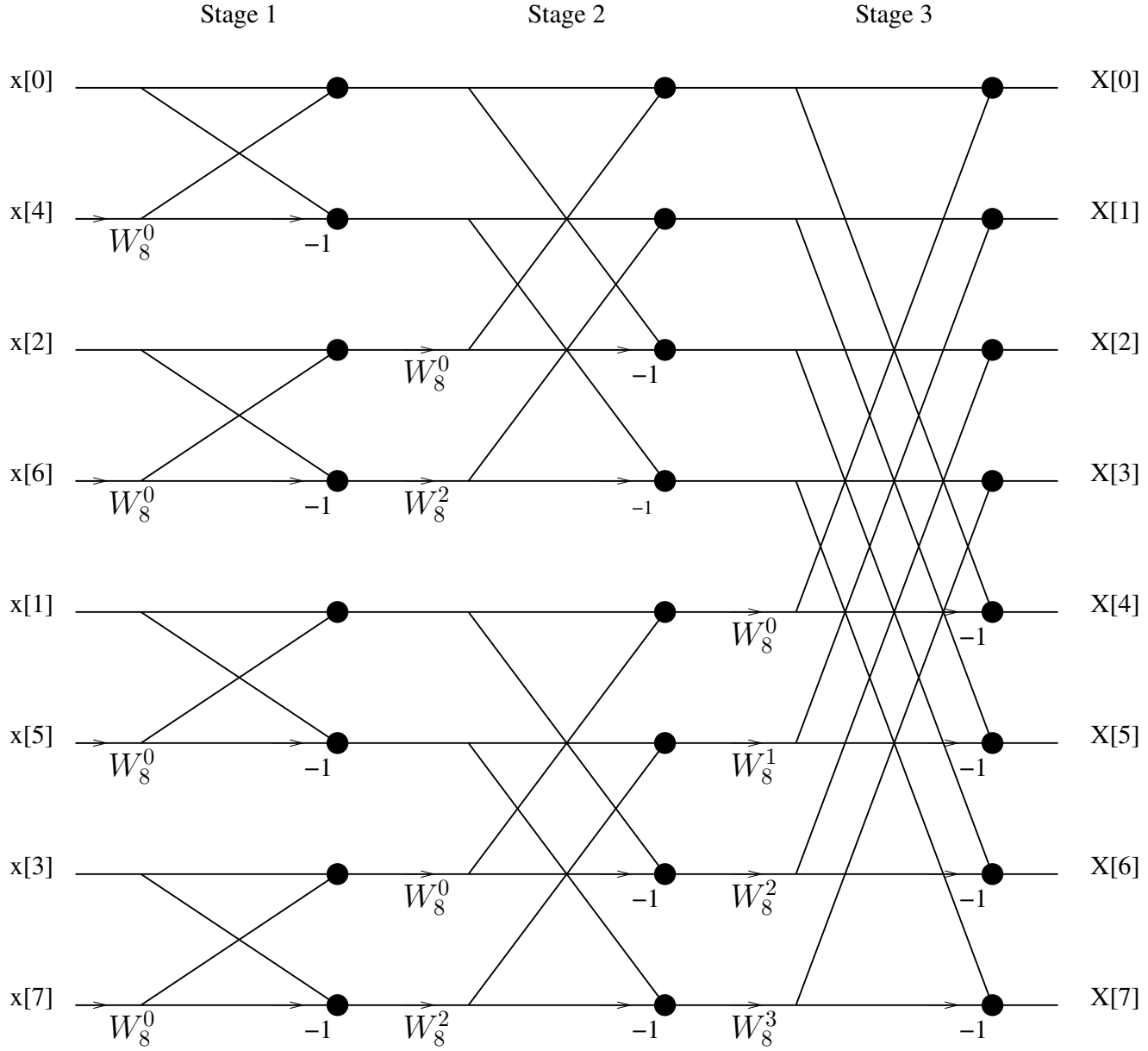
Figure 6.5, 6.6

Caution: Fig. 6.6 on p. 460 has superfluous dots in it

2-point DFT: ***Picture of 2-point butterfly!***

$$X[k] = \sum_{n=0}^{1} x[n] \, \mathrm{e}^{-\jmath \frac{2\pi}{2} kn} = x[0] + \mathrm{e}^{-\jmath \pi k} \, x[1] \implies \begin{array}{l} X[0] = x[0] + x[1] \\ X[1] = x[0] - x[1] \end{array}$$

$N = 8$-point decimation-in-time FFT algorithm



$$W_N = \mathrm{e}^{-\jmath \frac{2\pi}{N}}$$

Each dot represents a complex addition.
Each arrow represents a complex multiplication.

6.2

Applications of FFT

There are many.

6.2.1

Computing DFT of 2 real sequences

Suppose $x[n]$ and $h[n]$ are real and we want:

$x[n] \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X[k]$

$h[n] \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} H[k]$

Naive approach takes $N/2 \log_2 N$ complex multiplies each, for a total of $N \log_2 N$ complex multiplies.

Trick: form new complex sequence:

$$w[n] = x[n] + \jmath\, h[n]$$

and take its DFT:

$$w[n] \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} W[k] = X[k] + \jmath\, H[k] \,.$$

But $X[k]$ and $H[k]$ are not real in general, so we must think harder to extract $X[k]$ and $H[k]$ from the $W[k]$'s.

Trick:

$$x[n] = \text{real}(w[n]) = \frac{1}{2}\left(w[n] + w^*[n]\right) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} X[k] = \frac{1}{2}\left(W[k] + W^*[-k \bmod N]\right)$$

by real property of DFT. Similarly

$$h[n] = \text{imag}(h[n]) = \frac{1}{2\jmath}\left(w[n] - w^*[n]\right) \overset{\text{DFT}}{\underset{N}{\longleftrightarrow}} H[k] = \frac{1}{2\jmath}\left(W[k] - W^*[-k \bmod N]\right) .$$

Recipe:
- Form complex sequence $w[n]$
- Compute N-point DFT of $w[n]$
- $X[k] = \frac{1}{2}\left(W[k] + W^*[-k \bmod N]\right)$
- $H[k] = \frac{1}{2\jmath}\left(W[k] - W^*[-k \bmod N]\right)$

Requires about $N/2 \log_2 N$ complex multiplies, so a factor of 2 savings over naive approach.

6.2.2

Efficient computation of the DFT of a $2N$-point real sequence

6.2.3

Use of the FFT in linear filtering

6.3

Linear Filtering Approach to Computing the DFT ***skip***

6.4

Quantization Effects in Computing the DFT ***skip***

6.5

Summary
- The compute savings of the FFT relative to the DFT launched the age of digital signal processing.
- We emphasized radix-2 case, but good FFT implementations accommodate any N.
- A popular freeware implementation is the `fftw` package.

The discrete cosine transform (not covered due to lack of time)

The DFT/FFT are excellent for convolution, and useful for frequency-domain analysis of sampled analog signals. So why did someone invent a new transform, the DCT?

For image compression, we would like **energy compaction**; we would like a transform that reduces the signals of interest to a small number of nonzero coefficients. Image compression is often done in blocks. Suppose we select a small block from some natural image. The DFT of the block gives us the values of the discrete Fourier series of the periodic extension of that signal. Suppose the periodic extension has a discontinuity at the block boundaries. Then the DFT coefficients will decay slowly, just like the FT of a square wave (discontinuous) decay as $1/k$, whereas those of a triangle wave decay as $1/k^2$. So *any* discontinuities in an image, include at the boundary of a block, lead to poor energy compaction of the coefficients.

As an additional drawback of the DFT, if the image is real, then its coefficients are complex. All other things being equal, when developing image compression methods one would usually prefer real valued quantities over complex values if the original image is real.

To overcome these drawbacks of the DFT, **discrete cosine transform** (**DCT**) uses the trick of taking the image (block) and forming a symmetrized version of it before computing a DFT. This symmetrization has the effect of
- eliminating discontinuities at the block edges, and
- yielding real coefficients.

Interestingly though, the DCT is derived via the DFT. So Fourier analysis is still the fundamental tool, even for this new transform.

1D DCT

Consider the signal of extent $N = 4$: $x[n] = \{\underline{2}, 4, 6, 8\}$. Its 4-point circular extension is $\tilde{x}[n] = \{\ldots, 2, 4, 6, 8, \underline{2}, 4, 6, 8, 2, \ldots\}$. Note the discontinuity.

Now consider the new signal of length $2N$:

$$y[n] = \begin{cases} x[n], & 0 \le n \le N-1 \\ x[2N-1-n], & N \le n \le 2N-1 \\ 0, & \text{otherwise}, \end{cases}$$

which is $y[n] = \{\underline{2}, 4, 6, 8, 8, 6, 4, 2\}$ in the example.
This signal has no jumps at the boundaries. We now derive the DCT via the DFT. For $0 \le k \le 2N-1$:

$$\begin{aligned} Y[k] &= \sum_{n=0}^{2N-1} y[n]\, \mathrm{e}^{-\jmath \frac{2\pi}{2N} kn} = \sum_{n=0}^{2N-1} y[n]\, W_{2N}^{kn} \\ &= \sum_{n=0}^{N-1} x[n]\, W_{2N}^{kn} + \sum_{n=N}^{2N-1} x[2N-1-n]\, W_{2N}^{kn} \\ &= \sum_{n=0}^{N-1} x[n]\, W_{2N}^{kn} + \sum_{n=0}^{N-1} x[n]\, W_{2N}^{k(2N-1-n)} = \sum_{n=0}^{N-1} x[n]\, W_{2N}^{kn} [1 + W_{2N}^{-k}] \\ &= W_{2N}^{-k/2} \sum_{n=0}^{N-1} x[n] \left(W_{2N}^{k(n+1/2)} + W_{2N}^{-k(n+1/2)} \right) = W_{2N}^{-k/2} \sum_{n=0}^{N-1} x[n]\, 2 \cos\left(\frac{2\pi k(2n+1)}{4N} \right) \end{aligned}$$

where $W_N \triangleq \mathrm{e}^{-\jmath \frac{2\pi}{N}}$.

Suppose we started with a length N signal $x[n]$. Now we have $2N$ *complex* DFT coefficients $Y[0], \ldots, Y[2N-1]$. This hardly seems like progress towards compaction! But obviously there must be some redundancies in the $Y[k]$'s.
- Note that $Y[N] = 0$, as a consequence of the symmetry in the construction of $y[n]$.
- $W_{2N}^{k/2}\, Y[k]$ is *real* if $x[n]$ is real, since it is a sum of $x[n]$'s times a cosine.
- One can verify that

$$-W_{2N}^{(2N-k)/2}\, Y[2N-k] = W_{2N}^{k/2}\, Y[k], \quad k = 1, \ldots, 2N-1,$$

  which is a form of odd symmetry. So we only really need to save *half* of the $Y[k]$'s.

In light of these properties, the 1D DCT is defined as follows:

$$C_x[k] \triangleq \begin{cases} W_{2N}^{k/2} Y[k], & 0 \le k \le N-1 \\ 0, & \text{otherwise.} \end{cases}$$

$$\boxed{C_x[k] = \sum_{n=0}^{N-1} 2\, x[n] \cos\left(\frac{\pi k (2n+1)}{2N} \right), \quad k = 0, \ldots, N-1.}$$

A few properties of the DCT:
- Maps an N-point sequence to another N-point sequence.
- If $x[n]$ is real, then so is its DCT.
- $C_x[0] = 2\, Y[0]$, so the 0th component of the DCT is twice that of the DFT.

Is the set of signals $\left\{ \cos\left(\frac{\pi k(2n+1)}{2N} \right) \right\}_{k=0}^{N-1}$ an orthogonal set over $n = 0, \ldots, N-1$?
Why does it matter? (Simplicity in reconstruction.)

For the inverse DCT, one can recover the $2N$-point DFT coefficients $Y[k]$ from the DCT coefficients $C_x[\cdot]$ as follows:

$$Y[k] = \begin{cases} W_{2N}^{-k/2} C_x[k], & k = 0, \ldots, N-1 \\ 0, & k = N \\ -W_{2N}^{-k/2} C_x[2N-k], & k = N+1, \ldots, 2N-1. \end{cases}$$

Substituting into the iDFT formula and simplifying (or applying the orthogonality of the DCT basis) yields:

$$\boxed{x[n] = y[n] = \frac{1}{N} \sum_{k=0}^{N-1} w_k\, C_x[k] \cos\left(\frac{\pi k (2n+1)}{2N} \right), \quad n = 0, \ldots, N-1,}$$

where

$$w_k \triangleq \begin{cases} 1/2, & k = 0 \\ 1, & \text{otherwise.} \end{cases} \tag{6-1}$$

(Note that $Y[0] = 2 \sum_{n=0}^{N-1} x[n]$.)

Rarely does one *implement* the DCT using the two boxed formulae above, since that would require $O(N^2)$ operations. Instead one uses an FFT-based algorithm.

Basic algorithm for 1D DCT
- extend $x[n]$ to form $y[n]$. (Use MATLAB's `fliplr` or `flipud` command.)
- compute $2N$-point DFT $Y[k]$ from $y[n]$. (Use MATLAB's `fft` command.)
- $C_x[k] = W_{2N}^{k/2} Y[k],\ k = 0, \ldots, N-1$ (Use MATLAB's `real` command after scaling by the $W_{2N}^{k/2}$'s since there will be some residual complex part due to finite precision.)

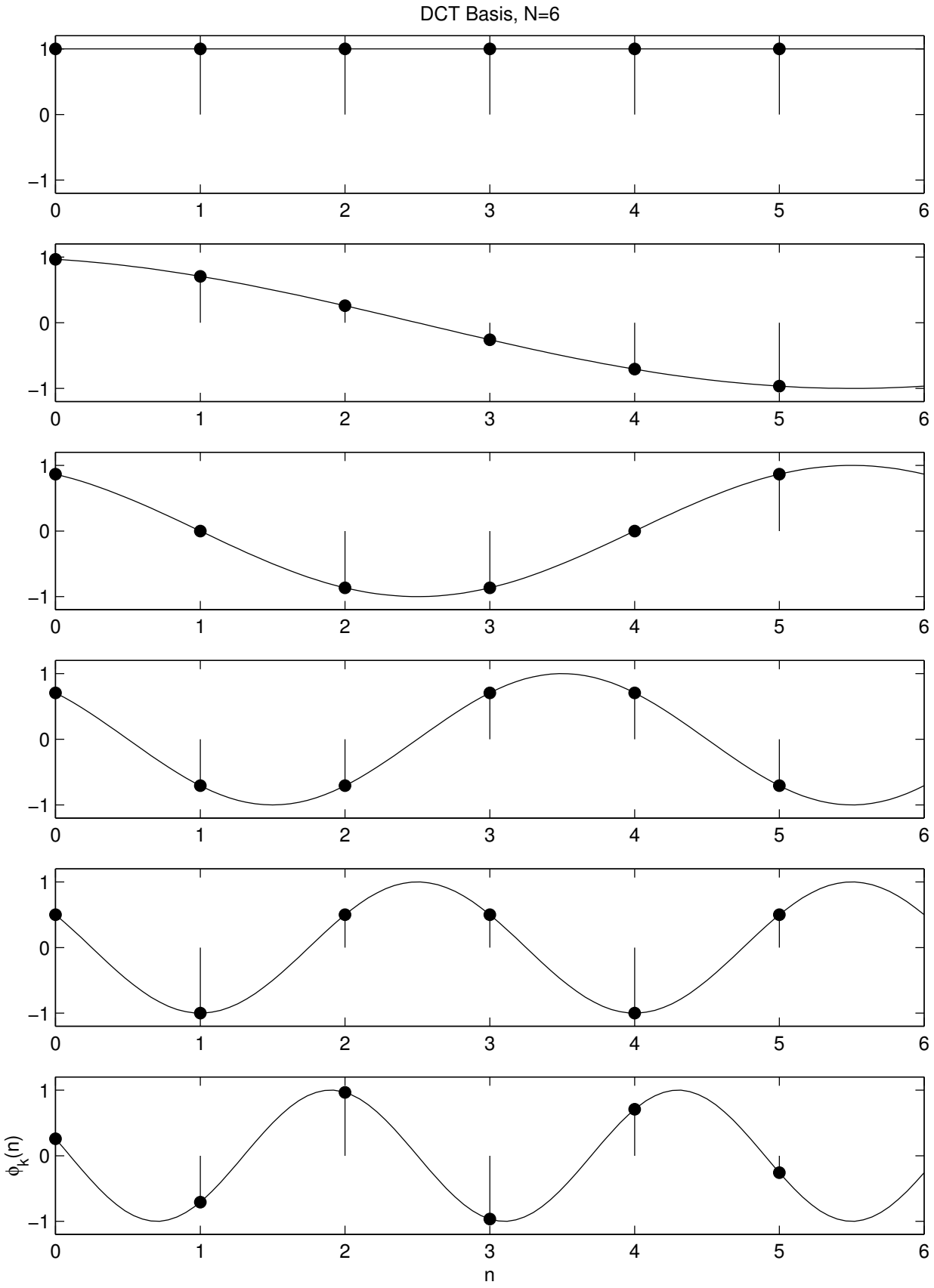
Similar for inverse DCT. In fact it can be done using N-point DFTs too. (A problem in Lim.)

Caution! MATLAB's `dct` command uses a slightly different definition of the DCT that is normalized so that it is an **orthonormal** transformation, following [4, p. 150-3].

Example: $x[n] = \{\underline{2}, 4, 6, 8\}$ has DFT $\{20, -4 + \jmath 4, -4, -4 - \jmath 4\}$. The DCT is $\{40, -12.6, 0, -0.897\}$, which has nominally better compaction since one of the entries is zero.

Since the DCT input sequence $y[n]$ has no extraneous sharp discontinuities, it will lead to better energy compaction in the frequency domain than the DFT input sequence $x[n]$, *i.e.*, more energy is concentrated in low frequency components.

What is the catch? What signals are better compacted by the DFT?


DCT Basis, N=6
$\phi_k(n)$
n