CMPN206: Multimedia

Lecture 6: Transform and Differential Coding

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2014

Agenda

- Transform Coding
  - Transform Properties
  - 1D Transforms
  - 2D Transforms
  - DCT Transform
- Differential Coding

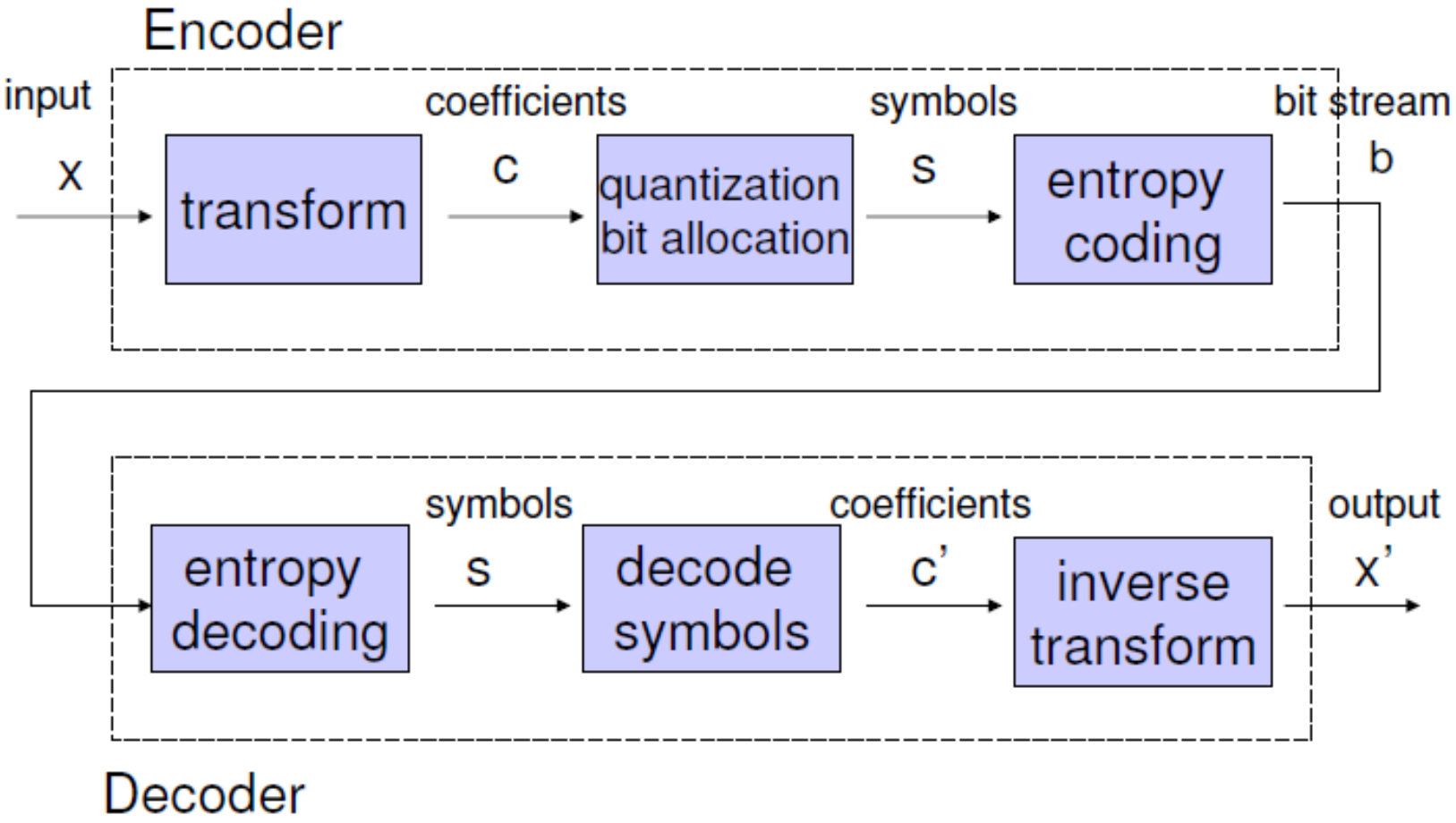
Acknowledgments: Most slides are adapted from Richard Ladner, from Li and Drew, and from Khaled Sayood.

Transform Coding

- *Transform* the input vector *X* into another vector *Y* such that:
  - Most of the entries of the vector *Y* are zeros or near zero
  - Most of the *energy* or *information* is compacted into a few *components* or *coefficients*
- Quantize the coefficients
  - This causes the information *loss*
  - Important coefficients are coded with higher precision i.e. more bits
- Code the quantized coefficients

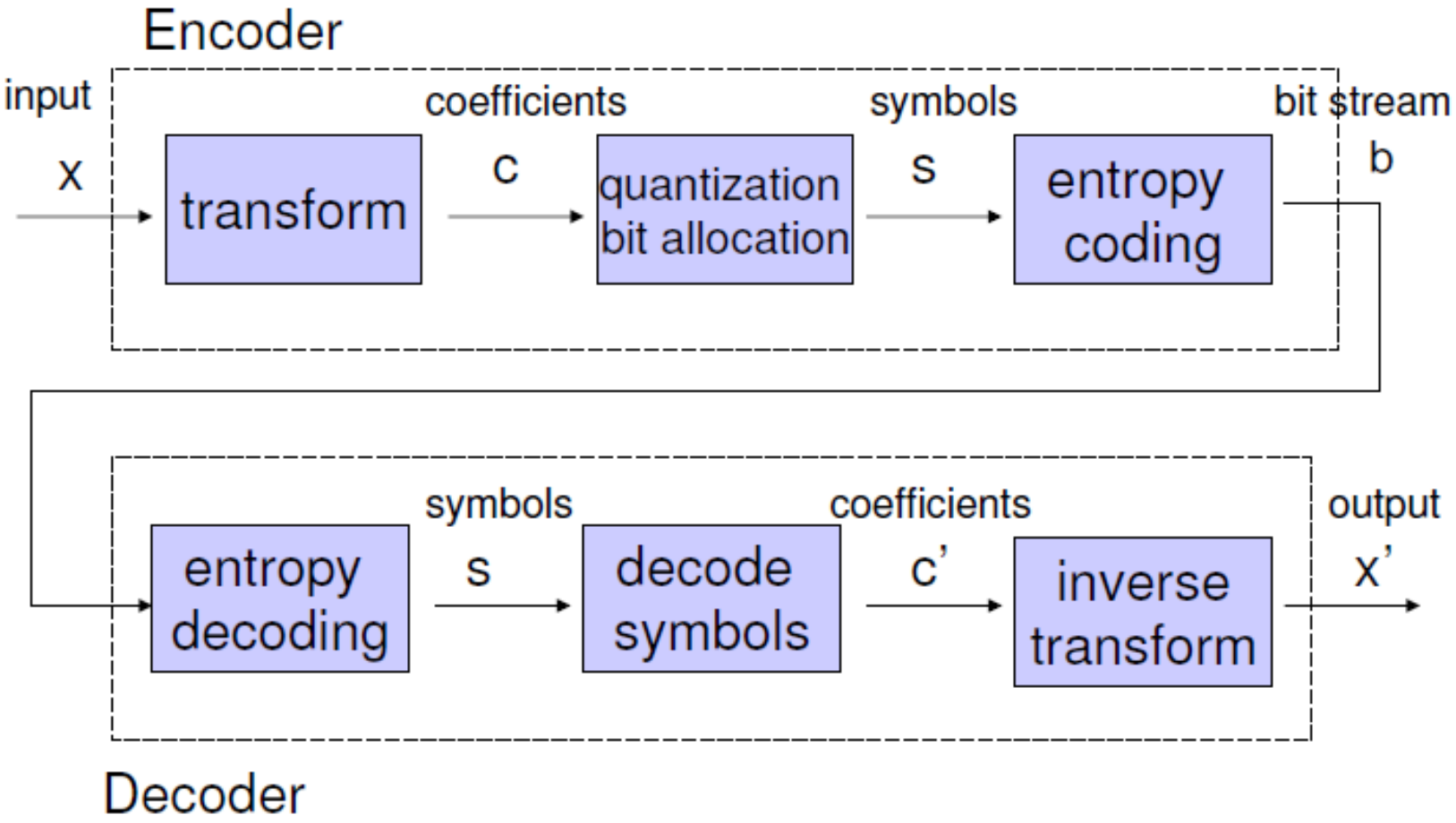
Transform Coding: Encoder

- Transform the input (source output)
- Quantize the *coefficients*
- Code the quantized coefficients using e.g. Huffman Coding



Transform Coding: Decoder

- Decode the entropy coded quantized coefficients
- Decode the quantized coefficients to obtain their approximation
- Apply the inverse transform to obtain the approximation of the input



Example Transform

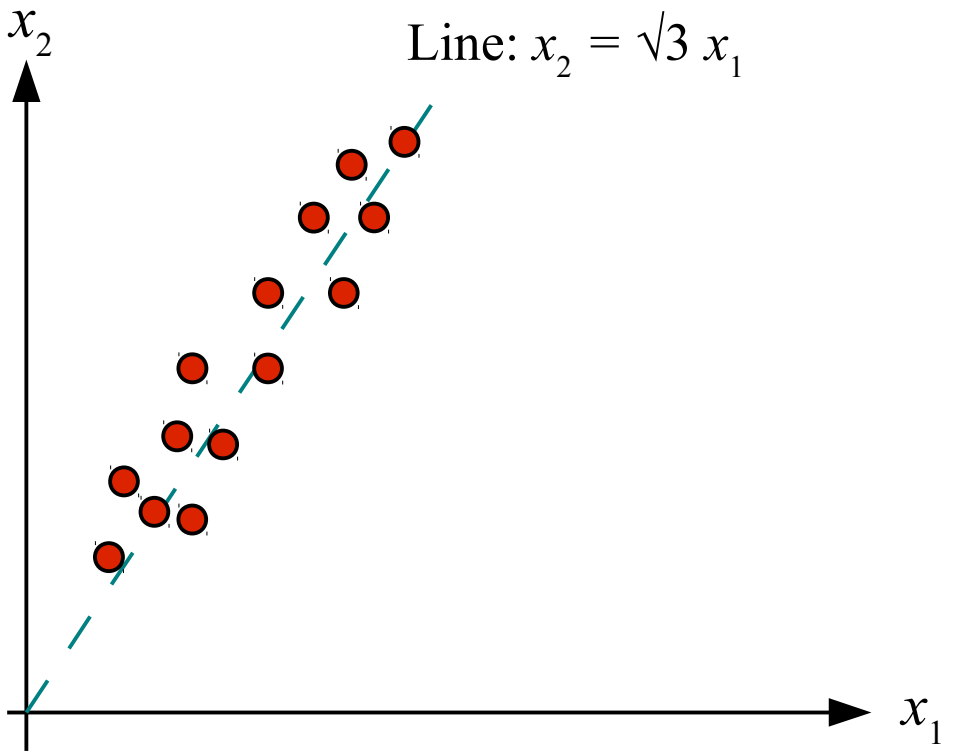
- Assume we have input 2D vectors x as shown
- The points are along the line $x_2 = \sqrt{3}\, x_1$
- We can approximate these input vectors by applying a *rotation* transformation T and keeping only the first coordinate of the transformed vectors y

$$y = T\, x$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- The transformation T is a rotation by 60 degrees:

$$T = \begin{bmatrix} \cos 60 & -\sin 60 \\ \sin 60 & \cos 60 \end{bmatrix} = \begin{bmatrix} 1/2 & -\sqrt{3}/2 \\ \sqrt{3}/2 & 1/2 \end{bmatrix}$$

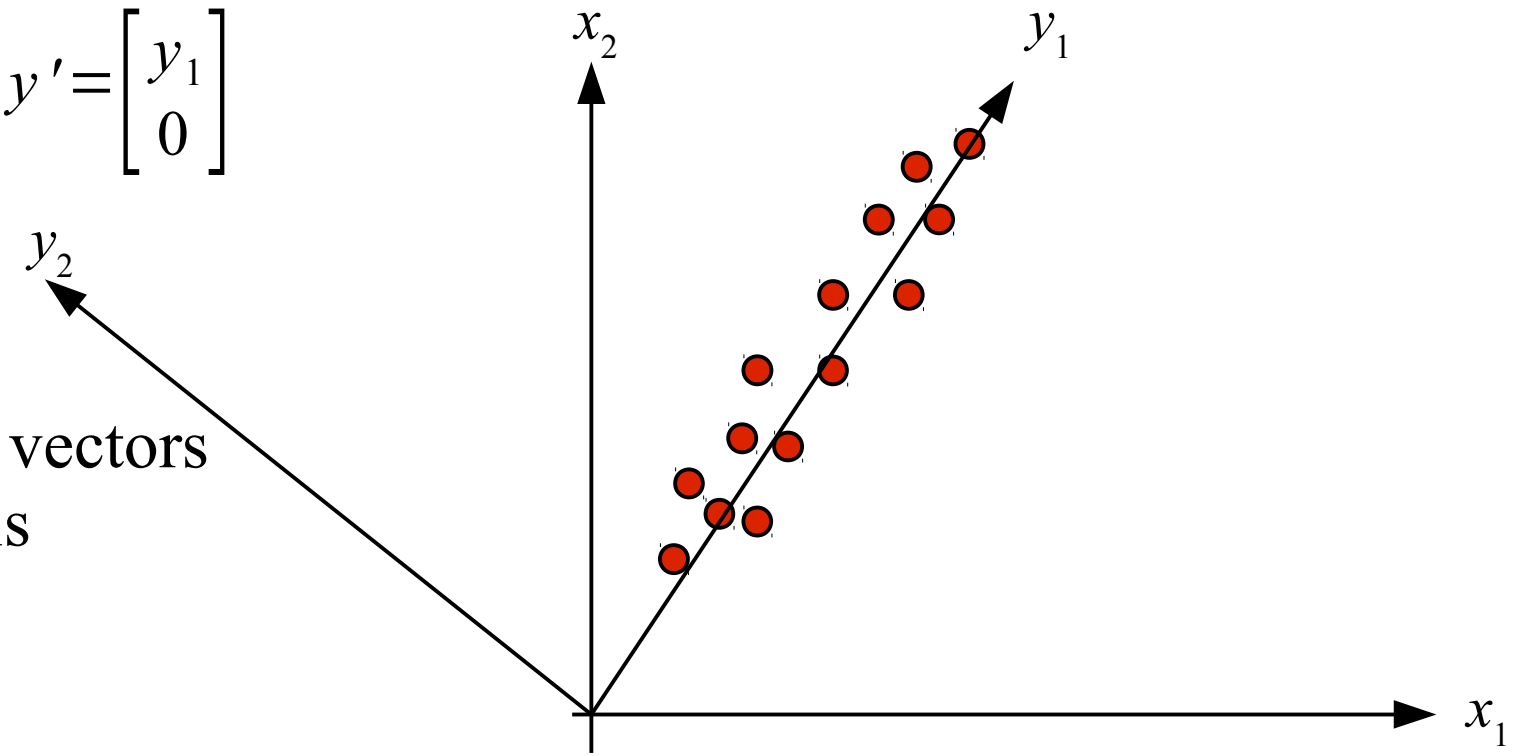


Example Transform

- This can be viewed as a change of coordinates
- The coordinate y_1 has most of the information (or variance) while the second coordinate y_2 is very close to zero for all the points
- So we can *transform* the input 2D vector x to the vector y, then only keep the first *coefficient* y_1 to get the vector y':

$$y' = \begin{bmatrix} y_1 \\ 0 \end{bmatrix}$$

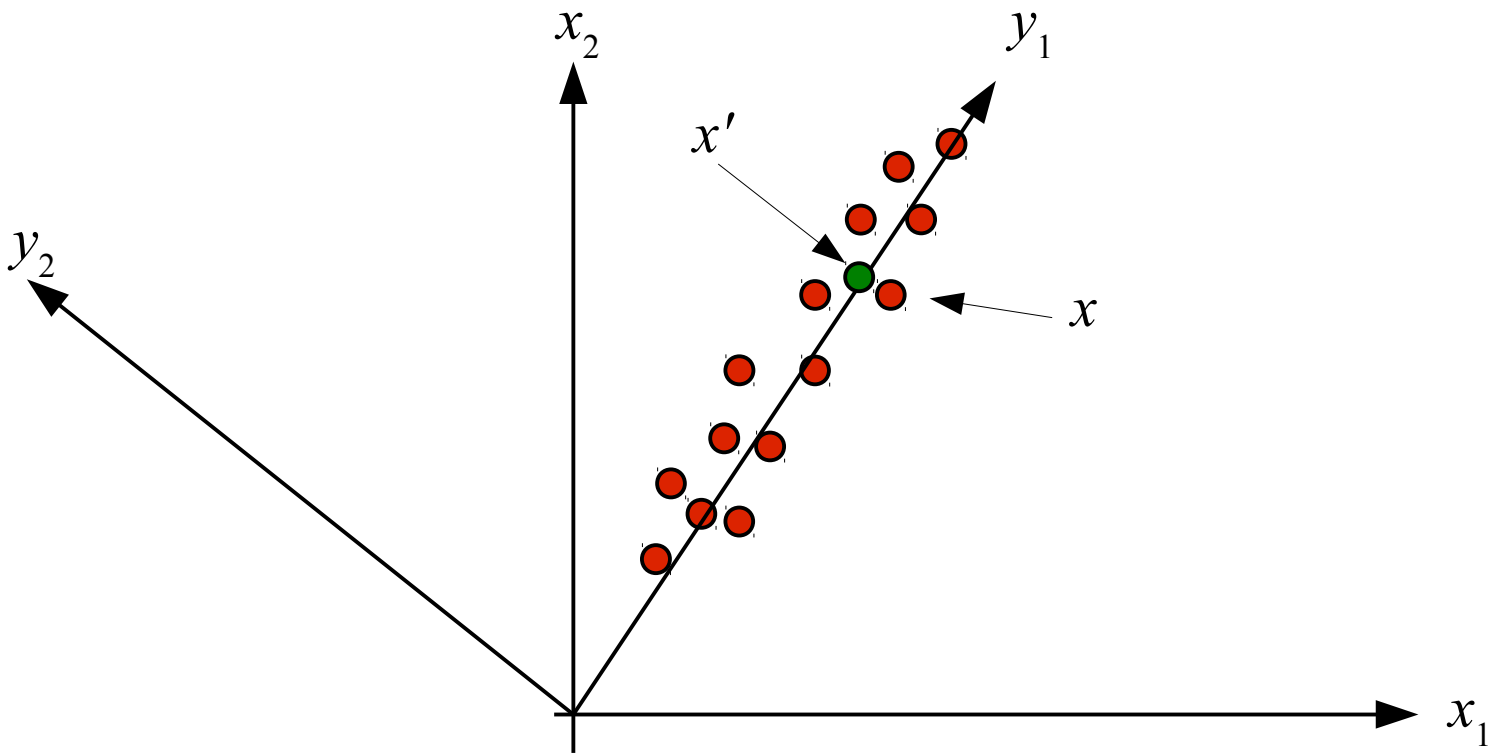
- This is equivalent to projecting the vectors onto the y_1 axis



Example Transform

- This will save a lot of space since we only keep one component instead of two
- The receiver can then reconstruct an approximation x' to the input x using the *inverse transform* T^{-1} of y':

$$x' = T^{-1} y' = T^{-1} \begin{bmatrix} y_1 \\ 0 \end{bmatrix} \approx x$$



Properties of Transforms

- We will consider only transforms that have two properties:
  - Linear
  - Orthonormal

Linear Transforms

- Satisfies the *linearity property*:

$$T(\alpha x+\beta y)=\alpha T(x)+\beta T(y)$$

- Defined by an $N\times N$ matrix A such that:

$$y=Ax$$

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix}=\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1,0} & a_{N-1,1} & \cdots & a_{N-1,N-1} \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

- Can also be written as a *sum*:

$$y_i=\sum_{j=0}^{N-1} a_{i,j}x_j \quad \text{for } i=0,\cdots,N-1$$

Orthonormal Transforms

- Satisfies the *orthonormality property*:

$$A^{-1} = A^T$$

$$A A^T = A^T A = I$$

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1,0} & a_{N-1,1} & \cdots & a_{N-1,N-1} \end{bmatrix}$$

- Which means:
  - Both the rows and columns are *unit vectors*.

$$\sum_j a_{i,j}^2 = 1 \quad \forall i$$

$$\sum_i a_{i,j}^2 = 1 \quad \forall j$$

  - The rows and columns are mutually *orthogonal*.

$$\sum_k a_{i,k} a_{k,j} = 1 \quad \forall i \neq j$$

Why Coefficients

- We have:

$$y = A\,x$$

$$A^{-1} = A^{T}$$

- Then:

$$x = A^{-1}\,y = A^{T}\,y$$

Rows of A

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{1,0} & \cdots & a_{N-1,0} \\ a_{0,1} & a_{1,1} & \cdots & a_{N-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{0,N-1} & a_{1,N-1} & \cdots & a_{N-1,N-1} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix}$$

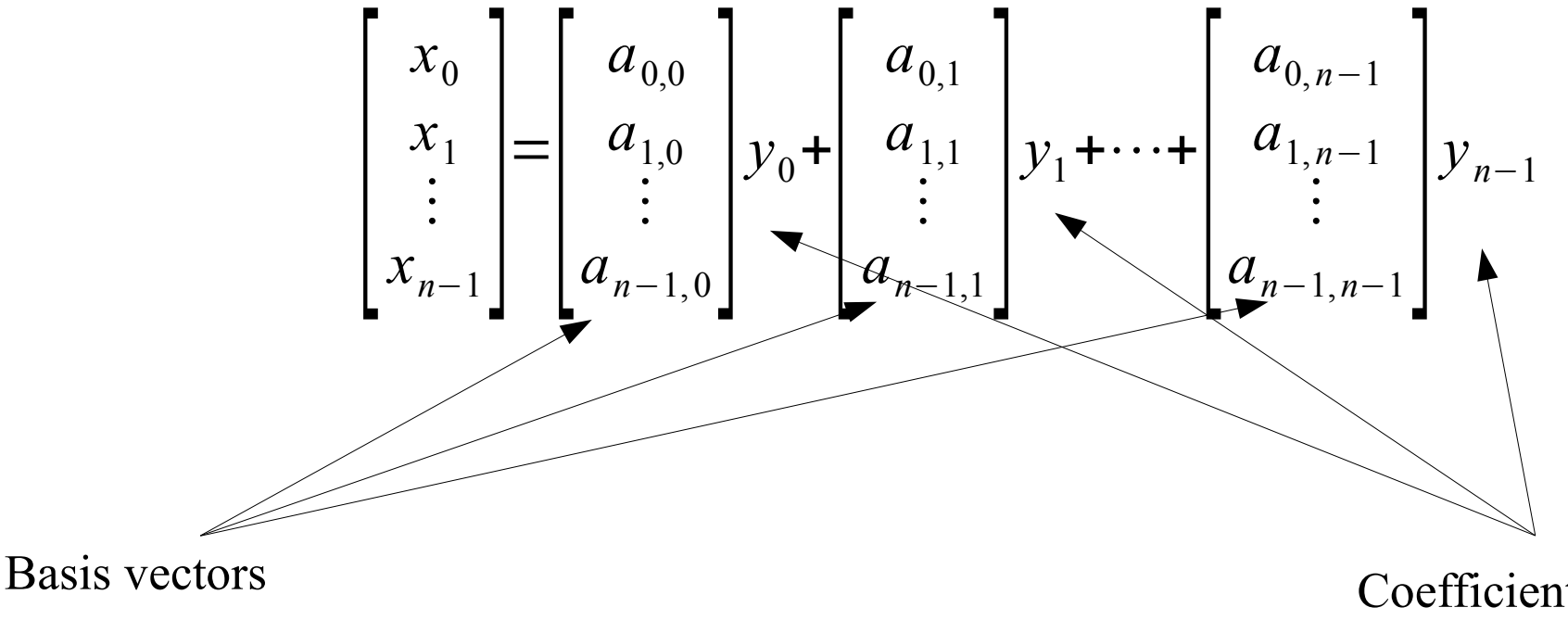
$$= \begin{bmatrix} a_{0,0} \\ a_{1,0} \\ \vdots \\ a_{N-1,0} \end{bmatrix} y_0 + \begin{bmatrix} a_{0,1} \\ a_{1,1} \\ \vdots \\ a_{N-1,1} \end{bmatrix} y_1 + \cdots + \begin{bmatrix} a_{0,N-1} \\ a_{1,N-1} \\ \vdots \\ a_{N-1,N-1} \end{bmatrix} y_{N-1}$$

Basis vectors

Coefficients

Why Coefficients

- We can view transformations as *decomposing* the vector *x* as a *linear combination* of *basis vectors* where the *coefficients* are the coordinates of the transform vector *y* and the *basis vectors* are the *rows* of the transform matrix:



Why Orthonormality

- The *norm* (energy) of the data equals the energy of the *coefficients*:

$$\sum_i y_i^2 = y^T y$$

$$= \left(x^T A^T\right)\left(A x\right)$$

$$= x^T \left(A^T A\right) x$$

$$= x^T x$$

$$= \sum_i x_i^2$$

Why Orthonormality

- This also means the *squared error* in *approximating* the coefficients (e.g. using fewer bits) equals the squared error in the original data:

Let $\epsilon_i = |y_i - y_i'|$

Error in approximating coefficients

$$\sum_i \epsilon_i^2 = \sum_i (y_i - y_i')^2$$

$$= (y - y')^T (y - y')$$

$$= (Ax - Ax')^T (Ax - Ax')$$

$$= (A[x - x'])^T (A[x - x'])$$

$$= [x - x']^T A^T A [x - x']$$

$$= [x - x']^T [x - x']$$

Error in approximating data

$$= \sum_i (x_i - x_i')^2$$

Linear 2D Transforms

- *Linear Transforms* can also work on *2D* inputs such as images or blocks of pixels
- Linear 2D transforms for an input block X of size $N \times N$ are defined as

$$y_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_{i,j,k,l} x_{i,j}$$

  where Y is the transformed coefficients of size $N \times N$ and $A_{k,l}$ defines the transform
- We will only consider transforms that are *orthonormal*, and *separable*

Separable 2D Transforms

- Most 2D transforms used are *separable* i.e. they can be decomposed into *two* 1D transforms by first transforming along one dimension (e.g. the *rows*) using the corresponding *1D transform*, then transforming along the second dimension (e.g. the *columns*):

$$y_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_{i,j,k,l} x_{i,j}$$

1D transform of the rows

$$= \sum_{i=0}^{N-1} a_{k,i} \sum_{j=0}^{N-1} a_{l,j} x_{i,j}$$

1D transform of the columns

$$= \sum_{i=0}^{N-1} a_{k,i} g_{i,l}$$

Separable 2D Transforms

$$y_{k,l} = \sum_{i=0}^{N-1} a_{k,i} \sum_{j=0}^{N-1} a_{l,j} x_{i,j}$$

- In *matrix* form, this can be represented by:

$$Y = A\, X\, A^T$$

where X is the input block of size $N \times N$, Y is the transformed coefficients of size $N \times N$ and A is an $N \times N$ matrix defining the corresponding 1D transform

- Since the transform is also *orthonormal*, we have

$$A^{-1} = A^T$$

and the *inverse transform* is:

$$X = A^T\, Y\, A$$

2D Basis Matrices

- We can also view 2D transforms as a decomposition of the input *matrix* (block) X as a *linear combination* of *basis matrices* formed by taking the *outer product* of the rows of the transform matrix A where the *coefficients* are the transformed matrix Y:

$$X = A^T Y A$$

$$= \begin{bmatrix} a_{0,0} & \cdots & a_{N-1,0} \\ a_{0,1} & \cdots & a_{N-1,1} \\ \vdots & \ddots & \vdots \\ a_{0,N-1} & \cdots & a_{N-1,N-1} \end{bmatrix} \begin{bmatrix} y_{0,0} & \cdots & y_{N-1,0} \\ y_{0,1} & \cdots & y_{N-1,1} \\ \vdots & \ddots & \vdots \\ y_{0,N-1} & \cdots & y_{N-1,N-1} \end{bmatrix} \begin{bmatrix} a_{0,0} & \cdots & a_{0,N-1} \\ a_{1,0} & \cdots & a_{1,N-1} \\ \vdots & \ddots & \vdots \\ a_{N-1,0} & \cdots & a_{N-1,N-1} \end{bmatrix}$$

$$= \sum_i \sum_j y_{i,j} \begin{bmatrix} a_{i,0} \\ a_{i,1} \\ \vdots \\ a_{i,N-1} \end{bmatrix} \begin{bmatrix} a_{j,0} & a_{j,1} & \cdots & a_{j,N-1} \end{bmatrix}$$

Outer product of rows of A

2D Basis Matrices

$$X = \sum_i \sum_j y_{i,j} \begin{bmatrix} a_{i,0} \\ a_{i,1} \\ \vdots \\ a_{i,N-1} \end{bmatrix} \begin{bmatrix} a_{j,0} & a_{j,1} & \cdots & a_{j,N-1} \end{bmatrix}$$

$$= \sum_i \sum_j y_{i,j} \alpha_{i,j}$$

where $\alpha_{i,j}$ define the basis matrices and defined as the outer product of the i^{th} row of and the j^{th} row of A:

$$\alpha_{i,j} = \begin{bmatrix} a_{i,0} \\ a_{i,1} \\ \vdots \\ a_{i,N-1} \end{bmatrix} \begin{bmatrix} a_{j,0} & a_{j,1} & \cdots & a_{j,N-1} \end{bmatrix}$$

$$= \begin{bmatrix} a_{i,0}a_{j,0} & a_{i,0}a_{j,1} & \cdots & a_{i,0}a_{j,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,N-1}a_{j,0} & a_{i,N-1}a_{j,1} & \cdots & a_{i,N-1}a_{j,N-1} \end{bmatrix}$$

Discrete Cosine Transform (DCT)

- The most widely used transform in *multimedia compression*
- Linear, orthonormal, and separable 2D transform
- Gets the name from the fact that its rows are obtained from *cosine* functions. The 1D DCT matrix A is defined as:

First row

$$a_{i,j}=\begin{cases}\sqrt{\frac{1}{N}} & \text{for } i=0 \text{ and } j=0,\cdots,N-1\\ \sqrt{\frac{2}{N}}\cos\frac{(2j+1)i\pi}{2N} & \text{for } i>0 \text{ and } j=0,\cdots,N-1\end{cases}$$

The rest of the rows

Discrete Cosine Transform (DCT)

- The 1D DCT can also be represented as:

$$y_i = \sqrt{\frac{1}{N}} x_0 + \sum_{j=1}^{N-1} \left(\sqrt{\frac{2}{N}} \cos \frac{(2j+1) i \pi}{2N} \right) x_j$$

$$y_i = \sum_{j=0}^{N-1} a_{i,j} x_j$$

$$a_{i,j} = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } i = 0 \text{ and } j = 0, \cdots, N-1 \\ \sqrt{\frac{2}{N}} \cos \frac{(2j+1) i \pi}{2N} & \text{for } i > 0 \text{ and } j = 0, \cdots, N-1 \end{cases}$$

Discrete Cosine Transform (DCT)

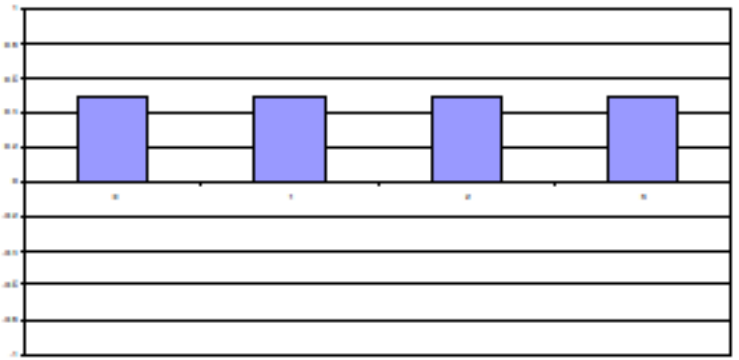
$$a_{i,j} = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } i = 0 \text{ and } j = 0, \cdots, N-1 \\ \sqrt{\frac{2}{N}} \cos \frac{(2j+1) i \pi}{2N} & \text{for } i > 0 \text{ and } j = 0, \cdots, N-1 \end{cases}$$

- It is also *orthonormal*: $A^{-1} = A^T$
- The *forward transform*: $y = A x$
- The *inverse transform*: $x = A^T y$

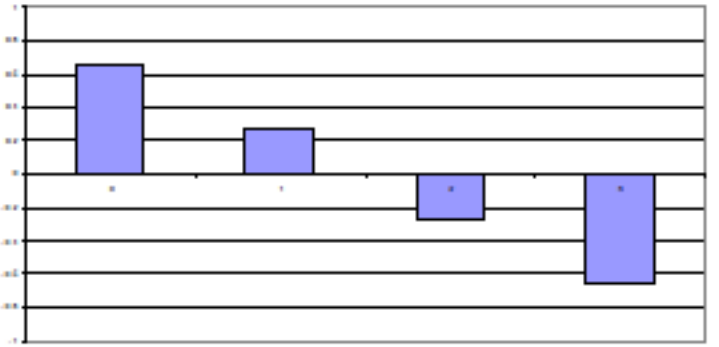
Example: N = 4

$$A = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.653 & 0.271 & -0.271 & -0.653 \\ 0.5 & -0.5 & -0.5 & 0.5 \\ 0.271 & -0.653 & 0.653 & -0.271 \end{bmatrix}$$

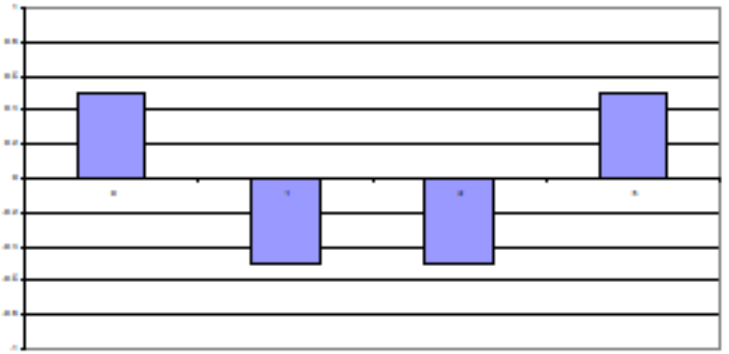
Basis Vectors



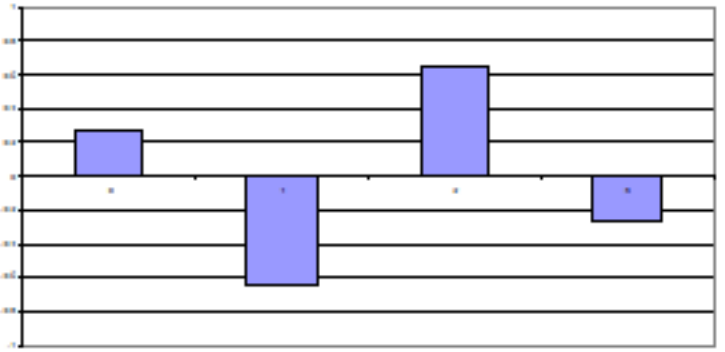
row 0



row 1



row 2

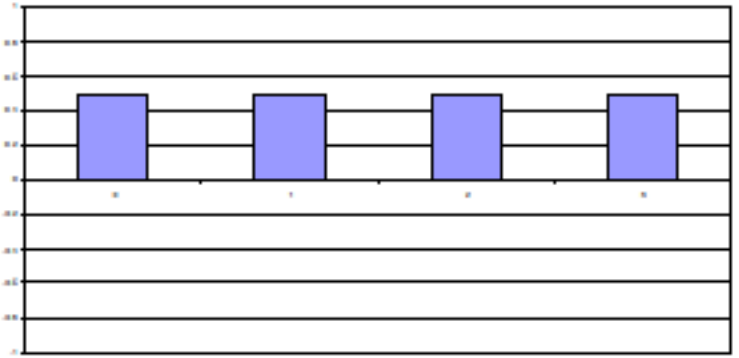


row 3

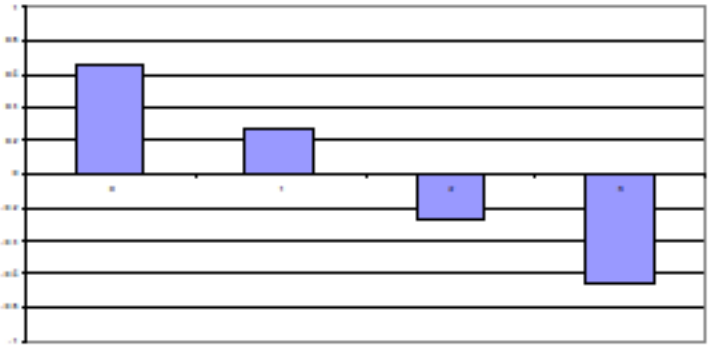
Example: N = 4

- The *first* basis vector is called the DC (Direct Current) component, since it represents a *constant* function
- The rest of the basis vectors represent functions at increasing *spatial frequency* i.e. rates of change in the values. They are called the AC (Alternating Current) components.

0 period



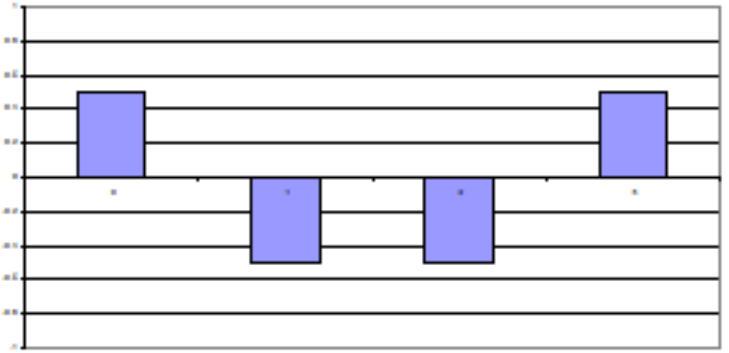
row 0



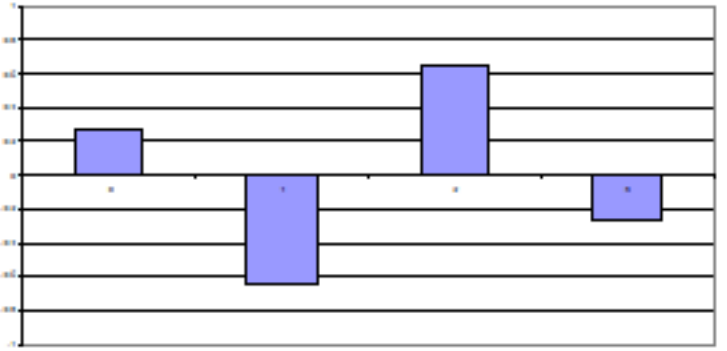
½ a period

row 1

1 period



row 2



1½ period

row 3

Example: N = 4

- We can decompose any 4-dimensional vector in terms of the basis vectors:

$$x = A^T y$$

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} y_0 + \begin{bmatrix} 0.653 \\ 0.271 \\ -0.271 \\ -0.653 \end{bmatrix} y_1 + \begin{bmatrix} 0.5 \\ -0.5 \\ -0.5 \\ 0.5 \end{bmatrix} y_2 + \begin{bmatrix} 0.271 \\ -0.653 \\ 0.653 \\ -0.271 \end{bmatrix} y_3$$

DC coefficient

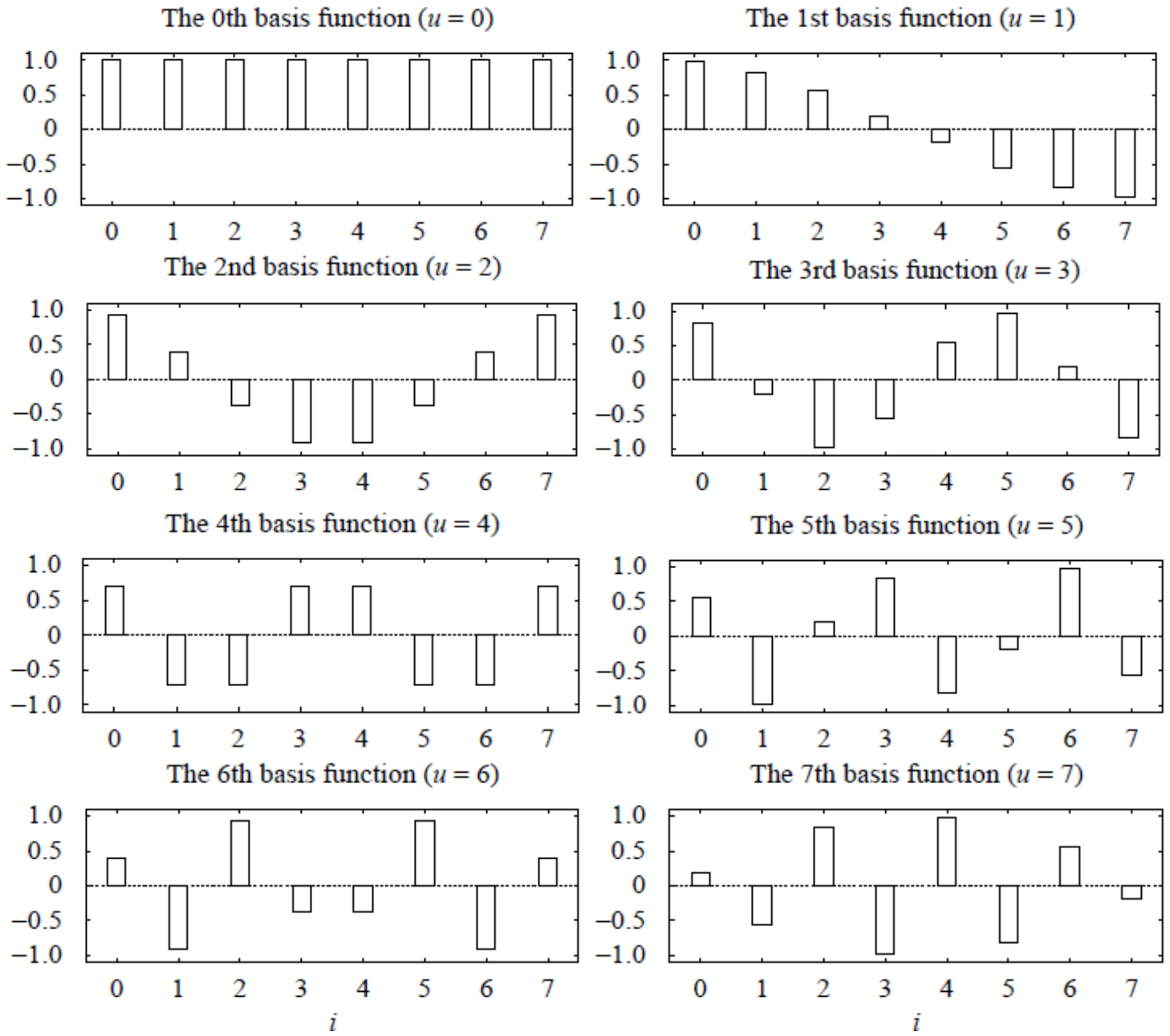
AC coefficients

Example: N = 8

$$A = \begin{bmatrix}
0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 & 0.353553 \\
0.490393 & 0.415735 & 0.277785 & 0.097545 & -0.097545 & -0.277785 & -0.415735 & -0.490393 \\
0.461940 & 0.191342 & -0.191342 & -0.461940 & -0.461940 & -0.191342 & 0.191342 & 0.461940 \\
0.415735 & -0.097545 & -0.490393 & -0.277785 & 0.277785 & 0.490393 & 0.097545 & -0.415735 \\
0.353553 & -0.353553 & -0.353553 & 0.353553 & 0.353553 & -0.353553 & -0.353553 & 0.353553 \\
0.277785 & -0.490393 & 0.097545 & 0.415735 & -0.415735 & -0.097545 & 0.490393 & -0.277785 \\
0.191342 & -0.461940 & 0.461940 & -0.191342 & -0.191342 & 0.461940 & -0.461940 & 0.191342 \\
0.097545 & -0.277785 & 0.415735 & -0.490393 & 0.490393 & -0.415735 & 0.277785 & -0.097545
\end{bmatrix}$$

Example: N = 8

Basis Vectors



Example: N = 8

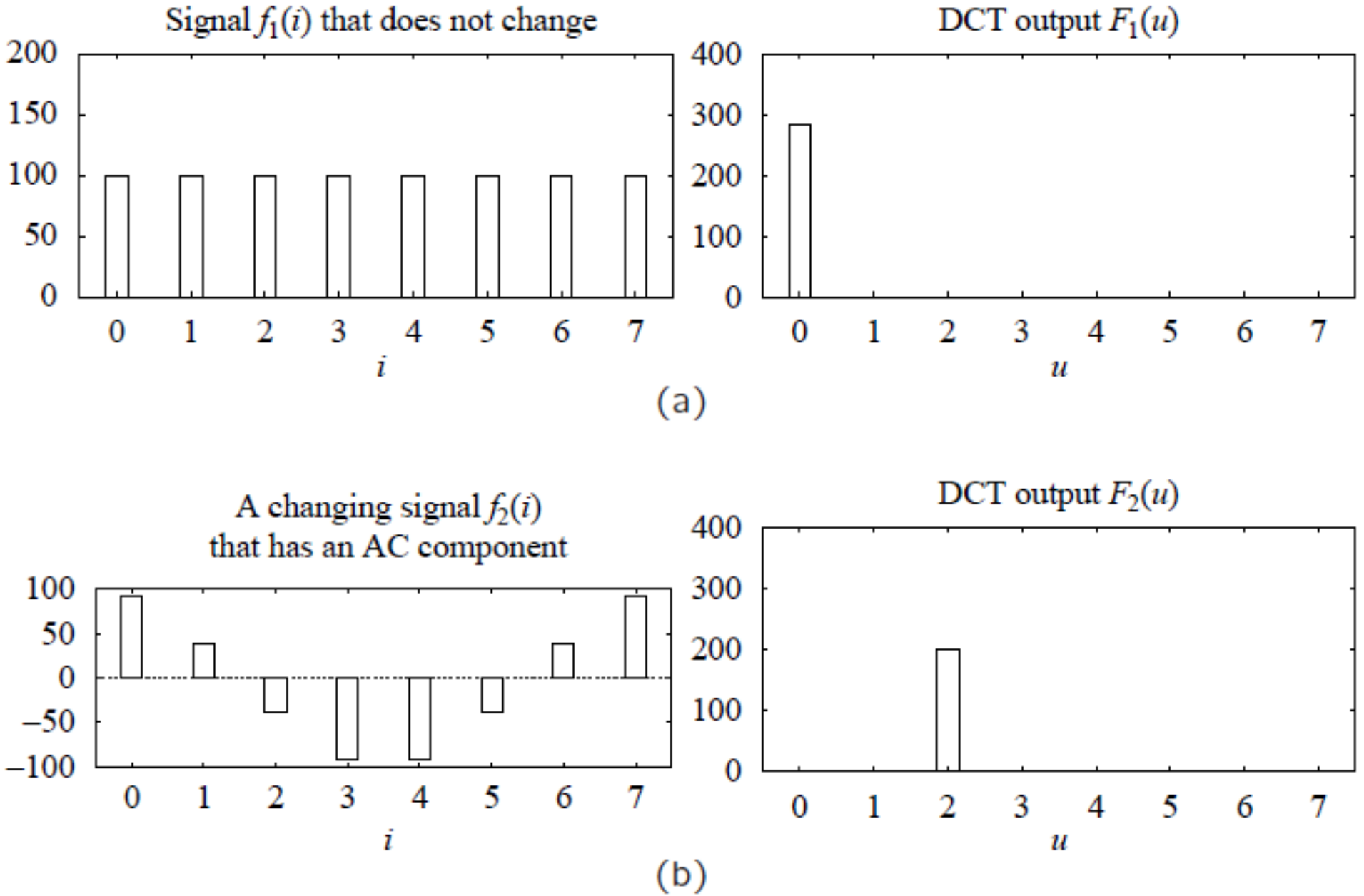


Fig. 8.7: Examples of 1D Discrete Cosine Transform: (a) A DC signal $f_1(i)$, (b) An AC signal $f_2(i)$.

Example: N = 8

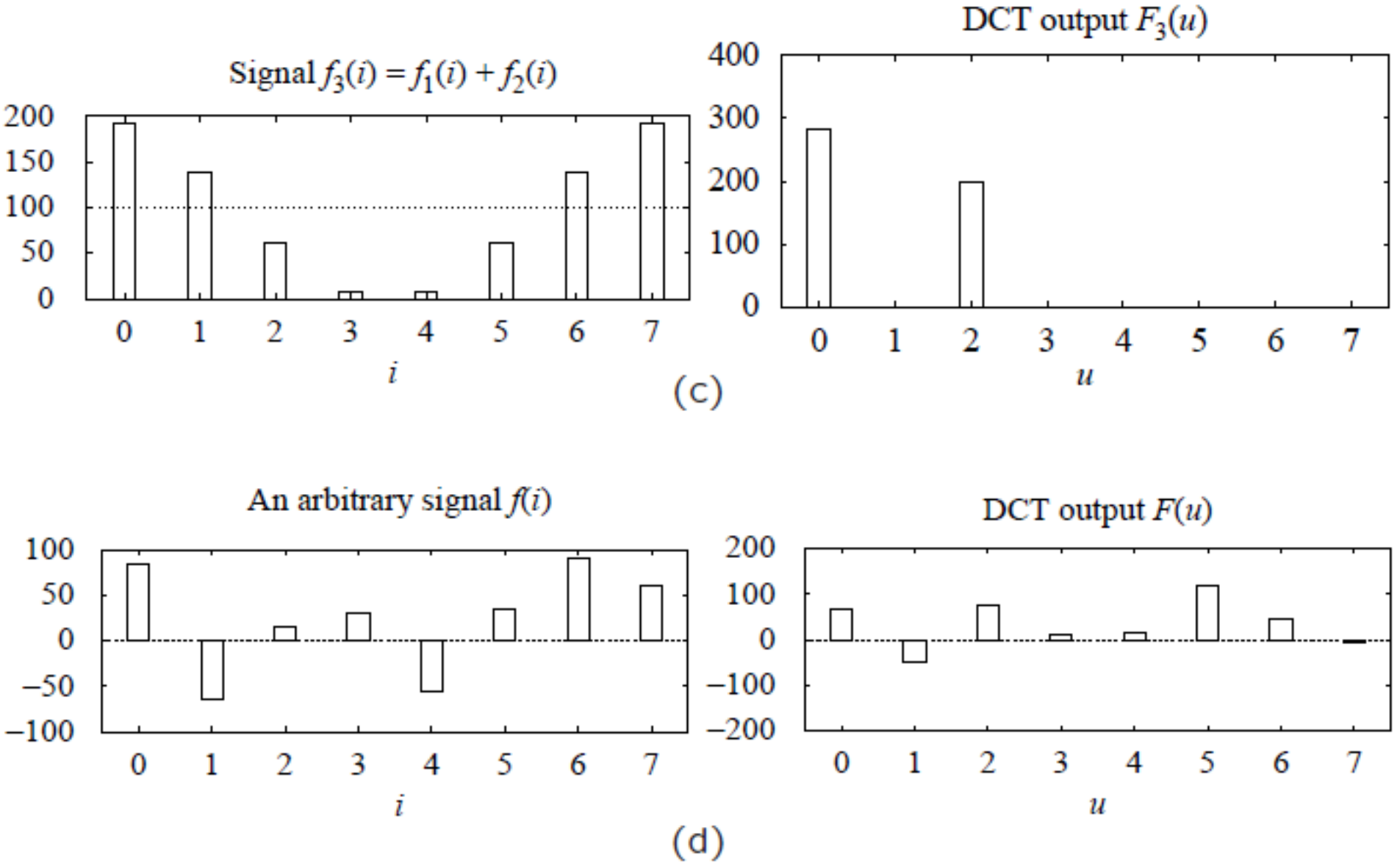


Fig. 8.7 (cont'd): Examples of 1D Discrete Cosine Transform: (c) $f_3(i) = f_1(i) + f_2(i)$, and (d) an arbitrary signal $f(i)$.

2D DCT

- The 1D DCT can be extended to 2D using the fact that the 2D DCT is a *separable* transform:

$$y_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(\beta(k)\beta(l) \cos\frac{(2i+1)k\pi}{2N} \cos\frac{(2j+1)l\pi}{2N} \right) x_{i,j}$$

$$y_{k,l} = \beta(k)\beta(l) \sum_{i=0}^{N-1} \cos\frac{(2i+1)k\pi}{2N} \left(\sum_{j=0}^{N-1} \cos\frac{(2j+1)l\pi}{2N} x_{i,j} \right)$$

$$\beta(i) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } i = 0 \\ \sqrt{\frac{2}{N}} & \text{for } i > 0 \end{cases}$$

2D DCT

- This can be written in *matrix form* using the same 1D DCT matrix A for $N \times N$ matrices X and Y:

$$Y = A\, X\, A^T$$

- The *inverse transform* is:

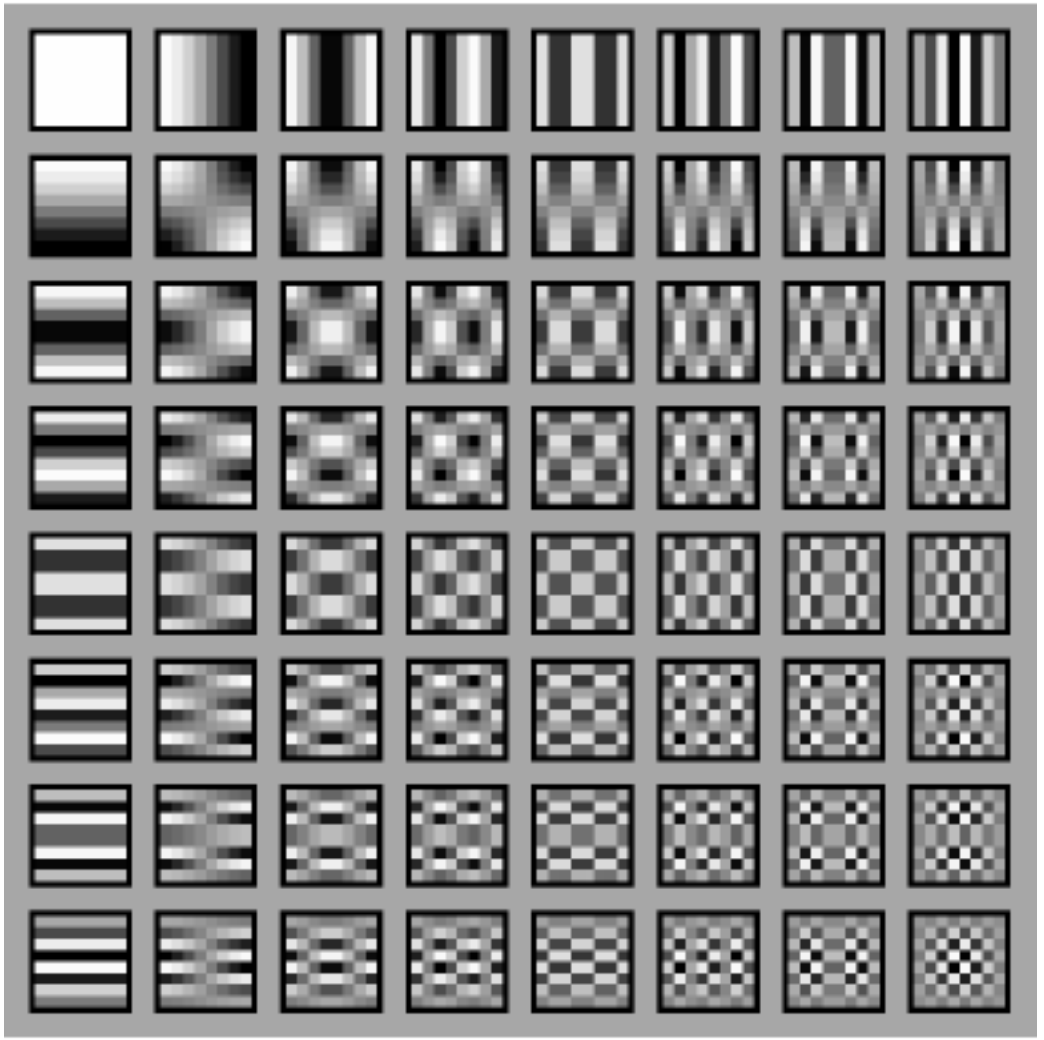
$$X = A^T\, Y\, A$$

- The *basis matrices* are the *outer products* of the rows of A:

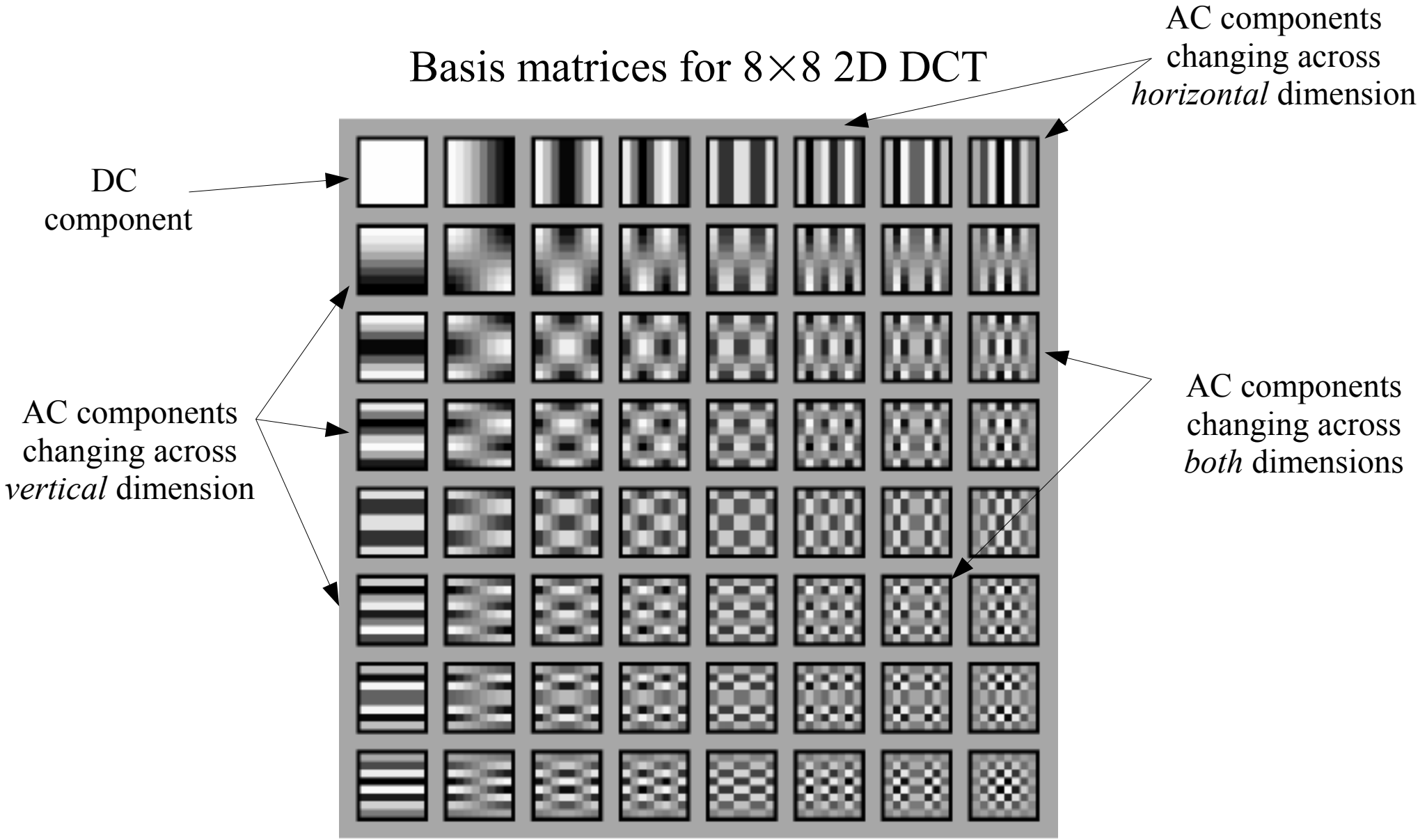
$$\alpha_{i,j} = \begin{bmatrix} a_{i,0} \\ a_{i,1} \\ \vdots \\ a_{i,N-1} \end{bmatrix} \begin{bmatrix} a_{j,0} & a_{j,1} & \cdots & a_{j,N-1} \end{bmatrix}$$

2D DCT Basis Matrices

Basis matrices for 8×8 2D DCT



2D DCT Basis Matrices



Importance of Coefficients

- The *coefficients* of the 2D DCT transform are not all equally important
  - The DC coefficient is the most important
  - The AC coefficients are less important, especially those with higher *frequency*
- More bits are allocated to important coefficients and fewer are assigned to less important ones

Importance of Coefficients

- Example bit allocation for 8×8 DCT blocks:

8	7	5	3	2	1	0	0
7	5	3	2	1	0	0	0
5	3	2	1	0	0	0	0
3	2	1	0	0	0	0	0
2	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

- Compression: 65 bits for 64 pixels instead of 512 bits i.e. 1.02 bit/pixel
- Compression ratio: 7.88

Quantizing Coefficients

- Usually *uniform scalar quantization* is used
- For $N \times N$ blocks, construct an $N \times N$ matrix Q called *Quantization Table*
- The element $q_{i,j}$ specifies the *step* of the *midtread* quantizer for the *coefficient* $y_{i,j}$
- *Larger* values for $q_{i,j}$ indices *fewer* bits

Quantizing Coefficients

- *Encoder*: each *coefficient* $y_{i,j}$ is encoded as the *label* $s_{i,j}$:

$$s_{i,j} = \left\lfloor \frac{y_{i,j}}{q_{i,j}} + 0.5 \right\rfloor$$

- *Decoder*: $y'_{i,j} = s_{i,j} q_{i,j}$

- Example:
  - $y = 54.2, q = 24 \rightarrow s = \text{floor}(2.78) = 2 \rightarrow y' = 2 * 24 = 48$
  - $y = 54.2, q = 12 \rightarrow s = \text{floor}(5.01) = 5 \rightarrow y' = 5 * 12 = 60$
  - $y = 54.2, q = 6 \rightarrow s = \text{floor}(9.03) = 9 \rightarrow y' = 9 * 6 = 54$

Example

Example 8×8 Quantization Table

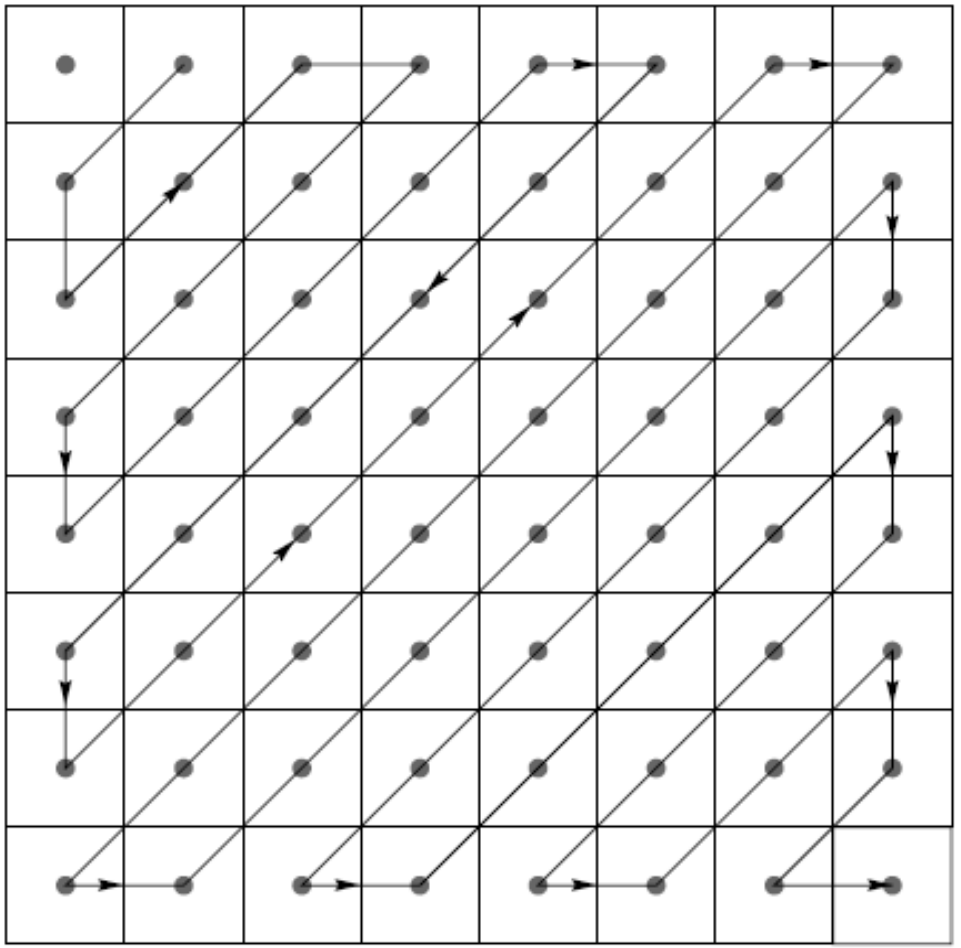
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	33	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Smaller q means more bits allocated

Larger q means less bits allocated

Zig-Zag Coding

- Since most of the high frequency AC coefficients have larger quantization steps, most of them will be *zeros*
- Coding the coefficients in a *zig-zag* order will result in a long run of zeros at the tail of the sequence



Applications of DCT

- Image Coding
  - JPEG
- Video Coding
  - MPEG
  - H.263, H.264
- Audio Coding
  - MP3

Other Transforms

- Discrete Sine Transform
- Discrete Walsh-Hadamard Transform
- Karhunen-Loeve Transform

Differential Coding

- As we saw for uniform quantizers, for the same number of quantization *levels* M, the *step* Δ of the quantizer depends on *variance* of the source input x
- For *uniformly distributed* values, the *variance* depends on the *dynamic range* $x_{\max} - x_{\min}$ i.e. the larger dynamic range the larger the step

$$\Delta = \frac{X_{\max} - X_{\min}}{M}$$

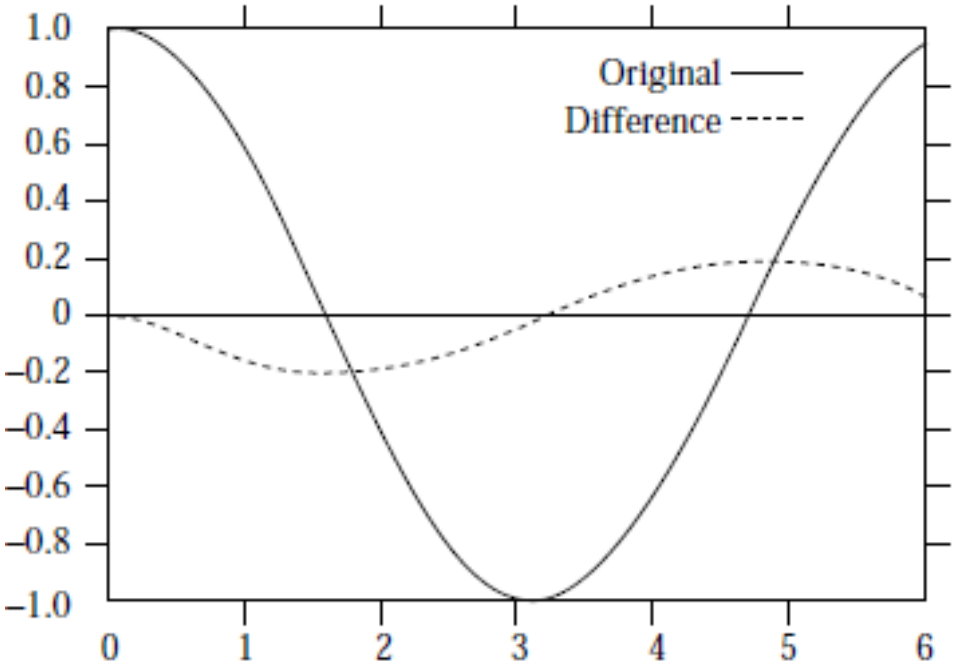
- The step also determines the amount of *quantization noise* incurred i.e. the larger the step the larger the noise, since the maximum error in any interval is: $\Delta / 2$

Differential Coding: Idea

- If the source output sequence $\{x_n\}$ (quantizer input) does not change much, the *difference* sequence $\{d_n = x_n - x_{n-1}\}$:
  - has *lower* variance and dynamic range than $\{x_n\}$
  - has a distribution that is highly peaked around 0

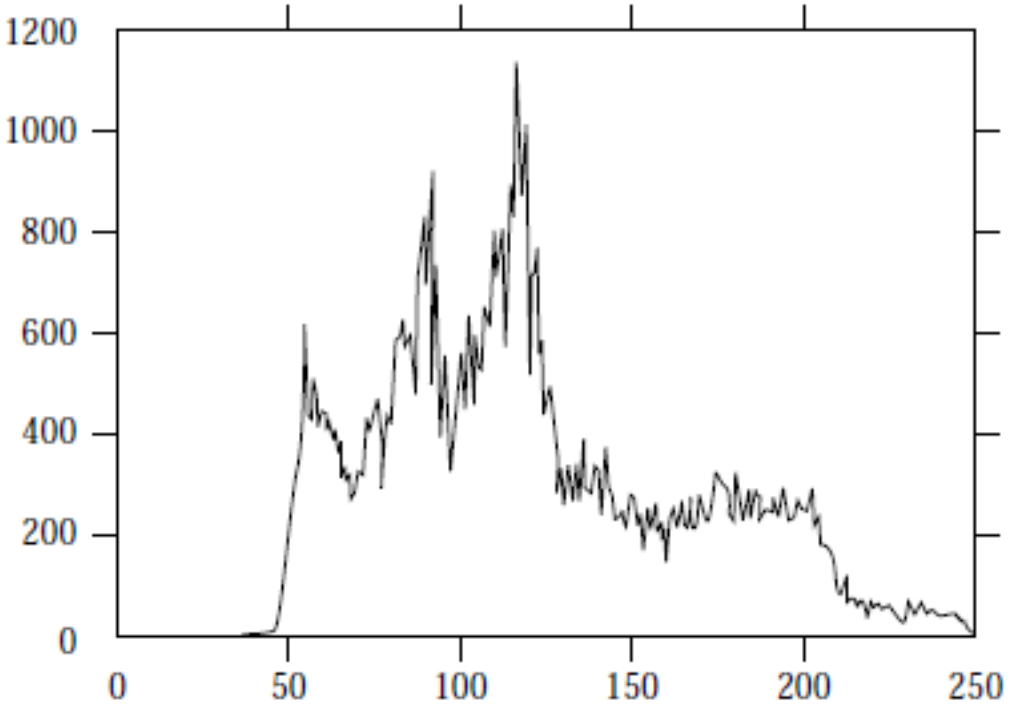
Example

- Consider the cosine signal below that was sampled at 30 samples per cycle
- The dynamic range of the sample signal is [–1, 1]
  - For 4-level quantizer, the step $\Delta = 0.5$
  - Errors lie in the range [–0.25, 0.25]
- If we take the difference between each sample and its predecessor, the dynamic range is [–0.2, 0.2]
  - For 4-level quantizer, the step $\Delta = 0.1$
  - Errors lie in the range [–0.05, 0.05]
- For the same number of bits, coding the difference signal instead of the signal itself we can reduce the error

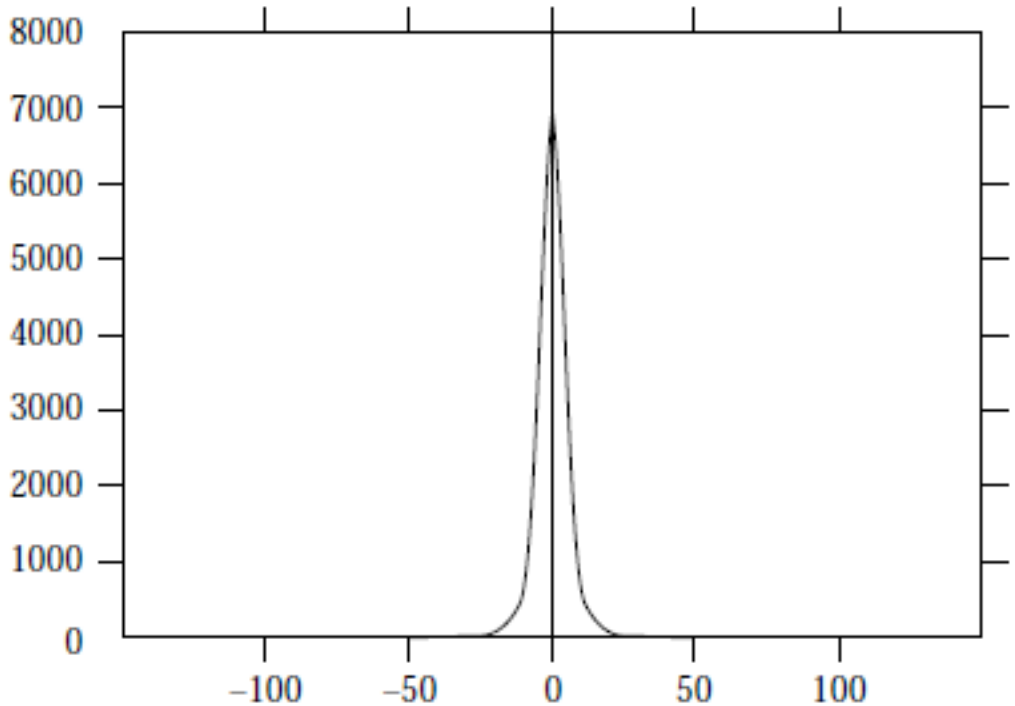


Example

- The *histogram* of the *pixel values* include almost all values in the range [0, 255]
- More than 99% of *pixel differences* lie in the range [–31, 31]
- We can code the image using 6 bits per difference, instead of 8 bits per pixel



Histogram pixels of the Sinan image



Histogram pixel differences

Differential Pulse Code Modulation (DPCM)

- The basic algorithm for *differential coding*
- Used in many applications, especially in speech and audio compression
- Used as a building block in other compression standards too

DPCM 1

- *Encoder*:
  - Starts with a sequence $\{x_n\}$ that needs to be coded
  - Forms the *difference* sequence $\{d_n = x_n - x_{n-1}\}$
  - This sequence is then *quantized* to obtain $\{\hat{d}_n = d_n + q_n\}$ where q_n is the *quantization error*
- *Decoder*:
  - Forms the *reconstructed* sequence $\{\hat{x}_n = \hat{x}_{n-1} + \hat{d}_n\}$

Example

- Encoder:
  - $\{x_n\}=\{6.2, 9.7, 13.2, 5.9, 8, 7.4, 4.2, 1.8\}$
  - $\{d_n\}=\{6.2, 3.6, 3.5, -7.3, 2.1, -0.6, -3.2, -2.4\}$
  - Using a 7-level quantizer:
    $\{\hat{d}_n\}=\{6, 4, 4, -6, 2, 0, -4, -2\}$

- Decoder:
  - $\{\hat{x}_n\}=\{6, 10, 14, 8, 10, 10, 6, 4\}$
  - Reconstruction error: $\{0.2, -0.3, -0.8, -2.1, -2, -2.6, -1.8, 2.2\}$

- Note that the *magnitude* of the reconstruction error *increases* over time

DPCM 1 Problem

- The *quantization errors* accumulate at the *decoder* over time
- Assume both the encoder and decoder start with x_0:

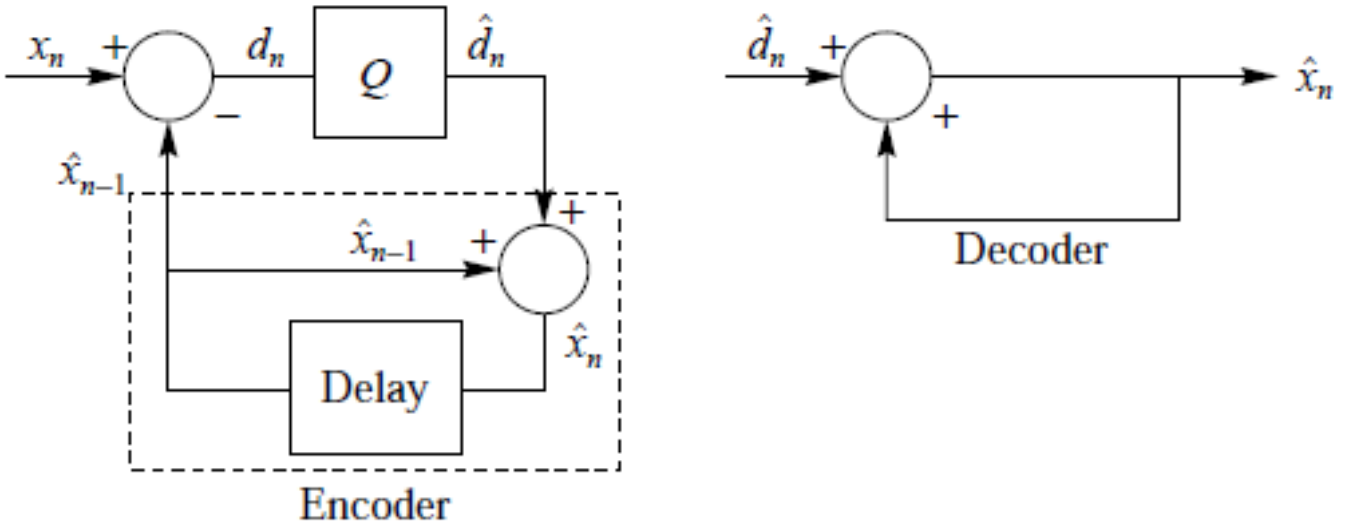
$$d_1 = x_1 - x_0$$
$$\hat{d}_1 = d_1 + q_1$$
$$\hat{x}_1 = x_0 + \hat{d}_1 = x_0 + d_1 + q_1 = x_1 + q_1$$

$$d_2 = x_2 - x_1$$
$$\hat{d}_2 = d_2 + q_2$$
$$\hat{x}_2 = \hat{x}_1 + \hat{d}_2 = x_1 + q_1 + d_2 + q_2 = x_2 + q_1 + q_2$$
$$\vdots$$
$$\hat{x}_n = x_n + \sum_{k=1}^{n} q_k$$

- A better solution would be for the encoder to use the *predicted* values for computing the differences

DPCM 2

- *Encoder*:
  - Starts with a sequence $\{x_n\}$ and $\hat{x}_0 = x_0$
  - Forms the *difference* sequence $\{d_n = x_n - \hat{x}_{n-1}\}$
  - This sequence is then *quantized* to obtain $\{\hat{d}_n = d_n + q_n\}$ where q_n is the *quantization error*
- *Decoder*:
  - Forms the *reconstructed* sequence $\{\hat{x}_n = \hat{x}_{n-1} + \hat{d}_n\}$



Problem Solved

- The *quantization errors* do *not* accumulate anymore
- Assume both the encoder and decoder start with x_0:

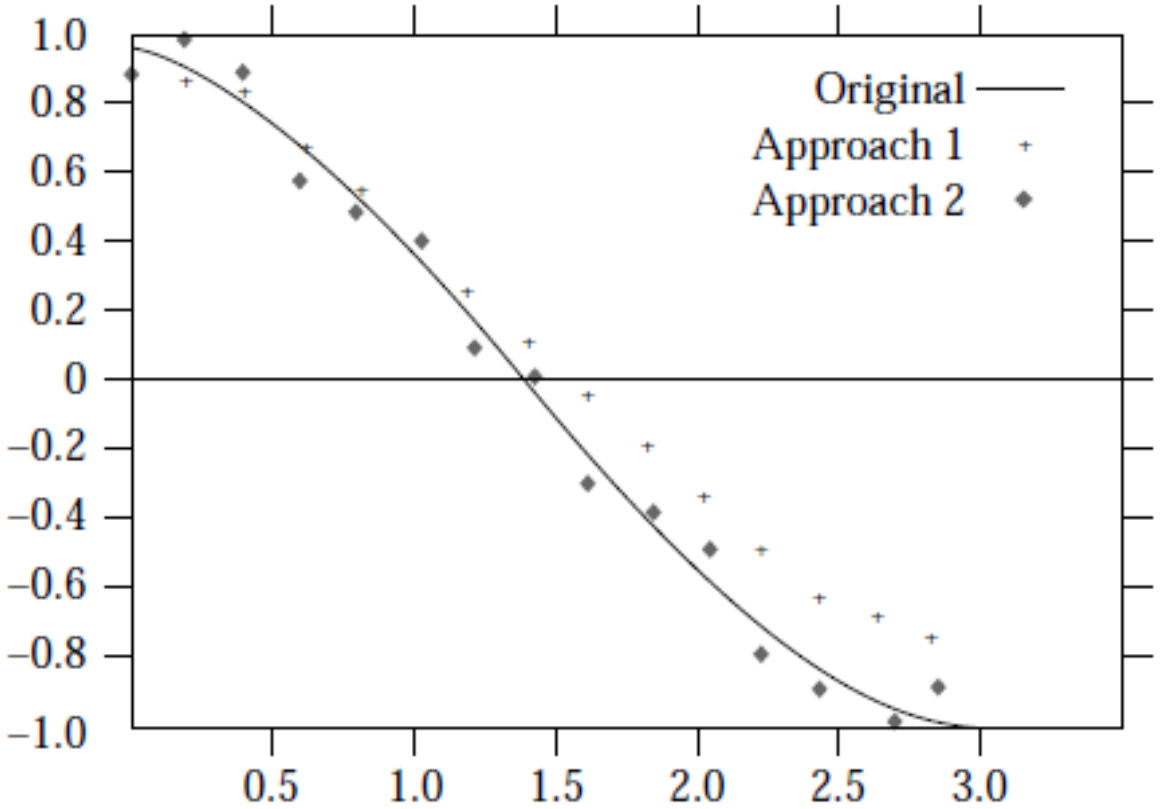
$$d_1 = x_1 - x_0$$
$$\hat{d}_1 = d_1 + q_1$$
$$\hat{x}_1 = x_0 + \hat{d}_1 = x_0 + d_1 + q_1 = x_1 + q_1$$

$$d_2 = x_2 - \hat{x}_1$$
$$\hat{d}_2 = d_2 + q_2$$
$$\hat{x}_2 = \hat{x}_1 + \hat{d}_2 = \hat{x}_1 + d_2 + q_2 = x_2 + q_2$$
$$\vdots$$
$$\hat{x}_n = x_n + q_n$$

Example

- Approach 1: $\{d_n = x_n - x_{n-1}\}$
- Approach 2: $\{d_n = x_n - \hat{x}_{n-1}\}$

- Second approach provides *lower* errors than first



DPCM 3

- In general, we want the difference values d_n to be small i.e. we want $\hat{x}_{n-1}$ to be as close as possible to x_n

$$\{d_n = x_n - \hat{x}_{n-1}\}$$

- For this to happen, we usually use a number of past values of the sequence $\{\hat{x}_n\}$ to *predict* the value x_n

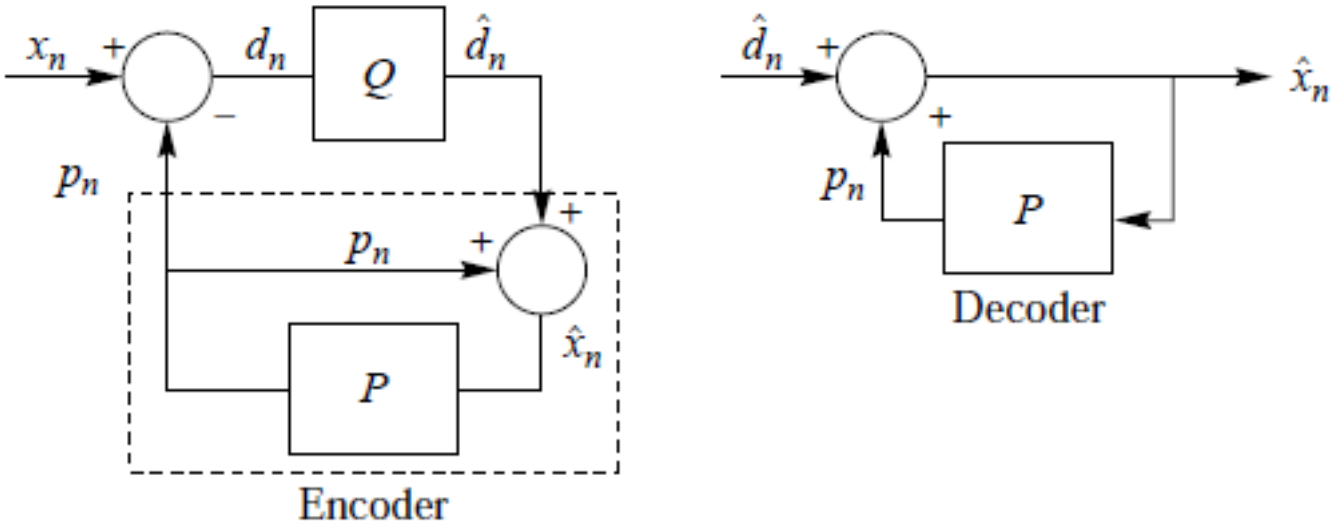
$$p_n = f(\hat{x}_{n-1}, \hat{x}_{n-2}, \cdots, \hat{x}_0)$$

and then use that to compute the *differences*:

$$\{d_n = x_n - p_n\}$$

DPCM 3

- *Encoder*:
  - Starts with a sequence $\{x_n\}$ and $p_0 = x_0$
  - Forms the *difference* sequence $\{d_n = x_n - p_n\}$ where the prediction p_n is $p_n = f(\hat{x}_{n-1}, \hat{x}_{n-2}, \cdots, \hat{x}_0)$
  - This sequence is then *quantized* to obtain $\{\hat{d}_n = d_n + q_n\}$ where q_n is the *quantization error*
- *Decoder*:
  - Forms the *reconstructed* sequence $\{\hat{x}_n = \hat{x}_{n-1} + \hat{d}_n\}$



Prediction

- The prediction is done in a way to *minimize* the *variance* of the difference sequence
- If we assume the *predictor* is a *linear* function of the past predicted values i.e.

$$p_n = \sum_{i=1}^{N} a_i \hat{x}_{n-i}$$

we can find the coefficients that minimize the variance using Wiener-Hopf equations

Example: Image Coding

- Using a simple pixel predictor

$$p[j,k] = \begin{cases} \hat{x}[j,k-1] & \text{for } k > 0 \\ \hat{x}[j-1,k] & \text{for } k = 0 \text{ and } j > 0 \\ 128 & \text{for } j = 0 \text{ and } k = 0 \end{cases}$$

  together with a 4-level uniform quantizer and Arithmetic Coding of the output

- Average bit rate of 1 bit/pixel
- Compare to JPEG for the same rate

Example: Image Coding

- DPCM image worse than JPEG at the same rate
- SNR = 22.3 dB compared to 32.5 dB for JPEG!

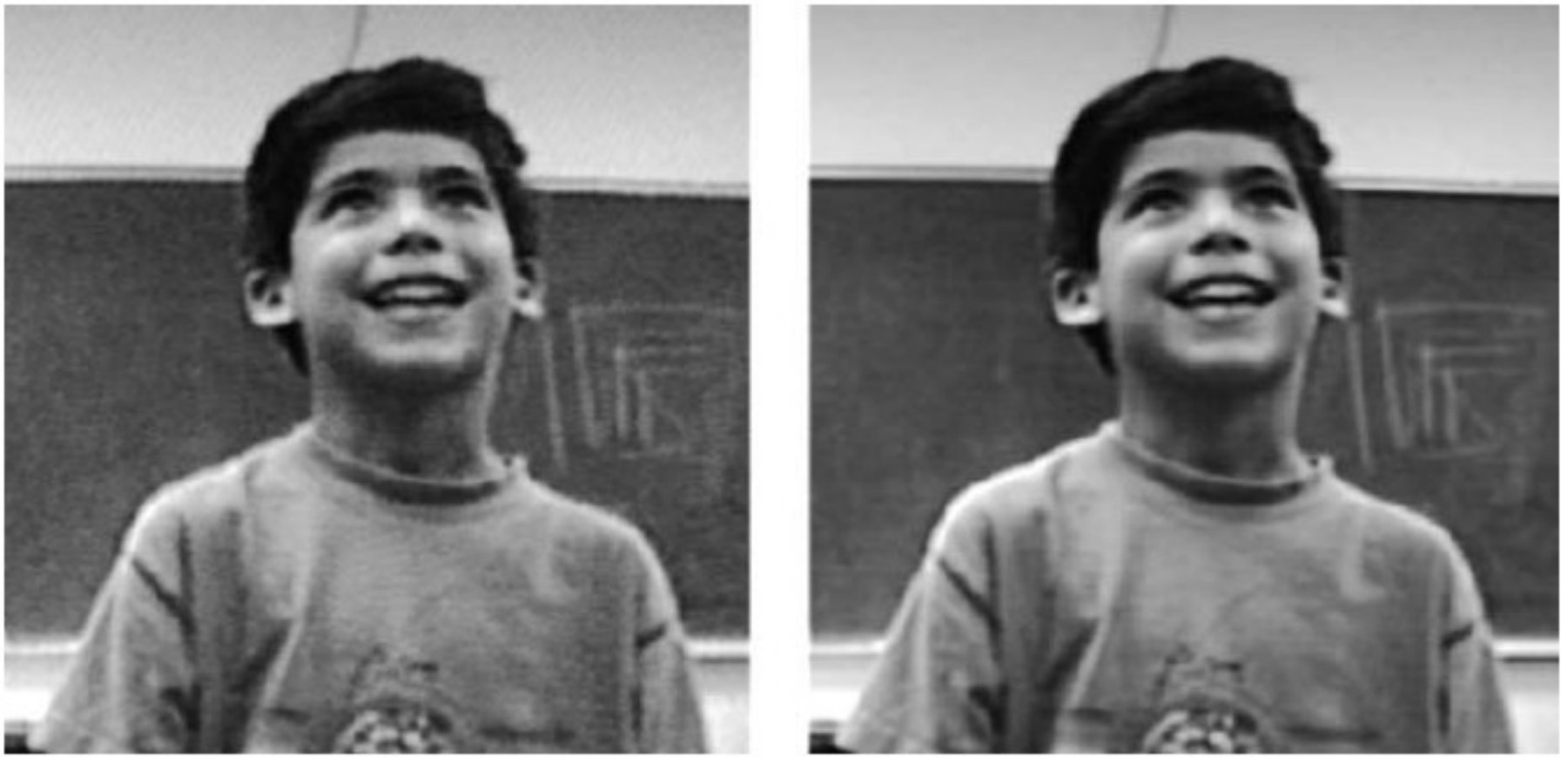


FIGURE 11. 18 **Left: Reconstructed image using differential encoding at 1 bit per pixel. Right: Reconstructed image using JPEG at 1 bit per pixel.**

Example: Image Coding

- A more advanced predictor: compute the values

$$p_1 = 0.5 \times \hat{x}[j-1,k] + 0.5 \times \hat{x}[j,k-1]$$
$$p_2 = 0.5 \times \hat{x}[j-1,k-1] + 0.5 \times \hat{x}[j,k-1]$$
$$p_3 = 0.5 \times \hat{x}[j-1,k-1] + 0.5 \times \hat{x}[j-1,k]$$

  and choose the prediction as $p[j,k] = \text{median}\{p_1, p_2, p_3\}$

- Average bit rate of 1 bit/pixel
- Compare to JPEG for the same rate

Example: Image Coding

- DPCM image still worse, but much better than before
- SNR = 29.2 dB (22.3 previously) compared to 32.5 dB for JPEG!
- Differential Coding can be very useful with good prediction models!!

FIGURE 11. 19 **Left: Reconstructed image using differential encoding at 1 bit per pixel using median predictor and recursively indexed quantizer. Right: Reconstructed image using JPEG at 1 bit per pixel.**

Recap

- Transform Coding
  - Transform Properties
  - 1D Transforms
  - 2D Transforms
  - DCT Transform
- Differential Coding

- Next: Subband and Wavelet Coding

- More information: Chapter 12, 13 [**IDC**]