6. Transform-Based Coding

6.1 Introduction

- **Previous chapter…**
  - Why Cosine-based series preferred to Sine or Cosine&Sine
- **Now, begin discussing specific transforms…**
  - Changing data from one representation to another
  - Specifically focus on the various 1-D and 2-D transforms most often used in image/video processing
- **Following this…**
  - Introduce the concepts of *quantization* and *bit-allocation*

    → processes that allow us to exploit transformed data towards data compression

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega x + \sum_{n=1}^{\infty} b_n \sin n\omega x$$

6.1.1 Sampled Data

$$a_n = \frac{2}{L}\int_0^L f(x)\cos n\omega x dx$$

$$b_n = \frac{2}{L}\int_0^L f(x)\sin n\omega x dx$$

$$a_0 = \frac{2}{L}\int_0^L f(x)dx$$

- **Recall (previous chapter)…**
  - Applying Fourier Analysis to non-periodic signal *f(x)*
    → Must proceed 'block-by-block'

    (i.e. extract subsection of *f(x)*; consider it as one period of artificial periodic function; apply Fourier; then move onto next block)
  - Applying Fourier analysis on a sampled signal *f(x)*…
    - Since *f(x)* only defined at discrete points, to calculate Fourier coeffs…

      → Compare *f(x)* with sampled versions of the sines/cosine basis functions (defined at the same points) via dot-products

RULE: Fourier analysis of discrete signal *f(x)* → num. of expansion terms required to fully represent *f(x)* = the number of samples in *f(x)*

(→ infinite number required in the continuous case!)

6.1.2 Simple Discrete Linear Transform

- **What do we mean by transforming?**
  - Given a set of data…

    → augment its representation ('re-phrase it')

    So aspects other than its currently known values are revealed (i.e. certain patterns, groupings, alignments, etc.)

- **Why do we want to transform data?**
  - In the expectation that in its new format → data may be more susceptible to exploitation in some way
    - E.g. see what we can do without

      → Compression! ☺

6.1.2 Simple Discrete Linear Transform

Example: Transforming data to reveal new aspects:

Data Samples: X = {13, 22, 23, 25, 45, 132, 24, 8}

Transform via: $y_n = x_n - x_{n-1}$ (→ represents the rate-of-change in X!)

Transformed data: Y = {13, 9, 1, 2, 20, 87, -108, -16}

→ new aspect revealed: rate-of-change!

Retrieve original data via: $x_n = y_n + x_{n-1}$ (→ 'the reverse transform')

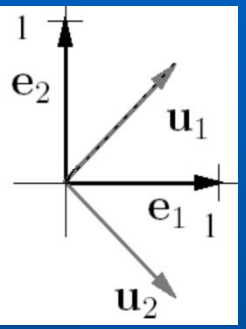
6.1.2 Simple Discrete Lin. Transf. (cont.)

- **Another transformation example…**
  - Task: Discrete data sequence X_n of length two to be transformed by an '*invertible orthogonal transform*'

    $X_n = [2, 3]$
  - Note 1: Invertible ≡ lossless
    - Will need the same number of coeffs in the transformed domain as original samples to fully represent it – i.e. 2 coeffs weighting the contribution of 2 basis vectors, against which the data is compared.
  - Note 2: Basis Functions?
    - $\boldsymbol{X_n}$ is discrete: Basis functions to be defined at same points as input signal
      - i.e. each basis vector to have length 2 (same as input)
  - Note 3: Orthogonal basis functions
    - Typical specification → ensures they represent completely independent aspects of the data (e.g. parallel basis vectors)

6.1.2 Simple Discrete Lin. Transf. (cont.)

- **E.g. solution…**
  - A standard pair of basis vectors that meets these requirements…



$$\mathbf{u}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \& \quad \mathbf{u}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

i.e. orthogonal, length-2, and there are two of them!

- **As before, we calculate corresponding transform coefficents by 'comparing' input signal with basis functions…**
  - i.e. calculating the dot product between input data values and each basis vector…
  - Resultant coeffs represent <u>similarity</u> between i/p vector and basis function

$$X_1 = \mathbf{u}_1 \cdot \mathbf{x} = \mathbf{u}_1^T \mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \frac{5}{\sqrt{2}}$$

$$X_2 = \mathbf{u}_2 \cdot \mathbf{x} = \mathbf{u}_2^T \mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \frac{-1}{\sqrt{2}}$$

6.1.2 Simple Discrete Lin. Transf. (cont.)

- **N.B. Transformation coeffs *X1* and *X2* represent (completely!) the original data in the transformed domain → they simply now reference a different basis system (i.e. u_1 & u_2)!**
  - Any processing performed on these values in the new domain will affect the reconstruction of the original data
- **Reconstruction from transformed domain?**
  - Have coeffs
  - Have basis functions

  } Transformed Domain
  - Retrieve original data?
    - Simply carry out (calculate) the weighted sum of the expansion
      - i.e. the sum of the basis vector values with the transform coefficients as the weights…

N.B. Perfect reconstruction

$$\mathbf{x} = X_1\mathbf{u}_1 + X_2\mathbf{u}_2 = \left(\frac{5}{\sqrt{2}}\right)\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} + \left(\frac{-1}{\sqrt{2}}\right)\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = \begin{bmatrix}2\\3\end{bmatrix}$$

6.1.2 Simple Discrete Lin. Transf. (cont.)

- **Frequency interpretation** (trivial example, but illustrative!)
  - Recall calculations involved in computing the transform coefficients

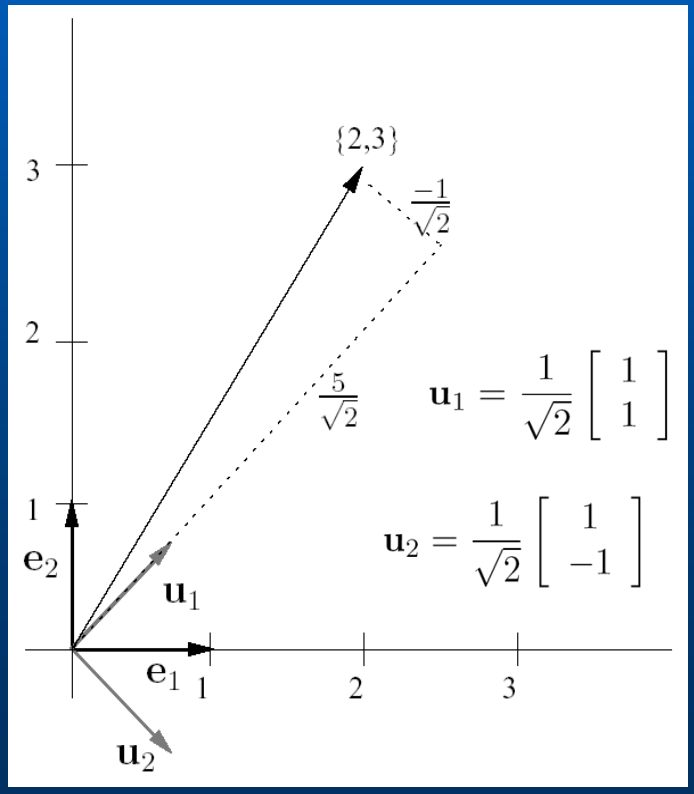
$$X_1 = \mathbf{u}_1 \cdot \mathbf{x} = \mathbf{u}_1^T \mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \frac{5}{\sqrt{2}} = \frac{1}{\sqrt{2}} [x_1 + x_2]$$

$$X_2 = \mathbf{u}_2 \cdot \mathbf{x} = \mathbf{u}_2^T \mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \frac{-1}{\sqrt{2}} = \frac{1}{\sqrt{2}} [x_1 - x_2]$$

  - Because of the basis functions used…
    - Calculating X_1 corresponds to finding the sum/average of the input data values
      - Akin to finding a 'low-frequency' coefficient
    - Calculating X_2 corresponds to finding the difference of the input data values…
      - Akin to finding the extent to which there is change ('high-frequency') in the data

6.1.2 Simple Discrete Lin. Transf. (cont.)

- **Geometrical Interpretation**
  - Note 1: Basis vectors used (u_1, u_2) are simply a rotated (+reflected) version of the std. basis vectors e_1 & e_2
  - Note 2: 'energy' of input data vector {2,3} along e_1 & e_2 is approx equal
  - Note 3: 'energy' of input data vector along u_1 larger than that along u_2
    - Hence in transformed domain representation…
      → Smaller *reconstruction error* would be generated by eliminating u_2 from the expansion than eliminating u_1
    - Why do this?
      - Data saving → compression!



$$\mathbf{x} = X_1\mathbf{u}_1 + X_2\mathbf{u}_2 = \left(\frac{5}{\sqrt{2}}\right)\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} + \left(\frac{-1}{\sqrt{2}}\right)\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} = \begin{bmatrix}2\\3\end{bmatrix}$$

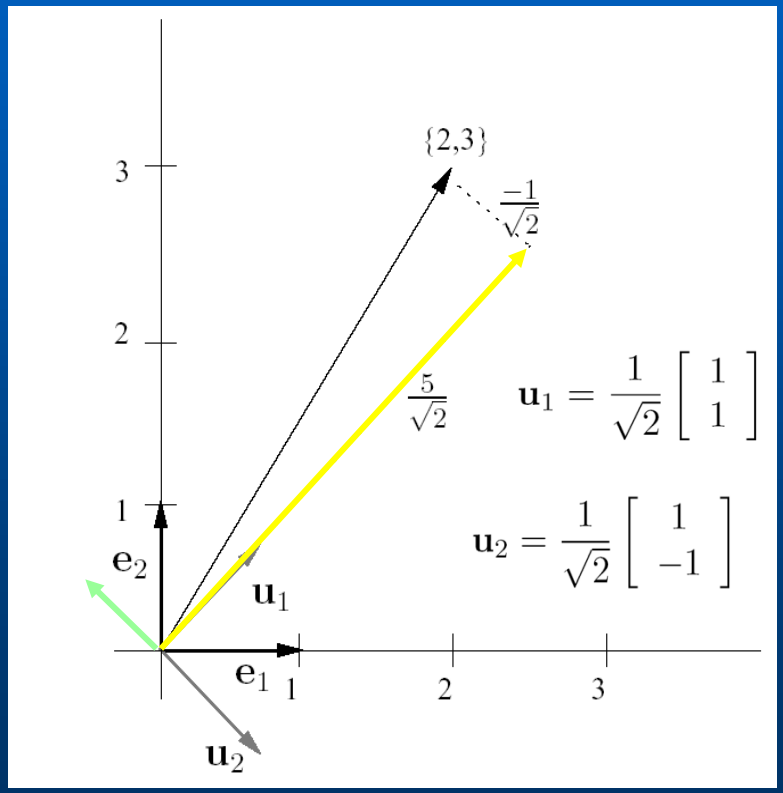
6.1.2 Simple Discrete Lin. Transf. (cont.)

- **Illustration:**

  $ReconU_1 = X_1U_1 = [2.5\ 2.5]$

  $ReconU_2 = X_2U_2 = [-0.5\ 0.5]$

  → $ReconU_1$ [2.5 2.5] closest to orig vector x [2 3]



NOTE: Original dataset [2,3] is said to be **decorrelated** in the transformed domain

→ Energy originally shared between two components, now more compacted (concentrated) on one component at the expense of the other

→ E.g. of how transforming data can reveal different (more exploitable) patterns

6.1.2 Simple Discrete Lin. Transf. (cont.)

- **Quick Note…**
  - More convenient to express a linear transform i.t.o. matrices
  - E.g. write transform basis vectors u_1 & u_2 together as rows of a matrix…

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

  - Can then express transform coefficients in matrix form as…

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \mathbf{A}\mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Laws of matrix multiplication implement the dot product

  - Expression for the data reconstruction then given by…

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{B}\mathbf{X} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

$$\mathbf{B} = \mathbf{A}^{-1} = \mathbf{A}$$

6.1.3 General Transf. Properties

- **Some more definitions & properties…**
  - Previous example: Basis vectors u_1 & u_2, as well as being orthogonal, both had length = 1
    - Known as '*orthonormal*' vectors
  - Recall, [u_1 & u_2] simply a rotation & reflection of the std. co-ordinate vectors (on which the original data defined)
    - Hence → Magnitude of the energy in the original data preserved in the transformed domain (no scaling)
      - i.e. the sum of the energy of the transform coefficients = the sum of the energy of the original data points

        → Important property of orthonormal transforms (see later)

6. Transform-Based Coding

6.2 1-D Discrete Orthogonal Transforms

- **This section: A background study**
  - Discuss some of the most efficient and well-studied **discrete** orthogonal transforms for signal processing (in their 1-D forms)
- **Following this…**
  - Examine the 2-D equivalents of these transforms
- **Quick note…**
  - Any properties associated with series' mentioned in terms of **continuous** domain processing → carry through to **discrete** equivalents
  - E.g. from Ch. 4 → "Cosine Series defined on a continuous interval exhibits good energy compaction"

    → Discrete cosine series expansion: similar compaction performance! ☺

6.2.1 The Discrete Cosine Transform

- **Recall (Ch. 4)…**
  - Key advantages of cosine expansion are…
    - Real arithmetic - no Complex implementation required (as in the DFT)
    - Faster convergence than sine-based equivalent
- **The discrete equivalent of cosine expansion…**
  - Known as the *Discrete Cosine Transform* (DCT)…
    - Retains these properties
- **Reminder…**
  - Matrical relationship between the original data (*x* - length *n*), the corresponding transform coefficients (*X* – length *n*), and the *n* *n*-dim transform basis vectors (*u*)

Rule1: # coeffs = # i/p data points
Rule2: # basis functions = # coeffs
Rule3: length basis = length i/p

$$\begin{bmatrix} X_0 \\ \vdots \\ X_{n-1} \end{bmatrix} = \begin{bmatrix} \longleftarrow \mathbf{u}_0^T \longrightarrow \\ \vdots \\ \longleftarrow \mathbf{u}_{n-1}^T \longrightarrow \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

6.2.1 The Discrete Cosine Transf. (cont.)

- Note, 'specifying' the DCT → corresponds to specifying the basis functions (matrix) onto which we project (compare) our data

$$\begin{bmatrix} X_0 \\ \vdots \\ X_{n-1} \end{bmatrix} = \begin{bmatrix} \longleftarrow \mathbf{c}_0^T \longrightarrow \\ \vdots \\ \longleftarrow \mathbf{c}_{n-1}^T \longrightarrow \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

- **NxN DCT basis matrix…**
  - First row defined by…

$$\mathbf{c}_0 = \frac{1}{\sqrt{N}}$$

  - Remaining rows (k=1 to N-1) are given by…

$$\mathbf{c}_k = \sqrt{\frac{2}{N}} \cos \frac{(2n+1)k\pi}{2N}$$

… where n ranges from 0 to n-1 across each row

6.2.1 The Discrete Cosine Transf. (cont.)

$$\mathbf{c}_0 = \frac{1}{\sqrt{N}} \qquad \mathbf{c}_k = \sqrt{\frac{2}{N}} \cos \frac{(2n+1)k\pi}{2N}$$

- **E.g. 1-D length-2 i/p data to be transformed…**
  - Corresponding N=2 DCT matrix given by…

$$\mathbf{C}_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{4} & \cos\frac{3\pi}{4} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- **E.g. 1-D length-4 i/p data to be transformed…**
  - Corresponding N=4 DCT matrix given by…

$$\mathbf{C}_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ a & b & -b & -a \\ 1 & -1 & -1 & 1 \\ b & -a & a & -b \end{bmatrix}$$

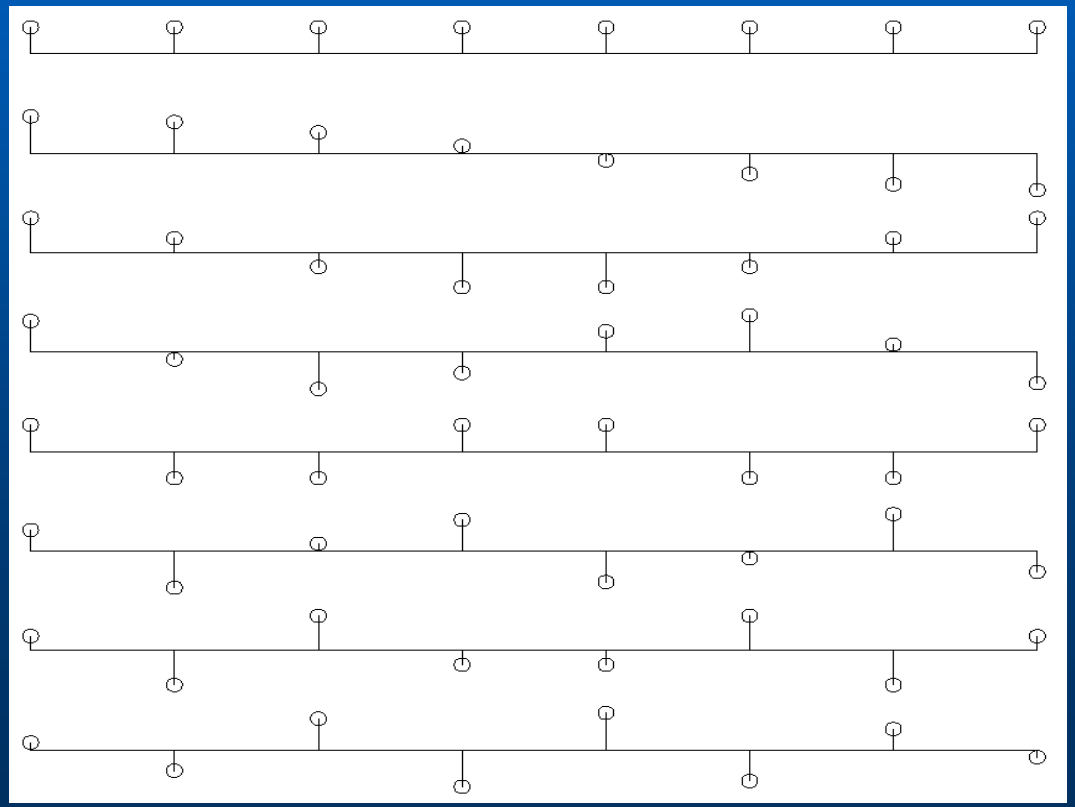
$$a = \sqrt{2}\cos\frac{\pi}{8} = 1.306$$

$$b = \sqrt{2}\cos\frac{3\pi}{8} = 0.541$$

6.2.1 The Discrete Cosine Transf. (cont.)

- **E.g. 1-D length-8 i/p data to be transformed**
  - Plot of the basis function trends for corresponding N=8 1-D DCT…

N.B. Vectors of increasing rate of change (freq)



6.2.1 The Discrete Cosine Transf. (cont.)

- **General properties of the DCT (to be remembered!)…**
  - Matrix values are real
  - Basis vectors are orthogonal
    - i.e. they represent completely independent aspects of the data
  - The forward and inverse transform have identical matrices

$$\mathbf{X} = \mathbf{A}\mathbf{x} \quad \& \quad \mathbf{x} = \mathbf{B}\mathbf{X}$$

… where $\mathbf{B} = \mathbf{A}^{-1} = \mathbf{A}$

  - DCT has a fast implementation version associated with it
    - Based on the Fast-Fourier Transform (FFT)
  - Cosine-based series → exhibits very good energy compaction for correlated data (images!)
  - Transform used in JPEG, H.261, MPEG-1.

6.2.2 The Discrete Sine Transform

- **Very quickly (not really relevant to us)…**
  - Row values of the NxN DST matrix are given for k=0 to N-1 as…

$$\mathbf{s}_k = \sqrt{\frac{2}{N+1}} \sin \frac{(n+1)(k+1)\pi}{N+1}$$

… where n ranges from 0 to N-1 across each row

  - **Properties of the DST…**
    - DST matrix is real and symmetric… $\mathbf{S} = \mathbf{S}^* = \mathbf{S}^T$
    - Orthogonal basis
    - The forward and inverse transform are identical… $\mathbf{S} = \mathbf{S}^{-1}$
    - The DST has a 'fast' version based on the FFT
    - Sine-based series → exhibits lesser energy compaction for correlated signals (better for uncorrelated data, e.g. text images)

6.2.3 The Hadamard Transform

- **Overview**
  - Orthonormal transform, used in signal processing and image/video compression
  - Examples…
    - The smallest HT matrix is… $$\mathbf{H}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$
    - In general, to construct higher order HT matrices → use 'tiling'…

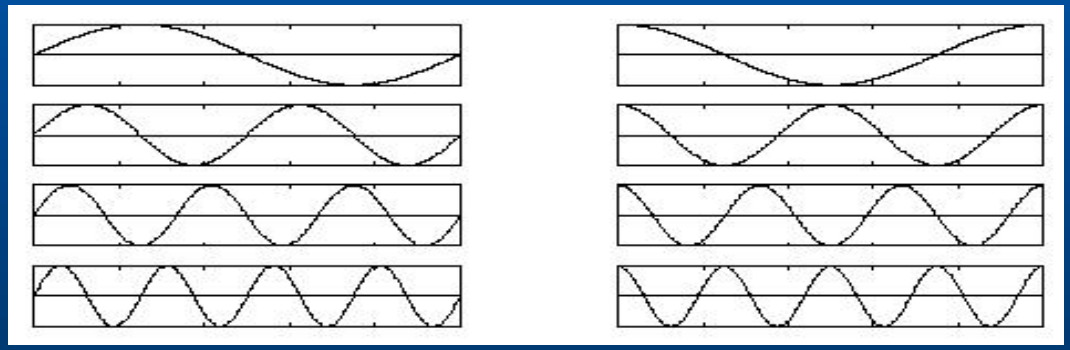
$$\mathbf{H}_{2N} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{H}_N & \mathbf{H}_N \\ \mathbf{H}_N & -\mathbf{H}_N \end{bmatrix}$$

$$\mathbf{H}_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

6.2.3 The Hadamard Transform (cont.)

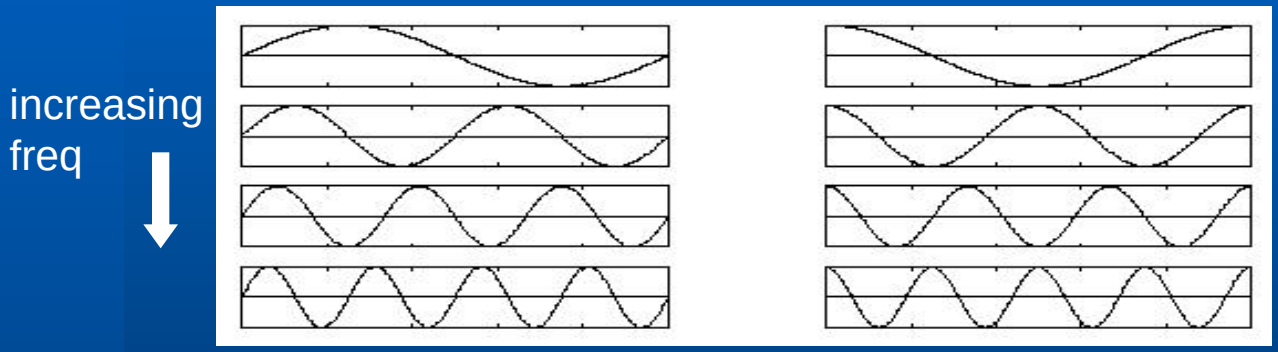
- **Frequency Interpretation of HT**
  - What concept of frequency interpretation do we have for the HT?
  - First, recall Fourier's Theorem…

    Periodic signal can be represented as a linear combination of the set of all integer-cycle sines/cosines defined in the period interval



6.2.3 The Hadamard Transform (cont.)

- Note direct relationship between frequency of the basis functions and the number of zero-crossings…

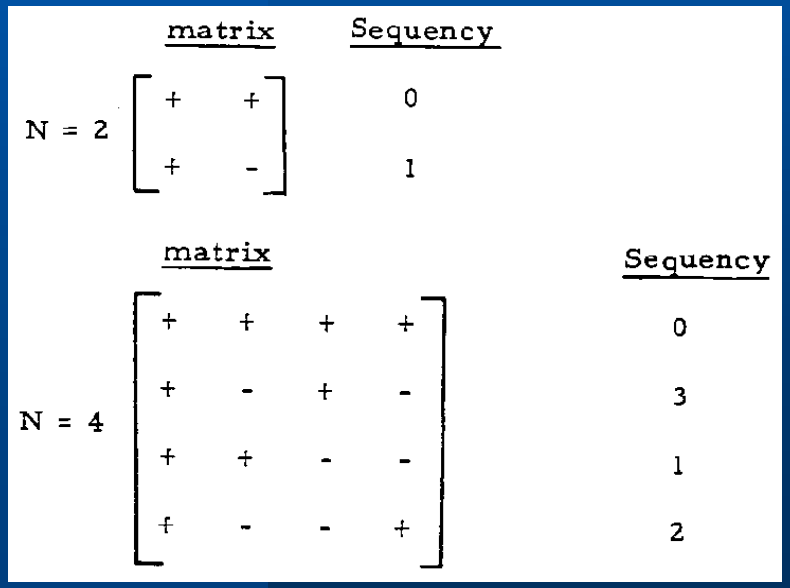


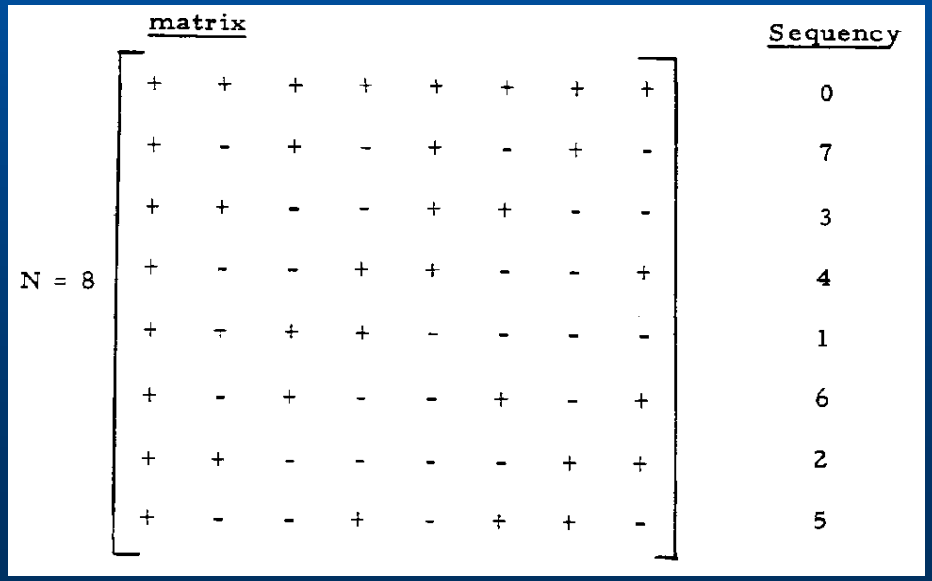
- However, Hadamard Transform (like many other transforms)…
  - based on orthogonal functions (vectors) that are not sinusoidal
- However → can exploit the concept of zero-crossings to infer a frequency style interpretation for non-sinusoidal transforms
  → known as '*transform frequency*' (as opposed to spatial frequency)

6.2.3 The Hadamard Transform (cont.)

- **The *transform frequency* of the HT basis functions…**
  - Given a special name → '*sequency*'
    - Relates to the num. of sign changes in the basis vectors (rows of the matrix)

      (N.B. some books → taken as half the number of sign changes!)
  - Some examples (HT basis matrices + sequency of each functions)…





6.2.3 The Hadamard Transform (cont.)

- **Variant on HT: Walsh-Hadamard Transform**
  - Typically → Rows of the HT matrix are ordered by sequency
    - Known as the *Walsh-Hadamard Transform* (WHT)
- **E.g.**
  - Non sequency-ordered HT matrix…(from before)

$$\mathbf{H}_4 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

'Hadamard order'

  - Corresponding sequency-ordered WHT Matrix…

$$\mathbf{H}'_4 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

6.2.3 The Hadamard Transform (cont.)

- **Hadamard Transform: Properties**
  - Matrices of the HT and the WHT are real, symmetric and orthogonal
  - Has a fast version associated with it
    - Furthermore, since basis vectors only have elements ±1 (scaled)…
      - No multiplications are required in the transform calculations
        - V. good from an implementation point of view!
  - Exhibits reasonable energy compaction for strongly correlated signals
- **Applications**
  - Image compression, e.g. *HD Photo*
  - Video compression, e.g. H.264/MPEG-4 AVC (compress DC-DCT coeffs)
  - Video coding (ME block matching): implement SATDs (as opposed to SADs)

6.2.4 The Haar Transform

- **Overview**
  - Orthogonal transform, used in signal processing and data compression
  - Matrix elements: powers of $\sqrt{2}$, ± 1 and/or 0
- **Example N=2…**

$$H_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

- **Frequency Interpretation**
  - Works by transforming array of i/p values into array of averages (‘*approximations*’) and differences-from-the-average (‘*details*’)
  - N.B. Use of the $\boldsymbol{H_2}$ transform illustrated in trivial example earlier (6.1.2)
    - Explained frequency interpretation of averages and differences
    - Showed decorrelation power

6.2.4 The Haar Transform

- **In general…**
  - Higher-order Haar matrices → continue this approach of extracting 'differences from the average', but at increasing resolutions
- **Example N=4…**

$$\mathbf{H}_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

- **Example N=8…**

$$\mathbf{H}_8 = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

6.2.4 The Haar Transform (cont.)

- **Properties of the Haar Transform…**
  - Haar matrix is real and orthogonal
  - Very fast
    - For N data points it can be implemented in O(N) operations!
  - Exhibits *sequency*-ordered basis vectors
  - Poor energy compaction for strongly correlated ('low-passed') data
    - Hence, not very efficient for general images that tend to have substantial areas of spatial correlation
    - Typically useful for compressing high-frequency images

The Haar Transform (usage example of Haar wavelets)

Non-normalized Haar matrices (N=4):

$$A_1 = \begin{bmatrix} 0.5 & 0 & 0.5 & 0 \\ 0.5 & 0 & -0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0.5 & 0 & -0.5 \end{bmatrix}$$

y – input vector
y_2 – transformed vector

Thus, the transformation matrices:

$$W_1 = A_1$$
$$W_2 = A_1 A_2$$
$$W = A_1 A_2 \ldots$$

$$y_1 = yA_1$$
$$y_2 = y_1 A_2$$
$$y_2 = yA_1 A_2$$

$$A_2 = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 \\ 0.5 & -0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Compression algorithm:

1. Convert the image into a matrix format (I).

$$I = \begin{bmatrix} \cdots & \cdots \\ \cdots & \cdots \end{bmatrix}$$

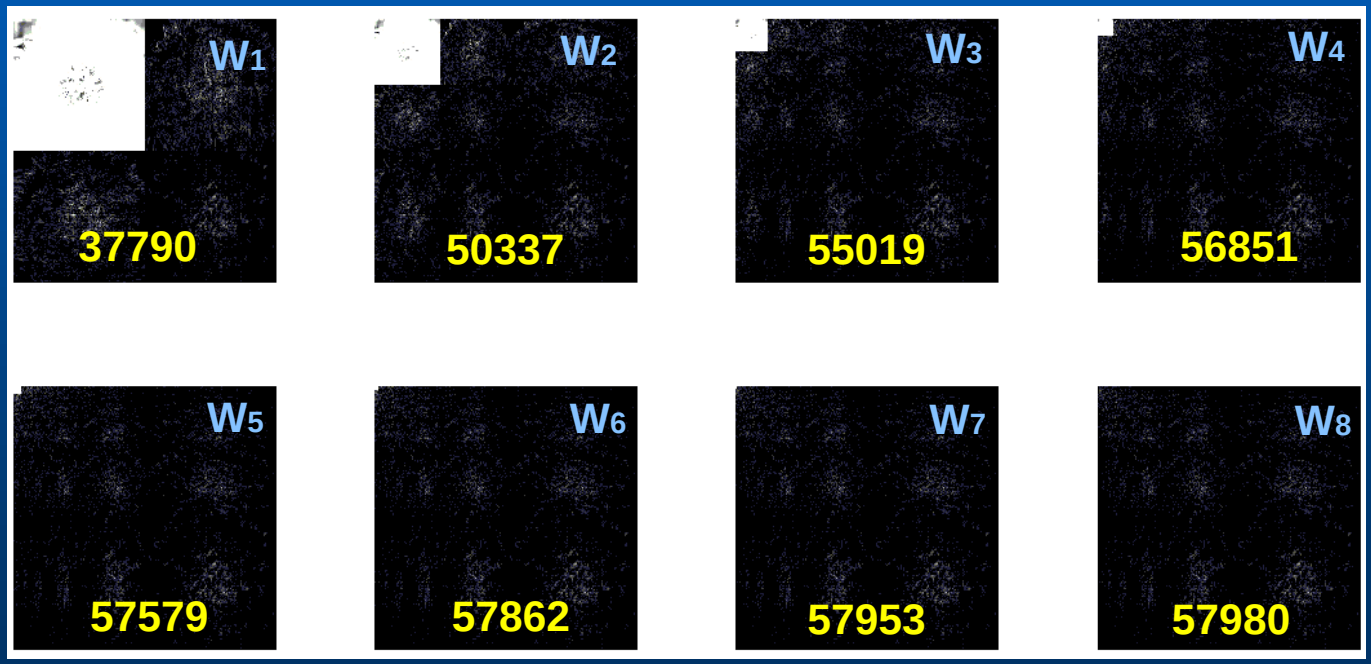
The Haar Transform (usage example cont.)

Compression algorithm cont.:

2. Calculate the row-and-column transformed matrix (T) using following equation:

$$T = W^T I W$$

Number of values smaller than 5 in original image = **28**



The transformed matrix should be relatively sparse.

The Haar Transform (usage example cont.)

Compression algorithm cont.:

3. Select a threshold value *e* (in our case *e=5*), and replace any element of *T* less than with a zero. This will result in a sparse matrix denoted as *S*.
4. To get our reconstructed matrix *R* from matrix *S*, we use an equation similar to:

$$R = \left(W^T\right)^{-1} S W^{-1}$$

but, because the inverse of an orthogonal matrix is equal to its transpose, we modify the equation as follows:

$$R = WSW^{-1} = f(S) \qquad e.g., \quad f_1(S_1) = W_1 S_1 W_1^T$$

Properties:

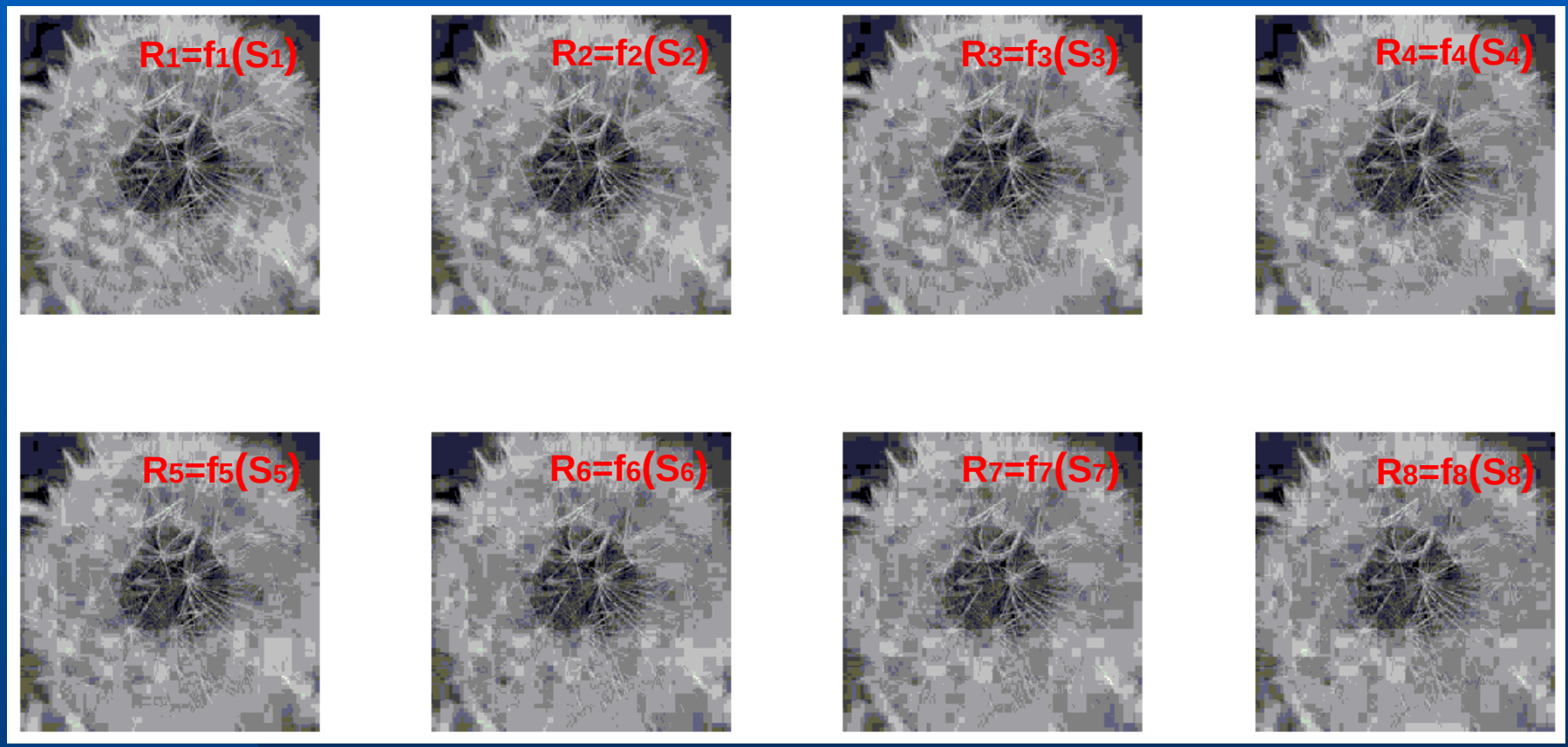
✓ If $e = 0$, then $S=T$ and therefore $R=I$ (lossless compression).

✓ If $e > 0$, then elements of *T* are reset to zero, and some original data is lost. The reconstituted image will contain distortions (lossy compression).

✓ The secret of optimal compression is to choose *e* so that compression is maximized while distortions in the reconstituted image are minimized.

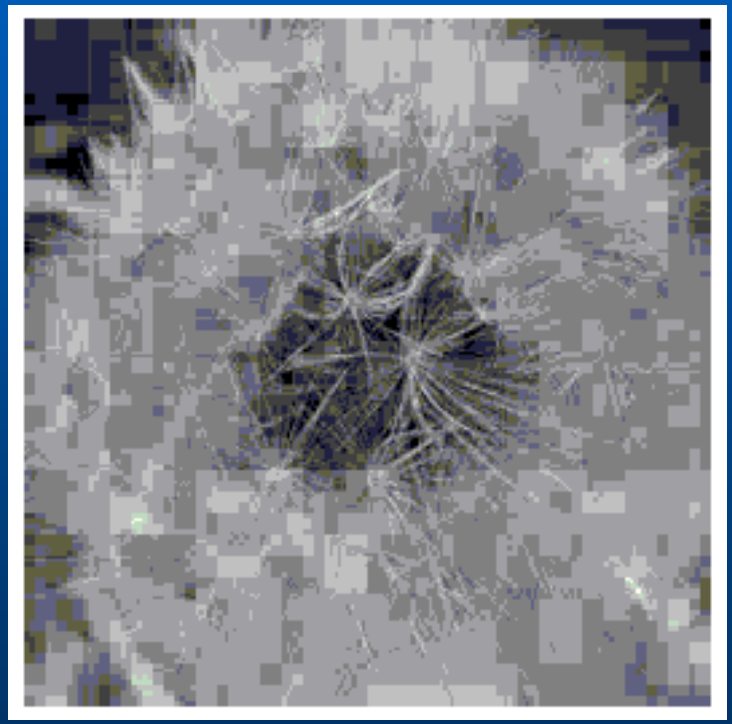
The Haar Transform (usage example cont.)

Decompression results:



The Haar Transform (usage example cont.)

Normalized vs. non-normalized:

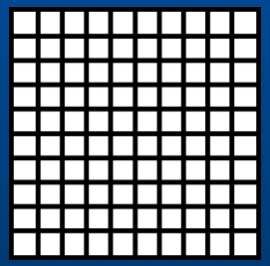


6. Transform-Based Coding

6.3 2-D Discrete Transforms

- **So far…**
  - Talked about transforms for 1-D space/time signals
    - E.g. 1-D time signal: Height of EE554's best performing student per year (temporal)
    - E.g. 1-D spatial signal: Average student height per lecture theatre row in todays lecture (spatial snapshot)

- **Images…**
  - 2-D spatial signals!
    - i.e. {width, height}



- **How transform 2-D signal?**

6.3 2-D Discrete Transforms

- **Two different ways of transforming 2-D data…**
  - Rearrange the 2-D data into *concatenated 1-D form*…
    - Then apply 1-D methods as before
      → Not very sophisticated
      → Quite cumbersome
  - Alternative: Extend the 1-D transforms into proper 2-D versions
    - i.e. construct seperable 2-D versions of the 1-D transforms
      → Most common approach, and the focus here!

6.3 2-D Discrete Transforms (cont.)

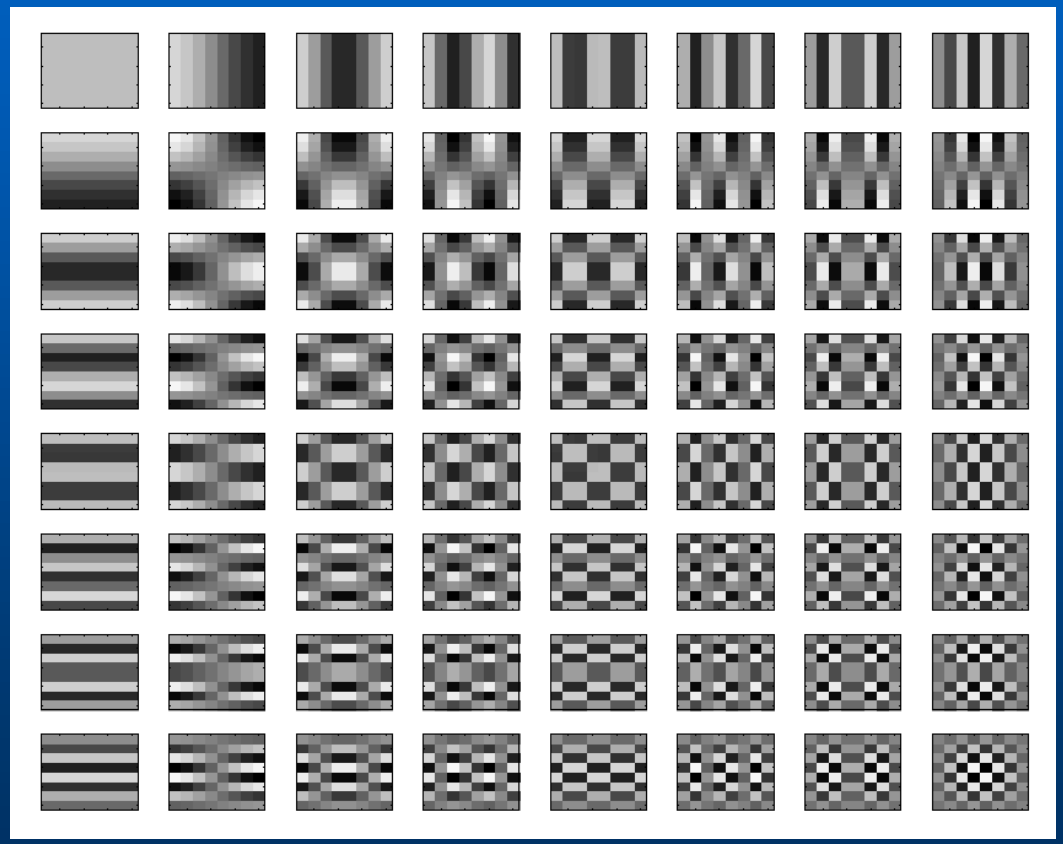
- So far: Transforming a 1-D block of data of length *N* (samples)…
  → Means we express it in terms of 1-D (length-N) basis vectors

- What is the 2-D equivalent?
  - E.g. NxN block of data to be transformed…
    - Transforming means we should be expressing it in terms of corresponding NxN basis vectors
      - With an associated 2-D set of transform coefficients
  - N.B. 2-D block of data = 'image' ➔ 2-D basis functions also 'images'!

- **Implementation of 2-D Transforms…**
  - A 1-D transform is applied to each *row* of the data block
  - Another 1-D transform is then applied to each *column* of this result
    - i.e. two separate 1-D transforms!

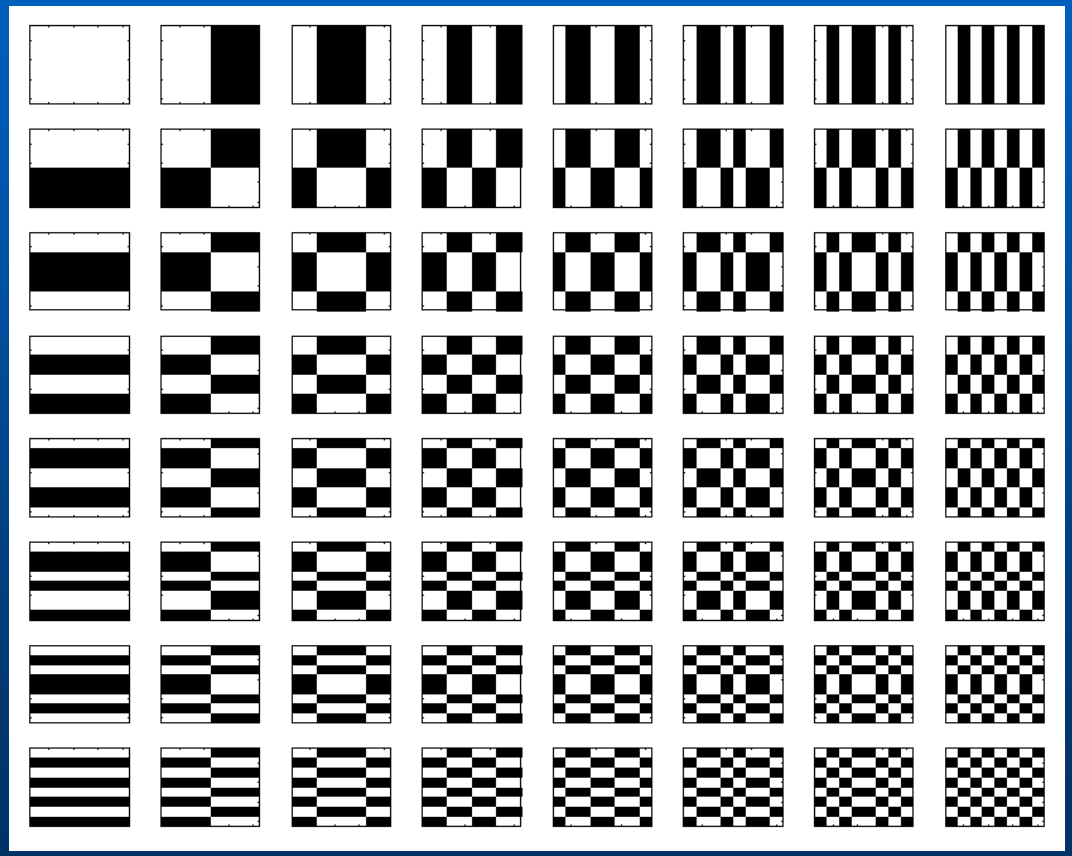
6.3.1 The 2-D Discrete Cosine Transform

- **E.g. 8x8 2-D DCT**
  - Image representations of the 2-D basis vectors…



6.3.2 The 2-D Discrete Walsh-Hadamard

- **E.g. 8x8 2-D DWHT**
  - Image representations of the 2-D basis vectors…



6. Transform-Based Coding

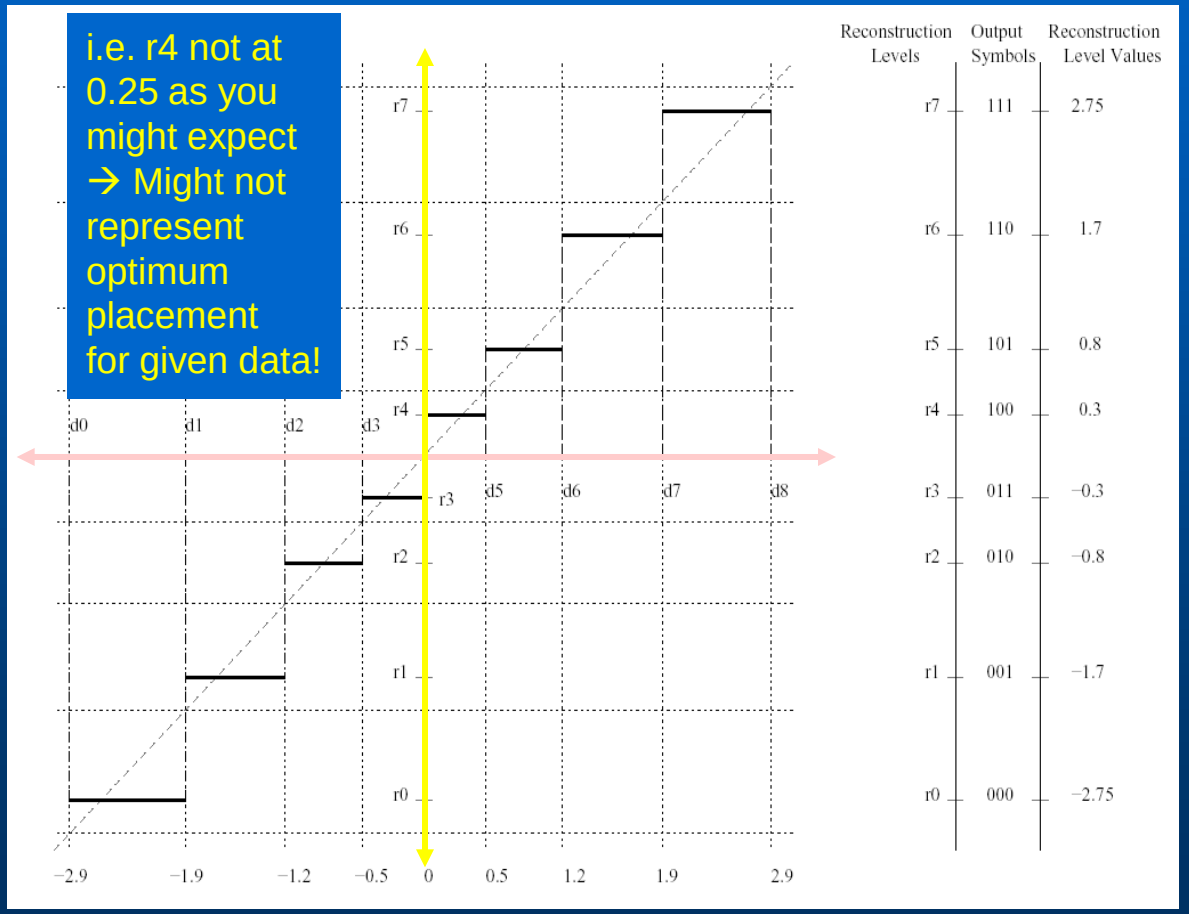
6.4.1 Scalar Quantization

- **Quantization: a quick definition…**
  - Procedure of constraining a measurement exhibiting a continuous domain (e.g. the real nos.) to a discrete (finite) set (e.g. the integers)
- **Role in image/video compression significant…**
  - E.g. reducing number of colours allowed to represent an image
    → E.g. Only 256 pre-defined colour levels per RBG channel (16M col.)
- **Simplest form: Scalar Quantization (Qzn)…**
  - Value of a scalar quantity is compared with a set of decision levels
    - Then, depending on within which band the value resides…
      → Assigned a corresponding code (representing nearest allowable value)
- **Challenge of quantizer design…**
  - How many decision levels to have? Evenly spaced or not?
    - Objective: minimize the distortion between i/p-o/p values for target data

6.4.1 Scalar Quantization (cont.)

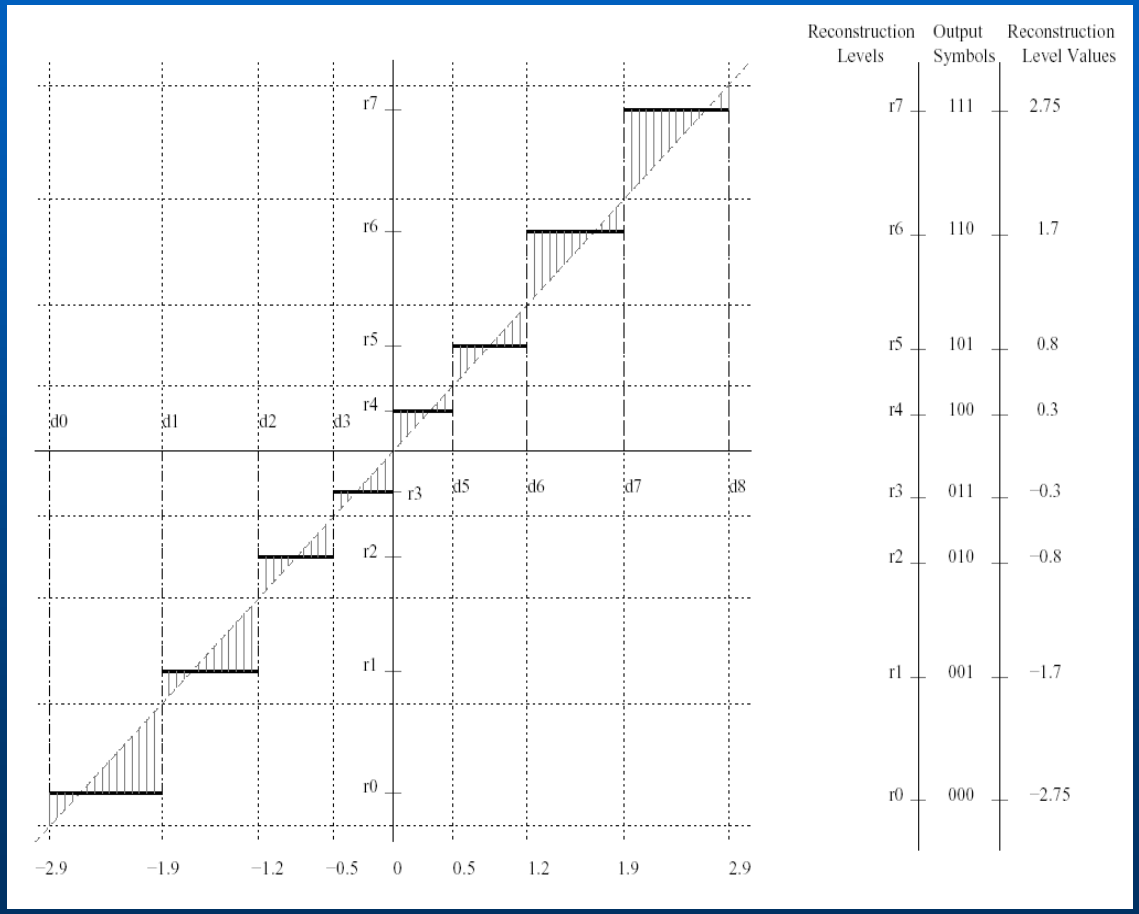
N.B. length-3 bit restriction for this scalar correponding to 8 possible reconstruction levels ⇓

- **Example…**
  - Input decision levels (d0-d8) along the (input) horizontal axis (non-uniform!)
  - Reconstruction levels (r0-r7) corresponding to each band along the vertical (output) axis (N.B. not centred!)
  - Also shown → the binary symbols for each decision band and the real number for each level



6.4.1 Scalar Quantization (cont.)

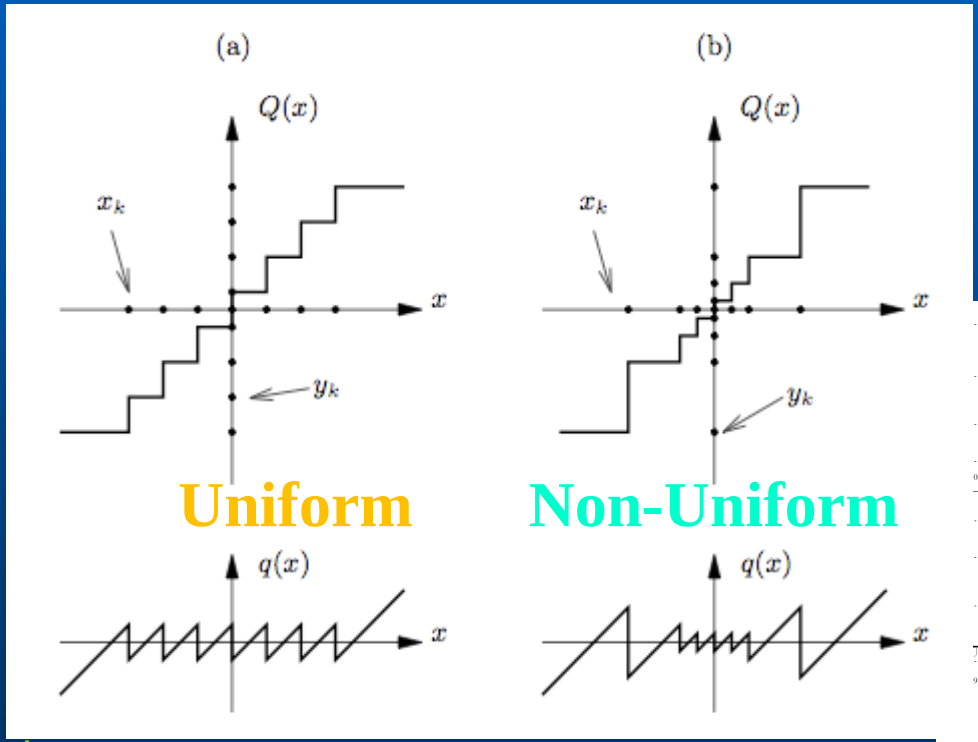
- **Example… (cont.)**
  - Illustration of the error introduced by this process
  - Vertical shade shows the difference between the quantized (reconstructed) output values and what the output values would be ordinarily
  - Magnitude of the error depends on the position of the i/p value relative to output value within each respective band

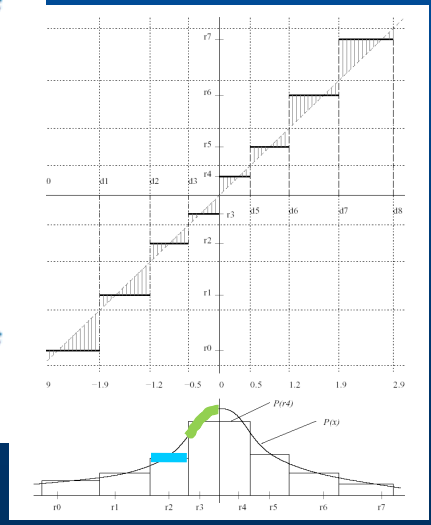


6.4.1 *Types of Scalar Quantizers*

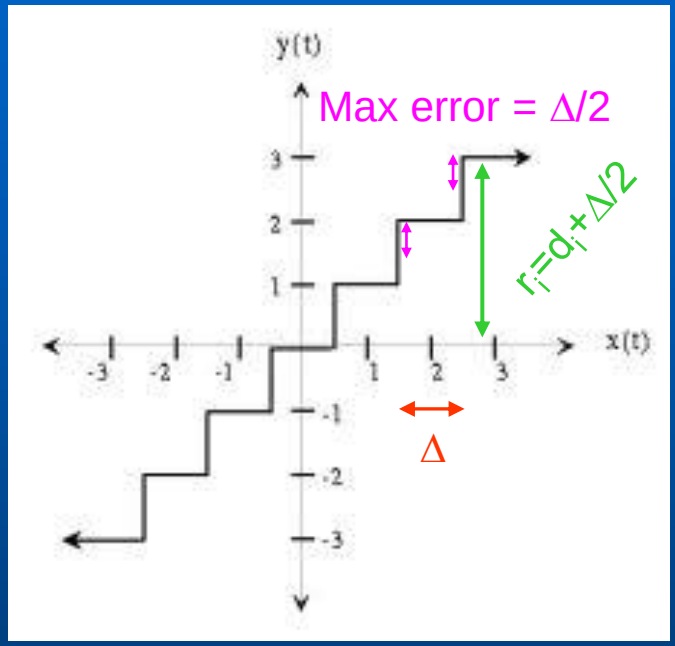
- Constant probability distribution (function) in the quantization range
- Stable behavior of error function in the quantization range
- Probability distribution no longer constant
- Error function can change its amplitude for different quantization bands
- Two cases:
  - Uniform distribution within a band
  - Non-uniform distribution within a band

Uniform Distr. vs Non-Uniform Distr. Quantizers





6.4.1 *Uniform Distribution Quantizers*



- **An even more specialised case…**
  - *P(x)* uniform over the entire signal range
  - Optimum quantizer obvious: Uniform!
    - Decision bands all the same width
    - Reconstruction levels equally spaced
- **Calculating the SNR for this quantiser…**
  - Let Δ = width between decision levels, i.e…

    $$\Delta = d_{i+1} - d_i$$
  - Corresponding reconstruction levels are at…

    $r_i = d_i + \Delta/2$ (a value half way between the decision level bounds)
  - Hence for *x* assuming values within particular band $\{d_i, d_{i+1}\}$…
    - Qzn error magnitude will never be larger than $\Delta/2$
    - And since all input values in the band equally likely…

      → Overall qzr error will be uniformly distributed over the interval $\{-\Delta/2, \Delta/2\}$

6.4.1 *Uniform Distribution Qzrs (cont.)*

- **Calculating the SNR for this quantiser (cont.)…**
  - Now exploit known result:
    - **Variance** of a uniform random variable with range R = **$R^2/12$**
  - Assuming we allocate *n* bits to the (uniform) quantizer
    - i.e. we have 2^n decision levels covering the input interval {-A/2, A/2}
      - This means $\Delta = A/2^n$ (decision level width = input range ÷ #levels)
  - In our case: Qzn error uniformly distributed over interval $\{-\Delta/2, \Delta/2\}$
    - So, can say **variance of Qzn error = $\Delta^2/12$**
  - Furthermore: Assuming the input is also uniformly distributed - over the interval {-A/2, A/2}…
    - Can say **input has variance = $A^2/12$**
  - **MS-SNR** for this system given by ratio of the input variance and Qzn error variance:

$$\frac{A^2/12}{\Delta^2/12} = \frac{1}{(1/2^n)^2} = 2^{2n}$$

in dBs: $10\text{Log}_{10}(2^{2n}) = 6n$ dB
i.e. SNR improves by 6dB per bit

6.4.1 *Optimum MS Quantizers (Lloyd-Max)*

- **Optimising Qzr design** → Need to consider probability of particular data values occuring…
  - E.g. dataset where particular range of input values occur frequently…
    - Vital to minimize the Qzn errors for these values in particular (i.e. have a fine decision level spacing for this range)

- **On this note…**
  - Consider an input variable *x* quantized with an N-level Qzr…
    - Overall Qzn error…

$$e_q = \sum_{i=0}^{N-1} \int_{d_i}^{d_{i+1}} (x - r_i)^2 \, P(x) \, dx$$

r_i = reconstruction levels
$P(x)$ = probability of x

→ the sum of the MSEs introduced in each individual band - i.e. between each consecutive pair of decision level points (d_i)

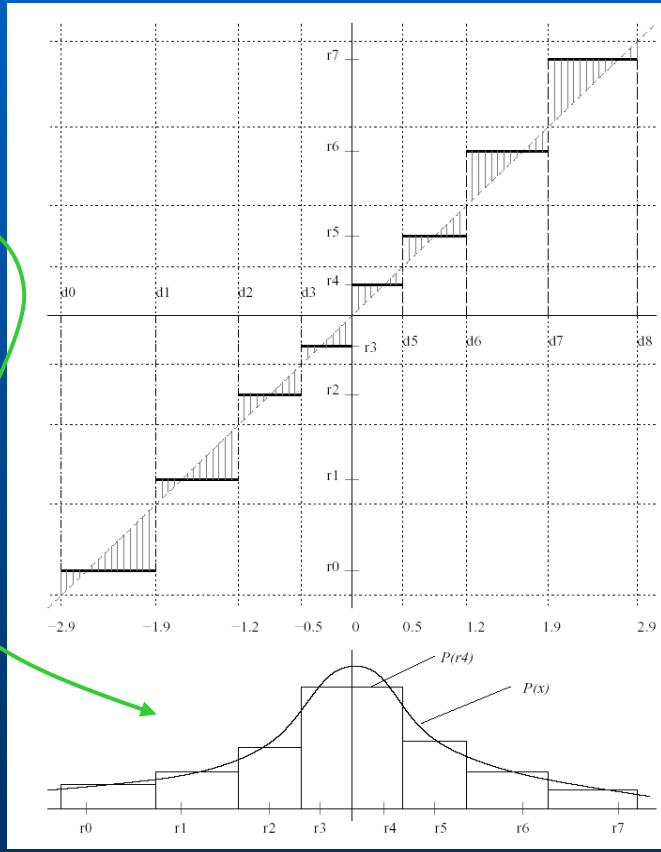
$$e_q = \sum_{i=0}^{N-1} \int_{d_i}^{d_{i+1}} (x - r_i)^2 \, P(x) \, dx$$

6.4.1 *Optimum MS Quantizers (cont.)*

- **Note, sometimes can assume (e.g. N large)**
  - *P(x)* constant over each quantization band
    - *i.e. probability of x assuming any of the values with a given band is equal! (still different for each band)*
  - Hence → can say $P(x) \equiv P(r_i)$
    - *Probability of x occurring → same as probabilty of its corresponding reconstructed level occurring*
  - Can now rewrite error expression as…

$$e_q = \sum_{i=0}^{N-1} P(r_i) \int_{d_i}^{d_{i+1}} (x - r_i)^2 \, dx$$

$$= \frac{1}{3} \sum_{i=0}^{N-1} P(r_i) \left[(d_{i+1} - r_i)^3 - (d_i - r_i)^3 \right]$$



Band-wise approximation of non-uniform distribution of x

$$e_q = \frac{1}{3}\sum_{i=0}^{N-1} P(r_i)\left[(d_{i+1}-r_i)^3 - (d_i - r_i)^3\right]$$

6.4.1 *Optimum MS Quantizers (cont.)*

- **So, finding the optimum quantizer…**
  - Have expression for overall MSE introduced by quantization process
  - Finding the **optimum solution** corresponds to finding locations of the r_i and d_i that minimize this expression
- **Optimal r_i**
  - Calculating $\frac{de_q}{dr_i} = 0$

… corresponds to finding the location of r_i that minimizes the Qzn error → answer is: $r_i = \frac{d_{i+1} + d_i}{2}$ → unsurprising for uniform P(x) scenario!

- **Optimal d_i**
  - Finding optimum locations of decision levels → complex maths
    - E.g. Lagrange Multipliers used to minimize expression for **d_i**

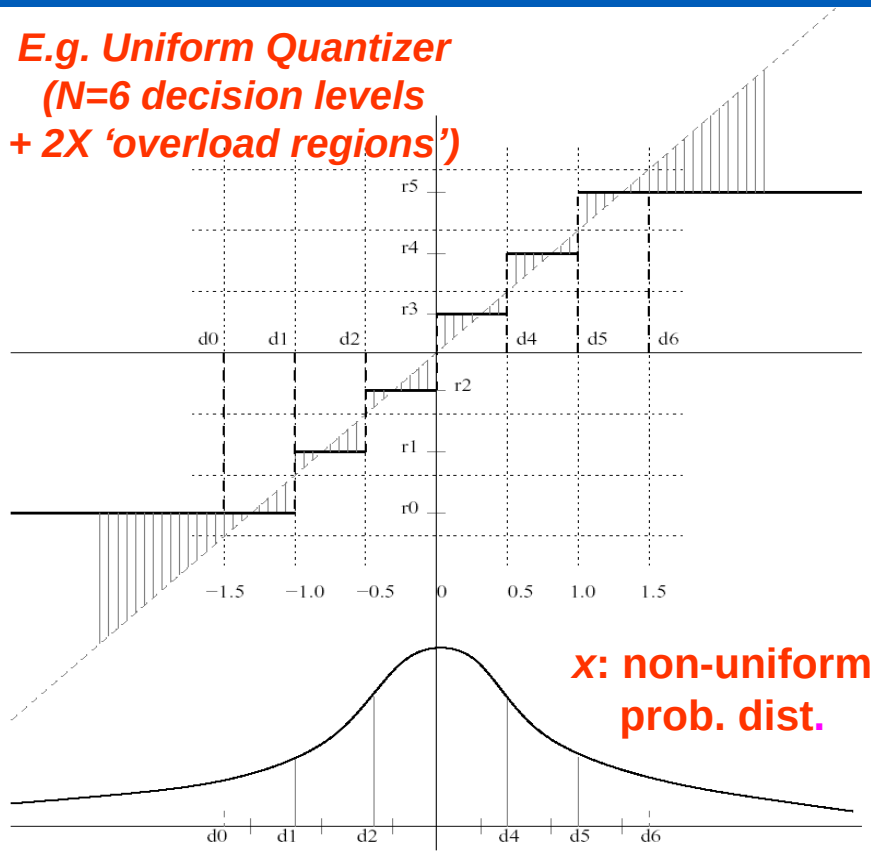
6.4.1 *Uniform Quantizers: Non-Uniform P(x)*

- **N.B. Sometimes required (e.g. implementation)…**
  - An input signal *x* with non-uniform probability distribution

    → quantized by uniform qzr (i.e. equally spaced decision levels), with an even number of decision levels (*N*)
- **What's the error involved?**
  - May be shown (see notes)…

$$e_q = \sum_{i=1}^{N-2} \int_{d_i}^{d_{i+1}} (x - r_i)^2 \, P(x) \, dx + 2 \int_{d_{N-2}}^{\infty} (x - r_{N-1})^2 \, P(x) \, dx$$

E.g. Uniform Quantizer (N=6 decision levels + 2X 'overload regions')



6.4.1 *Uniform Quantizers: Non-Uniform P(x)*

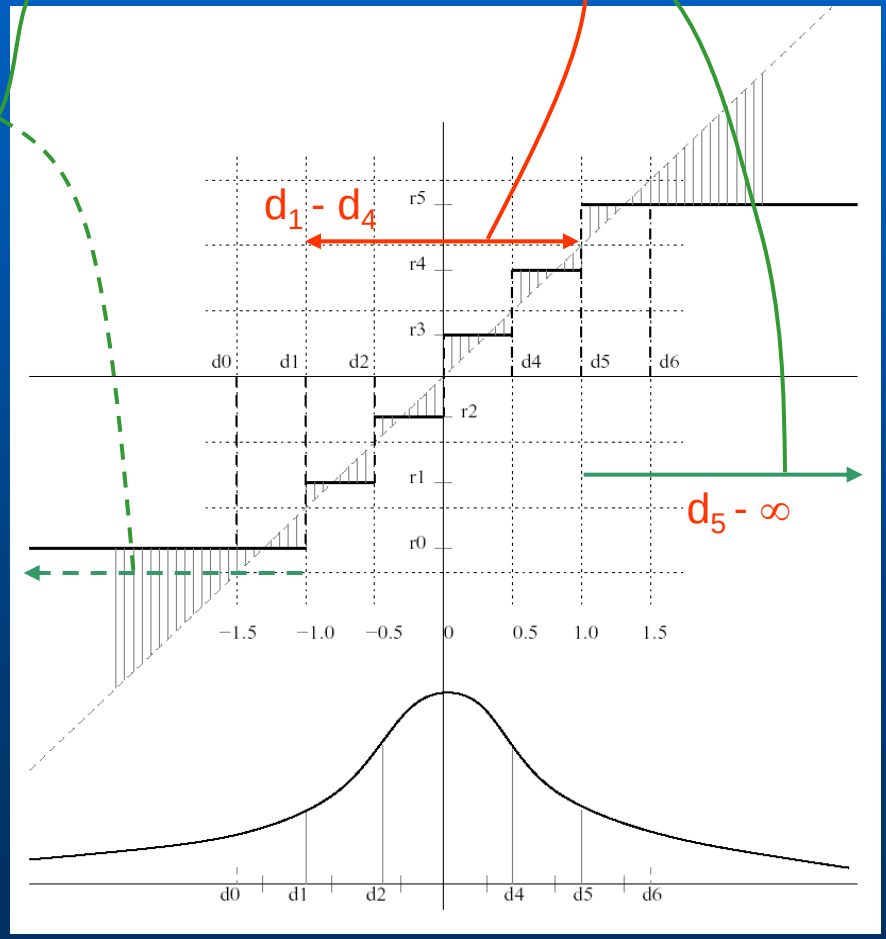
$$e_q = \sum_{i=1}^{N-2} \int_{d_i}^{d_{i+1}} (x - r_i)^2 \, P(x) \, dx + 2 \int_{d_{N-2}}^{\infty} (x - r_{N-1})^2 \, P(x) \, dx$$

- 1st term involves counting from 2nd to the 2nd-last Qzn band
- 2nd term represents error arising from first & last band, plus extent of distribution not covered by any bands ('*overload regions*')

The overload rules

→ i.e. all $x > d_N$ quantized to previous reconstruction level r_{N-1}

→ all $x < d_0$ quantized to first reconstruction level r_0



6.4.1 *Non-Uniform Distribution Quantizers*

- **So far looked at cases where…**
  1. P(x) uniform across entire range of input signal
     Optimum r_i & d_i obvious
  2. P(x) uniform within each quantisation band…
     - Optimum r_i & d_i may be found via error expression minimization
  3. When assumption of uniformity does not hold at all…
     - Numerical methods required to find optimum locations of r_i & d_i

- **However…**
  - Many signals exhibit standard distributions (Gaussian, Laplacian, Rayleigh, etc.)
  - Have corresponding tables of optimum d_i and r_i locations for given number of decision levels
    - These solution sets are known as *Lloyd-Max Quantisers* (a.k.a. *Optimum Mean-Square Quantizers) for Non-Uniform Distributions*

6.4.1 *Companding Quantizers*

- **So far;** Quantizer design based on minimizing MSE
- **Alternative to reduced-error quantizer design: 'Compandors'**
  - Scalar values *x* are first transformed, where non-linear tranformation used corresponds to prob. distribution of *x*
  - Idea is this results in transformed domain values that then have uniform probability distribution
  - Resulting values are then quantized by a uniform distribution qzr
  - Quantized values then inverse-transformed by inverse of the original transformation
  - Performance comparable to that of optimal MSE Qzrs

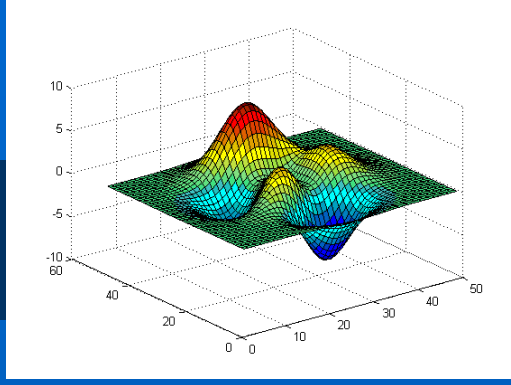
6.4.1 *Overall Coding Package*

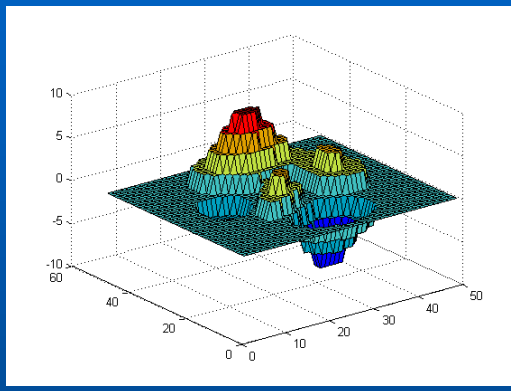
- **Note: Consider input signal *x* exhibiting…**
  - **1. Gaussian prob.distribution**
    - Uniform Qzr → SNR approx. 2dB poorer than Optimum MS Qzr
  - **2. Laplacian prob. distribution**
    - Uniform Qzr → SNR approx. 4dB poorer than Optimum MS Qzr
  - → May be led to think optimum uniform qzrs are no use!
- **It turns out…**
  - An optimum uniform Qzr + entropy coding (e.g. Huffman)
    - Gives lower distortion than optimum MS Qzr without entropy coding!
- **Hence → must not consider Qzrs in isolation!**
  - i.e. consider as part of an **overall** 'rate-distortion optimized package'

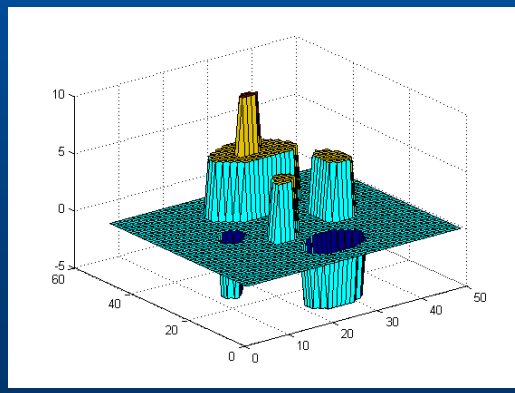
6.4.1 *Quantizers For Visual Data*

- **So far** → discussed *Scalar Quantization*
  - i.e. how we typically limit the precision of the set of values assumed by a single input variable, ***x***
- **Relevant issue for us**
  - Quantizing image pixel samples!
  - What scheme suitable for image data?
- **For the initial digitization of pixel values…**
  - Lower limit on qzn noise is the onset of '*contouring*' (differences between successive reconstruction levels become large and visually noticeable)
  - Uniform qzr → contouring @ 6 bits per pixel or less
  - In practice → 8-bit uniform quantization typically used (256 levels)

6.4.2 Vector Quantization







- **N.B. Image constitutes 2-D matrix of individual scalars**
  - **Use individual quantizer for each pixel?**
- Actually more efficient to first group scalars into 'vectors' and perform qzn at the 'vector' level:
  - i.e. consider *N* data sam. as el. of an *N*x1 vector
    - corresponds to single *point* in an *N*-dim space
  - Then imagine space partitioned into dec. *regions*
    - Input vectors transformed into quantized vectors at the output

→ **Turns out that overall average quantization error introduced to the system is reduced compared to processing scalar-by-scalar!**

→ **Hence VQ sometimes used instead**

Summary: Quantization

- Scalar Quantisation → Challenge of Quantizer design (positioning d_i and r_i)
- Optimum MS Quantizers: *P(x)* assumed constant over each decision band
  - How to derive optimum positions for d_i & r_i
- Uniform Distribution Quantizers – P(x) assumed uniform over input range
  - Expression for optimum SNR (d_i same width, r_i equally spaced)
- Non-Uniform Distribution Quantizers - no *P(x)* uniformity assumed
  - Numerical methods to find optimum
  - Known solutions exist for standard *P(x)*
- Companding Quantizers
  - Transform to uniform *P(X)*, then quantize
- Uniform Quantizers on Non-Uniform *P(x)*
  - Discussed error introduced
- Also: Quantizers for image data; Vector quantization

6. Transform-Based Coding

6.5 Transform Coding

- **So far…**
  - Looked at Transforms & Quantization as independent concepts
    → Now need to consider them together!

- **Outline of overall transform coding scheme…**
  1. Data values grouped into blocks
  2. Data blocks transformed as individual units
     - Transformation scheme should exhibit good energy compaction
       - Concentrates signal energy in as few transform coeffs as possible
  3. Each transform coefficient quantized - according to #bits it is allocated & chosen quantizer ($\mathbf{d}_i$ & $\mathbf{r}_i$ positions)

We look at this now

Q/ Given overall bit-quota for block → how is it determined how many bits allocated to each coefficient? Equal? Non-Equal?

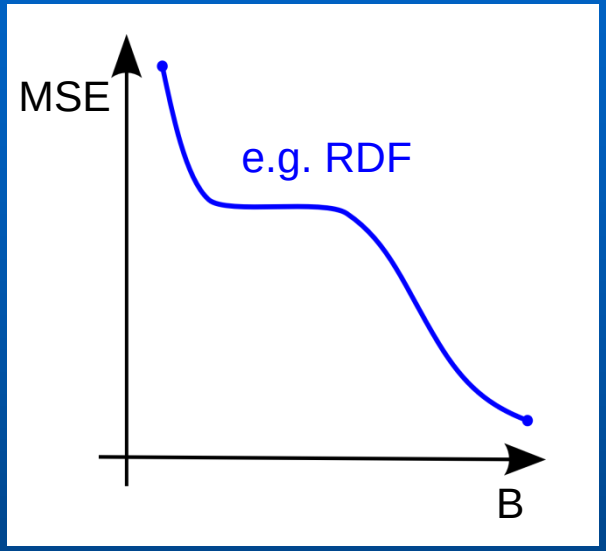
A/ Allocations should focus on minimizing the difference between original coeff values and their reconstructed (quantized) values

6.5.1 Bit Allocation Algorithms

$$n_{av} = \frac{1}{N} \sum_{i=0}^{N-1} n_i$$

- **Consider quantizing block of 8x8 image coeffs…**
  (e.g. 8x8 (length-64) DCT coeff block)
  - Coeffs will naturally have differing prob distns and/or variances
  - Hence each coeff should…
    - be quantized with its own specific quantizer
      - i.e. $\mathbf{d}_i$ and $\mathbf{r}_i$ positions specific to its case
    - have its own particular bit allocation ($\boldsymbol{n}_i$) according to its need
- So, given an average bit rate (n_{av}) limit for the overall scheme
  → uniform bit allocation across all coeffs not necessarily optimal!
    - Hence the challenge of optimal bit-allocation
      → corresponds to finding individual bit allocations ($\boldsymbol{n}_i$) for each coeff that minimize the average distortion…

6.4.1 *Rate-Distortion Function (RDF)*



- **RDF: Important concept in relation to bit-allocation**
  - Represents the MSE of the qzr as a function of the number of bits (*B*)
    - Usually of the form… $f(B) = a2^{-bB}$

      … i.e. a *monotonically decreasing* function of *B*
  - Note1: Intuitive since #bits corresponds directly to #reconstruction levels
  - Note2: For both *Optimum MS Qzrs* and *Uniform Qzrs* applied to Gaussian distributed data, resulting RDFs well known…

i.e. using a partic. Qzr → tells the expected MSE for given #bits

Quantizer	$0 \leq 2^B < 5$		$5 \leq 2^B < 36$		$36 \leq 2^B < 512$	
	a	b	a	b	a	b
Mean square Gaussian	1	1.5047	1.5253	1.8274	2.2573	1.9626
Optimum uniform Gaussian	1	1.5012	1.2477	1.6883	1.5414	1.7562

6.5.1 Bit Allocation Algorithms

- **Expressing average distortion…**
  - Previously derived expressions for distortion introduced by quantizing a single scalar i.t.o. d_i & r_i
  - In quantizing a set of coefficients, can express average distortion introduced i.t.o. each quantizer's RDF as follows:

$$e_q = \frac{1}{N} \sum_{i=0}^{N-1} \sigma_i^2 \, f(n_i)$$

σ_i^2 is the variance of the transform coeff X_i
$f(n_i)$ is the value of RDF for n_i
n_i is the #bits allocated to X_i

  - So, distortion introduced by quantising a single coeff X_i
    = its variance multiplied by value of its quantizer RDF for n_i

$$D_i(n_i) = \sigma_i^2 \, f(n_i)$$

6.5.1 Bit Allocation Algorithms (cont.)

- **Example: Deriving optimal bit allocation for two coeffs**
  - X_0 and X_1 are two coeffs with variances σ_0^2 and σ_1^2 respectively
  - We know → Distortion introduced by quantizing a single coeff…

$$D_i(n_i) = \sigma_i^2 \, f(n_i)$$

  - Therefore → average distortion of their quantizing scheme given by…

$$D_{av} = \frac{1}{2}\left[D_0(n_0) + D_1(n_1)\right] = \frac{1}{2}\left[\sigma_0^2 \, f(n_0) + \sigma_1^2 \, f(n_1)\right] \quad \bigstar$$

  - And average bitrate of the scheme is obviously the mean of the two individual bit allocations…

$$n_{av} = \frac{1}{2}\left[n_0 + n_1\right]$$

$$n_{av} = \frac{1}{2}[n_0 + n_1]$$

$$D_{av} = \frac{1}{2}\left[D_0(n_0) + D_1(n_1)\right] = \frac{1}{2}[\sigma_0^2 f(n_0) + \sigma_1^2 f(n_1)] \quad \bigstar$$

6.5.1 Bit Allocation Algorithms (cont.)

- **Example: Deriving optimal bit allocation for two coeffs (cont.)**
  - Optimising: need to find least distorting division of n_{av} between the two coeffs (e.g. $\boldsymbol{n_{av}=5}$; So n_0=4 & n_1=6 ? Or n_0=3 & n_1=7 ? Etc.)
    - Write n_1 in terms of n_{av} and n_0 … $n_1 = 2n_{av} - n_0$
    - Substitute this into expression for D_{av} (★) then set… $\frac{d\,D_{av}}{d\,n_0} = 0$
      - i.e. minimizing the average distortion w.r.t. the bit allocation n_0
    - Result is… $n_0 = n_{av} + \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}, \quad n_1 = n_{av} - \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}$

RDF form: $f(B) = a2^{-bB}$

N.B. Plugging into (★) → gives min. average distortion (i.t.o. n_{av}) of…

$$D_{av,min}(n_{av}) = \sigma_0\,\sigma_1\,f(n_{av})$$

6.5.1 Bit Allocation Algorithms (cont.)

- **Interpretation of result…**
  - Now have expressions for optimal division of bits i.t.o. their variance…

$$n_0 = n_{av} + \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}, \quad n_1 = n_{av} - \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}$$

  - i.e. #bits allocated to each coeff will determined by respective variances
    - e.g. high variance coeff → more bits than the lower variance coeff
  - Hence → signal energy more efficiently represented than with uniform bit allocation

- **Recall:** Advantage of transform coding…
  - Purposely concentrates energy in some coeffs at the expense of others giving them larger variance ('decorrelation' of original data)
  - Variance-based bit allocation exploits this nicely! ☺

$$D_{av} = \frac{1}{2}\left[D_0(n_0) + D_1(n_1)\right] = \frac{1}{2}\left[\sigma_0^2\, f(n_0) + \sigma_1^2\, f(n_1)\right] \quad \bigstar$$

6.5.1 Bit Allocation Algorithms (cont.)

- **Quantifying variance-based Vs uniform allocation…**
  - Compare variance-weighted approach with equal bit assignment ('PCM')
  - Already derived expression for the minimized average distortion of the optimal variance-based allocation…

$$D_{av,min}(n_{av}) = \sigma_0\, \sigma_1\, f(n_{av})$$

  - For PCM → each coefficient assigned same #bits
    - i.e. in ★ n_0 *and* n_1 *both* = n_{av}, giving average distortion for PCM…

$$D_{av,PCM}(n_{av}) = \tfrac{1}{2}\left[\sigma_0^2 + \sigma_1^2\right] f(n_{av})$$

  - Can quantify gain derived from choosing variance-weighted allocation over uniform by examining ratio of the average distortions…

$$\frac{D_{av,PCM}}{D_{av,min}} = \frac{\frac{1}{2}\left[\sigma_0^2 + \sigma_1^2\right]}{\sigma_0\, \sigma_1} = \frac{\frac{1}{2}\left[\sigma_0^2 + \sigma_1^2\right]}{\left[\sigma_0^2\, \sigma_1^2\right]^{\frac{1}{2}}}$$

σ = std. dev
σ^2 = variance

6.5.1 Bit Allocation Algorithms (cont.)

- **Interpreting this result…**

$$\frac{D_{av,PCM}}{D_{av,min}} = \frac{\frac{1}{2}[\sigma_0^2 + \sigma_1^2]}{\sigma_0\,\sigma_1} = \frac{\frac{1}{2}[\sigma_0^2 + \sigma_1^2]}{[\sigma_0^2\,\sigma_1^2]^{\frac{1}{2}}}$$

  - Very important result in transform coding theory
  - Corresponds to expression of the ratio of the *arithmetic mean* to the *geometric mean* of the coefficient variances
  - Suggests that…
    - At worst, transform coding will be as good as PCM
      - i.e. gain = 1 for $\sigma_0 = \sigma_1$
    - With the coding gain improving as the coeff variances deviate further from uniformity
      - i.e. gain > 1 for $\sigma_0 \neq \sigma_1$

Justifies variance based allocation approach!

6.5.1 Bit Allocation Algorithms (cont.)

- **Generalised expressions**
  - Moving away from the 2-coeff example…
  - Now considering N coeffs…
  - Same issue → given an average bitrate n_{av}, find the minimum distortion & optimal division of bits
  - Result is just a generalisation of the N=2 case
    - i.e. before we had…

$$n_0 = n_{av} + \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}, \quad n_1 = n_{av} - \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}$$

    - Now we have…

$$n_i = n_{av} + \frac{1}{b}\log_2\frac{\sigma_i^2}{\left[\prod_{j=0}^{N-1}\sigma_j^2\right]^{1/N}}$$

6.5.1 *Practical Bit Allocation*

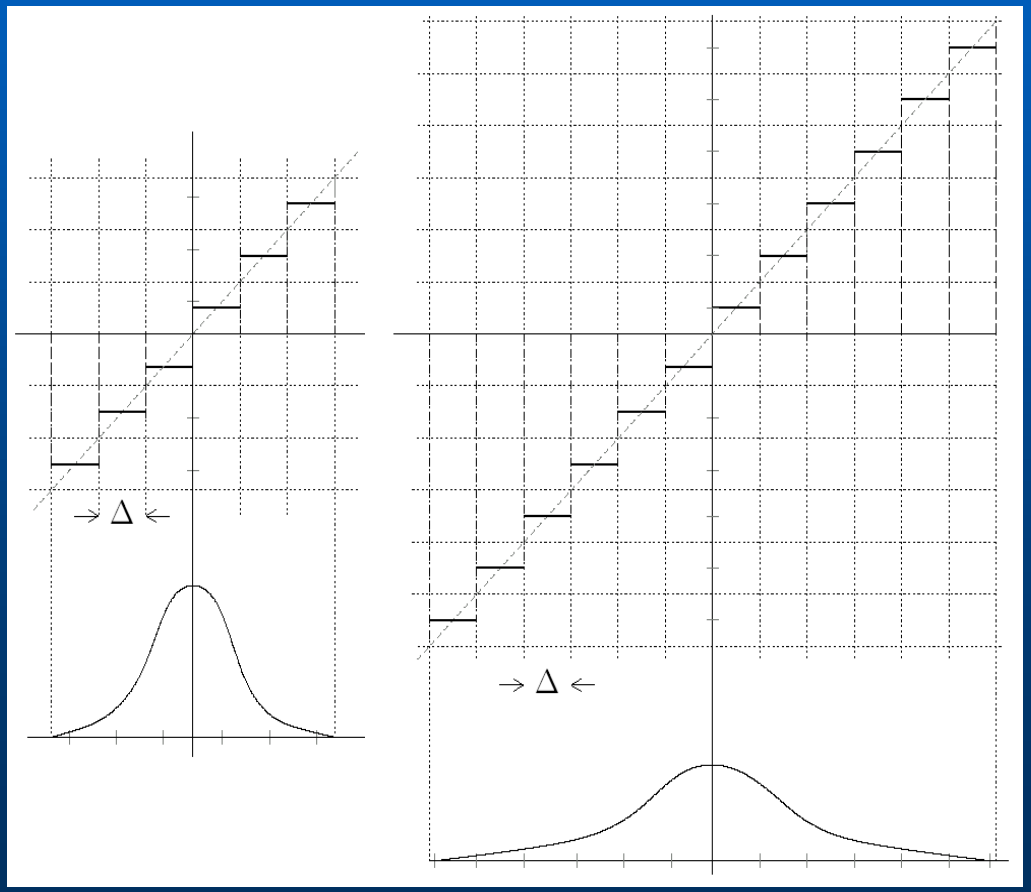
- **Note…**
  - Optimal bit allocation scheme just outlined may give **non-integer** (or even **negative**) values for n_i !
- **In practice: Non-integer bit allocations…**
  1. May be simply rounded up
  2. May be implemented (extra qzn levels representing fractional parts)
- **Negative bit allocations…**
  - May occur when the variance of a coeff is significantly smaller than the geometric mean of all the coeff variances (mathematical phenomenon)
  - Need to be set to zero
  - Then the other coeff bitrates adjusted accordingly
    - i.e. lowered to compensate for the raising of the negative bitrate (→ to maintain the target average bitrate n_{av})

6.5.1 *Practical Bit Allocation (cont.)*

- **Other points…**
  1. Coeffs allocated small bitrates
     - Considering the overhead involved in coding these…
       - It may be more efficient to set to zero
         - Then reallocate spare bits to other more 'needy' coeffs
  2. Coefficients with very large variance…
     - May be allocated bitrates exceeding the perceptual requirements of the human visual system
       - May decide to trim back if exceeds some limit n_{max}
         - Then reallocate to other needy coeffs
- **In short…**
  - In a real-world system…
    - A supplementary procedure is required to allocate actual bitrates to coeffs (i.e. beyond that prescribed by the optimal bit allocation mathematics)

6.5.1 Bit Allocation Algorithms (cont.)

- **Illustration:** Optimal bitrate quantizers for two coeffs
  - Non-uniform prob. distn for each coeff
  - One coeff larger range than the other
  - Step-sizes equal to ensure identical distortion introduced across both coeffs
  - Unequal bit allocation corresponding to extra quantization steps needed for coeff with larger range



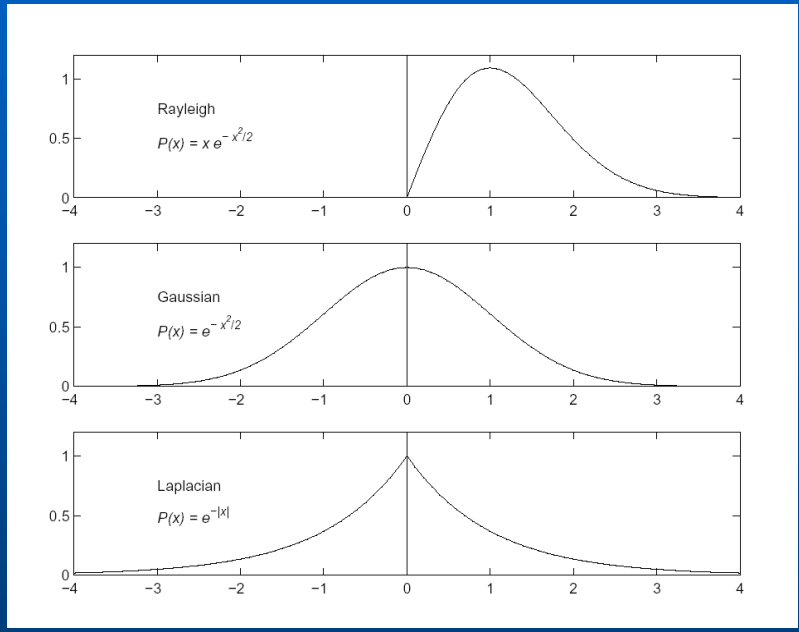
6.5.2 Transform Coding of Images

- **Our focus here → transforming images!**
- **In working with image data…**
  - Various options on block sizes
    - 4x4, 8x8, 16x16, etc.
      - Represent various tradeoffs of hardware/algorithmic complexity and coding efficiency

    → 8x8 block size the most common choice
- **N.B. Various bit allocation algorithms exist for implementing the practical division of bits for transformed image data (e.g. DCT coeffs)**
  - Aim to approximate the theoretical optimal allocations in real world scenarios

6.5.2 *Quantizer Design*

- **Quantizer types for images…**
  - E.g. DCT coeff prob. distributions…
    - DC coeffs are non-negative
      - Often modeled by a Rayleigh or one-sided Gaussian/Laplacian
    - Remaining (AC) coeffs usually have zero mean
      - Often modeled by symmetric distributions (e.g. Gaussian/Laplacian)



  - So, typically one quantizer will be needed for the DC coefficient (unique type)
  - And up to 8 different quantizers to process the remaining coeffs
    - 8-bit, 7-bit, 6-bit, 5-bit,…
      (coeffs with similar variance share quantizers)

Coeffs allocated anything from 8 → 0 bits

6.5.2 *Zonal Coding*

- **N.B. For DCT transformed pixel data…**
  - Typical for coeffs sharing similar variance characteristics (quantization requirements) to spatially occur together ('zones')
  - Phenomenon sometimes exploited for the process of bit allocation
    - i.e. different 'zones' assigned <u>fixed bit allocation</u>

      E.g. **8x8 DCT Zonal Coding**
      #bits used to quantize coeffs fixed according to 'zone' of coeff

  - Known as 'zonal coding' or 'zonal bit-allocation'

8	8	6	6	6	4	4	4
8	6	6	6	4	4	4	2
6	6	6	4	4	4	2	2
6	6	4	4	4	2	2	2
6	4	4	4	2	2	2	0
4	4	4	2	2	2	0	0
4	4	2	2	2	0	0	0
4	2	2	2	0	0	0	0

6.5.2 *Threshold Coding*

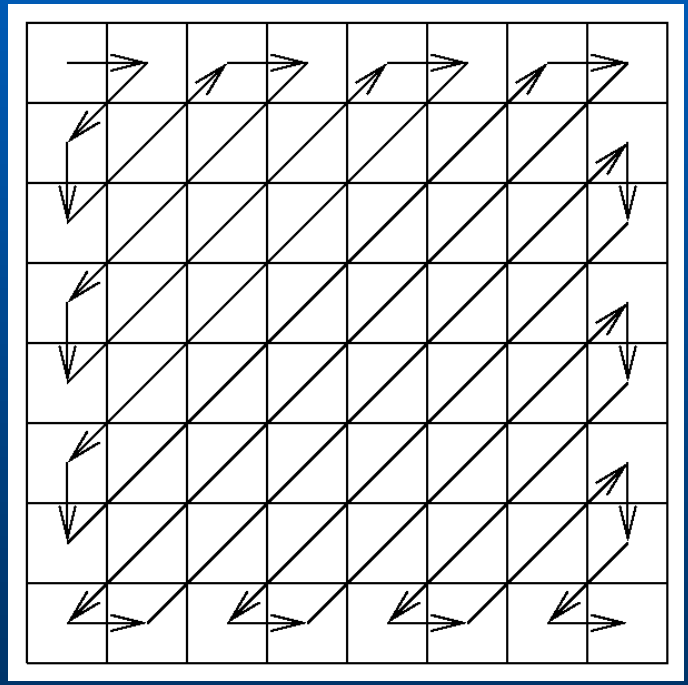
- **Zonal coding…**
  - Coeffs assigned bit allocation decided in advance
    - Based on typical variance characteristics
- **Another filter = 'Threshold Coding'**
  - Makes on-the-fly decisions about setting particular coeffs to zero
    - Based on amplitude of each coeff
- **E.g. 8x8 DCT transformed block** →
  - '1' indicates that the amplitude of the corresponding coeff was sufficiently large so as to be processed further
  - '0' indicates coeffs whose amplitude was below the threshold

    → Will not be processed further

1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	0
1	1	1	0	0	1	1	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0

6.5.2 *Zig-Zag Scanning*

- **Consider reading threshold-coded DCT coeffs into a bitstream…**
  - Even though the coeffs are uncorrelated
    - They are statistically related!
      - i.e. if horiz. freqs are present, then diagonal/vertical energy at the same freq is also likely to be present.



- **Hence…**
  - Makes sense to keep statistically related coeffs that close together when being processed
  - i.e. improves the efficiency of subsequent RLE, which exploits long runs of zeroes
    - Common when large #high freq coeffs get quantized/thresholded

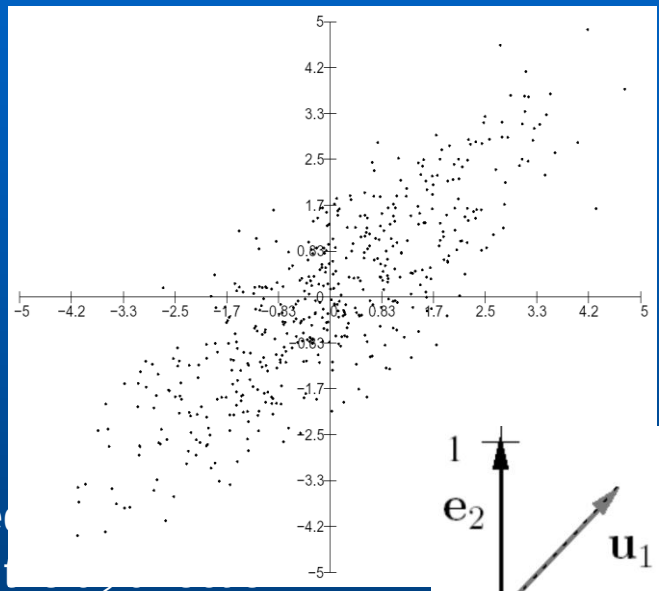
Example:

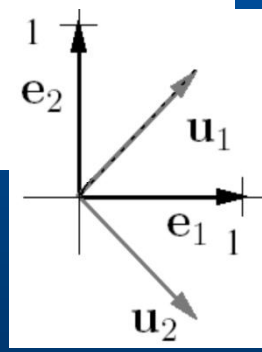
elm.eeng.dcu.ie/~ee554/java/DCT/

End of Chapter 6!

6.1.2 Simple Discrete Lin. Transf. (cont.)

- **Geometrical Interpretation (cont.)**
  - Suppose (e.g.) original data samples come from larger low-pass filtered dataset…
    - Large magnitude changes between samples eliminated (strong correlation between successive samples $x_n \approx x_{n+1}$)
      → Distribution, in general, occupies the diagonal $x_1 \approx x_2$
    - Overall:
      - Variance along e_1 & e_2 approximately e
      - Variance along the u_1 direction > along
    - Data is said to be '*decorrelated*' in the transform domain
      - Energy concentrated ('compacted') one one component at the of the other
      - E.g. of types of patterns only observable in different domain





- E.g. N=2…

6.5.1 *Practical Bit Allocation (cont.)*

- **Practical Bit Allocation: Example Algorithms…**

- **Example 1: Fractional bit allocations allowed…**

  Starting with optimal division of bits across coeffs…

  – Generate an excess bitrate by…
    - Setting some (small bitrate) coeffs to zero
    - Retrieving bits by trimming back allocation for coeffs given greater than required accuracy
  – Simply divide the excess evenly (hence the fractions!) across all other coeffs with $n_i < n_{max}$
  – Then, if these coeffs are brought above n_{max} → repeat

6.5.1 *Practical Bit Allocation (cont.)*

- **Practical Bit Allocation: Example Algorithms…**

- **Example 2: Fractional bit allocations <u>not allowed</u>…**
  - First set all the bit allocations for all coeffs to zero…
  - Then determine for which of the coeffs an additional bit will make the most difference, and assign the bits on a coeff-by-coeff basis
    - i.e. iterative procedure based on comparing the respective distortion decrements obtained for each coeff when assigned an extra bit
  - Finished when overall allocation realises n_{av} limit

6.5.1 *The equal distortion rule*

- So far considered OPTIMAL division of n_{av} bits as that which minimizes average distortion introduced across all coeffs being quantized
  - Actually corresponds to introducing **SAME** amount of distortion to each coeff
  - i.e. considering two coeffs X_0, X_1, $n_{av} = 4$

$$D_0(n_0) = D_1(n_1) = D_{av,min}(n_{av})$$

    - Optimal bit allocation of (e.g.) $n_0=3$, $n_1=5$ represents X_0 and X_1 being distorted to the same degree (as this relates to introducing a minimal average distortion)

- **N.B. couldn't be any other way, without contradicting optimality!**
  - Imagine X_0 and X_1 exhibited DIFFERENT levels of distortion, e.g. 2.5 and 6.0 (avg = 4.25) for $n_0=3$ and $n_1=5$
  - This means that for a one bit swap (i.e. $n_0=2$ and $n_1=6$), giving same n_{av}, we could get new distortion levels of (e.g.) 2.0 and 5.0 (unequally affected due to differences in variance)
  - This would mean even lower avg distortion (=3.5) for same n_{av} → implies original allocation not optimal!

6.5.1 Bit Allocation Algorithms (cont.)

- **So, optimal division of n_{av} bits…**
  - Each coeff equally distorted
  - i.e. a uniform qzn error introduced across each coeff

- **Recall:** uniform qzrs typically used even though non-uniform coeff probability distribution
- **Result from earlier:**
  - The qzn error variance of a uniform quantizer = $\Delta^2/12$
  - N.B. 'qzn error variance' actually measure of distortion
  - Hence distortion produced by uniform qzr = $\Delta^2/12$
    → determined by the decision-level width ('*step-size*') Δ

- So, quantizing set of optimally bit-allocated coeffs with uniform quantization → **qzrs have same step size!**