Previously we chose a DCT (the 2-dimensional DCT where we multiply a DCT from both sides) as a transform matrix. But the next question is, how do we choose a suitable, or optimum, transform matrix, out of the many possible? The ultimate goal of a video coder is to save bits, so we would like to have a transform matrix, which leads to a minimized number of bits for describing or coding an image. In a complete coder we apply irrelevance reduction (the properties of the eye), and redundancy reduction (properties of the source). Since this becomes somewhat difficult to solve for a simple transform matrix, we simplify our problem by looking at the properties of the source only (meaning redundancy reduction only or lossless coding). To minimize the needed bits for the lossless case is still mathematically a hard problem.

Hence we reformulate our problem: we would like the subband signals to be statistically independent of each other. Statistical independence of signals x and y means: probability $p(x)$ is the same as this probability conditioned to the other signal $y$: $p(x) = p(x|y)$,
which means **x contains no information about y**, which is what we actually want to avoid additional redundancies in our transmitted signal, to transmit information only once. Independence leads to uncorrelatedness, but uncorrelatedness leads not necessarily to independence, only for **gaussian distributed signals**. We obtain gaussian signals for instance if we add up many more or less random signals (law of large numbers).

But this statistical independence is again hard, and hence we choose the easier to obtain **uncorrelatedness** between the different subbands. We again reformulate our goal that we would like to have subbands \( y_k(m) \), which are **uncorrelated** to each other (instead of the lowest number of bits). This means that 
\[
E(y_k(m) \cdot y_l(m)) = 0 \quad \text{for} \quad k \neq l,
\]
meaning different subbands shall be uncorrelated, meaning each subband should carry new information.

In this way we can argue that we have minimized redundancy between the subbands in some respect (for gaussian signals), to obtain some sort of information separation between the subbands. In this way we have made this problem **mathematically more tractable**.
First assume a **one dimensional** signal $\mathbf{x}$, to simplify our notation. This signal is again divided into blocks of length $N$, obtaining blocks $\mathbf{x}(m)$, with block index $m$, and then each block is transformed using our transform matrix $\mathbf{T}$, 

$$
\mathbf{y}(m) = \mathbf{x}(m) \cdot \mathbf{T}
$$

where $\mathbf{y}(m)$ is a row vector. These vectors can now be assembled into a matrix, where the block index $m$ runs vertically, 

$$
\mathbf{Y} = 
\begin{bmatrix}
\mathbf{y}_0(0) & \mathbf{y}_1(0) & \cdots & \mathbf{y}_{N-1}(0) \\
\mathbf{y}_0(1) & \mathbf{y}_1(1) & \cdots & \mathbf{y}_{N-1}(1) \\
\mathbf{y}_0(2) & \cdots & \cdots & \cdots \\
\vdots & \cdots & \cdots & \cdots \\
\mathbf{y}_0(L-1) & \cdots & \cdots & \cdots 
\end{bmatrix}
$$

$\mathbf{Y}$ is a matrix of size $L \times N$, where $L$ is the number of blocks in the signal. The columns of this matrix contain the signals of the different subbands (one column is one subband, first column subband $k=0$, second column subband $k=1$..., so vertically we have the subband view). We would like to compute the correlations between the different subbands (at time lag 0 to again simplify matters), to obtain a transform matrix where this will be zero. We now obtain the correlation matrix of the subbands by multiplying the different subband column vectors (column vector of subband $k=0$ with itself results in its signal power, product of subband $k=0$ with $k=1$ results in their cross-
correlation with lag 0...), if we multiply the transpose with the matrix (in this way each subband column becomes a row from the left, which we then multiply with each subband column on the right),

\[
Y^T \cdot Y = \begin{bmatrix}
y_0(0) & \ldots & y_0(L-1) \\
\vdots & \ddots & \vdots \\
y_{N-1}(0) & \ldots & y_{N-1}(L-1)
\end{bmatrix} \begin{bmatrix}
y_0(0) & \ldots & y_{N-1}(0) \\
\vdots & \ddots & \vdots \\
y_0(L-1) & \ldots & y_{N-1}(L-1)
\end{bmatrix}
\]

which we name \( A_{yy} \),

\[
A_{yy} := Y^T \cdot Y \quad \text{(eq. 2)}
\]

The element at position i,j of matrix \( A_{yy} \) is

\[
\sum_{m=0}^{L-1} y_i(m) \cdot y_j(m)
\]

Hence \( A_{yy} \) is a NxN matrix, which now contains the autocorrelation and cross correlations of the different subbands at lag 0. On the resulting matrix \( A_{yy} \), each diagonal element contains the power of the corresponding subband, and the off-diagonal elements contain the cross-correlation of the corresponding subbands, which we want to be zero.

To obtain our goal of uncorrelated subbands, this correlation matrix needs to be a diagonal matrix.

\[
A_{yy} = \begin{bmatrix}
\sigma_0^2 & 0 & 0 \\
0 & \ddots & \vdots \\
0 & \ldots & \sigma_{N-1}
\end{bmatrix}
\]
So we need to find a solution which satisfies this goal. We stack our signal into the size LxN matrix \( X \) (as above),

\[
X = \begin{bmatrix}
  x(0) & \ldots & x(N-1) \\
  x(N) & \ldots & x(2N-1) \\
  x(2N) & \ldots & x(3N-1) \\
  \vdots & \ddots & \vdots \\
  \vdots & \ddots & \vdots 
\end{bmatrix}
\]

In this way we obtain the subband values as \( Y = X \cdot T \)

plugging this result into eq. (2) we obtain

\[
A_{yy} = Y^T \cdot Y = (X \cdot T)^T \cdot X \cdot T = T^T \cdot X^T \cdot X \cdot T
\]

Here we can observe that \( X^T \cdot X \) is the autocorrelation matrix of our signal blocks in the signal \( x \),

\[
A_{xx} := X^T \cdot X
\]

(Remember that the autocorrelation function of a signal \( x \) is defined as

\[
r_{xx}(n) = \sum_{n'} x(n') \cdot x(n' - n)
\]

, and the cross correlation between two signals \( x \) and \( y \) is

\[
r_{xy}(n) = \sum_{n'} x(n') \cdot y(n' - n)
\]

corresponds to the usual vector multiplication). Observe that \( X \) contains the blocks of the signal \( x \), which we here assume to be one-dimensional, for simplicity. Each row is one block of our 1-dimensional signal, for instance a block of a line of our image. This means that
each column corresponds to one position in each block, over the block index \( m \). In effect this is a **downsampled** signal \( x \) along each column of matrix \( X \), where each column corresponds to a different **phase** (instead of a different subband) of the downsampled signal. This means the correlation matrix \( A_{xx} \) corresponds to the correlations of the different downsampled versions of \( x \) (usually there will be a lot of correlation).

Remember that our goal was to make \( A_{yy} \) a diagonal matrix, which is equivalent to the subband signals being uncorrelated.

Observe that we now have

\[
A_{yy} = T^T \cdot A_{xx} \cdot T
\]

Given matrix \( A_{xx} \) we need to determine the **transform matrix** \( T \) such that \( A_{yy} \) becomes a **diagonal matrix**. The advantage now is that \( A_{xx} \) is much smaller than \( X \), it has a finite dimension, it is quadratic with dimension \( NxN \), and hence we can easily find a solution. Now we can see that if we only had one transform matrix (from just one side), it would only need to be the inverse of \( A_{xx} \) in order to get a diagonal matrix, since we would obtain the identity matrix, which is also a diagonal matrix. But we have the transform from both sides, so that doesn't work. So let's take a look at the
Eigenvectors of $A_{xx}$. An Eigenvector has the property that a multiplication of it with the matrix results in the same vector, but multiplied with the Eigenvalue:

$$A_{xx} \cdot e_0^T = \lambda_0 \cdot e_0^T$$

Also, since our matrix is symmetric, the Eigenvectors are orthogonal to each other, meaning $e_j \cdot e_k^T = 0$ for $j \neq k$, and $e_i \cdot e_i^T = 1$

Hence, we construct our transform matrix $T$ as having all Eigenvectors $e_k$ of $A_{xx}$ as its columns,

$$T = \begin{bmatrix} e_0^T & e_1^T & \ldots & e_{N-1}^T \end{bmatrix}$$

(important step!) This means that if we multiply this matrix from the right side to $A_{xx}$ we obtain the same Eigenvectors, but multiplied with their Eigenvalues $\lambda_k$.

$$A_{xx} \cdot T = \begin{bmatrix} \lambda_0 \cdot e_0^T & \lambda_1 \cdot e_1^T & \ldots & \lambda_{N-1} \cdot e_{N-1}^T \end{bmatrix}$$

(since we had $A_{xx} \cdot e_j^T = \lambda_j \cdot e_j^T$). Since the $e_j$ are orthogonal to each other, all what is left after the multiplication from the left hand side with $T^T$ (which has the eigenvectors in the rows), is a diagonal matrix! The diagonal elements now contain the Eigenvalues ( $e_j \cdot \lambda_j \cdot e_j^T = \lambda_j$),
$$A_{yy} = T^T \cdot A_{xx} \cdot T = \begin{bmatrix}
\lambda_0 & 0 & 0 & \ldots & 0 \\
0 & \lambda_1 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & \lambda_{N-1}
\end{bmatrix}$$

where $\lambda_j$ are the Eigenvalues of $A_{xx}$.

Observe that here we achieved our goal of a diagonal matrix as $A_{yy}$ for the uncorrelatedness of the subbands, using $T$ as a set of eigen-vectors of $A_{xx}$!

(Observe: If we define $D$ as the diagonal matrix containing the eigenvalues $\lambda_k$, then we get
$$T \cdot D = A_{xx} \cdot T$$
and
$$T \cdot D \cdot T^T = A_{xx}$$
which is the basic property of Eigenvectors, hence we get
$$A_{yy} = T^T \cdot A_{xx} \cdot T = T^T \cdot T \cdot D = D$$
because for the matrix of Eigenvectors we have $T^T \cdot T = I$)

In conclusion: This means we now reached our goal of diagonalizing our cross correlation matrix $A_{yy}$ with a transform matrix which consists of the Eigenvectors of autocorrelation matrix $A_{xx}$ in its columns! This is now the so-called Karhunen-Loeve
Transform (KLT).

In this way we obtain **one KLT** matrix for the **rows** of our picture, and one KLT matrix for the **columns** (hence two KLT matrices for the entire picture). The KLT for the rows is obtained of assembling **all** the rows of all blocks in a **picture** or set of pictures into the matrix $X$, use it to compute the correlation matrix $A_{xx}$, and then compute the eigenvectors to obtain the KLT as the set of these eigenvectors. Similarly, the **KLT for the columns** is obtained by assembling all columns of all blocks into a matrix $X$.

The problem with the KLT is, that it doesn't have a **fast implementation**, like a DCT, which can be implemented via an FFT. Also, if we want to have the optimal KLT for our image (or set of images), we would have to calculate it first, and then transmit it as **side-information** to the decoder, which would increase the necessary bit rate. This has the consequence that it is usually only a theoretical construct, but not really used in practical coders.

Observe that for the construction of the KLT we need useful, **reliable statistics**, and hence many blocks to compute our correlation matrix.
A_{xx} from. Hence it doesn't make sense to compute the KLT for single block of an image. It makes more sense to compute the correlation matrix from all blocks of an image, or from all blocks of a **collection of images or a video**.

**Python Example for the KLT:**

Take an image and read it into Python:

```python
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
from scipy import signal

pic=plt.imread('IMGP1690.JPG')
plt.imshow(pic)
plt.show()
```
Now convert it to the (2 dimensional) luminance component $y$,

$$Y = 0.299R + 0.587 G + 0.114 B$$

(library pyplot has R at index 0):

$$Y = 0.299 \cdot \text{pic}[:,:,0] + 0.587 \cdot \text{pic}[:,:,1] + 0.114 \cdot \text{pic}[:,:,2]$$

$Y$.shape

#(2448, 3264)

plt.figure()

plt.imshow(Y,cmap='gray')

plt.show()
meaning we have 2448 rows and 3264 columns.
Now we construct the matrix $x$, for blocks of size $N=4$. The dimension of this matrix is 4 columns and $2448 \times 3264 / 4 = 1997568$ rows. We use the command "reshape", which reads out a matrix column wise and puts this in a new matrix with given dimension: $\text{reshape} \ (Y, \ \text{SIZE})$. Since reshape reads out the values column wise, we need to transpose the matrices to read out the rows with length 4:

```python
x=np.reshape(Y,(-1,4), order='C')
```
So here we see that it has the correct size. Next we can compute the autocorrelation matrix
\[ A_{xx} = x^T \cdot x , \]
\[
Axx = \text{np.dot}(x.T, x)
\]
\[
Axx.shape
# (4, 4)
\]

Here we now have the autocorrelation matrix with the correct size (4x4).

Now we can compute the eigenvectors and eigenvalues.

To obtain the eigenvectors we use the command "eig" in Python:
\[
\text{Lambda, T} = \text{LA.eig}(Axx)
\]

where \( V \) contains the eigenvectors as column vectors.

\[
\text{Lambda, T} = \text{LA.eig(Axx)}
\]
\[
\text{Lambda, T}
# (array([ 1.42785375e+11, 5.55630080e+08, 9.00369086e+07, 1.67823115e+07]),
# array([[ 0.49948501, 0.63650916, 0.5080001 , -0.29547709],
#        [ 0.50057713, 0.30241337, -0.49924621, 0.63931363],
#        [ 0.50049634, -0.29275343, -0.49962342, -0.64356452],
#        [ 0.49944037, -0.64629534, 0.4930168 , 0.29965996]]))
\]

Here, T now is our KLT for the rows of our image. Since this is a 4x4 transform matrix, we
obtain 4 subband filters with 4 impulse responses. We can now also plot the 4 impulse responses of our KLT,

```python
plt.figure()
for k in range(0,4):
    plt.subplot(4,1,k+1)
    plt.plot(np.flipud(T[:,k]))
plt.show()
```

(x-axis is sample, y axis is value)

These are the equivalent subband filter impulse responses of our KLT. Observe that it looks like filters with increasingly higher (lower)
frequencies, on top is a low pass filter, below that a band pass, then another band pass, then a high pass. So simply by starting with a natural image has lead us to a subband decomposition, where we divide our frequency domain into different subbands! This also has some similarity to the DCT, and this is why a DCT works.

The frequency responses of these filters are

```python
plt.figure()

for k in range(0,4):
    w,frresp=signal.freqz(T[:,k])
    plt.subplot(4,1,k+1)
    plt.plot(w,20*np.log10(abs(frresp)))

plt.show()
```
Observe that the horizontal axis shows the normalized frequencies between 0 and pi.

Again we can see that the top figure shows a low pass filter, below two band pass filters, and at the bottom a high pass filter.

Now we can verify that the resulting subbands for the rows are indeed uncorrelated or orthogonal to each other. We compute the row-subbands with our KLT,

\[
y_r = \text{np.dot}(x, T)\]
yr.shape
#(1997568, 4)

Observe: each column of yr represents a subband (we have 4 subbands, each with 1997568 samples).

Next compute the correlations between the subbands:

np.dot(yr.T, yr)

array([[ 1.42785375e+11, -1.19530940e-04, -1.55719417e-04, -6.95257859e-05],
       [-1.19530940e-04,  5.55630080e+08, -5.46917661e-05, -7.18245548e-05],
       [-1.55719417e-04, -5.46917661e-05,  9.00369086e+07,  2.76820191e-05],
       [-6.95257859e-05, -7.18245548e-05,  2.76820191e-05,  1.67823115e+07]])

Here we can see that the diagonal elements are of significant size (the autocorrelations of the subbands), and the off-diagonal elements are of insignificant size (the cross-correlations), for all practical purposes they can be neglected. This can also be seen by looking at the ratio of the larger and smaller values. We have a ratio of about $10^{11}/10^{-5}=10^{16}$. This is roughly the computational accuracy of Python and corresponds to 160dB in power! (Since autocorrelation corresponds to a power). This is much more than the attenuation of our filters, and hence can be neglected. This means the subbands are indeed uncorrelated or orthogonal to each other, and
the KLT indeed works!
Next we still need to compute the KLT for the columns,

\[
x_{c} = \text{np.reshape}(Y.T, (-1, 4), \text{order}='C')
\]
\[
x_{c}.\text{shape}
\]
\[(1997568, 4)\]

\[
A_{xxc} = \text{np.dot}(x_{c}.T, x_{c})
\]
\[
A_{xxc}.\text{shape}
\]
\#[(4, 4)]
\[
\lambda_{c}, T_{c} = \text{LA.eig}(A_{xxc})
\]

\[
\lambda_{c}, T_{c}
\]
\#!(array([  1.42755754e+11,   5.64720492e+08,   1.08163754e+08, 1.91860936e+07]),
# array([[ 0.49978517,  0.62897522,  0.51863754, -0.29251869],
#        [ 0.50054465,  0.31400696, -0.51100746,  0.62428043],
#        [ 0.50042801, -0.2980642 , -0.48755008, -0.6504033 ],
#        [ 0.49924107, -0.64571475,  0.48179718,  0.31887662]]))
\]

Compare it with the KLT for the rows,
\[
T
\]
\![array([[ 0.49948501,  0.63650916,  0.5080001 , -0.29547709],
          [ 0.50057713,  0.30241337, -0.49924621,  0.63931363],
          [ 0.50049634, -0.29275343, -0.49962342, -0.64356452],
          [ 0.49944037, -0.64629534,  0.4930168 ,  0.29965996]])]
\]

We can see that the KLT's for the rows and the columns are almost identical for this picture. The slight differences or similarities can also be seen from the correlation matrices for the rows
and columns, Axx

\[
\begin{bmatrix}
3.58726591e+10, & 3.57816851e+10, & 3.55717784e+10, & 3.54121574e+10 \\
3.57816851e+10, & 3.58589123e+10, & 3.57393845e+10, & 3.55700035e+10 \\
3.55717784e+10, & 3.57393845e+10, & 3.58442949e+10, & 3.57714986e+10 \\
3.54121574e+10, & 3.55700035e+10, & 3.57714986e+10, & 3.58719581e+10
\end{bmatrix}
\]

\[
\begin{bmatrix}
3.59124282e+10, & 3.57918248e+10, & 3.55745680e+10, & 3.54153428e+10 \\
3.57918248e+10, & 3.58581365e+10, & 3.57247015e+10, & 3.55362714e+10 \\
3.55745680e+10, & 3.57247015e+10, & 3.5834638e+10, & 3.57445731e+10 \\
3.54153428e+10, & 3.55362714e+10, & 3.57445731e+10, & 3.58431960e+10
\end{bmatrix}
\]

To apply these KLT's to transform an image, we first divide it into blocks \( B \) of size 4x4 (in this case) and then transform each block with the KLT for the columns \( T_c \) and for the rows \( T \), in the encoder with,

\[
y = T_c^T \cdot B \cdot T
\]

In the decoder we get the inverse operations,

\[
B = T_c^{-T} \cdot y \cdot T^{-1}
\]

Since we know that the KLT matrix consists of the eigenvectors of a symmetric matrix (our correlation matrix), we know that these eigenvectors are orthogonal to each other, and hence the inverse is identical to the transpose, as we can see trying it out,

\[
\text{LA.inv}(T) \cdot T \cdot T
\]

\[
\text{array([[ 2.77556766e-16, -3.33066907e-16, -1.11022302e-16, 0.00000000e+00],
             [ 2.44249065e-15, -6.49480469e-15, 1.02695630e-14, -5.66213743e-15],
             [-1.54321000e-14,  2.81966648e-14, -2.92543767e-14, 1.70419234e-14],
             [ 1.57096558e-14, -2.80886425e-14, -1.87627691e-14, 3.09197112e-14]])
\]
Here we see the difference between the inverse and the transpose is zero up the the computational accuracy of Python. Hence we can replace the inversion with a transposition, which is much simpler to compute,

\[ B = T_c^{-T} \cdot y \cdot T^{-1} = T_c \cdot y \cdot T^T \]

This is now the equation for the decoder.
The Discrete Cosine Transform (DCT)

Instead of a KLT, usually a DCT is used. If we look at the DCT transform matrix, we can see that it has some similarity to the KLT of usual natural pictures. This means that the DCT gives us similar compression performance as the KLT, but it has fixed matrix elements which can be standardized, and it has a fast implementation. For those reasons it is usually adopted in video coding standards.

There are actually several types of DCT, and the most commonly used for image and video coding is the so-called DCT Type 2. It is defined with its transform as

\[ y_k = c_k \frac{2}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) \cos \left( \frac{\pi}{N} (n + 0.5) k \right) \]

where \( c_0 = \sqrt{0.5} \), and \( c_k = 1 \) for \( k > 0 \), with \( k = 0, \ldots, N-1 \). Here, \( n \) is our space or time index, and \( k \) is our frequency index. Both run from 0 to \( N-1 \), such that we obtain a square transform matrix \( T \) of size \( N \times N \), with elements

\[ T_{n,k} = c_k \sqrt{\frac{2}{N}} \cos \left( \frac{\pi}{N} (n + 0.5) k \right) \]

Observe that the space index \( n \) has the shift of \( 0.5 \), but not the frequency index \( k \). This is the characteristic of the DCT of type 2. It is the most commonly used type in video and
image coding, because the higher bands have zeros of their frequency response at DC, hence suppressing the high DC energy of usual pictures for the higher subbands sufficiently. This is important for a good coding gain. In this way we don't have to encode the DC energy in higher subbands again. The DCT matrix is designed such that it is orthonormal, which means its inverse is identical to its transpose,

\[ T^{-1} = T^T \]

This automatically also means that it is invertible (by using its transpose), which is important for the decoder, were we would like to reconstruct the original blocks of our image. Observe that we decompose the image into smaller blocks to obtain a time(space) / frequency decomposition. We reduce the spacial resolution to the size of our blocks, but for that we obtain frequency information within each block. We can say in which block a certain frequency appears. In this way we obtain a different representation of our image, where we can take advantage of the frequency information, for instance to use that most energy is expected at low frequencies, or that the eye has the highest sensitivity at the lower frequencies. Hence we will put most of our bits into representing the low frequencies, and very few bits into the higher frequencies.
Python Example for the DCT
We again assume a number of subbands of N=4. We generate our transform matrix T,

\[
N=4;
T=np.zeros((N,N))
for k in range(4):
    for n in range(4):
        T[n,k]=np.cos(np.pi/N*(n+0.5)*(k))*np.sqrt(2.0/N);

T[:,0]=T[:,0]*np.sqrt(0.5)
T =

0.50000 0.65328 0.50000 0.27060
0.50000 0.27060 -0.50000 -0.65328
0.50000 -0.27060 -0.50000 0.65328
0.50000 -0.65328 0.50000 -0.27060

We check if our computed matrix is indeed orthonormal, as it should be, by checking if \( T \cdot T^T = I \) is the identity,

\[
>>> np.dot(T,T.T)
ans =

1.0000e+00  2.5452e-17  6.2206e-18  2.7132e-17
2.5452e-17  1.0000e+00  1.1268e-16  2.3285e-16
6.2206e-18  1.1268e-16  1.0000e+00  6.8522e-17
2.7132e-17  2.3285e-16  6.8522e-17  1.0000e+00

We see it is indeed the identity matrix, with the ones on the diagonal, and practically zeros on the off-diagonals.
Now we can also plot the impulse responses and the frequency responses of its 4
equivalent filters, as we did for the KLT,

```python
equivalent filters, as we did for the KLT,

```
These are now the impulse responses of our 4 equivalent DCT filters.

Compare them with the KLT,

Observe that both are indeed quite similar.

Now we can also plot the frequency responses for the DCT equivalent filters,

```python
for k in range(0,4):
    w, frresp = signal.freqz(T[:,k])
    plt.subplot(4,1,k+1)
    plt.plot(w, 20*np.log10(abs(frresp)))
plt.show()
```
Observe that this is again like subbands of a filter bank, with a low pass on top, below that band pass filters with increasing center frequency, and then a high pass. Also observe that except for the low pass filter, the higher filters have all zeros (the transfer function is zero in the linear scale), meaning **infinite attenuation** (-infinity dB in the dB scale), at **frequency zero**, meaning **DC**! This is important because natural images often have a lot of energy around DC, and this keeps it out of the higher filters, to save bits.
Again quite similar to the KLT,

The KLT frequency responses as a comparison.

Observe that the DCT makes no difference for rows and columns, hence it is the same for both sides, we make no difference between $T_c$ and $T$.

To transform our image, with blocks $B$, in the encoder we use

$$Y = T^T \cdot B \cdot T$$

and in the decoder we use

$$B = T \cdot Y \cdot T^T$$
Walsh-Hadamard Transform (WHT)

There is another interesting transform for images and video, and that is the so-called Walsh-Hadamard transform. It is defined as,

\[ H_1 := \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \]

It is extended to higher order with

\[ H_m := \frac{1}{\sqrt{2}} \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix} \]

Here we can see that this matrix only consists of +/-1, which means a transform needs no multiplications, only sign changes! The factors with \( \sqrt{2} \) are for making the transform orthonormal, they are not necessarily needed for an implementation. This makes it very computationally efficient!

But since the matrix entries are now less similar to a KLT, the compression performance is usually not as high as with a DCT (at least for usual images). Remember the KLT depends on the image, and there might be (e.g. artificial) images where the KLT becomes more similar to a WHT.

Observe that it also has similarity to a DCT where you only keep the signs (\( \text{sgn}(T) \)).