Lecture_5_code_snippet

March 16, 2017

1 Python Example

Take again the DCT2 as our transform matrix,

```
In [1]: import numpy as np
        import scipy.fftpack as sft

        I = np.eye(4)
        T = sft.dct(I, norm='ortho')
        T
```

```
Out[1]: array([[ 0.5       ,  0.65328148,  0.5       ,  0.27059805],
               [ 0.5       ,  0.27059805, -0.5       , -0.65328148],
               [ 0.5       , -0.27059805, -0.5       ,  0.65328148],
               [ 0.5       , -0.65328148,  0.5       , -0.27059805]])
```

```
In [2]: T = np.matrix(T)
        T
```

```
Out[2]: matrix([[ 0.5       ,  0.65328148,  0.5       ,  0.27059805],
                [ 0.5       ,  0.27059805, -0.5       , -0.65328148],
                [ 0.5       , -0.27059805, -0.5       ,  0.65328148],
                [ 0.5       , -0.65328148,  0.5       , -0.27059805]])
```

The inverse matrix then is used for the inverse transform (in the decoder),

```
In [3]: Tinv = T.I
        Tinv
```

```
Out[3]: matrix([[ 0.5       ,  0.5       ,  0.5       ,  0.5       ],
                [ 0.65328148,  0.27059805, -0.27059805, -0.65328148],
                [ 0.5       , -0.5       , -0.5       ,  0.5       ],
                [ 0.27059805, -0.65328148,  0.65328148, -0.27059805]])
```

Observe that the inverse matrix is identical to its transpose. This is because the DCT2 transform matrix is **Orthogonal** (the transpose needs a factor to become theinverse) or **Orthonormal** (the transpose needs no factor to become the inverse).

Here we can see that, for instance, the first row contains the impulse response of the **first** synthesis filter, $h_0(n)$, which here is indeed identical to the first analysis filter.

Or, take the 2^{nd} subband, $h_1(n)$, as an example. In the analysis part, we have the equivalent impulse response of the 2^{nd} analysis filter as,

```
In [4]: h1 = np.flipud(T[:,1])
        h1

Out[4]: matrix([[-0.65328148],
                [-0.27059805],
                [ 0.27059805],
                [ 0.65328148]])
```

The 2nd synthesis filter h 1 (n) is obtained with

```
In [5]: Tinv[1,:]

Out[5]: matrix([[ 0.65328148,  0.27059805, -0.27059805, -0.65328148]])
```

We can see that this 2nd synthesis impulse response (a row vector) is the time reverse version of the 2nd analysis impulse response (a column vector). Observe that this is generally true for orthogonal transform matrices (where the synthesis matrix is the (conjugate) transpose of the analysis matrix)!

Taking again our subband block y from our previous example (check for the Transform matrix in Lecture 4 code snippet),

```
In [6]: x = 0.3 * np.ones((4,4))
        y = np.dot(np.dot(np.transpose(T),x),T)
        y

Out[6]: matrix([[ 1.2,  0. ,  0. ,  0. ],
                [ 0. ,  0. ,  0. ,  0. ],
                [ 0. ,  0. ,  0. ,  0. ],
                [ 0. ,  0. ,  0. ,  0. ]])
```

we get the inverse thransform with,

```
In [7]: np.dot(np.dot(np.transpose(Tinv),y),Tinv)

Out[7]: matrix([[ 0.3,  0.3,  0.3,  0.3],
                [ 0.3,  0.3,  0.3,  0.3],
                [ 0.3,  0.3,  0.3,  0.3],
                [ 0.3,  0.3,  0.3,  0.3]])
```

Here we see that we indeed get our original block back! (Observe that inv(T') = inv(T)').

Also Observe that the beauty of this approach is, that we can deal with each block of our image independently of the others blocks, meaning there is no overlap of filters into neighbouring blocks!