blockdct

February 9, 2017

1 Program 1 - blockdct

Program to apply DCT type 2 on a float 8x8 array and then apply inverse DCT. * Import the relevant modules.

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

- Function to forward 8x8 DCT of type 2, orthogonal:

```
In [2]: def dct8x8(frame):
            #Usage: X=dct8x8(frame)
            #2D DCT of blocks of 8x8 pixels of a 2D float array "frame" in array X.

            #First reshaping the image to width 8 and applying the 1D DCT all rows, then reshape
            #then transpose it, and again reshape it to width 8 and apply the 1D DCT to each rou
            #and transpose it back.
            #with norm='ortho' for energy conservation in the subbands and for
            #invertibiltity without factor.
            #find the shape of "frame":
            [r,c]=frame.shape

            #First reshape frame as frame with rows of width 8, (rows: order= 'C' ),
            #and apply DCT to each row of length 8 of all blocks:
            frame=np.reshape(frame,(-1,8), order='C')
            X=sft.dct(frame,axis=1,norm='ortho')

            #shape it back to original shape:
            X=np.reshape(X,(-1,c), order='C')

            #Shape frame with columns of hight 8 by using transposition .T:
            X=np.reshape(X.T,(-1,8), order='C')
            X=sft.dct(X,axis=1,norm='ortho')

            #shape it back to original shape:
            X=(np.reshape(X,(-1,r), order='C')).T

            return X
```

- Inverse 2D DCT of type 2, orthogonal:

```
In [3]: def invdct8x8(X):
            #Usage: x=invdct8x8(X)
            #with X: array of coefficients of 8x8 DCT's
            #x: reconstructed frame.

            #find the shape of
            [r,c]=X.shape

            #Rows:
            X=np.reshape(X,(-1,8), order='C')
            X=sft.idct(X,axis=1,norm='ortho')

            #shape it back to original shape:
            X=np.reshape(X,(-1,c), order='C')

            #Shape frame with columns of hight 8 (columns: order='F' convention):
            X=np.reshape(X.T,(-1,8), order='C')
            x=sft.idct(X,axis=1,norm='ortho')

            #shape it back to original shape:
            x=(np.reshape(x,(-1,r), order='C')).T

            return x
```

- Test the functions on a frame of float values:

```
In [4]: if __name__ == '__main__':
            import numpy as np

            frame=np.ones((16,16));
            print "frame: \n", frame
```

```
frame:
[[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

```
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]]
```

```
In [5]:    X=dct8x8(frame)
           print "dct8x8(frame): \n", X
```

```
dct8x8(frame):
[[ 8.  0.  0.  0.  0.  0.  0.  0.  8.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 8.  0.  0.  0.  0.  0.  0.  0.  8.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

```
In [6]:    x=invdct8x8(X)
           print "reconstructed x: \n", x
```

```
reconstructed x:
[[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]]
```