Orthonormal bases

Standard basis and DCT

Chiara Ravazzi

National Research Council of Italy
Institute of Electronics, Computer and Telecommunication Engineering, c/o Politecnico di Torino, Italy
Systems Modeling & Control Group

Analisi tempo-frequenza e multiscala – 01RMQNG

Motivation

Main goals :

- use libraries (numpy, matplotlib.pylab, scipy)
- read and save images
- plot images
- compare images
- compare orthonormal bases : standard basis and 2-Dimensional DCT
- simulate transmissions errors
- compress images : linear/nonlinear approximation

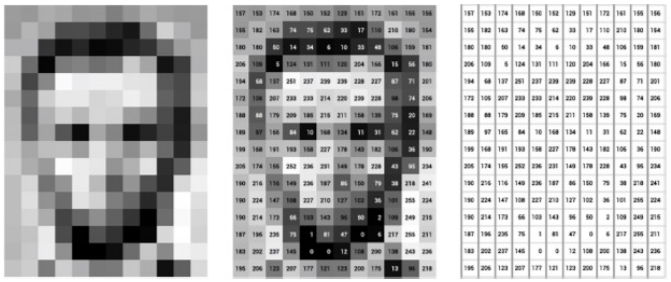
Digital images

Preliminaries

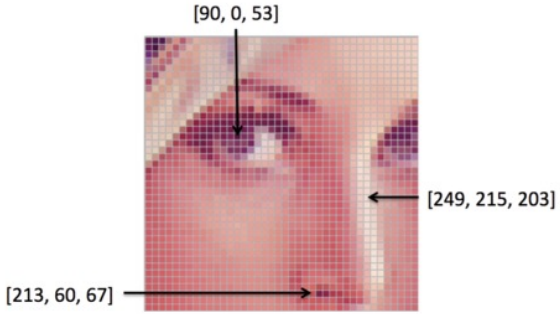
Think of images as functions mapping locations in images to pixel values $f(x, y)$

Gray Image

Grayscale Pixel Values
(intensity between 0 and 255)



Color Image



- $f(x, y)$ vector of three values
- RGB : colors from a combination of Red, Green, and Blue (RGB).
- Three channels + integer values from 0 to 255 $\Longrightarrow$ 256*256*256 = 16,777,216 combinations or color choices.

Consider

$$\{x : \mathbb{Z}_M \times \mathbb{Z}_N \to \mathbb{R}\}, \qquad \mathbb{Z}_N = \{0, \ldots, N-1\}$$

- $x_{m,n}$ intensity of pixel (m, n), $m = 0, \ldots, M-1$, $n = 0, \ldots, N-1$
- Inner product :

$$\langle x, y \rangle = \mathrm{Tr}(y^\top x) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_{m,n} y_{m,n}$$

- Frobenius norm :

$$\|x\|_F = \sqrt{\langle x, x \rangle} = \sqrt{\sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_{m,n}^2}$$

Before starting

Tips

Recall

- import libraries and modules (numpy, matplotlib.pylab, scipy)

```
Created on Tue Sep 17 13:18:10 2019

@author: chiara ravazzi
"""



import numpy as np
import matplotlib.pylab as plt
import scipy.io as sio
#from scipy import linalg
from functions_es2 import ps

```

```
"""
Created on Wed Oct 23 10:29:43 2019

@author: chiararavazzi
"""
import numpy as np

def DCT_matrix(N):
    C=np.zeros([N,N])
    for k in range(N):
        for n in range(N):
            if k==0:
                C[n,k]=1/np.sqrt(N)*np.cos(np.pi/N*(n+1/2)*k)
            else:
                C[n,k]=np.sqrt(2/N)*np.cos(np.pi/N*(n+1/2)*k)
    return C

def ps(A,B):
    return np.sum(A*B)


```

- identify functions that are required for your task (e.g. read and plot images)
- do a keyword search or get help and find documentation

```
In [33]: help(plt.imread)
Help on function imread in module matplotlib.pyplot:

imread(fname, format=None)
    Read an image from a file into an array.

    Parameters
    ----------
    fname : str or file-like
        The image file to read. This can be a filename,
        file-like object opened in read-binary mode.
```

Read images

matplotlib.pyplot.imread

`matplotlib.pyplot.imread(fname, format=None)` [source]

Read an image from a file into an array.

Note

This function exists for historical reasons. It is recommended to use `PIL.Image.open` instead for loading images.

Parameters: **fname** : *str or file-like*

The image file to read: a filename, a URL or a file-like object opened in read-binary mode.
Passing a URL is deprecated. Please open the URL for reading and pass the result to Pillow, e.g. with `np.array(PIL.Image.open(urllib.request.urlopen(url)))`.

format : *str, optional*

The image file format assumed for reading the data. The image is loaded as a PNG file if *format* is set to "png", if *fname* is a path or opened file with a ".png" extension, or if it is an URL. In all other cases, *format* is ignored and the format is auto-detected by `PIL.Image.open`.

Returns: `numpy.array`

The image data. The returned array has shape

- (M, N) for grayscale images.
- (M, N, 3) for RGB images.
- (M, N, 4) for RGBA images.

PNG images are returned as float arrays (0-1). All other formats are returned as int arrays, with a bit depth determined by the file's contents.

Plot images

matplotlib.pyplot.imshow

matplotlib.pyplot.imshow(*X*, *cmap=None*, *norm=None*, ***, *aspect=None*, *interpolation=None*, *alpha=None*, *vmin=None*, *vmax=None*, *origin=None*, *extent=None*, *interpolation_stage=None*, *filternorm=True*, *filterrad=4.0*, *resample=None*, *url=None*, *data=None*, ***kwargs*) [source]

Display data as an image, i.e., on a 2D regular raster.

The input may either be actual RGB(A) data, or 2D scalar data, which will be rendered as a pseudocolor image. For displaying a grayscale image set up the colormapping using the parameters `cmap='gray', vmin=0, vmax=255`.

The number of pixels used to render an image is set by the Axes size and the *dpi* of the figure. This can lead to aliasing artifacts when the image is resampled because the displayed image size will usually not match the size of *X* (see Image antialiasing). The resampling can be controlled via the *interpolation* parameter and/or `rcParams["image.interpolation"]` (default: `'antialiased'`).

Parameters:

X : *array-like or PIL image*

The image data. Supported array shapes are:

- (M, N): an image with scalar data. The values are mapped to colors using normalization and a colormap. See parameters *norm*, *cmap*, *vmin*, *vmax*.
- (M, N, 3): an image with RGB values (0-1 float or 0-255 int).
- (M, N, 4): an image with RGBA values (0-1 float or 0-255 int), i.e. including transparency.

The first two dimensions (M, N) define the rows and columns of the image.

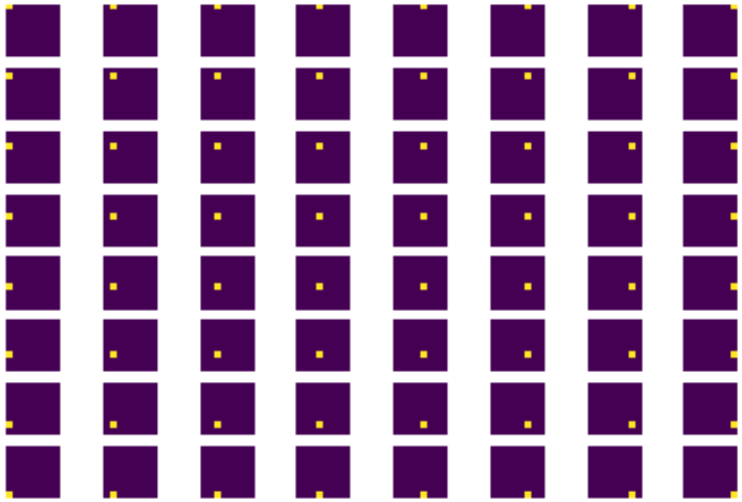
Out-of-range RGB(A) values are clipped.

cmap : *str or `Colormap`, default: `rcParams["image.cmap"]` (default: `'viridis'`)*

The Colormap instance or registered colormap name used to map scalar data to colors. This parameter is ignored for RGB(A) data.

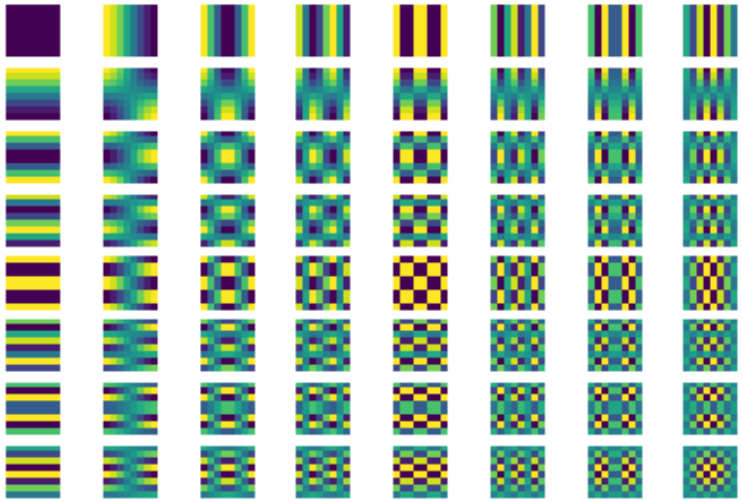
Standard basis & 2-Dimensional DCT

Standard basis



$N_1 = 8$ and $N_2 = 8$

Two-dimensional DCT



$N_1 = 8$ and $N_2 = 8$

Consider

$$\{x : \mathbb{Z}_M \times \mathbb{Z}_N \to \mathbb{R}\}, \qquad \mathbb{Z}_N = \{0, \dots, N-1\}$$

- $x_{m,n}$ intensity of pixel (m, n), $m = 0, \dots, M-1$, $n = 0, \dots, N-1$
- Inner product :

$$\langle x, y \rangle = \mathrm{Tr}(y^\top x) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_{m,n} y_{m,n}$$

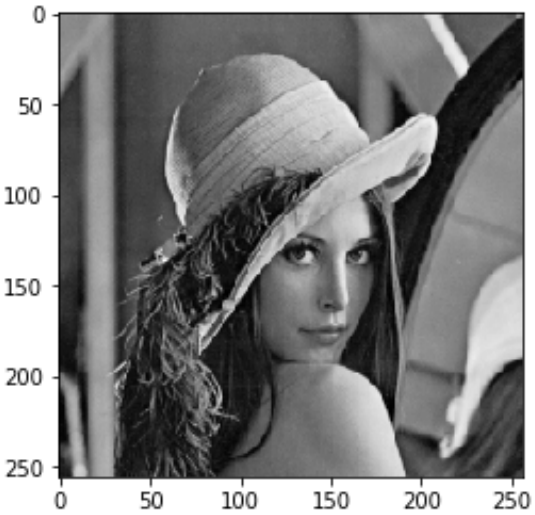
- Standard basis $\{E^{(k,\ell)}\}$ with $k = 0, \dots, M-1$, $\ell = 0, \dots, N-1$

$$E^{(k,\ell)}_{m,n} = \begin{cases} 1 & \text{if } m = k, n = \ell \\ 0 & \text{otherwise} \end{cases}$$

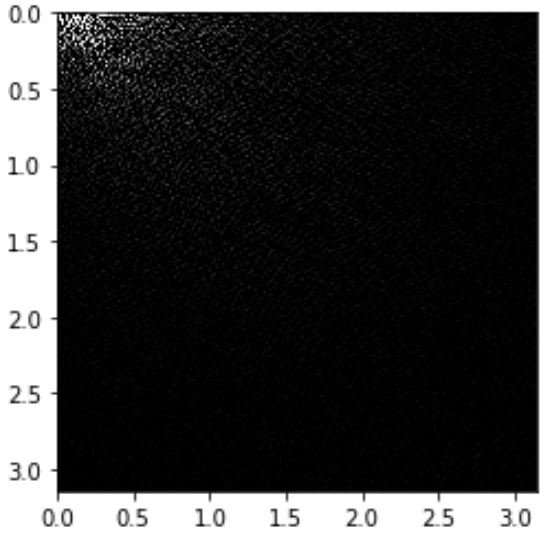
- Let $c_{(k,\ell)} = \langle x, E^{(k,\ell)} \rangle = x_{k,\ell}$ then

$$x = \sum_{k=0}^{M-1} \sum_{\ell=0}^{N-1} c_{(k,\ell)} E^{(k,\ell)} = \sum_{k=0}^{M-1} \sum_{\ell=0}^{N-1} x_{k,\ell} E^{(k,\ell)}$$

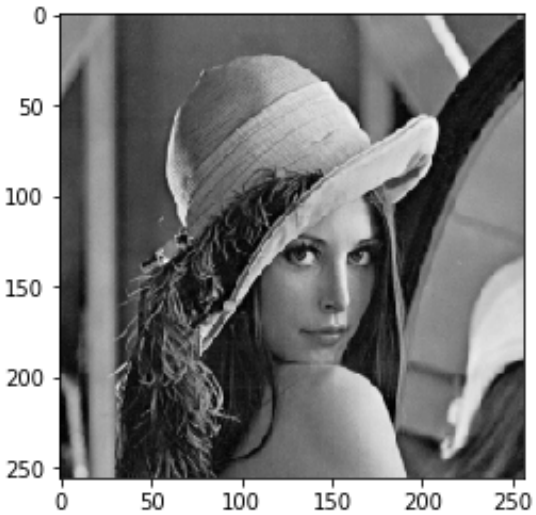
2-DCT and Inverse 2-DCT



Original image



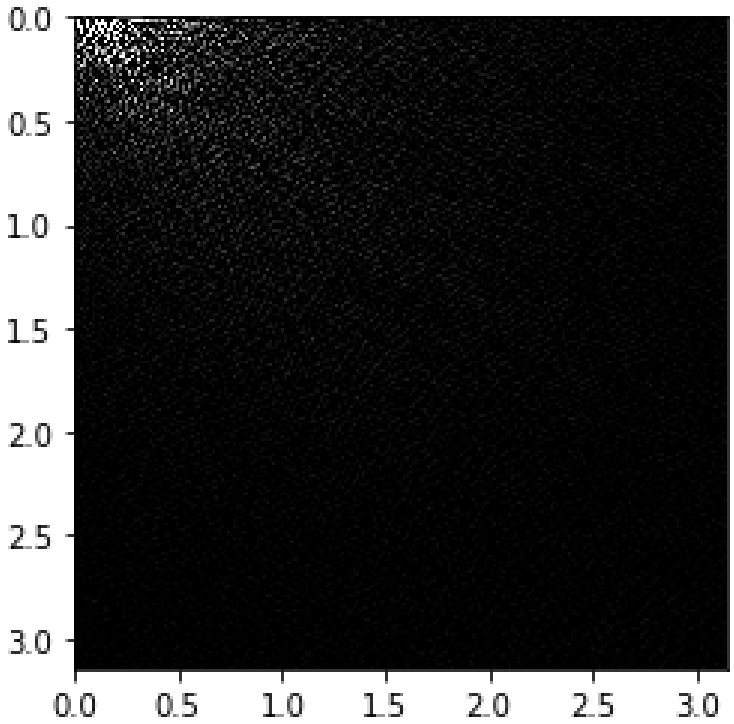
DCT Coefficients



Identity

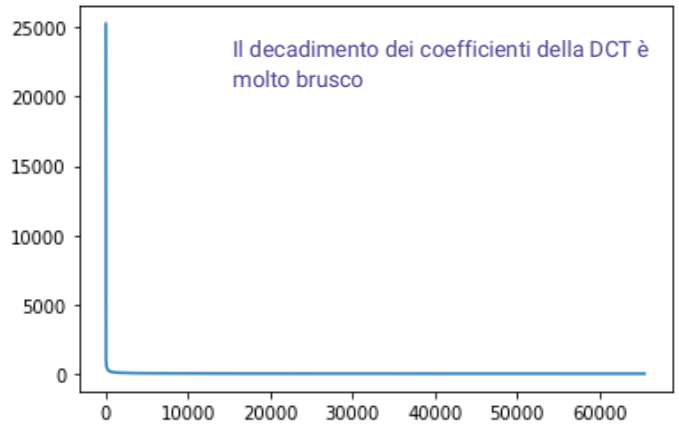
Approximation/Compression

Visualizziamo i valori assoluti dei coefficienti della DCT dell'immagine importata

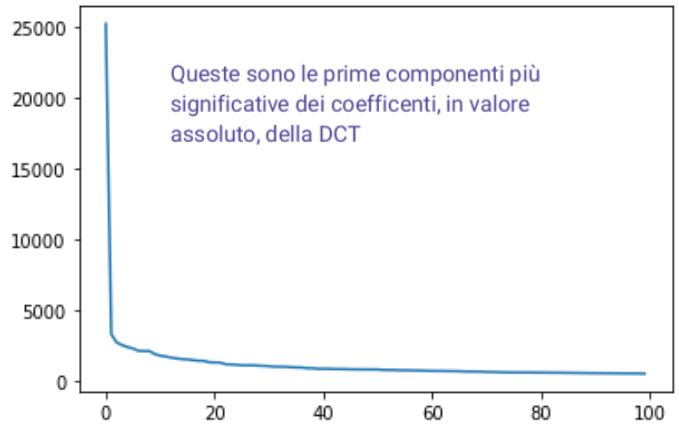


DCT Coefficients

Cosa comporta questo decadimento molto piccato? L'immagine può essere rappresentata abbastanza bene e rappresentata da pochi coefficienti della DCT



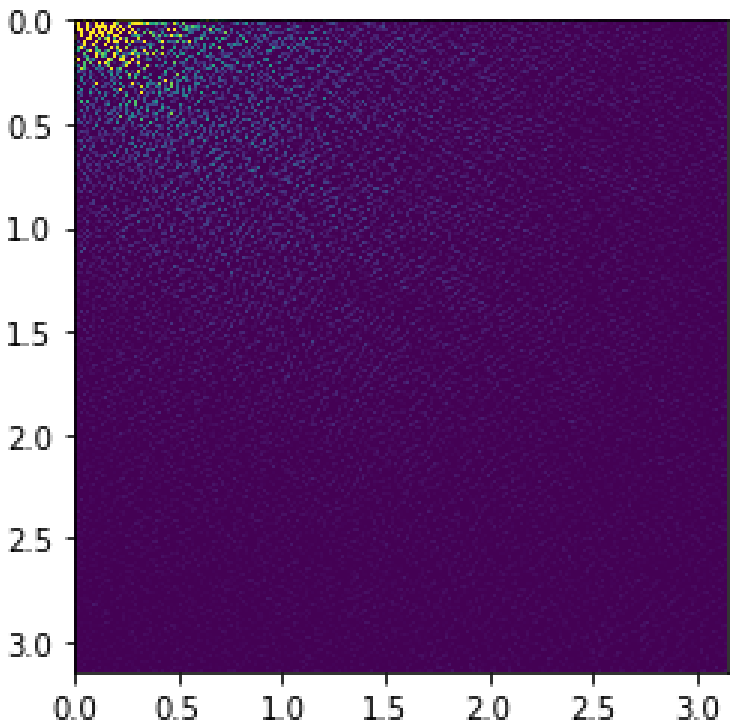
DCT coefficients' decay



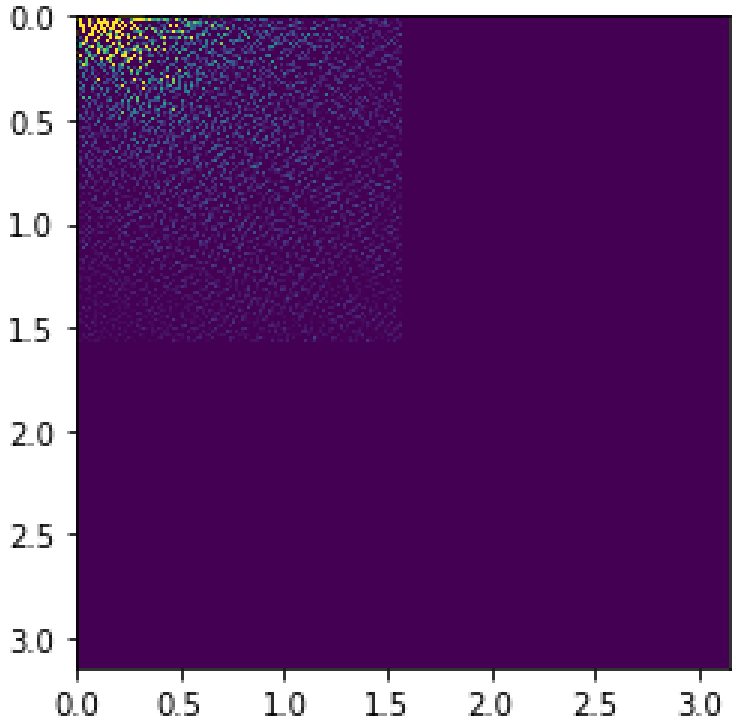
DCT coefficients' decay
(first 100 significant components)

Linear approximation

Retain DCT Coefficients in the upper corner : compression ratio = 0.25



DCT Coefficients

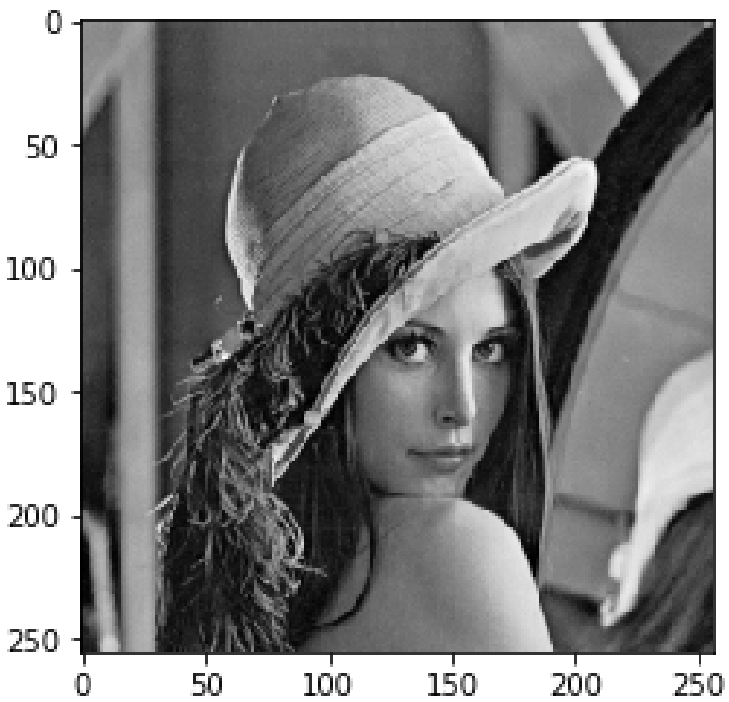


DCT Coefficients in upper corner

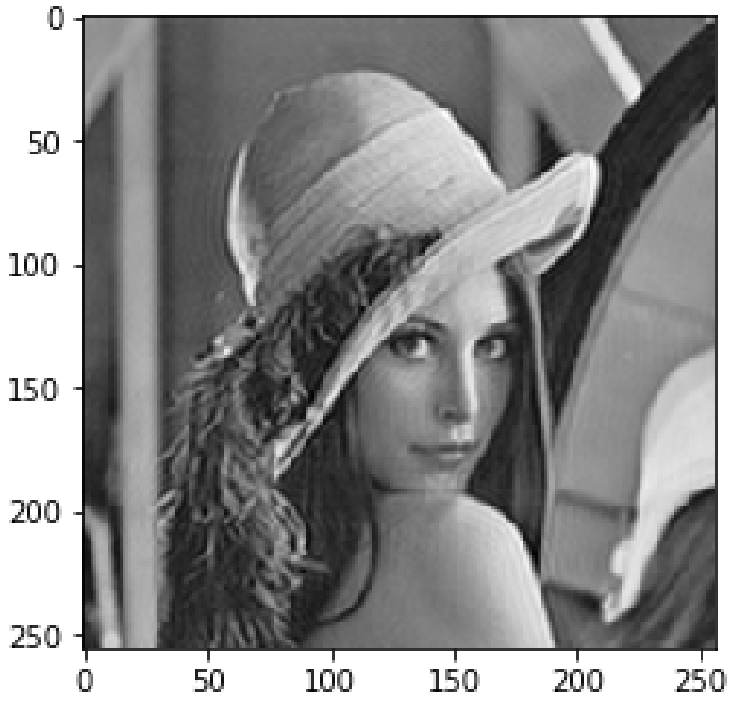
Consideriamo di tenere solo i coefficienti dell'angolo in alto a sinistra, quinid usando solo 1/4 dei coefficienti della DCT
Quindi scegliamo 1/4 senza motivi controllare il valore dei coefficienti, cioè gli scegliamo in modo casuale

Linear approximation

Compute the Inverse 2-DCT



DCT Coefficients



DCT Coefficients in upper corner

Con tutti i coefficienti otteniamo l'immagine originale perchè la base è ortonormale, quindi riusciamo a ritornare all'immagine originale senza perdere informazioni

Compression ratio = 0.25

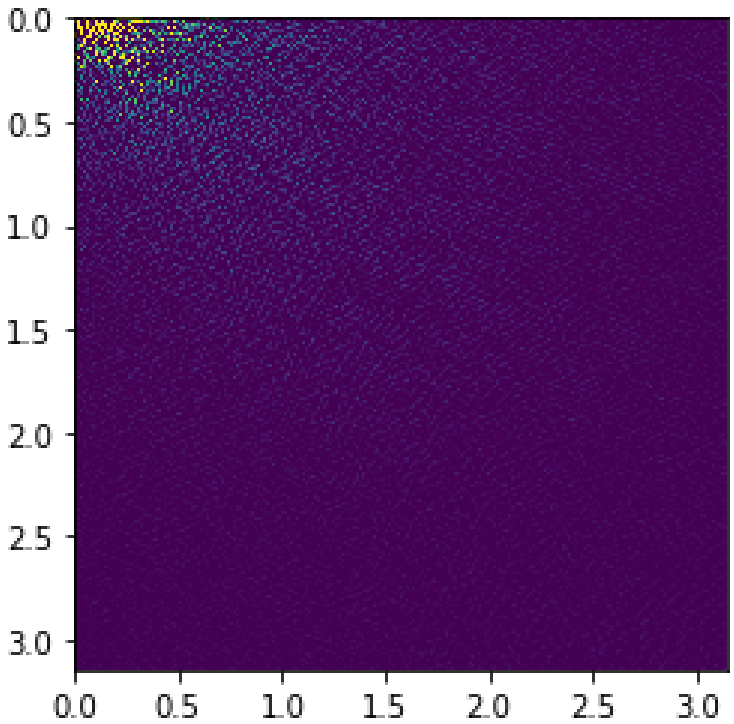
Relative Error = $\frac{\|I-\widehat{I}\|_F}{\|I\|_F} = 0.0673$

Usando 1/4 dei coefficienti otteniamo comunque un'immagine fedele, si vede che è leggermente sgranata in alcuni punti, ma si capiscono tutti i dettagli
Stiamo comprimendo l'immagine con rapporto di compressione 0.25
L'errore relativo è calcolato in norma di Frobenius

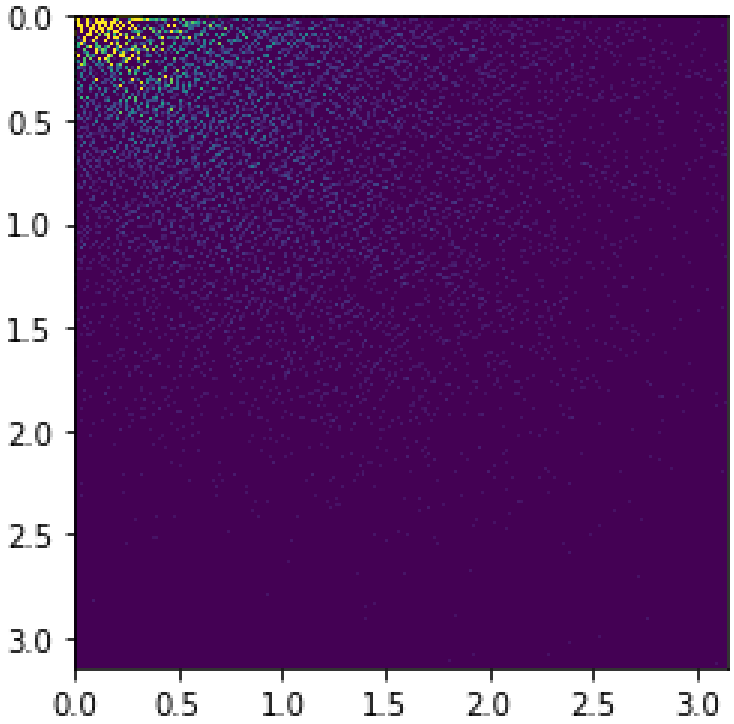
Nonlinear approximation

Cosa succede se operaiamo una approssimazione non lineare --> teniamo i coefficienti della DCT che sono più grandi in valore assoluto e ne teniamo sempre 1/4 (quindi prima ne abbiamo presi 1/4 a caso, ora buttiamo quelli più piccoli e teniamo solo i coefficienti della DCT più grandi)

Retain the largest DCT Coefficients : compression ratio = 0.25



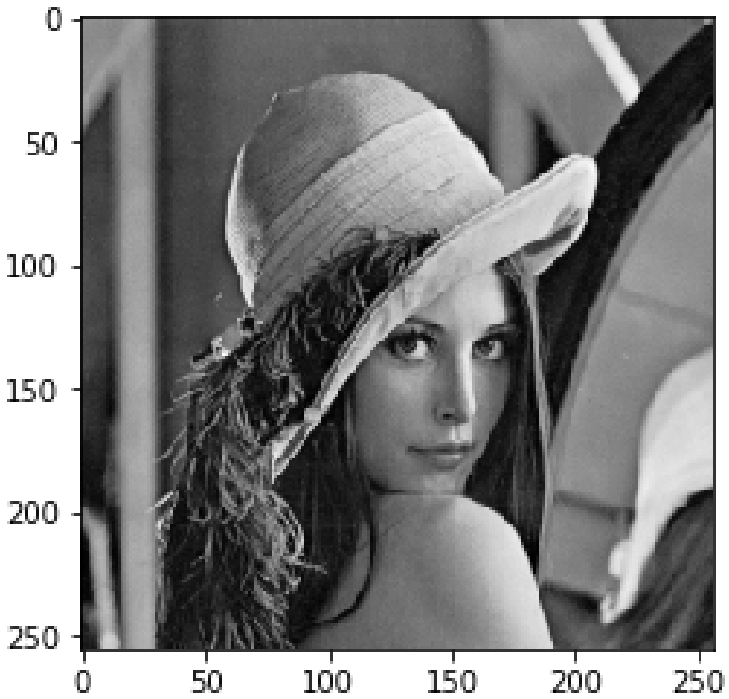
DCT Coefficients



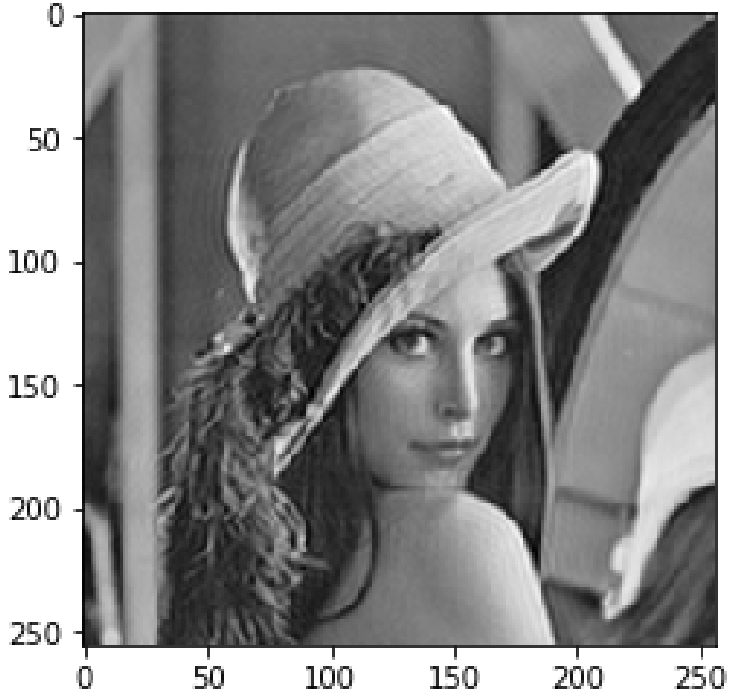
Best-*k* approximation

Nonlinear approximation

Compute the Inverse 2-DCT



DCT Coefficients



Best-k approximation

Vediamo che in questo modo l'errore relativo scende

Compression ratio = 0.25

Relative Error = $\frac{\|I-\widehat{I}\|_F}{\|I\|_F} = 0.0419$