Assignment 1
IFT 6268 Winter 2015

Date posted: February 4, 2015
Due: February 11, 2015, at the *beginning* of class

---

1. (2/10) The discrete Fourier Transform (DFT) of the one-dimensional signal $s(t)$ is defined as

$$S(k) = \sum_{t=0}^{T-1} s(t) e^{-i\frac{2\pi}{T}kt} \quad k = 0, \ldots, T-1$$

   Show in details what happens to the amplitude spectrum and to the phase spectrum of a signal, if we perform the translation:

$$s(t) \rightarrow s(t+\tau)$$

   You may assume that the translation is with "wrap-around" (modulo the length of the signal).

2. (2/10) Assume $s(t)$ to be a *real valued* signal. Prove or disprove:

$$S(l) = \bar{S}(T-l)$$

3. (3/10) Download the CIFAR-10 dataset from

   `http://www.cs.toronto.edu/~kriz/cifar.html`

   (If the 350MB size of the dataset causes problems, ask instructor for a smaller subset.)

   Use only the *training set* for this question. Turn the RGB-images into gray-value images using the formula

```
trainimages = (trainimages*numpy.array([[[0.299], [0.587], [0.144]]])).sum(1)
```

   Display one image from each class to get familiar with the dataset.

   Then, using an FFT package of your choice, make a plot of the *average amplitude spectrum*, where the average is over all the images in the training set, as well as 10 plots showing the amplitude spectrum averaged over all training images per class.

   Then show the amplitude spectrum of the first *automobile*-image as well as the first *cat*-image in the dataset. Finally, show the reconstruction of the automobile image by combining its own amplitude spectrum with the phase spectrum of the cat-image, and vice versa. What does the result tell you about the relative importance of amplitude vs. phase for recognizing the image content?

   Hints:

- The amplitude spectra should be clearly visible. Use a log transform if necessary.
- All spectra should be displayed with the **0**-frequency vector in the center.
- Display real-valued images only. If the inverse FFT returns complex numbers you will need to deal with this somehow.
- Use gray values not colors to display the spectra.
- For those using Python 2.x/pylab:
  - The complex number i is represented by the expression "1j" in python.
  - You can access the real and imaginary parts of a complex number c using "c.real" and "c.imag", respectively.
  - To display images and spectra you can use the function "imshow".
  - You may want to make use of the functions "numpy.fft.fft2", "numpy.fft.ifft2", "numpy.fft.fftshift", and "numpy.fft.ifftshift".
  - Be sure to represent your data as floating point numbers, otherwise strange things may happen.

What to hand in: (a) Ten images, each showing one training case per class, (b) one image of the average amplitude spectrum, (c) ten images showing the class-specific amplitude spectra, (d) four additional images: one image showing the cat amplitude spectrum, one image showing the automobile amplitude spectrum, one image showing the reconstruction of the cat image using its own amplitude spectrum but the phase spectrum of the automobile image, and vice versa. Add a short interpretation of that result.

Do *not* hand in any program code.

4. (3/10) Download the image

   `http://www.iro.umontreal.ca/~memisevr/teaching/ift6268_2015/nao_bw.jpg`

   and save it locally.

   Write a function $convolve(I_1, I_2)$, that computes the convolution of two images I_1 und I_2. Then low-pass filter the image above by convolving it with an appropriate filter mask. Use filter masks of various different sizes, for example, 5×5, 11×11, 21×21, 31×31, 41×41.

   First use your convolve function. Note down the run-times as a function of filter size. Then generate the same outputs (perhaps up to some small shift) by applying the filters in the frequency domain using the FFT. Again, note down the run-times as a function of filter size.

   Hints:

   - You may assume (if you want) that both images have an uneven number of rows and of columns.
   - You may assume (if you want) that the first argument, I_1 is larger than the second, I_2.

- Boundaries: The output should have the same size as I_1. You may assume that I_1 is 0 everywhere outside its defined size.
- Do *not* take the convolve function from a library, write your own. Efficiency is not important.
- If you use Python, you can use the argument "s" of the fft2 function to interpret the filter-masks as if they had the same dimensions as the image.
- The numpy-function "meshgrid" may be useful to create the filter-masks.
- To determine run-times you can use the Python-function time() in the module time (and accessible in the ipython environment with the %time or %timeit magic commands).

What to hand in: (a) The original image and the filtered versions, using both your convolve function and the FFT, (b) images showing the corresponding filter-masks, (c) a plot showing the run-times.