AIMS CDT Signal Processing Lab Session 1
Signal and Image Processing

Xiaowen Dong (xdong@robots.ox.ac.uk)

In this lab session[1], we look at several classical signal & image processing operations based on the representations enabled by the Fourier and wavelet transform. First, download the toolbox provided at `https://github.com/gpeyre/numerical-tours/raw/master/python/nt_toolbox.zip`, and make sure the folder *nt_toolbox* is in the current working directory. Then, install and import the necessary Python modules.

```
python3 -m pip install numpy scipy matplotlib scikit-learn scikit-image
import numpy as np
import scipy as scp
import pylab as pyl
import matplotlib.pyplot as plt
from nt_toolbox.general import *
from nt_toolbox.signal import *
```

Plot an image. Let's load and plot an example image.

```
n0 = 512
f = rescale(load_image("nt_toolbox/data/hibiscus.bmp", n0))
plt.figure(figsize = (5, 5))
imageplot(f, 'Image f')
```

We can zoom in to see more detail of one part of the image.

```
plt.figure(figsize = (5, 5))
imageplot(f[n0//2 - 32:n0//2 + 32, n0//2 - 32:n0//2 + 32], 'Zoom')
```

Fourier transform. Now let's perform the 2-D Fourier transform.

```
F = pyl.fft2(f)/n0 # forward transform

from pylab import linalg
print("Energy of Image:   %f" %linalg.norm(f))
print("Energy of Fourier: %f" %linalg.norm(F))
```

- Why is the signal energy preserved in the frequency domain?

Display the log amplitude of the Fourier coefficients. The *fftshift* function shifts the zero-frequency component to the centre of the spectrum.

```
L = pyl.fftshift(np.log(abs(F) + 1e-1))
plt.figure(figsize = (5, 5))
imageplot(f, 'Image', [1, 2, 1])
imageplot(L, 'Log(Fourier transform)', [1, 2, 2])
```

[1] This lab session was largely based on the Numerical Tours designed by Gabriel Peyré: `http://www.numerical-tours.com`.

Filtering. One of the most common signal processing operations is filtering. Let's consider a low-pass filtering example, where we keep coefficients corresponding to the M lowest frequencies while setting the rest to zero, before carrying out an inverse transform.

```
M = n0**2//64
q = int(np.sqrt(M))
F = pyl.fftshift(pyl.fft2(f))
Sel = np.zeros([n0, n0])
Sel[n0//2 - q//2:n0//2 + q//2, n0//2 - q//2:n0//2 + q//2] = 1
F_zeros = np.multiply(F, Sel)
imageplot(np.log(abs(F_zeros) + 1e-1), 'Log(Fourier transform)') # plot in frequency domain

f_lpf = np.real(pyl.ifft2(pyl.fftshift(F_zeros))) # inverse transform
plt.figure(figsize = (5, 5))
imageplot(f, 'Image', [1, 2, 1])
imageplot(clamp(f_lpf), 'Low-pass filtered image', [1, 2, 2])
```

- Can you observe the difference between the original image and the filtered one?
- Now perform a high-pass filtering (how can you do this?). Can you explain the filtering outcome?

Fourier-based vs wavelet-based approximation. The low-pass filtering example above corresponds to image approximation using Fourier coefficients corresponding to the M *lowest* frequencies. We can also use the wavelet transform for the same purpose and compare the outcome. Let's first perform the 2-D wavelet transform.

```
Jmin = 0
from nt_toolbox.perform_wavelet_transf import *
Fw = perform_wavelet_transf(f, Jmin, + 1) # forward transform
plt.figure(figsize = (5, 5))
plot_wavelet(Fw)
plt.title('Wavelet coefficients')
plt.show()
```

Let's implement a wavelet approximation by keeping the M *largest* wavelet coefficients while setting the rest to zero.

```
a = np.sort(np.ravel(abs(Fw)))[::-1] # sort a 1D copy of magnitude of fw in descending order
Fw_zeros = np.multiply(Fw,(abs(Fw) > a[M])) # keeping the M largest wavelet coefficients
plt.figure(figsize = (5, 5))
plot_wavelet(Fw_zeros) # comapre with the plot of coefficients above
plt.title('Wavelet coefficients (thresholded)')
plt.show()

f_wav = perform_wavelet_transf(Fw_zeros, Jmin, -1) # inverse transform
plt.figure(figsize = (5, 5))
imageplot(clamp(f_lpf), 'Fourier approximation', [1, 2, 1])
imageplot(clamp(f_wav), 'Wavelet approximation', [1, 2, 2])
```

- Can you comment on the comparison between the two approximations above. Why is one better than the other?

Linear denoising. Another typical signal processing task is image denoising, i.e., removing noise from an image. First let's generate and plot a noisy image.

```
n = 256
N = n**2
name = 'nt_toolbox/data/boat.bmp'
f = load_image(name, n)
plt.close('all')
```

```
imageplot(f)
sigma = .08
y = f + sigma*np.random.standard_normal(f.shape)
imageplot(clamp(y))
```

We will implement a linear denoising strategy via convolution. Define a Gaussian convolutional kernel as follows.

```
cconv = lambda a, b : np.real( pyl.ifft2(pyl.fft2(a)*pyl.fft2(b)) )
normalize = lambda h : h/sum(h.flatten())
t = np.transpose( np.concatenate( (np.arange(0, n/2), np.arange(-n/2, 0) ) ) )
[Y, X] = np.meshgrid(t, t)
h = lambda mu: normalize(np.exp(-(X**2 + Y**2)/ (2*mu**2)))
```

Now denoise the image by convolving it with a Gaussian kernel with a given width.

```
mu = 5
plt.figure(figsize = (5, 5))
imageplot(pyl.fftshift(h(mu)), 'kernel', [1, 2, 1])
imageplot(pyl.fftshift(np.real(pyl.fft2(h(mu)))), 'FT of kernel', [1, 2, 2])
denoise = lambda x, mu : cconv(h(mu), x)
plt.figure(figsize = (5, 5))
imageplot(f, 'Image', [1, 3, 1])
imageplot(y, 'Noisy image', [1, 3, 2])
imageplot(denoise(y, mu), 'Denoised image', [1, 3, 3])
```

- Why is this denoising strategy linear?
- Vary the kernel width μ, and compare the denoising outcome.
- Can you find an optimal μ that minimises the ℓ_2-norm of the difference between original and denoised image?

Non-linear denoising. Now let's try a non-linear denoising method using the wavelet transform. A simple way of achieving this is to express the image signal in an orthogonal wavelet basis $\{\boldsymbol{\psi}_m\}_m$ and threshold the wavelet coefficients before reconstruction. Mathematically, this can be expressed as:

$$\hat{\mathbf{f}} = \sum_m s_T^1(\langle \mathbf{f}, \boldsymbol{\psi}_m \rangle) \boldsymbol{\psi}_m, \tag{1}$$

where $s_T^1(\alpha) = \max(0, 1 - \frac{T}{|\alpha|})\alpha$ is the so-called soft thresholding (this is in contrast to hard thresholding which sets all coefficients with a magnitude smaller than a threshold to 0) operator with threshold $T \geq 0$.
Now implement a wavelet-based non-linear denoising via the following steps:

1. Define the soft thresholding operator;
2. Perform the forward wavelet transform as above;
3. Apply the soft thresholding operator with a certain threshold T to the wavelet coefficients;
4. Perform the inverse wavelet transform;
5. Repeat with different values of the threshold T and compare.

- Can you comment on the differences between linear and non-linear denoising methods?
- In addition to visual comparison, it is common to compute the signal-to-noise ratio (SNR) of the denoised image. This can be done by calling the 'snr' function built in 'signal.py' in the toolbox. The higher the SNR, the better the denoising effect in general.

[Optional] You can try more advanced image processing exercises as part of the Numerical Tours at `http://www.numerical-tours.com/python/`.

[Checkpoint] You are now done with this lab session. Please ask a lab demonstrator to evaluate your work.