Exercise set 12 – MATH-250 Advanced Numerical Analysis I

There is no quiz this week. The exercises marked with **(Python)** are implementation based and can be solved in the Jupyter notebooks which are available on Moodle/Noto.

Exercises

Problem 1.
An audio signal $y_i, i = 0, 1, \ldots, n-1$ can be compressed by computing its discrete Fourier transform (DFT) $\hat{y}_i, i = 0, 1, \ldots, n-1$ and setting all the coefficients whose absolute value relative to the maximum amplitude of the signal is below a certain level, based on a threshold τ, to zero:

$$\hat{y}_i^{(c)} = \begin{cases} 0 & \text{if } |\hat{y}_i| < \tau \max_{j=0,1,\ldots,n-1} |\hat{y}_j| \\ \hat{y}_i & \text{else} \end{cases}, i = 0, 1, \ldots, n-1$$

To convert the compressed DFT coefficients to the compressed audio signal, simply apply the inverse DFT to the compressed coefficients $\hat{y}_i^{(c)}, i = 0, 1, \ldots, n-1$.

The Python functions `np.fft.rfft` and `np.fft.irfft` allow to calculate respectively the DFT and the inverse DFT of a real-valued signal using the *Fast Fourier Transform* algorithm. With the help of these functions, implement the function `compress_audio` which takes as input an audio signal `y` and a threshold value `threshold`, and returns the compressed audio signal `y_compressed`.

Test your compression algorithm on the audio signal `foryoublue.wav` which you can download from Moodle and read using the SciPy function `sp.io.wavfile.read`. Test different threshold values $\tau = [0.1, 0.2, 0.5]$ and plot the original and compressed signals for a small extract of the audio (roughly 5000 beats).

Problem 2.
Throughout this problem, for a vector $\mathbf{x} \in \mathbb{R}^n$, we start indexing vectors from 0, i.e., x_0 is the first element of $\mathbf{x}$. In addition, indexing should be understood as taking modulo n. This means, $x_k = x_{k+n}$ for all k. We let $\hat{\mathbf{x}}$ denote the DFT of $\mathbf{x}$:

$$\hat{x}_k = \sum_{j=0}^{n-1} x_j \exp\left(-\frac{i2\pi kj}{n}\right) = \sum_{j=0}^{n-1} x_j \omega_n^{kj} \tag{1}$$

where $\omega_n = \exp(-i\frac{2\pi}{n})$.

(a) For a vector $\mathbf{x} \in \mathbb{R}^n$ recall that its DFT can be written as $\hat{\mathbf{x}} = F_n\mathbf{x}$, where $\frac{1}{\sqrt{n}}F_n$ is unitary (by Lemma 7.5). For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ show that $\langle \mathbf{x}, \mathbf{y} \rangle = n^{-1}\langle \hat{\mathbf{x}}, \hat{\mathbf{y}} \rangle$, where $\hat{\mathbf{y}}$ is the DFT of $\mathbf{y}$. Conclude that $\|\mathbf{x}\|_2 = n^{-1/2}\|\hat{\mathbf{x}}\|_2$.

(b) The convolution of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, is the vector $\mathbf{z} = \mathbf{x} * \mathbf{y} \in \mathbb{R}^n$ defined by

$$z_k = \sum_{j=0}^{n-1} x_j y_{k-j}, \quad k = 0, \ldots, n-1.$$

Show that $\hat{z}_i = \hat{x}_i \hat{y}_i$.

Hint: Write out the DFT of $\mathbf{z}$ explicitly using (1) and use that indexing should be understood by taking modulo n.

(c) Using (b) and the function `np.fft.rfft` and `np.fft.irfft` in Python, implement the convolution operator in Python. For $\mathbf{x} = \begin{pmatrix} 3 & 4 & \cdots & 12 \end{pmatrix}^\top$ and $\mathbf{y} = \begin{pmatrix} 12 & 11 & \cdots & 3 \end{pmatrix}^\top$ display the output from your implementation.

(d) Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ contain the coefficients of two polynomials:

$$p_{\mathbf{x}}(t) = x_0 + x_1 t + \cdots + x_{n-1} t^{n-1}, \quad p_{\mathbf{y}}(t) = y_0 + y_1 t + \cdots + y_{n-1} t^{n-1}.$$

Let $\mathbf{z}$ contain the coefficients of the product $p_{\mathbf{x}}(t) \cdot p_{\mathbf{y}}(t)$.

Show how $\mathbf{z}$ can be computed in $O(n \log_2 n)$ complexity from a convolution of the vectors $\mathbf{x}, \mathbf{y}$ padded with zeros in an appropriate manner. You may use the fact that the functions `rfft` and `irfft` require $O(n \log_2 n)$ operations to compute the DFT of a vector.

Implement a Python function `fft_convolve` that, given $\mathbf{x}$,$\mathbf{y}$, returns $\mathbf{z}$ using your function from (c). Test your implementation for the polynomials

$$p_{\mathbf{x}}(t) = 2 + 3t + t^2 + 2t^3, \quad p_{\mathbf{y}}(t) = 1 + t + 2t^2 + 2t^3.$$

Problem 3. For a real vector $\mathbf{f} \in \mathbb{R}^n$, the ℓth component in the discrete cosine-transform (DCT) [1] is given by

$$\hat{f}_\ell = \frac{f_0}{2} + \sum_{k=1}^{n-1} f_k \cos\left(\frac{(2k+1)\pi\ell}{2n}\right), \quad \ell = 0, \ldots, n-1. \tag{2}$$

(a) Show how the DCT can be computed from the real part of the DFT of a vector of length $2n$.

(b) Derive an algorithm requiring only $O(n \log(n))$ operations to compute the DCT of a vector.

[1] There are several versions of the DCT. This type is called DCT-III.