Problem Set

This quiz contains 2 problems, each containing a few steps. Answer the questions concisely and report the answer to each step as and when you complete them. If you are asked to write a function to implement an algorithm, you are expected to write the function yourself. Duplication from other sources must be cited. Dependencies used (MATLAB toolboxes and/or Python packages) must also be cited. Send your answers via email in a .pdf format with the codes attached.

Problem 1: The Annihilating Filter

Let $(x_n)_{n\in\mathbb{N}}$ be a discrete sequence defined by $x_n = \sum_{k=1}^{K} a_k^n$. It is of interest to estimate the parameter set $\{a_k\}_{k=1}^{K}$ using the samples of the elements of the sequence $(x_n)_{n\in\mathbb{N}}$. Design a method to estimate $\{a_k\}_{k=1}^{K}$ with the fewest number of elements as possible.

Step 1: Consider the case when $K = 1$. The sequence $x_n = a_1^n$ contains one unknown a_1. Design a simple method to estimate the unknown parameter a_1.

Step 2: Consider a causal discrete filter H with the Z-transform of the type $H(z) = 1 - \alpha z^{-1}$. Design a system using the filter H to estimate the unknown parameter a_1. Find the minimum number of elements of the sequence needed to estimate a_1 using this system. Compare it with the method designed in *Step 1.*

Step 3: Consider the sum of exponentials $x_n = \sum_{k=1}^{K} a_k^n$ with unknowns $\{a_k\}_{k=1}^{K}$. Design a filter using the prototype H to estimate the unknowns $\{a_k\}_{k=1}^{K}$. Find the minimum number of elements of the sequence needed to estimate all the unknowns $\{a_k\}_{k=1}^{K}$.

Consider the discrete samples of some signal $x_n = \sum_{k=1}^{K} c_k e^{\mathrm{j}\omega_k n}$ with the set of parameters $\{c_k, \omega_k\}_{k=1}^{K}$. Use the method designed above to estimate the parameters $\{\omega_k\}_{k=1}^{K}$. Describe concisely how the parameters $\{c_k\}_{k=1}^{K}$ could be estimated. Write a MATLAB/Python function to implement this estimation algorithm. The function must take the sequence and the number of unknowns K and output the unknowns $\{\omega_k\}_{k=1}^{K}$.

Problem 2: Corner Detection

Detecting features (such as edges, corners etc) from images is an important preprocessing step in most computer vision applications. Our goal is to reliably detect corners in an image. You have to implement this algorithm using only the *numpy* and *matplotlib* (for plottong images) library in Python, with as few other dependencies as possible or using MATLAB, without using the *Signal Processing* or *Image Processing Toolbox.*

Step 1: Write a function to compute 2D convolution. The function must take an input $N \times N$ image A and the $(2M+1) \times (2M+1)$ filter kernel K and must output the convolution (size $n \times N$) of the image and the kernel. Zero pad the input image by adding M extra rows of zero to the top and bottom of image and by adding M columns of zeros to the left and right of the image. For example If $2M + 1 = 3$ and $N = 256$ then after zero

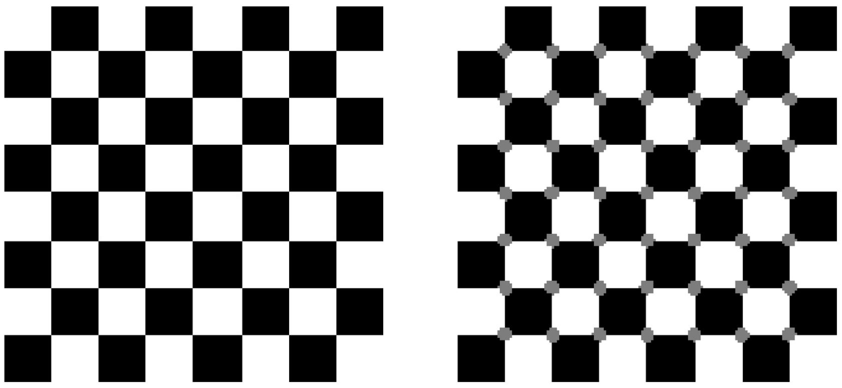

Figure 1: The gray coloured pixels in the right image above show successfully detected corners, with the left image as input.

padding, the new image should have the dimension 258 × 258. Then compute the convolution B as:

$$B[i,j] = \sum_{m=-M}^{M} \sum_{n=-M}^{M} K[m,n]A[i-m, j-n]$$

Step 2: Use this convolution function to compute the x and y image gradients using the kernels:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Let the resultant images be A_x and A_y respectively.

Step 3: Using the image gradients, compute $A_{x^2} = A_x \odot A_x$, $A_{y^2} = A_y \odot A_y$, $A_{xy} = A_x \odot A_y$. Use a 3 × 3 Gaussian kernel to smooth A_{x^2}, A_{y^2}, A_{xy} to obtain J_{x^2}, J_{y^2}, J_{xy}.

Step 4: For each pixel $[i,j]$ in the image, define a matrix:

$$M = \begin{bmatrix} J_{x^2} & J_{xy} \\ J_{xy} & J_{y^2} \end{bmatrix}$$

For each pixel, compute the parameter $R = \det(M) - 0.04 \cdot (\text{trace}(M))^2$. Generate a binary image by comparing R for each pixel with a threshold. If R is greater than threshold, assign 1 to the corresponding pixel and 0 otherwise. Try different values of threshold in the range 10^4 to 10^8.