

Homework 3

DFT and Block Transform

- Deadline: Thursday, 2020-06-11, 23:45
- Submission: Upload your code on Cody CourseworkTM Mathworks platform <https://grader.mathworks.com/courses/12790-dsp-ss20> according to the instructions given there for all problems marked with (Cx.y).
- For all problems marked with (Ax.y) refer to the quiz questions at <https://www.moodle.tum.de/course/view.php?id=54118>. There you can also find the homework rules and more information.
- Result verification: Check your functions as often as you want with Cody CourseworkTM.
- Lab session for MATLAB questions – see website, room 0943
- Notation: `variable name`, *file name*, `MATLAB_function()`

1 Discrete Fourier Transform

1. (C1.1) In this problem you will perform filtering of the image in frequency domain using bandpass filter with specified kernel parameters. For approximation of the bandpass filter we use the difference of two Gaussian filters with standard deviations σ_1 and σ_2 , respectively. Thus, the resulting filter is given as $G = G_1 - G_2$. For this write a function that given input image, size and filter parameters returns a filtered image (spatial and frequency domains). Make sure result is identical to the equivalent output of filtering in spatial domain, i.e. image is filtered using appropriate border options. (8 points) **fft(filterG,m,n) false**
2. (A1.1) When performing filtering in frequency domain on the image in step (1), analyze what effects do you observe at the border. How can you reproduce this effect when you perform filtering solely in the spatial domain? (5 points)
3. (A1.2) Observe the effects of bandpass filtering in step (1), when the input image `im` is distorted with 'salt & pepper' noise with a noise density of 0.05. Is band-pass filtering in principle influenced by salt and pepper noise? (5 points)
4. (A1.3) Compare the number of multiplications required in spatial and frequency domain. Hint: The complexity of the FFT is discussed e.g. on Wikipedia. (5 points)
5. (C1.2) The frequency transform of a real-valued signal should be conjugate point-symmetric around the (0,0) coefficient. Write a program to verify this property on `imf`. It should return true in case the property holds for this 2D signal and false otherwise.
Hint: For this and following problems, work with a normalized frequency space in the range $f \in [-1, 1[$, as used by `freqz2()`. The function `fftshift()` can be used to shift the quadrants of the frequency space between this convention and the convention of `fft2()`. (9 points)
6. (C1.3) The Discrete Fourier Transform¹ (DFT) was previously used for frequency analysis of filters. Apart from that, it exhibits some other properties which can be used to speed up certain processing tasks considerably. One of these properties we will consider here is the Fourier shift theorem, which relates a shift in the spatial domain of image i_1 by (x_0, y_0) to a phase term in frequency domain:

$$i_2(x, y) = i_1(x - x_0, y - y_0) \circ \bullet \mathcal{I}_2(\xi, v) = e^{-j2\pi(\xi x_0 + v y_0)} \cdot \mathcal{I}_1(\xi, v) \quad \text{with } \mathcal{I}_n = \mathcal{F}(i_n)$$

The phase term is isolated using the cross-power spectrum:

$$\mathcal{T} = \frac{\mathcal{I}_1(\xi, v) \mathcal{I}_2^*(\xi, v)}{|\mathcal{I}_1(\xi, v) \mathcal{I}_2^*(\xi, v)|} = e^{j2\pi(\xi x_0 + v y_0)} \quad , \text{ where } \mathcal{I}^* \text{ denotes complex conjugate.}$$

¹The Fast Fourier Transform (FFT) is an efficient implementation of the DFT for certain block sizes


By back-transforming \mathcal{T} to the spatial domain, the spatial shift is found using:

$$(t_x, t_y) = \arg \max_{t_x, t_y} \mathcal{F}^{-1}(\mathcal{T}^*)$$

inverse fft matrix
why find the
biggest term

the smaller template
id part of original
picture

fft different not only
in terms of phase-
shift

The Fourier shift theorem can help us find a spatial shift of the smaller template created from the original image by cutting out a part of it. Thus, such template can be considered a result of multiplication of the larger image with a 2D rect function (see *template1.png*). To find the spatial shift in the frequency domain you have to use the formulas given above. You need to pad the template to obtain the same size as the original image. The found coordinates of spatial shift should be returned in  \mathbf{x}, \mathbf{y} . (8 points)

2 Separable Block Transforms

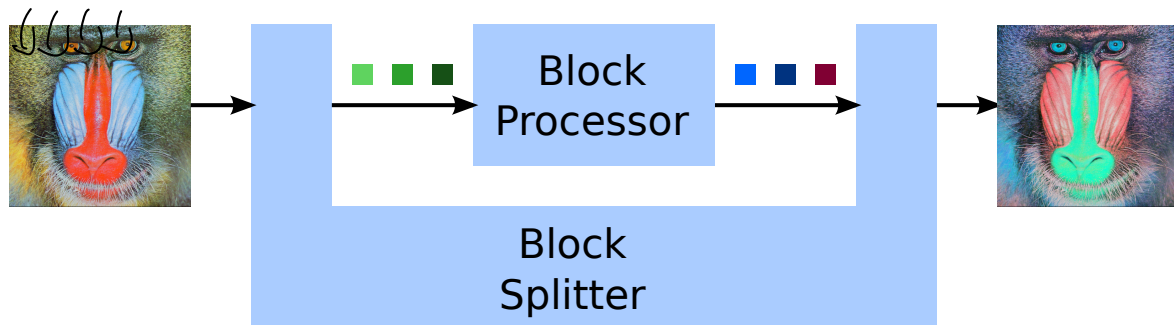


Figure 1: Block-based image processing system

When processing an image with block transformations, you generally need two types of functions: A block splitter, which splits the image into blocks of a predefined size, and a block processor which performs the actual transformation on one block. The block splitter calls the block processor once for each block and stitches the outputs of the block processor back to an image. On multicore processors or graphical processing units (GPU) several block processors can run independently in parallel, providing a simple mean of speeding up processing.

In this problem, separable block transforms of the form $Y = AXA^T$ are considered.

1. (C2.1) Implement a versatile block splitter for any block size with special support for border blocks: If there are not enough pixels at the right or bottom image border to create a full block, pad the block by repeating the last row/column using standard matrix operations. After processing, remove the padding again.
In the following tasks you will create some block processors. Test your block splitter and each block processor with `lenna.gray.jpg` and different quadratic block sizes. It is recommended to work with function handles (`doc function_handle`) in order to “plug in” any block processor into the block splitter. (10 points)
2. (C2.2) Write a simple block processor function that takes an image block as input and flips the block left to right and decreases the brightness by subtracting 0.3 from each pixel. Make sure to stay in the correct intensity range using clipping. (5 points)
3. (A2.1) Is the operation described in step (2) linear? (3 points)
4. (C2.3) In this problem, you need to apply the given Haar transform matrix to the image. For this you will implement a function that takes as input an image and Haar matrix and outputs the transformed image. You are given the function handle to the block splitter, so that you do not need to implement it again. Function arguments for the block splitter are the same as in problem (1). (7 points)
5. (C2.4) The resulting image of a block-based transform is rather meaningless for a human observer. However, when reordering the pixels such that all coefficients from the same basis function are grouped together, we can grasp the properties of the transform much better. Write a reordering routine that groups together the pixels of a Haar-transformed image in this way. For an $N \times N$ transform, you will obtain $N \times N$ sub-images of size $\frac{w}{N} \times \frac{h}{N}$ as illustrated in the lecture (with image dimensions w, h). The order of the sub-images should be the same as in the problem above – hence, the top-left sub-image should be a smaller version of the original image. (7 points)
6. (C2.5) When transforming the image using Haar transform, it is useful to analyze statistics for each coefficient over all blocks. Implement a function to calculate mean `im_mean` and variance `im_var` for each coefficient over all blocks. The resulting variables should have the same dimensions as a single block. (9 points)

-

[illegible]

blur_lenna																		
512x512 double																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0.3022	0.4468	0.4887	0.4998	0.4988	0.4919	0.4635	0.4135	0.3728	0.3537	0.3451	0.3456	0.3475	0.3463	0.3453	0.2574	0.2681	0.3576
2	0.4054	0.5988	0.6534	0.6672	0.6659	0.6571	0.6191	0.5525	0.4983	0.4721	0.4596	0.4600	0.4630	0.4610	0.4591	0.3419	0.3574	0.4777
3	0.4056	0.5988	0.6525	0.6659	0.6650	0.6564	0.6189	0.5525	0.4978	0.4703	0.4571	0.4583	0.4627	0.4615	0.4591	0.3414	0.3574	0.4789
4	0.4034	0.5951	0.6505	0.6676	0.6674	0.6564	0.6179	0.5520	0.4968	0.4694	0.4571	0.4591	0.4635	0.4620	0.4596	0.3417	0.3569	0.4789
5	0.3650	0.5431	0.6061	0.6377	0.6468	0.6382	0.6047	0.5473	0.4983	0.4750	0.4654	0.4662	0.4664	0.4615	0.4586	0.3414	0.3571	0.4792
6	0.2706	0.4118	0.4794	0.5304	0.5596	0.5691	0.5596	0.5292	0.4995	0.4858	0.4799	0.4787	0.4730	0.4637	0.4603	0.3436	0.3600	0.4831
7	0.1848	0.2794	0.3311	0.4272	0.4578	0.4831	0.4926	0.4934	0.4949	0.4936	0.4924	0.4833	0.4716	0.4689	0.3515	0.3645	0.3640	0.4897
8	0.1490	0.2098	0.2319	0.2650	0.3002	0.3402	0.3892	0.4355	0.4713	0.4924	0.5002	0.4995	0.4907	0.4836	0.4846	0.3642	0.3676	0.4944
9	0.1404	0.1870	0.1902	0.2039	0.2240	0.2559	0.3076	0.3706	0.4272	0.4684	0.4914	0.4951	0.4922	0.4949	0.4980	0.3725	0.3728	0.4988
10	0.1402	0.1850	0.1841	0.1897	0.1978	0.2147	0.2517	0.3078	0.3674	0.4201	0.4598	0.4794	0.4902	0.5010	0.5037	0.3745	0.3836	0.5074
11	0.1422	0.1885	0.1873	0.1897	0.1902	0.1934	0.2118	0.2500	0.2995	0.3544	0.4088	0.4502	0.4794	0.5010	0.5113	0.3838	0.3924	0.5145
12	0.1424	0.1890	0.1865	0.1865	0.1836	0.1797	0.1870	0.2098	0.2422	0.2892	0.3510	0.4105	0.4583	0.4929	0.5137	0.3912	0.3946	0.5169
13	0.1431	0.1895	0.1850	0.1826	0.1784	0.1733	0.1777	0.1978	0.2221	0.2488	0.2946	0.3586	0.4218	0.4681	0.4946	0.3789	0.3953	0.5201
14	0.1473	0.1958	0.1912	0.1865	0.1804	0.1750	0.1792	0.2056	0.2331	0.2338	0.2426	0.2907	0.3578	0.4152	0.4532	0.3534	0.3902	0.5196
15	0.1488	0.2010	0.1990	0.1941	0.1865	0.1799	0.1828	0.2076	0.2289	0.2137	0.2005	0.2282	0.2821	0.3402	0.3941	0.3203	0.3699	0.5056
16	0.1098	0.1505	0.1510	0.1478	0.1419	0.1363	0.1375	0.1498	0.1556	0.1422	0.1338	0.1485	0.1777	0.2157	0.2618	0.2206	0.2625	0.3676
17	0.1130	0.1395	0.1289	0.1306	0.1390	0.1434	0.1417	0.1375	0.1331	0.1289	0.1292	0.1355	0.1402	0.1473	0.1745	0.1529	0.2115	0.3130
18	0.1480	0.1853	0.1740	0.1762	0.1848	0.1890	0.1885	0.1836	0.1757	0.1725	0.1755	0.1801	0.1797	0.1809	0.2056	0.1770	0.2000	0.3755
19	0.1446	0.1848	0.1776	0.1777	0.1833	0.1890	0.1958	0.1946	0.1828	0.1784	0.1821	0.1855	0.1838	0.1762	0.1784	0.1		