Discussion of Homework 5
Autoregressive Models and Pyramids

2020-07-09

- Submission: Upload your code on Cody Coursework™ 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 Coursework™.
- Notation: `variable name`, *`file name`*, `MATLAB_function()`

1 Autoregressive Models

An autoregressive process is a generator for correlated random sequences which show similar local statistics as natural images. The simplest case for a 2D signal is a separable AR(1) process, which exhibits an autocovariance of:

$$C_{xx}(k,l) = \sigma_x^2 \cdot \rho_h^{|k|} \cdot \rho_v^{|l|}; \qquad \sigma_x^2 = \frac{\sigma_z^2}{(1-\rho_h^2)\,(1-\rho_v^2)} \tag{1}$$

The covariance between neighbouring pixels may be obtained by plugging in their distance k, l (in the horizontal and vertical direction respectively). The model has 3 parameters σ_x, ρ_h, ρ_v and can be implemented by a causal recursive filter according to:

$$x_{AR}(x,y) = \rho_h x_{AR}(x-1,y) + \rho_v x_{AR}(x,y-1) - \rho_h \rho_v x_{AR}(x-1,y-1) + \eta_{\sigma_z}(x,y), \tag{2}$$

using a zero-mean Gaussian noise source η_{σ_z} with standard deviation σ_z.

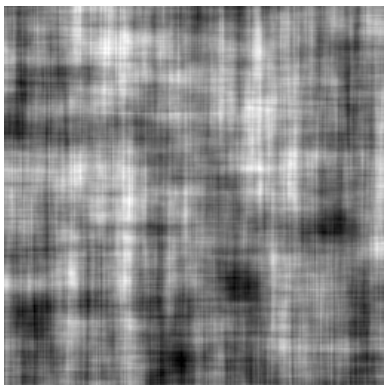


Figure 1: A 256×256 image generated by an AR(1) process

1. (C1.1) Calculate the autocovariance of a natural image I, using specified neighbourhood 11×11 for each pixel as follows:

$$C_{II}(k,l) = E\left[I(x,y) \cdot I(x+k, y+l)\right] \forall x, y \quad \text{with} \quad -5 \le k, l \le 5$$

Before processing, do not forget to subtract the mean value of the image. Save the autocovariance matrix to ☞ `acov`. Taking into account a 5 pixel border, you will calculate the covariances using $(512-10) \times (512-10)$ measurements assuming the image of size 512×512. Extract the variance (σ_x^2, ☞ `var`) and the covariances for 1 pixel horizontal (☞ `cov_h`) and vertical displacement (☞ `cov_v`). *(10 points)*

```
function [acov,  var, cov_h, cov_v] = autocov(im)

neighborhood = 11;
nh2 = floor(neighborhood/2); %Neighborhood: nh2 elem. forward/backward

% 1. subtract mean
imm = im - mean(im(:));

% 2. exclude borders
sz = size(im);
nh = nh2 * 2 + 1;
yrange = 1 + nh2:sz(1) - nh2; %Submatrix to process; exclude borders
xrange = 1 + nh2:sz(2) - nh2;
acov = zeros(nh);

% 3. compute autocovariance matrix
for t_y = -nh2:nh2
    for t_x = -nh2:nh2
        tmp = imm(yrange, xrange) .* imm(yrange+t_y, xrange+t_x);
        acov(t_y+nh2+1, t_x+nh2+1) = mean(tmp(:));
    end
end

% 4. get variance, covariance in horizontal and vertical directions
var = acov(nh2+1, nh2+1);
cov_h = acov(nh2+1, nh2+1+1);
cov_v = acov(nh2+1+1, nh2+1);

end
```

2. (A1.1) Discuss the values for `cov_v`, `cov_h`, `acov`, `var` you obtained in step (1) for the image ***lenna_gray.png*** and explain the structure of the autocovariance matrix. Plot autocovariance matrix as a 3D surface. *(10 points)*

   $var = 0.0354, cov\_h = 0.0344, cov\_v = 0.0349.$

   The center of the autocovariance matrix (i.e. the $(6,6)$ element) is the "correlation of each pixel with itself", i.e. the variance or energy of the pixels (0.035 in this case). Around the center element, we find the autocovariances of each pixel with its right neighbour $(6,7)$, its left neighbour $(6,5)$, its top neighbour $(5,6)$, etc., which amount to $\approx 97\%$ of the variance here. As we measure all pixels, the autocovariance matrix should be point symmetric around its center point (except for border effects). With increasing distance between the pixels, the autocovariance drops almost linearly. A slightly higher covariance is observed in the vertical direction.

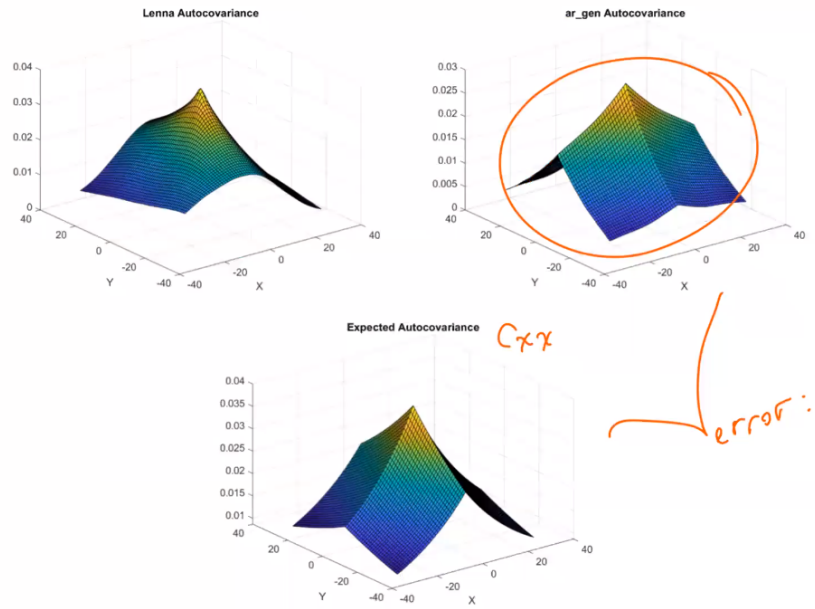
3. (C1.2) Generate a 512×512 image I_{AR} (☞ `ar_gen`) using the separable AR(1) process Eqn. (2). For this, you are given covariance in horizontal `cov_h` and vertical directions `cov_v` as well as variance `var`. Based on this and Eqn. (1), you would need to compute rv, rh and σ_z. The process will need some pixels to "start up", so add a temporary border of 100 pixels filled with zeros to the top and to the left. To verify your results, you can measure the autocovariance of this image in an 11×11 neighbourhood. *(10 points)*

```
function [ar_gen, rh, rv, std_z] = generate_ar_image(cov_h,cov_v,var)

n = 512; % Size of the image
b = 100; % "Startup" border

% compute parameters rh,rv,std_z
rh=cov_h/var;
rv=cov_v/var;
std_z=sqrt(var*(1-rh^2)*(1-rv^2));

% allocate and append border
im=zeros(b+n,b+n);

% generate AR process
for i=2:b+n
    for j=2:b+n
        im(i,j)=rv*im(i-1,j)+rh*im(i,j-1)-rv*rh*im(i-1,j-1)+std_z*
            randn(1);
    end
end

% remove border
ar_gen=im(b+1:b+n,b+1:b+n);

end
```

4. (A1.2) Compute parameters of the equivalent AR process such that C_{xx} in Eqn. (1) yields the three values you obtained from the natural image in step (1) at the corresponding positions (k, l). Compare the autocovariance of the image I_{AR} (☞ `ar_gen`) (from step (3)) and compare it to Eqn. (1). Identify the shortcomings of the autocovariance matrix of I_{AR} computed in step (3) compared to the natural autocovariance matrix from step (1). *(10 points)*

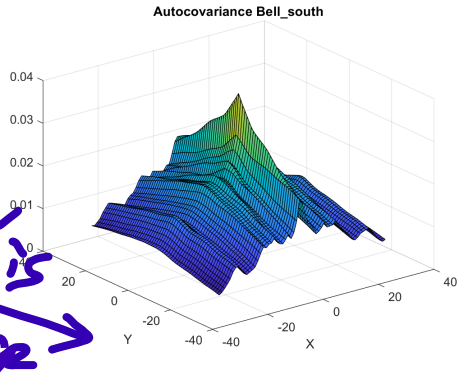
```
% Lenna Autocovariance
im = im2double(imread('data/lenna_gray.png'));
nh2 = 30;
[acov,  var, cov_h, cov_v] = autocov_(im, nh2);
figure, surf(-nh2:nh2, -nh2:nh2, acov);
title('Lenna Autocovariance'); xlabel('X'); ylabel('Y');

% ar_gen Image Autocovariance
[ar_gen, rh, rv, std_z] = generate_ar_image(cov_h, cov_v, var);
[acov_ar_gen,  var_ar_gen, cov_h_ar_gen, cov_v_ar_gen] = autocov_(
    ar_gen,nh2);
figure, surf(-nh2:nh2, -nh2:nh2, acov_ar_gen);
title('ar\_gen Autocovariance'); xlabel('X'); ylabel('Y');

% Expected Autocovariance
[t_h, t_v] = meshgrid(-nh2:nh2, -nh2:nh2);
rxx_exp = var * rh.^abs(t_h) .* rv.^abs(t_v);
figure, surf(-nh2:nh2, -nh2:nh2, rxx_exp);
title('Expected Autocovariance'); xlabel('X'); ylabel('Y');
```

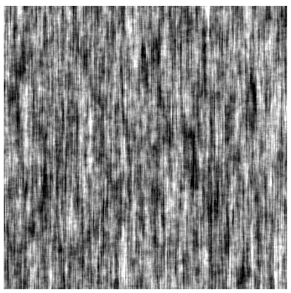
The separable AR(1) process produces an autocovariance which is piecewise planar in the four quadrants. This results in a modelling error which increases from the center towards the border along $k = 0$ or $l = 0$. Furthermore, one observes a constant offset between the measured autocovariance of the AR(1) image and Eqn. (1). That error is minimized by using a larger "startup border".



5. (A1.3) Repeat step (1) with image `bell-south.jpg` and save the resulting autocovariance matrix. Explain why this matrix looks considerably different, even though it is also obtained from a natural image. How would an image generated by an AR(1) process with parameters measured from `bell-south.jpg` look like? *(5 points)*

The resulting autocovariance matrix exhibits a significantly higher correlation in the vertical direction, due to the dominant vertical structures in the image. An AR(1) image created with corresponding parameters exhibits vertically dominating structures.





Left: Autocovariance for `bell-south.jpg`, right: image generated using AR(1) process with parameters computed from `bell-south.jpg`.

2 Pyramid Representations

Pyramid or multiscale representations are very useful for many image processing tasks. For instance, they form the fundamental preprocessing step in modern image feature transforms such as SURF[1], which are used in computer vision applications.

1. (C2.1) A simple 3×3 approximation of a Gaussian lowpass filter is given on lecture page 109. In this problem, assume you used this filter for 2:1 reduction (horizontally and vertically). Now you need to reconstruct the image to the original resolution. For this, write a routine to compute an appropriate 3x3 reconstruction filter for 1:2 upsampling. The filter can be found by inspection. *(8 points)*

   After the 1:2 upsampling step, each second pixel in the horizontal and vertical direction is unknown and set to zero. The reconstruction filter fills in these gaps by linear interpolation with equal weights from two or four neighbours. The value of known pixels should not change. This leaves only one possible solution for a 3×3 reconstruction filter, which can be found by inspection.

```
function G = compute_reconstruction_filter()
G = 1/4 * [ 1 2 1 ; 2 4 2 ; 1 2 1 ];
end
```

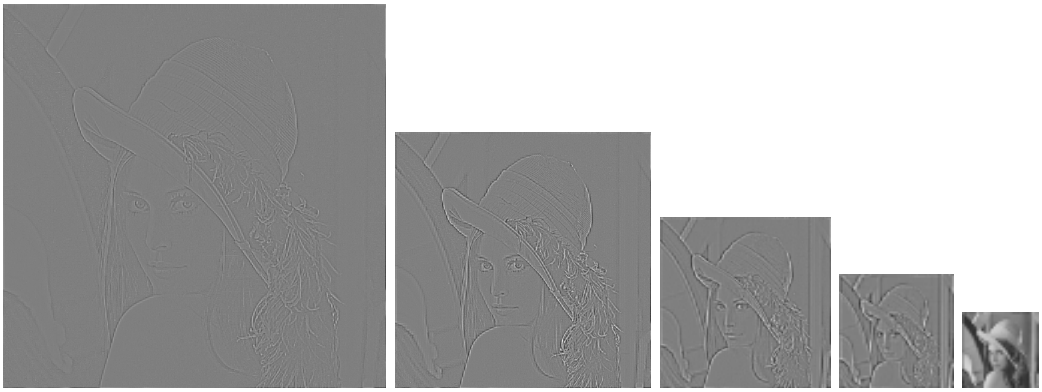
2. (C2.2) Generate a Laplacian pyramid of the given image *lenna_gray.jpg* using the filters from step 1 with 5 levels. Level 1 holds the finest details at full resolution (512×512), while level 5 corresponds to the lowest level of detail (of resolution 32×32). Like that, the scaling factor between two levels is 2:1 horizontally and vertically. Use `imfilter` with the `replicate` option and return your results from the function for all levels as a cell array ☞ `laplacian_pyr{1,2,3,4,5}`. *(10 points)*

```
function laplacian_pyr = generate_laplacian_pyr(im,F, G)

NL = 5; % number of levels
laplacian_pyr = cell(1,5); % allocate memory for the Laplacian pyramid
x = im;

for level=1:(NL-1)
    % Filter & Downsample
    xf = imfilter(x, F, 'replicate');
    xf = xf(1:2:end, 1:2:end);
    % Upsample & filter
    xr = zeros(2*size(xf));
    xr(1:2:end, 1:2:end) = xf;
    xr = imfilter(xr, G, 'replicate');
    % Difference image
    laplacian_pyr{level} = x - xr;
    % For next loop:
    x = xf;
end
laplacian_pyr{level+1} = x;

end
```

symmetric
Conv = convolution

low subsampled version as input for next iteration

[1] Bay et al., SURF: Speeded Up Robust Features, European Conference on Computer Vision, 2006



Levels 1 to 5 of the Laplacian pyramid.
The scaling between two levels is 2, but depicted as 1.5 here.

512 256 128 64 32

3. (C2.3) In this problem you will work with the 9-7 Cohen-Daubechies-Feauveau wavelet[2]. In particular, you are asked to obtain the 2D filter kernels for analysis and synthesis steps, ☞ `F` and ☞ `G` respectively. You are already given separable 1D lowpass filter coefficients for these wavelets. *(5 points)*

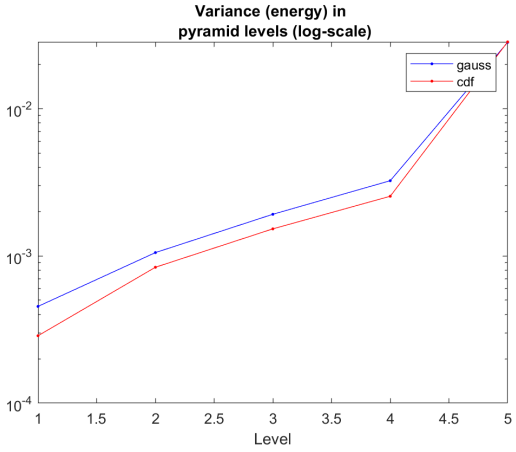
```
function [F,G] = compute_CDF_filter_kernel(f1d, g1d)
F = f1d * f1d';
G = g1d * g1d';
end
```

4. (C2.4) Write a function that calculates the variances of each level for your pyramid representation (you will observe results for pyramids from steps 2 & 3 and compare the variances when using Gaussian filters and 9-7 Cohen-Daubechies-Feauveau wavelets). Return the variances in 5×1 vector ☞ `var_pyramid`. To neglect border effects, remove a border of 4 pixels around the images at each level. *(7 points)*

```
function var_pyramid = calculate_statistics_pyramid(pyramid_structure)

NL = 5;
var_pyramid = zeros(NL,1);
b = 4;    % border margins
for i = 1:NL
    cim1 = pyramid_structure{i};
    cim1 = cim1(b+1:end-b, b+1:end-b);
    var_pyramid(i) = var(cim1(:));
end

end
```

[2] http://en.wikipedia.org/wiki/Cohen-Daubechies-Feauveau_wavelet

5. (A2.1) Explain the difference you observe for variances when using 9-7 Cohen-Daubechies-Feauveau wavelet and Gaussian filters. *(5 points)*



In general, the variances (or energies) of the difference levels (1-4) are lower than in the last level 5. With CDF filters, a little more energy is packed into level 5 (but below the tolerance of 0.001), but the lower levels (finer details) exhibit considerably less energy.

6. (C2.5) In this problem you will need to reconstruct the image from the pyramid representation. Instead of taking all levels of the pyramid, set the two levels with the finest details to 0. After that reconstruct the images at full resolution and return them in ☞ `im_reconstruct`. Calculate the PSNR of the reconstructed images and return it in ☞ `im_psnr` – again neglecting a border of 4 pixels. *(10 points)*

```
function [im_reconstruct, im_psnr] = reconstruct_image(pyramid, im, G)
NL = numel(pyramid);
mode = 'replicate';
pyramid{1} = 0*pyramid{1};
pyramid{2} = 0*pyramid{2};
b = 4;

xcur = pyramid{NL};
for level = NL-1:-1:1
    % Upscale & filter previous (coarser) level:
    xr = zeros(size(pyramid{level}));
    xr(1:2:end, 1:2:end) = xcur;
    xr = imfilter(xr, G, mode);
    % Add current level (difference image):
    xcur = xr + pyramid{level};
end
im_reconstruct = xcur;
mse = (im - im_reconstruct).^2;
mse = mse(b+1:end-b, b+1:end-b);  % Ignore border effects
mse = mean(mse(:));
im_psnr = 10 * log10(1/mse);

end
```

upsample filter interpolate

7. (A2.2) Explain the values you obtain for PSNR for two reconstruction filters in step 6. What kind of artifacts do you observe? Is there a difference between the representations from steps 2 and 3? *(10 points)*

The artifacts are essentially those of a separable low-pass filter applied to the image. There are no block artifacts. With the pyramid based on Gaussian filters, we see more blurring than with the pyramid based on CDF.

The PSNR for Gaussian filters is around 28 dB. Comparing Gaussian and CDF filters, we see an improvement of 1.28 dB with the latter ones. This can be explained by the fact that as CDF filters pack more energy into higher levels, less information is lost when setting lower levels to zero.