Discussion of Homework 4
Transformations: DCT and KLT

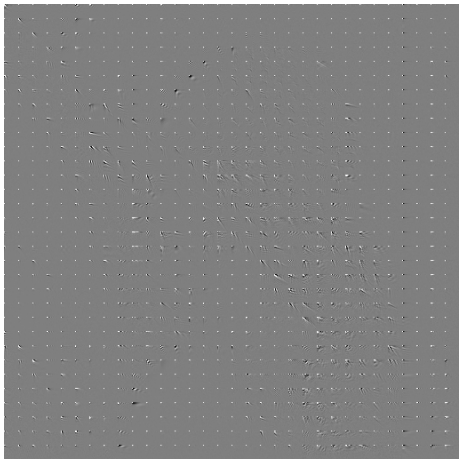
2020-06-25

- 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 Discrete Cosine Transform (DCT)

The Discrete Cosine Transform (DCT) is a separable linear transform used in image and video compression. It exhibits very beneficial properties for energy compaction in natural images. Contrary to the DFT, the DCT is a real-valued transformation.

1. (C1.1) Here you will transform images to DCT domain using block processing. For this write a function that takes as input DCT transformation matrix, function handle to block splitter and image for transformation. Its output will be the image transformed into DCT domain. The provided handle to block splitter has the same signature as in previous problems. When displaying the resulting image for visualization purposes, remember to use appropriate scaling. *(5 points)*



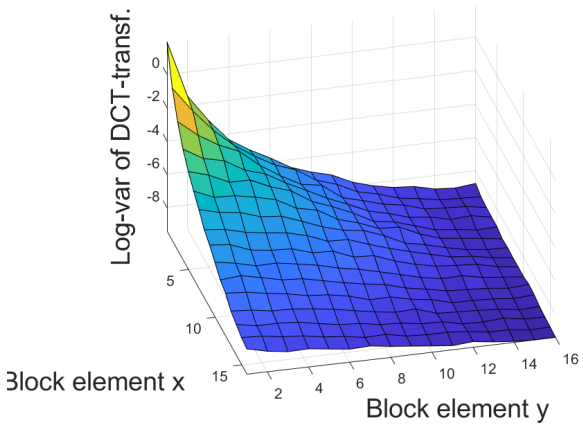
DCT-transformed image

```
function dct_image = transform_to_dct_domain(im, dct_matrix, ...
    fcnHandleBlkSplitter)
% DCT on a block : A * block_in * A' where A is the transform matrix
linear_block_transform = @(block_in, A) A * block_in * A';
block_sz = size(dct_matrix,1);
% Apply block splitter and block processor with the given ...
    fcnHandleBlockSplitter
dct_image = fcnHandleBlkSplitter(im, block_sz, linear_block_transform,...
     dct_matrix);
end
```

For simple, almost uniform blocks, only the DC or low frequency coefficients exhibit energy.Most other coefficients are close to zero. In more complicated areas of the image, such as the hair, there is a large number of non-zero high-frequency coefficients. Therefore, energy compaction is less efficient here. A characteristic coefficient pattern is observed in blocks showing a clear edge: The active coefficients are located in a region which is perpendicular or parallel to the original edge.

2. (A1.1) How do the coefficients in each block in step (1) correspond to the contents of the original image? Calculate the variances and means of the DCT coefficients, using all blocks from the transformed image. Use your previously implemented function `calculate_statistics()`, from **Homework 3**. How does the mean matrix look like? How is it related to the mean of the untransformed image? Create a 3D plot of the variances. What "rule" for the decay of the variances do you observe? *(6 points)*

The mean of the DC-coefficient is proportional to the mean of the untransformed image in the respective area. AC-coefficients exhibit zero mean. The variance of the DC coefficient is largest, and it decreases with increasing spatial frequency. Variances in horizontal directions exhibit slightly higher amplitude when compared to the vertical direction.



Variances of DCT coefficients (log scaling)

```
% Stats for DCT-transformed image
[dct_mean, dct_var] = calculate_statistics(dct_image, M);

figure;
surf(log(dct_var));
xlim([1 size(dct_var,1)]);
ylim([1 size(dct_var,2)]);
zlim([min(log(dct_var(:))), max(log(dct_var(:)))]);
zlabel('Log-var of DCT-transf.', 'fontsize',18);
xlabel('Block element x', 'fontsize',18);
ylabel('Block element y', 'fontsize',18);
view(75.5, 38);
```

3. (C1.2) In this task you will perform compression of a DCT-transformed image. For each transformed block of the Lenna image (block size 16×16), keep only the first N coefficients and set the remaining coefficients to zero. Use an appropriate quadratic sub-block of side length $\sqrt{N}$, starting

with the DC coefficient. Return the image (in spatial domain) resulting from compression with the given coefficients in ☞ `im_rec_comp`. You are also given as arguments the function handle to split image into blocks as well as DCT transformation matrix. *(7 points)*

```
function im_rec_comp = compress_dct(dct_image, dct_matrix, ↩
    fcnHandleBlkSplitter, N)

block_size = 16;
block_processor_linear = @(block_in, A) A * block_in * A';

% 1. remove all but first N coefficients
d = zeros(block_size,1);
d(1:sqrt(N)) = 1;
A_ = diag(d);
im_dct_comp = fcnHandleBlkSplitter(dct_image, block_size, ↩
    block_processor_linear, A_);

% 2. back-transform the image using given function handles
im_rec_comp = fcnHandleBlkSplitter(im_dct_comp, block_size, ↩
    block_processor_linear, dct_matrix');

end
```

4. (A1.2) Calculate the PSNR for each compression level for image in step (3). What effects do you observe for the different compression levels ($N = 9, 16, 36, 64$) in step (3)? What are the critical image regions? Do your observations justify equal compression for all image blocks? In JPEG, coefficients are ordered using the zig-zag scheme. Explain why this is a better way to order them "by importance". *(6 points)*

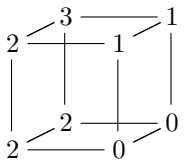
require high frequency components

Significant block effects are observed in image regions with much detail, such as the hair and the eyes, if only 9 coefficients are kept. Only minor block effects are observed with 64 coefficients, which still corresponds to a compression factor of $\frac{256}{64} = 4$. *The various images regions obviously require different numbers of coefficients in order to look well for a human observer: Simple image regions, such as the uniform background blocks, require only few coefficients, while for blocks with complex details, even 64 coefficients are not enough. The PSNR values range from 27.1 dB (for 9 coefficients) to 35.4 dB (for 64 coefficients) with a 2 – 3 dB increase per step.*

depend on Block content

Reconstructed images after compression, keeping only the first 9 (left) or 64 (right) coefficients

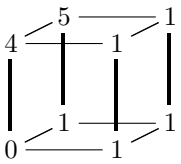
5. (C1.3) Because of its separable property DCT transformation can be efficiently extended to 3 dimensions. With the following example, we provide how 3D DCT can be applied seperately in each dimension of the following 3D block:



Since the M-D version of the DCT is separable, we can successively apply the 1-D transform given by `dct_matrix` along the three dimensions independently.

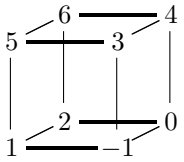
In no particular order, we choose to apply it first on all the vertical columns:

$$\texttt{dct\_matrix}\left[\begin{array}{c|c|c|c} 2 & 1 & 3 & 1 \\ 2 & 0 & 2 & 0 \end{array}\right] = \tfrac{1}{\sqrt{2}}\left[\begin{array}{c|c|c|c} 4 & 1 & 5 & 1 \\ 0 & 1 & 1 & 1 \end{array}\right] \quad \Longrightarrow \quad \tfrac{1}{\sqrt{2}} \cdot$$



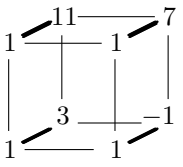
then on all the horizontal rows of the result thereof:

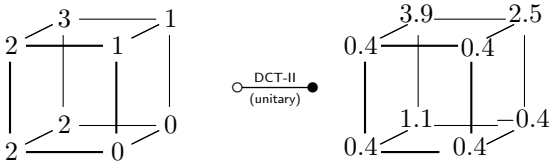
$$\texttt{dct\_matrix}\left[\begin{array}{c|c|c|c} 4 & 0 & 5 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right] \tfrac{1}{\sqrt{2}} = \tfrac{1}{2}\left[\begin{array}{c|c|c|c} 5 & 1 & 6 & 2 \\ 3 & -1 & 4 & 0 \end{array}\right] \quad \Longrightarrow \quad \tfrac{1}{2} \cdot$$



and eventually, we apply `dct_matrix` on the "out-of-plane rows" to obtain the final result:

$$\texttt{dct\_matrix}\left[\begin{array}{c|c|c|c} 6 & 4 & 2 & 0 \\ 5 & 3 & 1 & -1 \end{array}\right] \tfrac{1}{2} = \tfrac{1}{\sqrt{8}}\left[\begin{array}{c|c|c|c} 11 & 7 & 3 & -1 \\ 1 & 1 & 1 & 1 \end{array}\right] \quad \Longrightarrow \quad \tfrac{1}{\sqrt{8}} \cdot$$





In this question you will apply DCT matrix along each of the 3 dimensions subsequently to get the 3D transformed DCT image. Make sure that you get the same result as in the above example. *(9 points)*

```
function im_dct = apply_3ddct(im_in, dct_mat)

% 1. vertical columns
  for i=1:size(im_in,3)
      for j=1:size(im_in,2)
          vert(:,j,i)=dct_mat*im_in(:,j,i);
      end
  end

  % 2. horizontal rows
  for i=1:size(im_in,3)
      for j=1:size(im_in,1)
          hor(j,:,i)=dct_mat*vert(j,:,i)';
      end
  end

  % 3. out-of-plane rotations
  for i=1:size(im_in,1)
      for j=1:size(im_in,2)
          im_dct(i,j,size(im_in,3):-1:1)=dct_mat*squeeze(hor(i,j,end
              :-1:1));
      end
  end
end
```

6. (C1.4) Here you will apply the previously computed 3D DCT for analysis of the 3D sequence, in this case video sequence. This way, the third dimension extends along the time dimension of the sequence and allows to encode temporal correlation efficiently. You are given the handle to the 3D DCT transformation function (signature as in problem before). Return the DC and high-frequency components of the resulting DCT transformed sequence. You can ignore padding of the sequence for this problem. *(7 points)*

```
function [video_dc, video_hf] = threed_dct_analysis(video, dct_mat, ...
    fcnHandle3DDCTTransformation)

    sz_b = 2; % block size
    sz_v = size(video);
    video_out = zeros(sz_v);
    for i=1:sz_b:sz_v(1)-1
        for j=1:sz_b:sz_v(2)-1
            for k=1:sz_b:sz_v(3)-1
                temp = video(i:i+1,j:j+1,k:k+1);
                im_out = fcnHandle3DDCTTransformation(temp, dct_mat);
                video_out(i:i+1,j:j+1,k:k+1) = im_out;
            end
        end
    end
    video_dc = video_out(1:sz_b:end, 1:sz_b:end,2:sz_b:end);
    video_hf = video_out(2:sz_b:end, 2:sz_b:end,1:sz_b:end);
end
```

7. (A1.3) Which DCT component of the 3D DCT transformed sequence contains the most energy? What is the best strategy for video compression using 3D DCT? *(6 points)*

The most energy is stored in DC component and the energy decreases along the major axes with increasing distance from 0,0,0 coordinate. The best strategy depends on the content. For sequences with very dynamic content but with areas of same color it is best to preserve more DCT coefficients corresponding to time dimensions as compared to DCT coefficients of spatial information. For sequences with high level of spatial detail and static content it is best to preserve the coefficients corresponding to the spatial domain.

2 Karhunen-Loève-Transform

The Karhunen-Loève-Transform (KLT) performs a complete decorrelation of correlated data and thus allows for the best energy concentration using a linear transform. It must be trained from a specific dataset (e.g. a set of images or a set of image blocks), but shows certain general properties, which depend on the statistics of the dataset. Work with formulas given in the lecture using the autocorrelation matrix C_{xx} and do not use built-in MATLAB functions for correlation blindly.

1. (C2.1) *Preparatory step:* As explained in the lecture, 2D image blocks are reshaped into 1D column vectors for the KLT, which allows us to use matrix-vector notation. The block is read out column by column, starting from the top-left (column-major order!). Like that, an 32×32 (base) image is reshaped into a 1024×1 vector. Implement a block processor which reshapes the block into a vector, adds 0.1 to the vector elements `[529:544,272:32:784]` and reshapes the result back to a block. Transform the image with block size 32 and save the result to ☞ `im_reshape`. Make sure you understand how reshaping is performed and how the resulting image is generated. *(5 points)*

```
function im_reshape = reshape_image(im, block_splitter)
N=32;
v = zeros(N^2,1);
v([529:544,272:32:784]) = 0.1;
func_handle = @(block, add) reshape(block(:) + add, N, N);
im_reshape = block_splitter(im, N, func_handle, v);
end
```

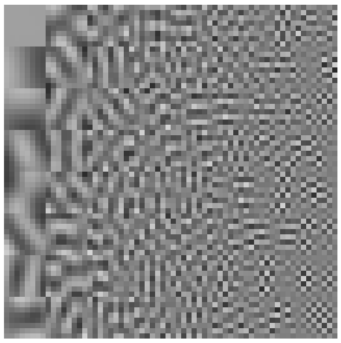
2. (C2.2) Calculate the KLT basis functions from the *`lenna_gray.jpg`* image for 8×8 blocks. The following steps are needed to be performed:

   - First, restructure image into blocks of 8×8 size and return the result in ☞ `blocks`.
   - Calculate the autocorrelation matrix for blocks of 8×8 size and save the result to ☞ `C_im`. `C_im` should have the size of 64×64.
   - Perform the eigenvalue decomposition of the autocorrelation matrix `C_im`.
   - Sort the resulting eigenvalues with descending order and correspondingly the eigenvectors.
   - The resulting eigenvector constitutes the KLT basis images. Reshape each vector to a block of 8×8 and pack them all into a 64×64 image in column-major order.

   Your function should return the restructured image in ☞ `blocks`, autocorrelation matrix ☞ `C_im`, and the KLT basis functions in ☞ `klt_base`. *(10 points)*

```
function [blocks, C_im, klt_base] = compute_klt_basis(im)

N=8;
% 1. Allocate memory for blocks
sz = size(im);
x_arr = 1:N:(sz(2) - N+1);
y_arr = 1:N:(sz(1) - N+1);
blocks = zeros(length(x_arr) * length(y_arr), N*N);

%2. Regroup the input image into blocks
for iy = 1:length(y_arr)
    for ix = 1:length(x_arr)
        x_start  = x_arr(ix);  y_start = y_arr(iy);
```

```
        block_in = im(y_start:y_start+N-1,  x_start:x_start+N-1);
        blocks((iy-1)*length(x_arr) + ix, :) = block_in(:);
    end
end

%3. Compute autocorrelation over all blocks
C_im = blocks'*blocks/size(blocks,1);

%4. Compute eigenvalue decomposition of the autocorrelation matrix and
    thus get the KLT basis
[klt_base_v,ew] = eig(C_im);
[ew,ew_idx] = sort(diag(ew), 'descend');
klt_base_v = klt_base_v(:, ew_idx);
klt_base = zeros(N^2);

for x = 0:N-1
    for y = 0:N-1
        idx = x*N+y+1;
        klt_base(y*N+1:(y+1)*N, x*N+1:(x+1)*N) = ...
            reshape(klt_base_v(:, idx), N, N);
    end
end

end
```



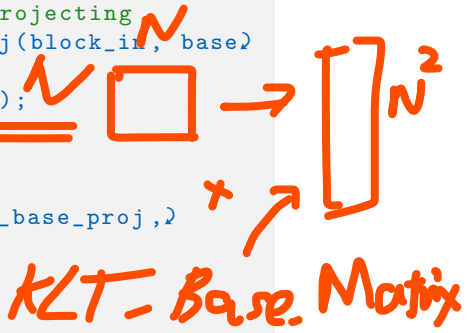
KLT basis images for image `lenna_gray.jpg`

3. (A2.1) Compare KLT basis images when using different images for training. *(5 points)*

   KLT is the most efficient transform in terms of decorrelation and energy packing; however, it is dependent on image statistics. As in the above question, autocorrelation of a specific image is calculated and KLT basis images are derived from that. Hence, for different images with different statistics, the resulting KLT basis images would be different.

4. (C2.3) Write a function to project the image to the (same) KLT basis which was obtained above. Return the resulting images in ☞ `im_projected`. Coefficients should be ordered the same way as the basis images. Also compute the back-projected image and return in ☞ `im_backprojected`. Compare the back-transformed image with the original ones and return the calculated PSNR in ☞ `klt_psnr`. *(8 points)*

```
function [im_projected, im_backprojected, klt_psnr] = ...
    project_reconstruct_klt_basis(im, klt_base_v, block_splitter)

N = sqrt(size(klt_base_v,1)); % block size

% 1.a First defined a block processor for orthogonal base for ...
    projecting
function block_out = block_processor_orth_base_proj(block_in, base)
    block_out=reshape(base'*reshape(block_in,N*N,1),N,N);
end

% 1.b. Define block processor for othogonal base backprojecting
function block_out = block_processor_orth_base_backproj(block_in, base...
    )
    block_out=reshape(base*reshape(block_in,N*N,1),N,N);
end

% 2. Then using a block splitter project the image
im_projected=block_splitter(im,N,@block_processor_orth_base_proj,...
    klt_base_v);

% 3. Then backproject to the original image
im_backprojected=block_splitter(im_projected,N,...
    @block_processor_orth_base_backproj,klt_base_v);

% 4. And compute PSNR of the difference
im_diff = (im_backprojected - im).^2;
klt_psnr = 10 * log10(1/mean(im_diff(:)));

end
```

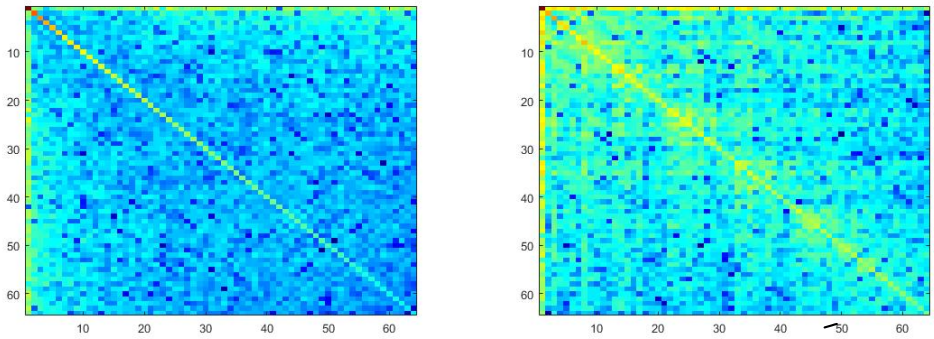


5. (C2.4) Write a function to calculate the autocorrelation matrix of all blocks from KLT transformation and return in ☞ `autocorr`. Here you can reuse your approach for block grouping and autocorrelation computation from Problem 2.2. *(5 points)*

```
function autocorr = compute_autocorrelation_klt_images(...
    klt_projected_image, N)

% 1. Allocate memory for blocks
sz = size(klt_projected_image);
x_arr = 1:N:(sz(2) - N+1);
y_arr = 1:N:(sz(1) - N+1);
blocks = zeros(length(x_arr) * length(y_arr), N*N);

%2. Regroup the input image into blocks
for iy = 1:length(y_arr)
    for ix = 1:length(x_arr)
        x_start  = x_arr(ix);  y_start = y_arr(iy);
        block_in = klt_projected_image(y_start:y_start+N-1,  x_start:...
            x_start+N-1);
        blocks((iy-1)*length(x_arr) + ix, :) = block_in(:);
    end
```

```
end

%3. Compute autocorrelation over all blocks
autocorr = blocks'*blocks/size(blocks,1);

end
```

6. (A2.2) How should the autocorrelation matrix of KLT blocks look like? When evaluating function from step (5), compare results for image *lenna_gray.jpg* and *mandrill_gray.png*. When transforming *mandrill_gray.png* use the KLT basis images trained from *lenna_gray.jpg*. Explain the result for the *mandrill_gray.png* image considering the properties of the KLT. Is KLT suitable as a generic compression method? *(8 points)*



Autocorrelation matrices (log scale) for Lenna (left) and Mandrill (right)

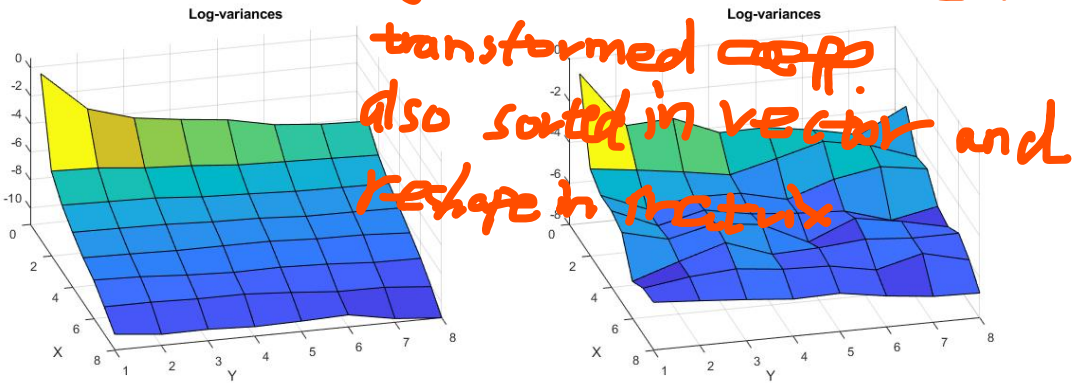
The KLT base was learned from the Lenna image, so it allows for perfect decorrelation (except for numerical errors). Thus, the autocorrelation matrix exhibits only elements on its diagonal and zero elsewhere. This is not the case for other images being projected on Lenna's base: The diagonal of the autocorrelation matrix still dominates, but there are significant non-diagonal elements. Therefore, the KLT is not suitable as a generic compression method.

7. (A2.3) Calculate the statistics, ☞ `klt_mean` and ☞ `klt_var`, of the KLT transformed images (using the KLT lenna basis images) *lenna_gray.jpg* and *mandrill.png*. You can use your previously implemented function `calculate_statistics()`, from **Homework 3**. Then, plot the resulting variances using `surf(log(klt_var))`. How do you explain the differences between the decay in variance (or energy) for the two images? *(8 points)*

```
% Lenna Image
im_lenna = im2double(imread('data/lenna_gray.jpg'));
% KLT basis images of Lenna image
load('klt_base_v.mat', 'klt_base_v');
block_splitter = @block_splitter_image;
[im_projected_lenna, im_backprojected_lenna, klt_psnr_lenna] = ...
    project_reconstruct_klt_basis(im_lenna,klt_base_v,block_splitter);
% Calculate Statistics
N = 8;
```

```
[klt_mean_lenna, klt_var_lenna] = calculate_statistics(
    im_projected_lenna,N );
figure,surf(log(klt_var_lenna)); title('Log-variances');  view (75.5
    ,38);
xlabel('X'); ylabel('Y');

% Mandrill Image
im_mandrill = im2double(imread('data/mandrill_gray.png'));
% KLT basis images of Lenna image
load('klt_base_v.mat', 'klt_base_v');
block_splitter = @block_splitter_image;
[im_projected_mandrill, im_backprojected_mandrill, klt_psnr_mandrill]
    = project_reconstruct_klt_basis(im_mandrill,klt_base_v,
    block_splitter);
% Calculate Statistics
N = 8;
[klt_mean_mandrill, klt_var_mandrill] = calculate_statistics(
    im_projected_mandrill,N );
figure,surf(log(klt_var_mandrill)); title('Log-variances');  view (75
    .5,38);
xlabel('X'); ylabel('Y');
```

eigenvector are sorted transformed coeff also sorted in vector and reshape in matrix



Variances of coefficients for the KLT Lenna (left) and Manrdill (right)

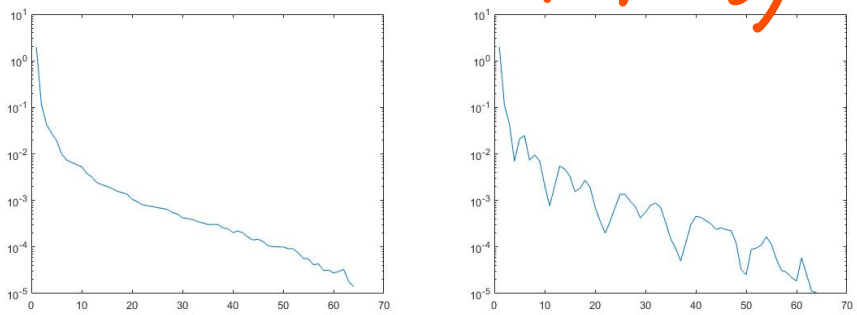
The KLT of Lenna on its own base exhibits a monotonic decrease of the variance of coefficients. For other images, the energy compaction is not optimal – therefore, the KLT of Mandrill on Lenna's base shows a noisy behaviour in the variance plot.

8. (A2.4) Repeat the previous step for DCT-transformed versions of the *lenna_gray.jpg* image, using this time `semilogy()` to visualize the variance of the transform coefficients in log scale. How does the performance of the DCT and KLT transformations differ? *(5 points)*.

```
% KLT
im_lenna = im2double(imread('data/lenna_gray.jpg'));
% KLT basis images of Lenna image
load('klt_base_v.mat', 'klt_base_v');
block_splitter = @block_splitter_image;
[im_projected_lenna, im_backprojected_lenna, klt_psnr_lenna] =
    project_reconstruct_klt_basis(im_lenna,klt_base_v,block_splitter);
% Calculate Statistics
N = 8; % block size
```

```
[klt_mean_lenna, klt_var_lenna] = calculate_statistics(...
    im_projected_lenna,N );
figure,semilogy(klt_var_lenna(:));

% DCT
im_lenna = im2double(imread('data/lenna_gray.jpg'));
M = 8; % block size
dct_matrix = dctmtx(M);
block_splitter = @block_splitter_image;
% dct transformed image
dct_image = transform_to_dct_domain(im_lenna, dct_matrix, ...
    block_splitter);
% Calculate Statistics
[dct_mean_lenna, dct_var_lenna] = calculate_statistics(dct_image, M);
dct_var_lenna_ordered = ZigZag8x8(dct_var_lenna);
figure,semilogy(dct_var_lenna_ordered(:));
```

frequency



Variances of coefficients for the KLT Lenna (left) and DCT Lenna (right)

The KLT of Lenna on its own base exhibits a monotonic decrease of the variance of coefficients and higher energy compaction efficiency to lower order coefficients compared to the DCT.