Discussion of Homework 2
Image Sampling and Filtering

2020-05-28

- 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 Image Sampling

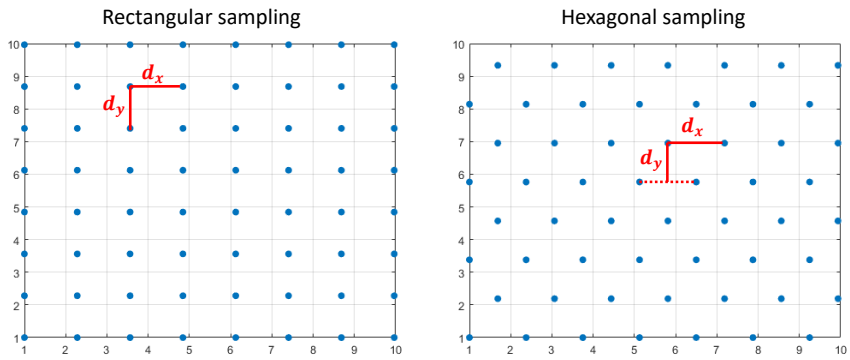


Figure 1: Sampling points for hexagonal and rectangular sampling grids

In this question we will work on image sampling with two different sampling grids, rectangular and hexagonal. In a hexagonal grid (Figure 1), the distances of a sampling point to all its 6 neighbors are equal, whereas in a rectangular grid there are 4 neighbors with distance a and 4 more neighbors with distance $\sqrt{2}a$. In this problem we investigate sampling effects on a natural image.

1. (C1.1) Generate a regular and rectangular sampling grid of dimensions $N \times N, N = 400$, which starts at $(1, 1)$ and spans (exactly!) the range $[1, 512]$ for x and y. Use `meshgrid()` and save the sampling coordinates as $N \times N$ matrices to ☞ `gridX` and ☞ `gridY`. Note the convention of `meshgrid()` to return sampling coordinates as matrices which resemble the spatial locations of the sampling points. *(5 points)*

```
function [gridX, gridY] = generate_rect_grid(sz, N)
grid_ = linspace(1, sz(1), N);
[gridX, gridY] = meshgrid(grid_);
end
```

2. (C1.2) Now you start working with the gray scale Lenna image, `im`. In order to find image intensity at non-integer coordinates, you need to use 2D interpolation: Create a smaller interpolated version of `im` of size $N \times N$ using `interp2()` with the grid from above and try the different interpolation options. As the resizing factor is small, skip the low-pass filtering step. Implement the corresponding function so that it performs interpolation using "linear" and "cubic" interpolation options and returns both images. Note: you do not need to implement the interpolation yourself, just reuse the built-in Matlab function. *(5 points)*

```
function [im_linear, im_cubic] = interp_im_rect_grid(im, gridX, gridY)
sz = min(size(im));
[imX, imY] = meshgrid(1:sz);  % image is given at these positions
% now implement the interpolation using  linear and cubic interp
im_linear = interp2(imX, imY, im, gridX, gridY, 'linear');
im_cubic = interp2(imX, imY, im, gridX, gridY, 'cubic');
end
```

3. (A1.1) Describe how the resulting images (i.e. images after cubic and linear interpolation) differ and compare the required runtime, e.g. with `tic / toc`. Make sure to repeat interpolation for 10 times when performing the timing experiments and accumulate the timing. Otherwise, the implementation differences in different Matlab versions can lead to inconsistent results. *(5 points)*

Image interpolated using cubic interpolation can better preserve edges and other fine components, at the same time linear interpolation results in loss of these details. The linear interpolation runtime is smaller comparing to cubic one. Timing and visual difference between linear and cubic interpolation: Cubic interpolation produces better visual result but is slower.

4. (C1.3) Next, reconstruct the image in its original size 512×512 from the smaller `im_small`. You are given grid coordinates as computed from problem (C1.1). Images of square size are assumed. Use the `TriScatteredInterp()` class with "linear" interpolation this time. It works similar to `interp2()`, but can also handle non-regular grids. The resulting function will return the reconstructed image `im_rec`. *(6 points)*

```
function im_rec = reconstruct_from_smaller_image(im_small, sz, gridX, ↩
    gridY)
N = size(gridX, 1);
[imX, imY] = meshgrid(1:sz);
% 1. Apply triscatteredinterp function
im_interp = TriScatteredInterp(gridX(:), gridY(:), im_small(:), '↩
    linear');
% 2. Then interpolate image to the original image coordinates
im_rec = im_interp(imX, imY);
end
```

5. (C1.4) Generate a horizontally aligned hexagonal grid with the first sampling point located at $(1, 1)$. The horizontal sampling distance is $d_x = \frac{511-1}{372-1}$ and the vertical sampling distance is $dy = \sqrt{3} \cdot 0.5 \cdot dx$, which are depicted in Figure 1. The sampling locations must span horizontally (exactly!) $x \in [1, 511 + \frac{1}{2}d]$ and vertically $y \in [1, 512[$. The function you implement should return the grid coordinates stored in two matrices ☞ `hexaX` and ☞ `hexaY`. *(10 points)*

Hints: This hexagonal grid will have a total of 159960 sampling points. Create it as follows:

- Generate the points of the grid first using `meshgrid()` and with the given d_x and d_y.
- Shift each second raw of the calculated grid by $0.5d_x$ to obtain the hexagonal grid.
- Verify that the grid fulfills the given constraints and that the distances of each point to all of its six neighbors are equal.

The function `meshgrid()` is used to generate regular 2D grids. For 1D, you can use `linspace()` or the `a:b:c` notation. Instead of generating even and odd rows of the hexagonal grid separately, we can generate a single grid and shift each second row.

```
function [hexaX, hexaY] = generate_hexagonal_grid()
sz = 512;
x_sz = sz-1;
dx = (x_sz-1)/(372-1); % horizontal displacement in hexaX
grid_hx = 1:dx:(x_sz);
dy = dx * sqrt(3)/2;   % vertical displacement in hexaY
grid_hy = 1:dy:sz(1);
[hexaX, hexaY] = meshgrid(grid_hx, grid_hy);
% Shift each 2nd row (instead of separate processing for even/odd):
hexaX(2:2:end, :) = hexaX(2:2:end, :) + 0.5*dx;
end
```

6. (C1.5) Similar to step (2), create a smaller version of `im`, interpolated at the hexagonal sampling positions using `interp2()`. Save the resulting image to ☞ `im_h`. After that repeat the reconstruction from step (4) with the hexagonally sampled image and return the resulting image. Make sure that you specify the correct coordinates of your input data for `TriScatteredInterp()`. *Hint:* Interpolation function returns NaN for border points, where no interpolation can be made. Simply ignore this fact for this problem. *(8 points)*

```
function [im_rec] = interpolate_reconstruct(im, hexaX, hexaY)
sz = min(size(im));
[imX, imY] = meshgrid(1:sz);
% Interpoolate to hexagonal grid positions
im_h = interp2(imX, imY, im, hexaX, hexaY, 'linear');
% Reconstruct the image to its original size
si_h_rec = TriScatteredInterp(hexaX(:), hexaY(:), im_h(:), 'linear');
im_rec = si_h_rec(imX, imY);
end
```

7. (C1.6) After you perform reconstruction, you are normally interested to know what reconstruction quality is achieved. For this, you usually need to compute PSNR of the difference image. In this problem, implement a function to compute PSNR of the difference image between the original and reconstructed versions. Remove a border of 5 px on all sides to neglect border effects. Save the values you obtain to ☞ `psnr_diff`. For images with a maximum intensity of 1, the PSNR is calculated from the mean squared error (MSE) as follows:

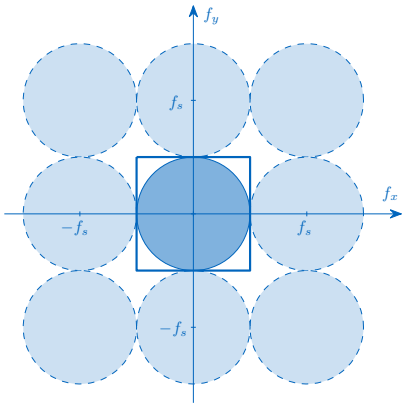
$$\text{PSNR [dB]} = 10\log_{10}\frac{1}{\text{MSE}}, \quad \text{MSE} = \frac{1}{N}\sum_{x,y\in\mathcal{I}}(i_1(x,y)-i_2(x,y))^2,$$

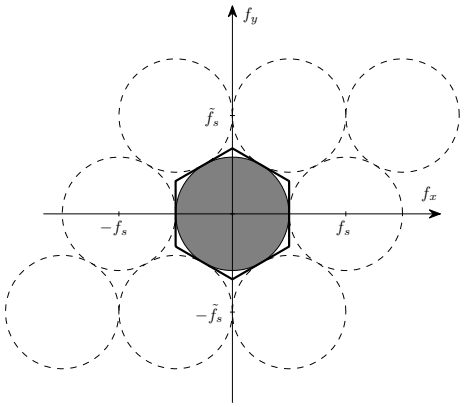
where N is number of considered pixels in the image. Note: Keep in mind that Matlab has a function with a name `psnr()`, therefore avoid calling your variables with the same name. This leads to errors that are hard to trace back. *(5 points)*

```
function psnr_diff = compute_psnr_diff(im, im_rec)
% implement psnr computation according to the formula
im_diff = im_rec - im;
border = 5;
mse = im_diff(border+1:end-border, border+1:end-border).^2;
mse = mean(mse(:));
psnr_diff = 10 * log10(1/mse);
end
```

8. (A1.2) Analyze PSNR computed in step (7) for both reconstructed images. Which image has a better (higher) PSNR? How do you explain this difference? *(5 points)*

A higher PSNR is expected for hexagonal sampling, as it exhibits a better spectral efficiency.





2 Zone Plate

A circular zone plate is an image of a sinusoidal circular wave which increases in frequency with the distance to a central point. It is useful to analyze effects of filtering, sampling and aliasing. In this problem, you will create a zone plate with the central point at the top-left pixel and the maximum possible frequency along the image diagonal in the bottom-right corner (see Figure 2). Using a linear frequency chirp $f(t) = kt$, the circular zone plate image is calculated as follows:

$$I_{zp}(x,y) = \frac{1}{2} + \frac{1}{2}\cos\left[2\pi \int_0^{d(x,y)} f(t)\,dt\right] = \frac{1}{2} + \frac{1}{2}\cos\left(2\pi\frac{k}{2}d(x,y)^2\right)$$

with distance from central point $d(x,y) \in [0, d_{max}]$ and constant k which adjusts the maximum frequency.

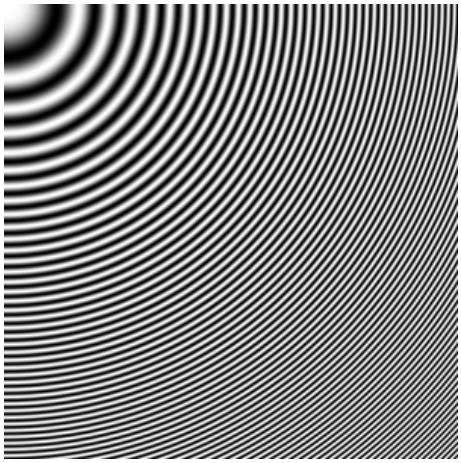


Figure 2: Image of a zone plate

1. (C2.1) Write a function to generate a $N \times N$ image matrix ☞ `D` whose entries represent the Euclidean distance in pixel units of the respective pixel to the top-left pixel. For the top-left element (`D(1,1)` in MATLAB) it is 0, for the bottom-right element (`D(N,N)` in MATLAB) $d_{max} = \sqrt{2}(N-1)$. Verify image `D` and display it for instance with `imagesc()`. Do not use for-loops – the functions `repmat()` or `meshgrid()` provide helpful initializers to calculate `D`. The maximum frequency of the zone plate is observed in the bottom right corner. Calculate ☞ `k` such that the phase change along the last two diagonal pixels is exactly π. Note that the distance of two pixels along the diagonal is $\sqrt{2}$. Now calculate the zone plate image ☞ `Izp` according to the given formula. *(10 points)*

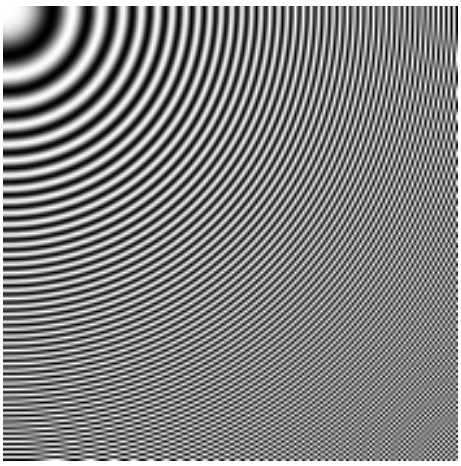
```
function [D, k, Izp] = calculate_zone_plate(N)
t = (0:N-1);
X = repmat(t,N,1);          Y = X';
D = sqrt(X.^2 + Y.^2);
dmax = max(D(:));
k = 1 / (D(end,end)^2 - D(end-1,end-1)^2);
cosarg = 2*pi * k/2 * D.^2;
Izp = 0.5 + 0.5*cos(cosarg);
end
```

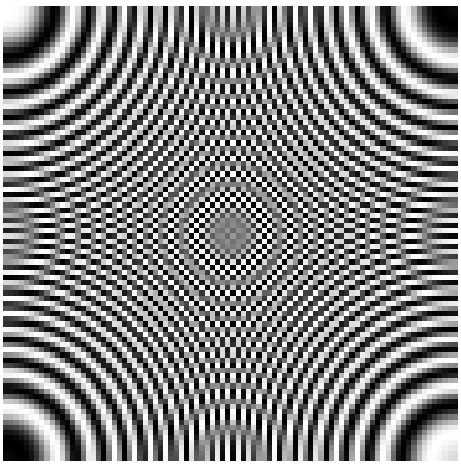
2. (A2.1) Sub-sample `Izp` with factors 2:1 and 4:1 using sub-matrix addressing. Display the resulting images and explain the effects you observe. Make sure you visualize the image with 1:1 scaling. *(6 points)*

supersample

What effects do you observe in the subsampled image: Images with factor 2:1 and 4:1 exhibit artifacts in the bottom and right parts. This is because high frequency components in an image are projected to lower frequency components after subsampling. For instance, in the case of 4:1 sub-sampling, the high frequencies in the bottom-right corner overlap with the new DC component.

```
N = 350;
[D, k, Izp] = calculate_zone_plate(N);

Izp_21 = Izp(1:2:end,1:2:end);
Izp_41 = Izp(1:4:end,1:4:end);

figure
title('Zone plate 2:1 subsampling');
imshow(Izp_21);

figure
title('Zone plate 4:1 subsampling');
imshow(Izp_41);


imwrite(Izp_21, 'res/Izp_21.png');
imwrite(Izp_41, 'res/Izp_41.png');
```





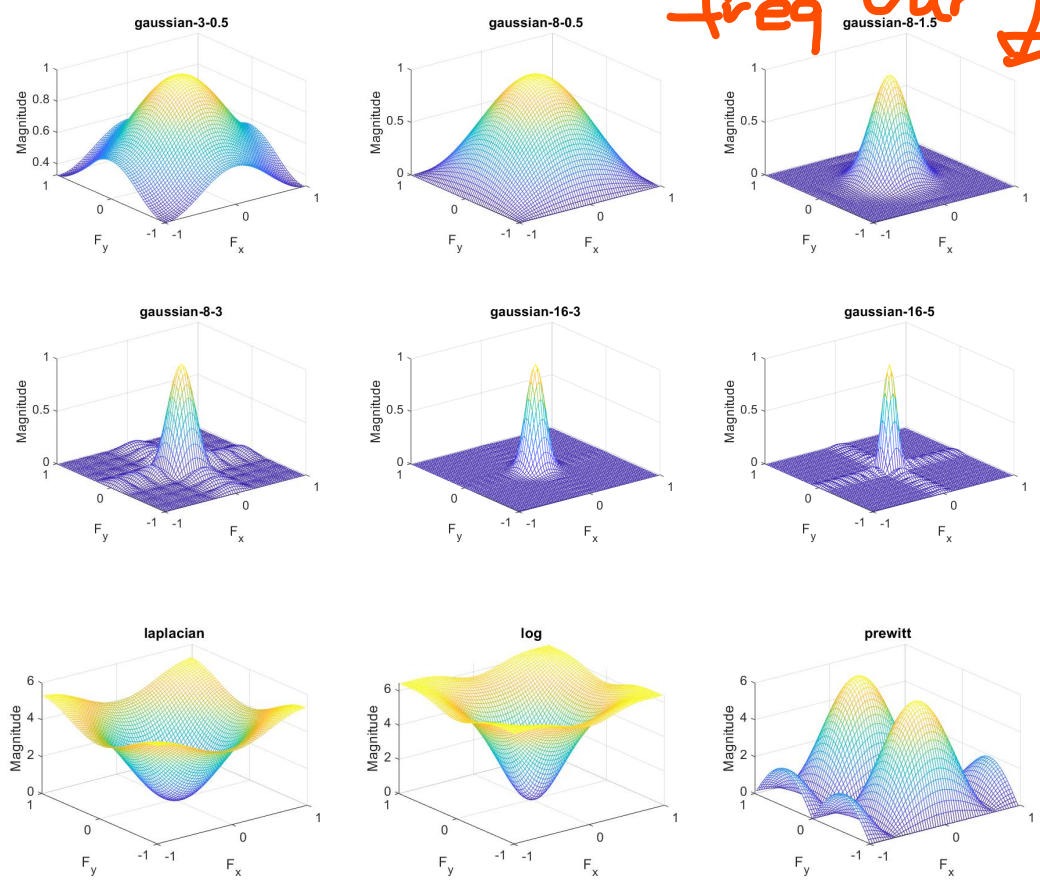
3 Image Filters

In this problem, you will work with several 2D filters. Their effects are observed on natural images and on the zone plate image.

1. (A3.1) Use the `fspecial()` function to generate image filters of types "gaussian","laplacian", "log", "prewitt". Work with sizes $3 \times 3, 8 \times 8$ and 16×16 as well as with different parameters. Plot the filter kernel with `surf()`. Then, analyze the frequency response of the generated filters using `freqz2()`. How does the filter size influence the frequency response? *(9 points)*

   - *Gaussian filters act as low pass filters, attenuating the high frequency components.*
   - *Smaller filters in space have a wider frequency spectrum.*
   - *Laplacian filter is most similar to the high-pass filter as it attenuates the low-frequency components and lets through the high-frequency ones.*
   - *Laplacian of Gaussian filter is also a high pass filter; however, in order to reduce its sensitivity to noise a Gaussian filter is used as well, smoothing the image.*

```
% Gaussian filters
Hg1 = fspecial('gaussian', 3, 0.5);
figure; freqz2(Hg1);    title('gaussian-3-0.5');
Hg2 = fspecial('gaussian', 8, 0.5);
figure; freqz2(Hg2);    title('gaussian-8-0.5');
Hg3 = fspecial('gaussian', 8, 1.5);
figure; freqz2(Hg3);    title('gaussian-8-1.5');
Hg4 = fspecial('gaussian', 8, 3);
figure; freqz2(Hg3);    title('gaussian-8-3');
Hg5 = fspecial('gaussian', 64, 3);
figure; freqz2(Hg5);    title('gaussian-16-3');
Hg6 = fspecial('gaussian', 16, 5);
figure; freqz2(Hg6);    title('gaussian-16-5');

% Laplacian filter
Hl1 = fspecial('laplacian');
figure; freqz2(Hl1);    title('laplacian');

% LoG: Laplacian of Gaussian filter
Hlog1 = fspecial('log');
figure; freqz2(Hlog1);  title('log');

% Prewitt
Hp1 = fspecial('prewitt')';
figure; freqz2(Hp1);    title('prewitt');
```



2. (C3.1) Find out how `freqz2()` generates its plot by inspecting its source code and write a function to do the same manually using the `fft2()` function. The function should return z values ☞ `prew_z`, same as what `freqz2()` plots as z-values for the "prewitt" filter (assume default options for `fspecial()` and `freqz2()`). The top-left element of `prewitt_z` corresponds to $(x, y) = (-1, -1)$, this way zero frequency component is located at the center of the spectrum. *(5 points)*

```
function prew_z = prewitt_z()
% Define the prewitt filter
Hp1 = fspecial('prewitt');
N=64;% default for: freqz2(h) uses [Ny Nx] = [64 64]
% Compute the NxN frequency spectrum with 0-padding:
Fp1 = fft2(Hp1, N, N);
prew_z = abs(fftshift(Fp1));
end
```

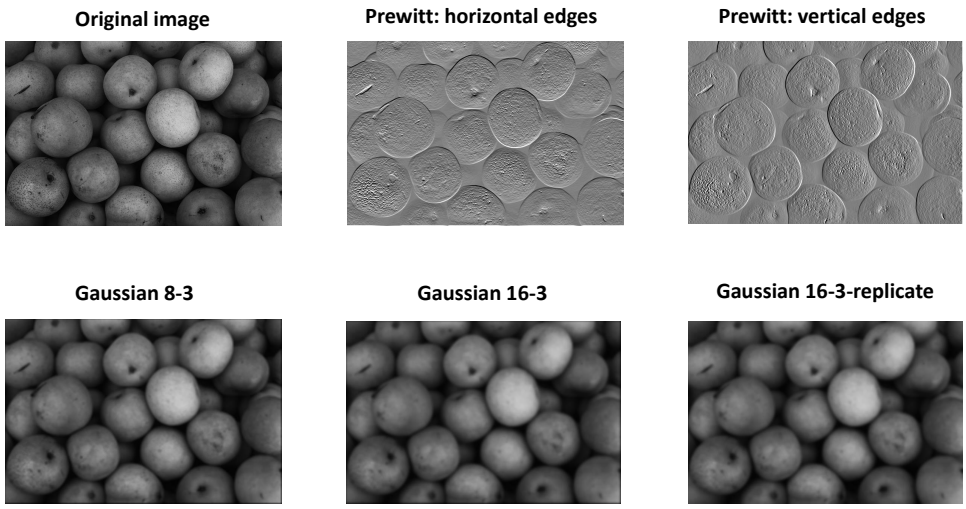
3. (C3.2) Write a function to filter the image with a Gaussian filter (16×16, $\sigma = 3$) and "replicate" border processing. *(5 points)*

```
function im_filtered = filter_image(im)
Hg5 = fspecial('gaussian', 16, 3);
im_filtered = imfilter(im, Hg5, 'replicate');
end
```

4. (A3.2) Load image *pears.jpg* as double and filter it with the created filters (using `imfilter()`), without introducing any shift or scaling. Can you explain the observed effect from the frequency response of the filter? What is the effect of prewitt filter? What is the effect of a large filter size? How do you have to choose the size and σ value of a Gaussian low-pass filter to blur out fine image details, such as the marks on the pears? *(6 points)*

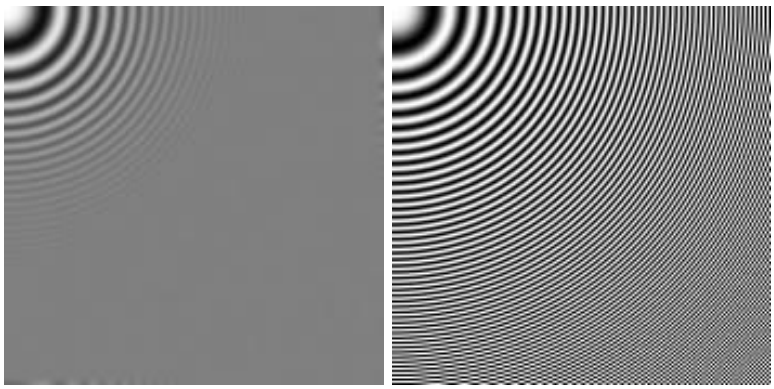
   - *The "prewitt" filter shows 2 strong peaks at "mid-range" vertical frequencies. This results in an image which shows only horizontal edges.*
   - *A dark border is observed especially with large filter kernels because outside pixels are considered to be 0 by default. The "replicate" options in `imfilter()` copies the closest border pixel to the outside range, which typically produces a better result.*
   - *A Gaussian filter with e.g. $\sigma = 1$ results in a clearly visible blurring (e.g. removal of high-frequncy components), which removes fine image details and noise. Larger filter size leads to larger border errors.*

```
% Load original image
im = im2double(imread('data/pears.jpg'));
figure;  imshow(im); title('Original image');

% Prewitt
im_p1 = 0.5+imfilter(im, Hp1);
figure;  imshow(im_p1);  title('prewitt horizontal edges');

im_p2 = 0.5+imfilter(im, Hp1');
figure;  imshow(im_p2);  title('prewitt vertical edges');

% Gaussian filters
im_g4 = imfilter(im, Hg4);
figure;  imshow(im_g4);  title('gaussian-8-3');

im_g5 = imfilter(im, Hg5);
figure;  imshow(im_g5);  title('gaussian-32-3');

imf_g5_replicate = imfilter(im, Hg5, 'replicate');
figure;  imshow(imf_g5_replicate);  title('gaussian-16-3 replicate');
```



5. (C3.3) Now you will work with the zone plate image `Izp` from the problem C2.1 (1). Write a function that filters `Izp` with a Gaussian (16×16, $\sigma = 3$, boundary option "symmetric") and subsequently subsamples it with 2:1. The function should output the main diagonal of the resulting image. The second output variable is diagonal of non-filtered image with subsampling 2:1. *(5 points)*

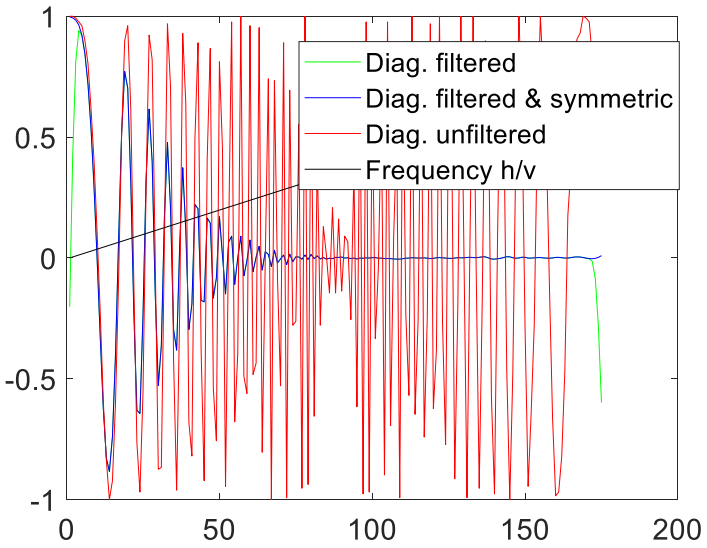
```
function [diag_filter_subsampled, diag_nonfilter_subsampled] = ...
    filter_subsample_zone_plate(Izp)
F = fspecial('gaussian', 16, 3);
tmp1 = imfilter(Izp, F, 'symmetric'); % filter image
tmp1 = tmp1(1:2:end,1:2:end); % subsample
diag_filter_subsampled = diag(tmp1);
diag_nonfilter_subsampled = diag(Izp(1:2:end,1:2:end));
end
```



Zone plate Gaussian Filtering and subsampling with 2:1 (left), withouth Gaussian filter (right)

6. (A3.3) Compare plots of the diagonals from the previous problem (5). How do they differ? What happens if you use MATLAB's default boundary option instead of the "symmetric" option? Compare the damping you observe with the frequency response of the Gaussian filter. *(5 points)*

In the unfiltered and sub-sampled diagonal, aliasing is apparent. The amplitude of the filtered diagonal decays according to the frequency response of the filter. With the default border processing option (extension with 0 values), the plot drops to 0 at the beginning and at the end. The "symmetric" option extends the zone plate image quite well, such that no drop can be observed.



Diagonals of the zone plate image. The intensity values are mapped to $[-1; 1]$.