Medical Image Processing - Worksheet 4

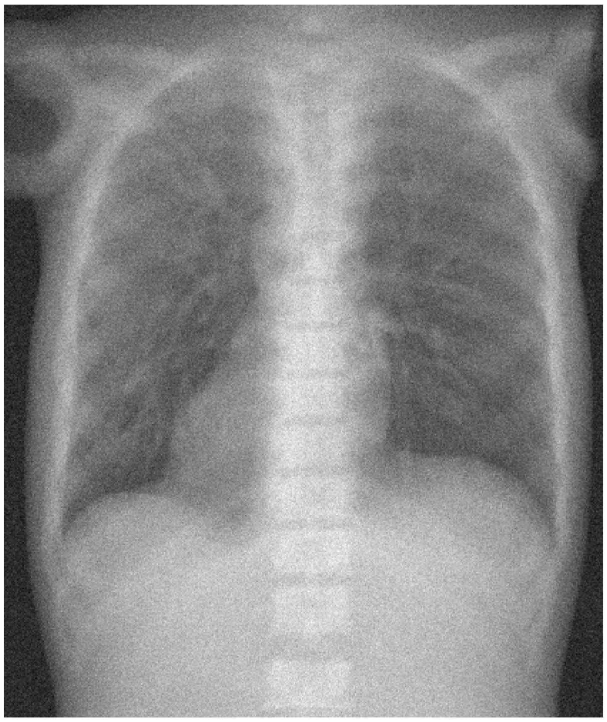
Preparation

Set up the initial image

```
image_file = fopen("data/ThoraxXray_500x420_noise");
xray_image = fread(image_file, [500, 420], "float32", "ieee-be");
```

Display the image

```
imshow(xray_image, [])
```



Define global parameters

```
pixel_size = 0.8 % mm
```

```
pixel_size =
0.8000
```

Task 1: Filtering in spatial domain

Build your own function called SpatialFilt that filters an image with a Gaussian filter.

SpacialFilt function

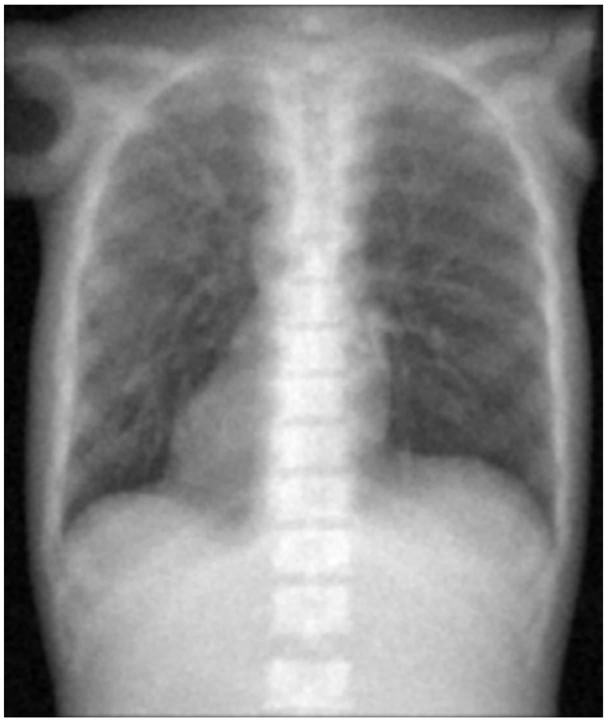
```
function filt_image = SpatialFilt(myimage, im_pixel_size, k_size, k_sigma,
edge_fill)
    arguments
        myimage (:,:) {mustBeNumeric}   % Original image (2D numeric array)
        im_pixel_size (1,1) {mustBeNumeric, mustBePositive} % Pixel size in
the original image
        k_size (1,1) {mustBeNumeric, mustBeInteger, mustBePositive} %  Size
of the kernel
        k_sigma (1,1) {mustBeNumeric, mustBePositive} % Sigma value (σ) for
the Gaussian filter in mm
        edge_fill (1,1) {mustBeMember(edge_fill, [0, 1])} % Option for
filtering pixels at the edges - 0: zeros, 1: repeat last pixel value
    end
    if mod(k_size, 2) == 0
        error("SpatialFilt:InvalidKernelSize", "k_size must be an odd
integer.");
    end
    padding_size = floor(k_size / 2);
    if edge_fill == 0 % Zero-padding
        padded_image = padarray(myimage, [padding_size, padding_size], 0,
"both");
    elseif edge_fill == 1 % Replicating border pixels
        padded_image = padarray(myimage, [padding_size, padding_size],
"replicate", "both");
    else
        error("SpatialFilt:InvalidEdgeFillOption", "edge_fill must be 0 or
1.");
    end

    % Build the filter
    filter_center = (k_size + 1) / 2;
    [filter_pos_X, filter_pos_Y] = meshgrid(1:k_size, 1:k_size);
    filter_distances = sqrt((filter_pos_X - filter_center).^2 +
(filter_pos_Y - filter_center).^2); % Task desciption does not specify the
norm, so Euclidean is chosen
    gaussian_filter = exp(-(filter_distances.^2)/(2 * k_sigma /
im_pixel_size));
    gaussian_filter = gaussian_filter / sum(gaussian_filter(:));

    % Create the filtered image
    filt_image = zeros(size(myimage));

    for i = 1:size(filt_image, 1)
        for j = 1:size(filt_image, 2)
            % Extract local region from padded image
            % Note that an offset (padding_size) must be included when
indexing the padded image
            % i - padding_size + padding_size:i + padding_size + padding_size
            % j - padding_size + padding_size:j + padding_size + padding_size
            image_region = padded_image(i:i + 2 * padding_size, j:j + 2 *
padding_size);
            % Multiply region pixels with normalized kernel
            filt_image(i, j) = sum(image_region(:) .* gaussian_filter(:));
        end
    end
end
```

Testing SpatialFilt function

```
k_size = 21;
k_sigma = 1.5;
image_filtered_spatial = SpatialFilt(xray_image, pixel_size, k_size,
k_sigma, 0);
```

Display the image

```
imshow(image_filtered_spatial, [])
```



Task 2: Filtering in Fourier domain

Build your own function called FourierFilt that filters an image in the Fourier domain with a Gaussian low pass filter.

```
function filt_image = FourierFilt(myimage, im_pixel_size, k_sigma_spat, 
zero_pad)
    arguments
        myimage (:,:) {mustBeNumeric}   % Original image (2D numeric array)
        im_pixel_size (1,1) {mustBeNumeric, mustBePositive} % Pixel size in 
the original image
        k_sigma_spat (1,1) {mustBeNumeric, mustBePositive} % Sigma value (σ) 
for the Gaussian filter in mm
        zero_pad (1,1) {mustBeNumeric, mustBeNonnegative} % Number of zeros 
to use for zero padding
    end

    [size_x, size_y] = size(myimage);
    % Apply zero padding
    if size_x <= size_y
        padding_x = zero_pad / 2;
        padding_y = max(0, size_x + zero_pad - size_y) / 2; % Try to make 
the padded image square
    else
        padding_y = zero_pad / 2;
        padding_x = max(0, size_y + zero_pad - size_x) / 2; % Try to make 
the padded image square
    end
    padded_image = padarray(myimage, [padding_x, padding_y], 0, "both");

    [size_padded_x, size_padded_y] = size(padded_image);
    fov_x = size_padded_x * im_pixel_size;
    fov_y = size_padded_y * im_pixel_size;

    % Compute the Fourier Transform of the padded image
    F_image = fftshift(fft2(padded_image));

    % Build the filter for the Fourier domain
    [U, V] = meshgrid((-size_padded_y / 2:size_padded_y / 2 - 1) / fov_y, 
(-size_padded_x / 2:size_padded_x / 2 - 1) / fov_x);
    freq_radius = sqrt(U.^2 + V.^2);
    k_sigma_freq = 1 / (2 * pi * k_sigma_spat);
    F_gaussian = exp(-(freq_radius.^2) / (2 * k_sigma_freq^2));
    F_gaussian = F_gaussian / max(F_gaussian(:)); % Normalize the filter

    % Apply the filter to the image
    F_filtered = F_image .* F_gaussian;

    % Compute inverse Fourier transform and construct the output
    filtered_padded_image = real(ifft2(ifftshift(F_filtered)));
    filt_image = filtered_padded_image(padding_x + 1:padding_x + size_x, 
padding_y + 1:padding_y + size_y);
end
```

Testing FourierFilt

```
k_sigma_spat = 1.5;
zero_pad = 420;
image_filtered_fourier = FourierFilt(xray_image, pixel_size, k_sigma_spat,
zero_pad);
```

Display the image

```
imshow(image_filtered_fourier, [])
```