Question 1 (Section 2)

As described in the question, the formula $f(x)$ of this linear contrast stretch can be written as:

$$f(x) = \begin{cases} 0, & I(x) \leq 55 \\ 1.7586 \times I(x) - 96.7241, & 55 < I(x) < 200 \\ 255, & I(x) \geq 200 \end{cases}$$

In the MATLAB code, thresholding is applied to the image in order to generate two binary masks which are one everywhere when the condition is satisfied, and it is zero otherwise. The first mask selects those pixels of original image that have a value between 55 and 200. And the pixel value that is greater than or equal to 200 is set to 1 in the other mask.

After that, every pixel of original image is linearly mapped and multiplied by the first mask. Pixels that have a value between 55 and 200 are then stretched to 0-255, and other pixels are zero. Next, the second mask is multiplied by 255 and added to the previous image in order to satisfy the third equation of $f(x)$. This can make sure all pixel values is in the range 0-255. The code is as shown in Table 1, and the generated image is as illustrated in Figure 1.

```
file = "../lab1/Sigmedia15979.TIF";
I = imread(file);
gray = rgb2gray(I);
figure(1)
imshow(gray)
mask_1 = (gray > 55 & gray < 200);
mask_2 = (gray > 200 | gray == 200);
% apply linear transformation to every pixel
img_1 = 1.7586 * double(gray) - 96.7241;
img_2 = mask_1 .* img_1 + mask_2 * 255;
figure(2)
imshow(uint8(img_2))
```

Table 1: The code of Question 1

Figure 1: Left: Original jfk image; Right: Stretched image

Question 2 (Section 3)

Question 2.1

The MATLAB function that implements Gaussian filter is as shown in Table 2 with detailed comment.

```
function filter = gaussianf(var, size, type)
% Returns 1D or 2D Gaussian filters
% input:
%   var: the varaiance of a Gaussian filter (must be
positive))
%   size: filter size (must be a positive integer)
%   type:
%     'combined' - (default) returns a 'size x size'
Gaussian filter
%     'separable' - returns a '1 x size' Gaussian filter

% default parameter
if (nargin<3)
  type = 'combined';
end

% determine if 'var' is positive
assert(var > 0, 'The variance must be positive')
var = double(var);

% determine if 'size' is a positive integer
assert(size == ceil(size) & size > 0, ...
  'The variance must be a positive integer')

% get 1-D values of coordinates
limit = (size - 1) / 2;
values = linspace(-limit, limit, size);

% extend and then generate the X coordinate matrix
X = ones(size, 1) * values;

% extend and then generate the Y coordinate matrix
Y = values' * ones(1, size);

switch type
  case 'combined'
    % calculate the filter in one shot
    f = exp(-((X .^ 2 + Y .^ 2) / (2 * var ^ 2)));
    % normalisation
    filter = f ./ sum(f, 'all');
  case 'separable'
    f = exp(-(values .^ 2) / (2 * var ^ 2));
    filter = f ./ sum(f, 'all');
  otherwise
    error('Invalid filter type')
end
end
```

Table 2: The code of Question 2.1

Question 2.2

This normalisation strategy can make sure the average graylevel of the image remains the same. If values of a filter sum up to greater than 1, then the generated image will become brighter after filtering. If they sum up to less than 1, then the image will get darker afterwards.

Question 2.3 & 2.4

The code is as shown in Table 3, and the time taken is written in the comment.

```
file = "../lab1/Sigmedia15979.TIF";
I = imread(file);
gray = rgb2gray(I);

% get a separable gaussian filter
fs = gaussianf(1.5, 21, 'separable');
% apply 1D gaussian filter to both the rows and the
columns of the image
tic
for i = 1 : 1000
  gaus_1 = imfilter(gray, fs);
  gaus_2 = imfilter(gaus_1, fs');
end
toc

% get a combined gaussian filter
fc = gaussianf(1.5, 21, 'combined');
% apply 2D gaussian filter
tic
for i = 1 : 1000
  gaus_3 = imfilter(gray, fc);
end
toc

%%%%%%%%%%%%%% log %%%%%%%%%%%%%%%%
% Elapsed time is 1.208801 seconds.
% Elapsed time is 3.770891 seconds.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Table 3: The code of Question 2.3 & 2.4

Question 2.5

The result shows that 'combined' gaussian filter convolution takes about three times as long as than 'separable' convolution, and two convolution methods obtain the same result (only slight differences due to floating point calculation loss). Theoretically, computation for a row and column filtering is more efficient, and 21×21 case computation is reduced by a factor of 10.5.

$$\frac{h \times w \times 21 \times 21}{h \times w \times 21 + h \times w \times 21} = 10.5$$

Question 3 (Section 4)

Question 3.1

The code is as shown in Table 4. The original image is substracted by a image that is filtered by a 2-D low-pass Gussian filter. The result is the high frequency component of original image. Next, it is multiplied by a factor *f*, which is a 'volumn controller', and added by the original image to obtain the enhenced image.

```
file = "../lab1/Sigmedia06907.TIF";
I = imread(file);
figure(1)
imshow(I)

fs = gaussianf(2.5, 15, 'combined');
% apply low pass filtering
I_ = imfilter(I, fs);

f = 0.7;
I_out = f * (I - I_) + I;
figure(2)
imshow(I_out)
```

Table 3: The code of Question 3

Question 3.2 & 3.3 & 3.4

The value of *f* is set to 0.7, and the result is as illustrated in Figure 2(b). We can notice that the edges and details of images are effectively enhenced, but film-grain noise also become easier to be observed.

Other values of *f* are also tested, and results is shown in Figure 2(c)(d). It shows that the edges of image get white when *f* become bigger. Also, when *f* is less than 0.5, the enhencement is not that effective. The best value range of *f* should be between 0.5 to 1.

Question 3.5

Based on the same settings, this image does not show a significant improvement in quality as previous case (Figure 3), and even the noise in the background is enhenced. The reason is that this image contains less detail information but more areas with steady color changes, such as the table and the floor, thus the noise in the background area is enhanced, while the effect of detail areas such as people is not obvious.

(a) Original image (b) $f = 0.7$

(c) $f = 3.0$ (d) $f = 0.1$

Figure 2: Original image and enhenced results

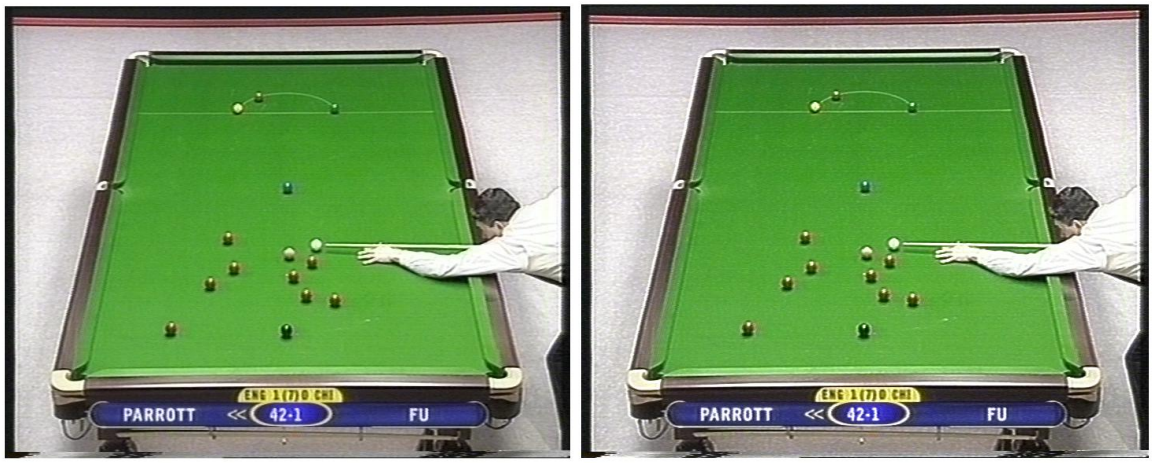

Figure 3: Original image and enhenced result