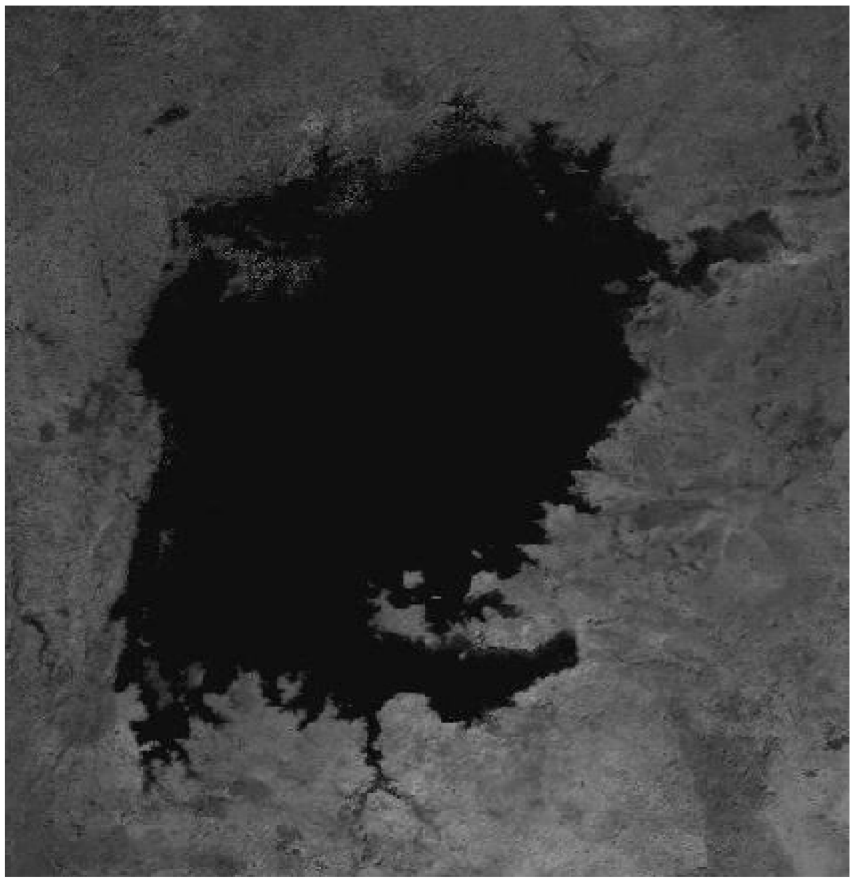
Task 2

Task 2.1

Load the lake and show it

```
lake = imread("../lake_gray.png");
figure;imshow(lake);
```

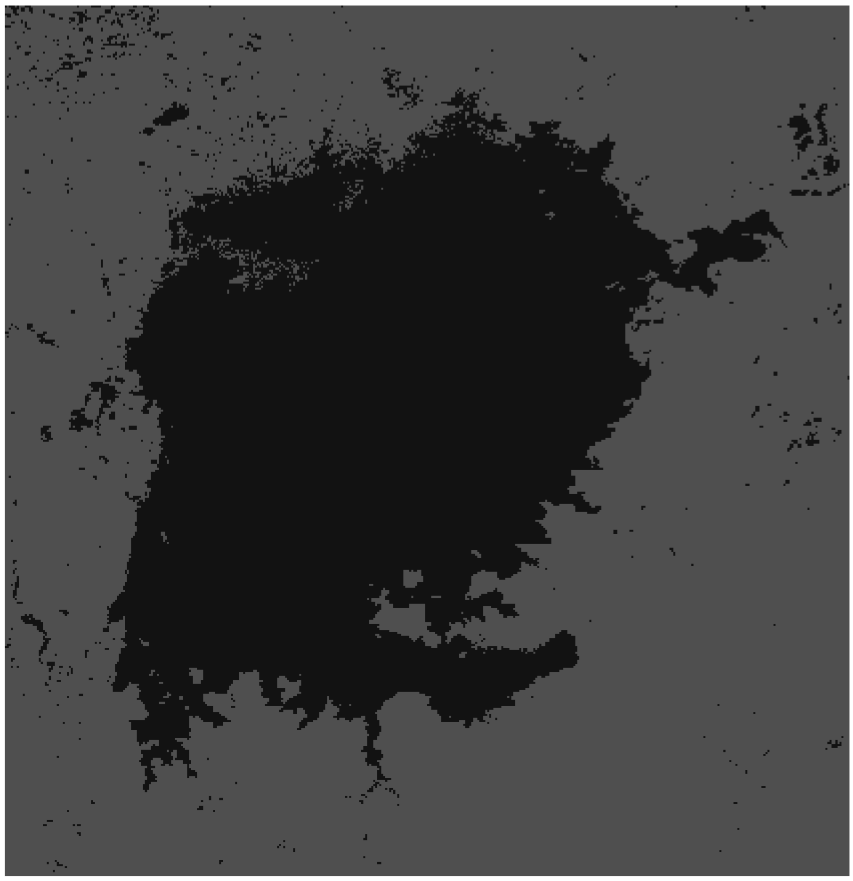


Use k = 2 since we want to classify the lake and a threshold for exiting equal to 0.004

```
k = 2;
th = 0.005;
```

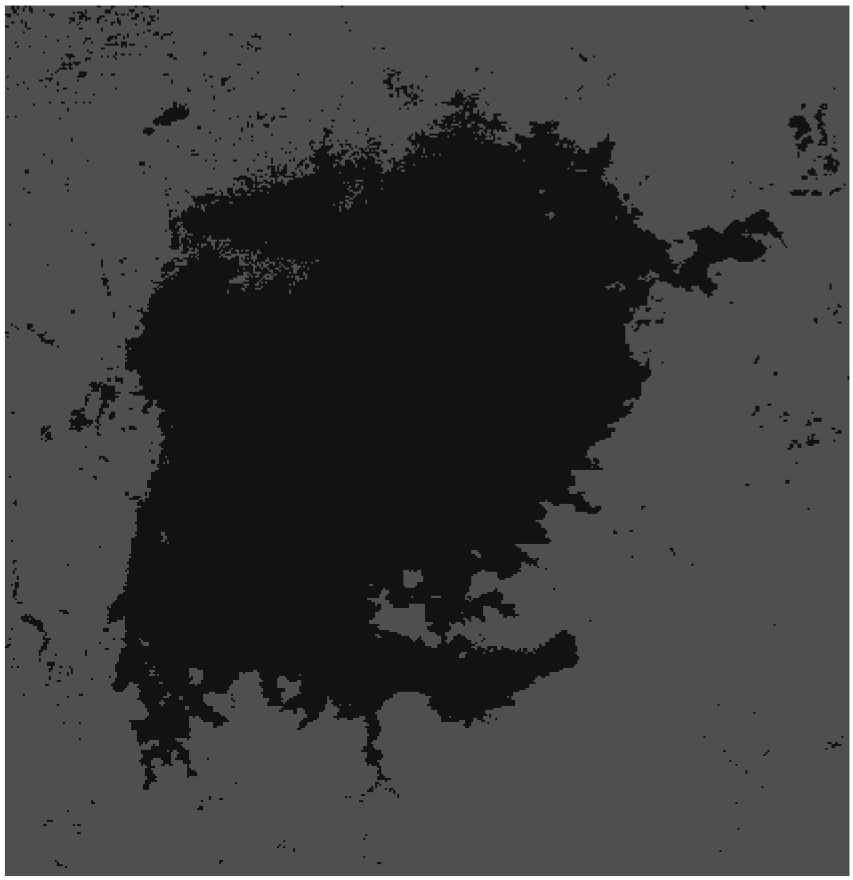
Call my implementation

```
[result,clutered_avg] = (k_means_clustering(lake,k,th));
figure;imshow(remap(result,clutered_avg));
```



Call the matlab implementation for comparison.

```
[l,desc]= imsegkmeans(lake,k);
figure;imshow(remap(l,desc));
```



It is quite apparant that the matlab implementation is faster, but the result achived is verry similar.

Here is the code I used for k_means_clustering

```
function [classed_image,cluster_avg]= k_means_clustering(image, k, T)
    % Does k-means clustering using grayscale images
    % Lets try this
    image = im2double(image);
    % P is a 1d vector of the pixels
    P = [image(:)];
    % Compute the clusters needed
    [p_clustered,cluster_avg] = k_means(P, k);
    soize = size(image);
    dim = size(image,2);
    classed_image = reshape(p_clustered,soize);

    function [p_clustered,cluster_avg] = k_means(P, k)
        % This assignment could be done much better
        means = ((1:k)-1)./k;
        new_means = means;

        while 1==1
          %if (dist(new_means, means) < T)
          %  break;
```

```
  %end
    if means ~= new_means
        if (sqrt(sum((new_means-means).^2)))<T
            break;
        end
    end
    means = new_means;
    % 1d vector of minimum values
    mins = (1:k) .* inf;
    % 1d vector of max values
    max = (1:k) .* (-inf);
    % 1d vector holding the number of elements in each class
    number_of_elements = (1:k) .* 0;
    % 1d vector holding s of all elements in class
    s = (1:k) .* 0;

    for pixel = 1:numel(P)
        % Compute closest mean for every pixel, this is really slow, O(n*k)
        [~, means, max, mins, s, number_of_elements] = closest_mean(P(pixel), means, max, mins, s, num
    end

    new_means = 1:k;

    for j = 1:k
        % Update means
        intermediate = s(j) / number_of_elements(j);
        if isnan(intermediate)
            intermediate = means(j);
        end
        new_means(j) = intermediate;
    end

end

p_clustered = P;
cluster_avg = s./number_of_elements;
% Now we have the segments
for itterator = 1:numel(P)
    index = assign_to_segment(P(itterator), max, mins);
    p_clustered(itterator) = index;
end


function index = assign_to_segment(pixel, max, min)
    index = 1;
    % returns what class a pixel belongs to
    for itter = 1:numel(means)

        if (pixel > min(itter) & pixel < max(itter))
            index = itter;
        end

    end

end

function [index, means, max, min, s, number_of_elements] = closest_mean(
```

```
                        pixel, means, max, min, s, number_of_elements
                minima = inf;
                index = 1;
                for i = 1:numel(means)
                    dist = abs(pixel-means(i));

                    if dist < minima
                        % We found a new best
                        minima = dist;
                        index = i;
                    end

                end

                % Append self to class
                if pixel < min(index)
                    min(index) = pixel;
                end

                if pixel > max(index)
                    max(index) = pixel;
                end

                s(index) = s(index) + pixel;
                number_of_elements(index) = number_of_elements(index) + 1;
            end

        end

    end
```

Task 2.2

First run SLIC super pixel algorithm from matlab, going to use matlabs k-means here too since it's faster

```
[l,n] = superpixels(lake,30);
img = zeros(size(lake),'like',lake);
super_pixels = label2idx(l);
for i = 1:n
    region = super_pixels{i};
    img(region) = mean(lake(region));
end
figure;
imshow(img);
```

Now run k means clustering

```
[result,clutered_avg] = (imsegkmeans(img,k));
figure;imshow(remap(result,clutered_avg));
```

After running the super pixel algorithm first, it seems to have removed a lot of noise.

Trying with a smaller amount of super pixels

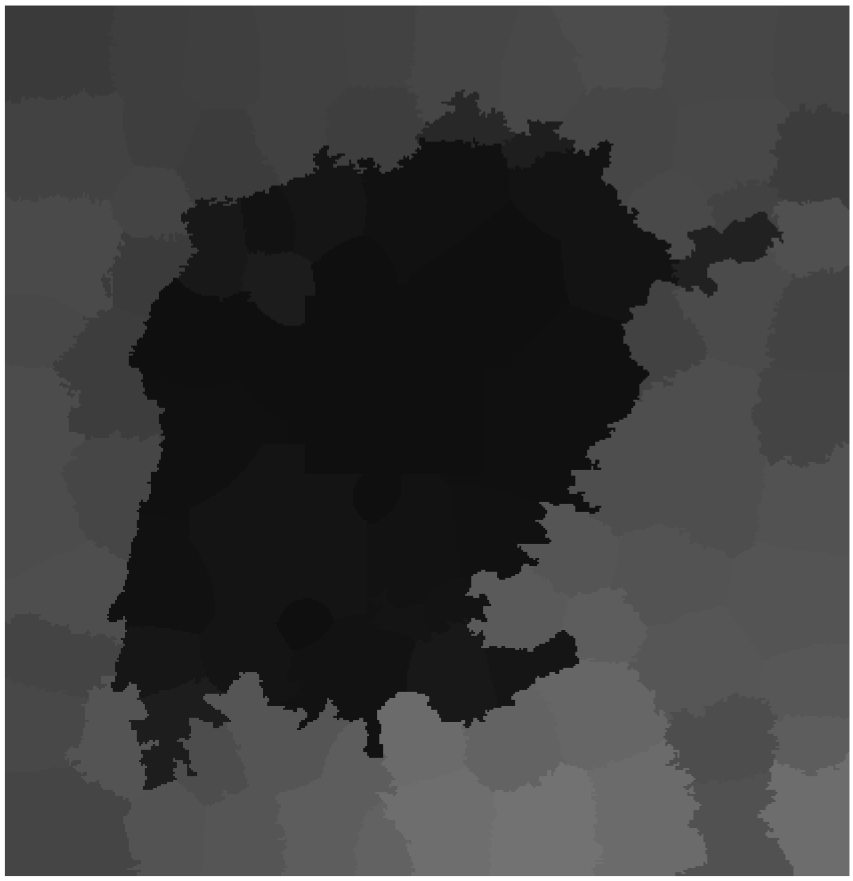
```
[l,n] = superpixels(lake,10);
img = zeros(size(lake),'like',lake);
super_pixels = label2idx(l);
for i = 1:n
    region = super_pixels{i};
    img(region) = mean(lake(region));
end
figure;
imshow(img);
```

Now run k means clustering

```
[result,clutered_avg] = (imsegkmeans(img,k));
figure;imshow(remap(result,clutered_avg));
```

Trying with a larger amount of super pixels

```
[l,n] = superpixels(lake,100);
img = zeros(size(lake),'like',lake);
super_pixels = label2idx(l);
for i = 1:n
    region = super_pixels{i};
    img(region) = mean(lake(region));
end
figure;
imshow(img);
```



Now run k means clustering

```
[result,clutered_avg] = (imsegkmeans(img,k));
figure;imshow(remap(result,clutered_avg));
```

Using super pixels first seems to improve the performance substantially, removing all noise. This is great since the lake is much better segmented after this. Using less super pixels makes us lose some detail in the edges, using more super pixels is better for this task since it allows us to find the outline more precisely

```
function o = remap(inp,map)
    o = inp;
    for i = 1:size(inp,1)
        for j = 1:size(inp,2)
            o(i,j) = map(inp(i,j));
        end
    end
end
```