```
%============================== segKmeans ==============================
%
%  function [J, K, means] = segKmeans(I, numk, iter, means)
%
%  Perform k-means segmentation on the grayscale image I.
%
%  Input:
%    I          - image I ( from |R^2 -> |R^d ).
%    numk    - The number k to use in k-means.
%    iter       - Maximum number of iterations (can stop earlier if no change).
%    means      - initial guess at means, each column is a mean value.
%              means should be provided row-wise.  [k x d matrix]
%    ncov       - covariance matrix to use in the scaling (use 1 if none).
%                      can be scalar for all means to use, or can be a unique
%                      value for each mean value.
%
%  Output:
%    J          - the segmentation map.
%    K          - the simplified image using the means and the segmentation.
%    means      - the final segmentation means.
%
%============================== segKmeans ==============================

%
%  Name:                segKmeans.m
%
%  Author:              Patricio A. Vela, pvela@gatech.edu
%
%  Created:             2010/01/05
%  Modified:    2012/04/07
%
%============================== segKmeans ==============================
function [J, K, means] = kmeans(I, iter, numk, means)

%--[0] Parse the input arguments, set to defaults if needed.
if (nargout == 0)               % If nothing expected, then don't bother
  return;                               %   doing the computations.
end

if (nargin < 3)                 % If first three not given, can't do much.
  disp('ERROR: Need at least the first three arguments');
  error('BadArgs');
end


%--[1] Prep workspace and variables.  Convert image data to vector data.
sz = size(I);
layers = 1
if length(sz) == 3
    layers = sz(3);
end
vec = reshape(I, sz(1) * sz(2), layers);
%vec = I(:);
J = zeros(sz);
%--[2] Perform k-means clustering.

if ( (nargin > 3) && ( numk == size(means,1) ) )
  % kmeans invocation goes here.  Make sure to set the optional
  % arguments properly for both iterations and initial guess.
```

```
  % Make sure to grab all that's needed for the triple output.
  [J, means] = kmeans(vec, numk, 'MaxIter', iter, 'Start', means);
else
  % let Matlab guess means.  Make sure to set the optional
  % arguments properly for iterations.
  % Make sure to grab all that's needed for the triple output.
  [J, means] = kmeans(vec, numk, 'MaxIter', iter);
end

J = reshape(J, sz(1), sz(2), 1);

%--[3] Prep additional output.  Convert cluster indices into actual
%      image data values by using the returned means.  This will
%      generate an image using only the k mean values.


if ((nargin == 3) || isempty(ncov))
  ncov = 1;                     % If no covariance, default is 1.
end

if (isscalar(ncov))             % If scalar, copy for each mean value.
  ncov = repmat(ncov, [layers, numk])';
end

%means = means .* ncov;
if (nargin > 1)
  K = J;
  for i=1:numk
    K(J == i) = means(i,1);
  end
end

end
```

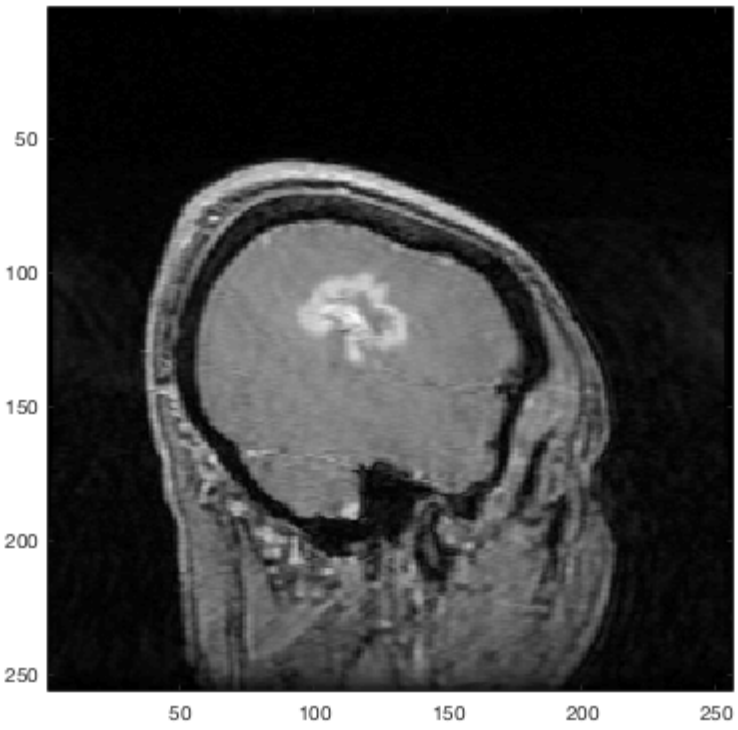
Published with MATLAB® R2016b

```
%============================== segmentM ==============================
%
%  script segmentM
%
%
%  This is a Matlab script that will run Matlab's k-means algorithm
%  for segmenting an image.   You should select two images from the
%  homework Matlab file and perform segmentation on them. The output will
%  be used by another script of mine for visualization.  You should also
%  pick a third color image of your own.
%
%
%  This code uses some Matlab tricks to be somewhat genecleric.  First,
%  all arguments are encapsulated into a cell array.  This works as
%  follows.  The cell array belows consists of two arguments:
%
%  >> sampleCellArray = {40, 34};
%
%  that when expanded as an argument to a function, provides two
%  inputs to the function,
%
%  >> plus( sampleCellArray{:} )
%
%  The output should be the addition of the two arguments:
%
%  ans =
%
%     74
%
%  Anyhow, this function expectes the same, but the arguments are
%  consistent with what the imkmeans function expects.
%
%============================== segmentK ==============================

load('segment.mat');
picks = [1 2];

for i = 1:length(picks)
  switch (picks(i))
    case 1,
      images{i} = westin;
      iparms{i} = { 6, 5 };
    case 2,
      images{i} = fish04;
      iparms{i} = { 6, 5 };
  end
end

% Add more cases if you want to do all of the images, and change
% "picks" accordingly.

% Add your own personal case by including the code below:
% While the segmentation images above are grayscale, yours should
% be a color image file to change things up.
images{3} = imread('macaw.jpg');
iparms{3} = { 10, 6 };

for i = 1:length(images)
  [segimg{i}, K, nmeans{i}] = imkmeans(double(images{i}), iparms{i}{:});
```

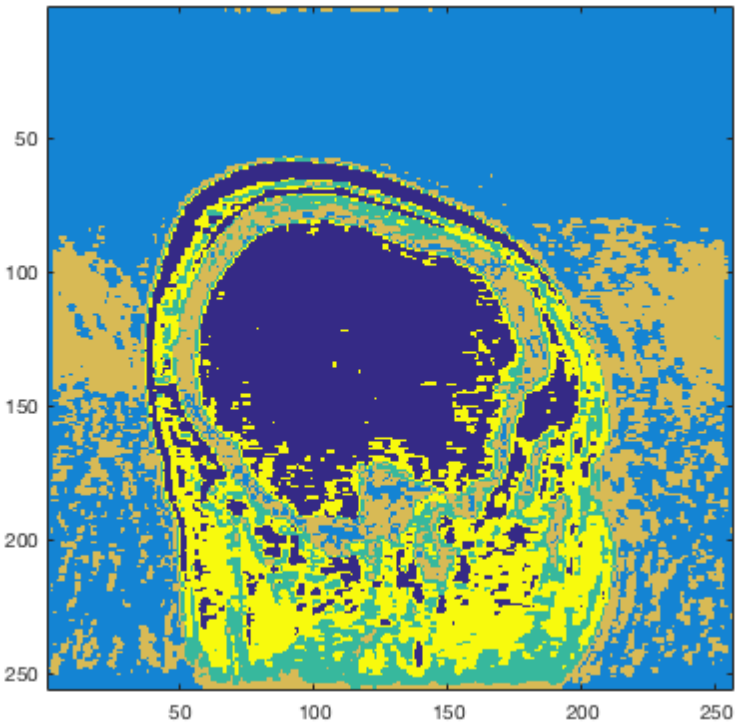
```
  figure(3 * i);
  imagesc(images{i});
  axis image;
  colormap('gray');

  figure(3 * i + 1);
  imagesc(uint8(segimg{i}));
  axis image;
  colormap('default');

  figure(3 * i + 2);
  imagesc(uint8(K));
  axis image;
  colormap('gray');

  nmeans{i};
end

%
%================================ segmentK ================================
```

```
layers =

     1

Warning: Failed to converge in 6 iterations.

layers =

     1

Warning: Failed to converge in 6 iterations.

layers =

     1

Warning: Failed to converge in 10 iterations.
```

Original Westin Image



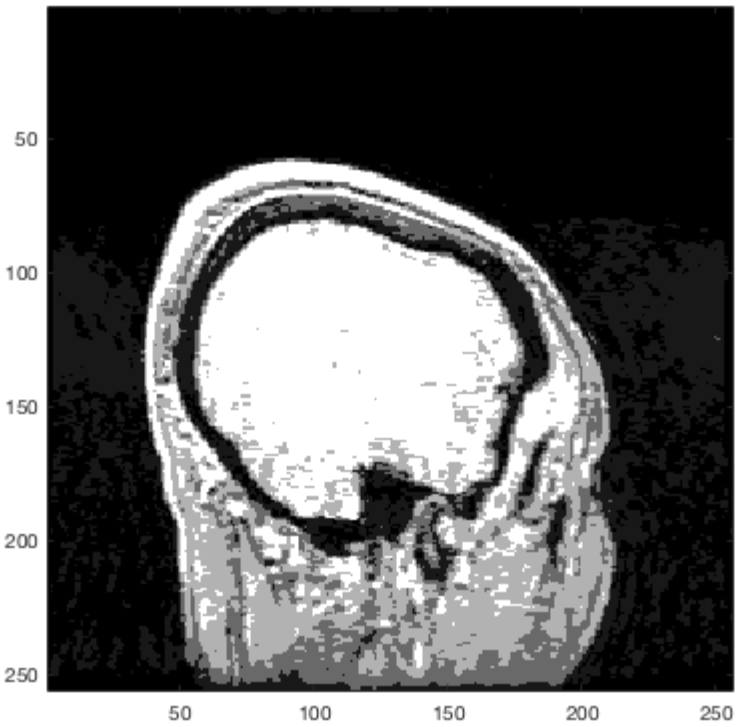
Segmented Westin Image Using kmeans Function



Max Iterations: 6
K Value: 5

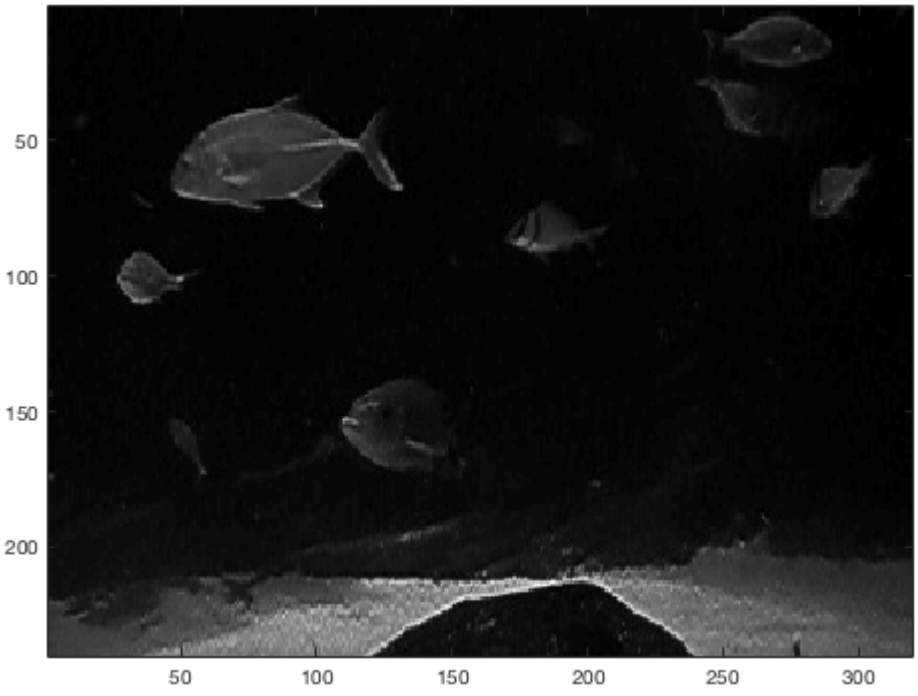
Since the builtin kmeans function was used with a low maximum iteration value and no initial input, the resulting labels were under-segmented. For instance, the black background has two different labels.

Segmented Westin Average Value Image

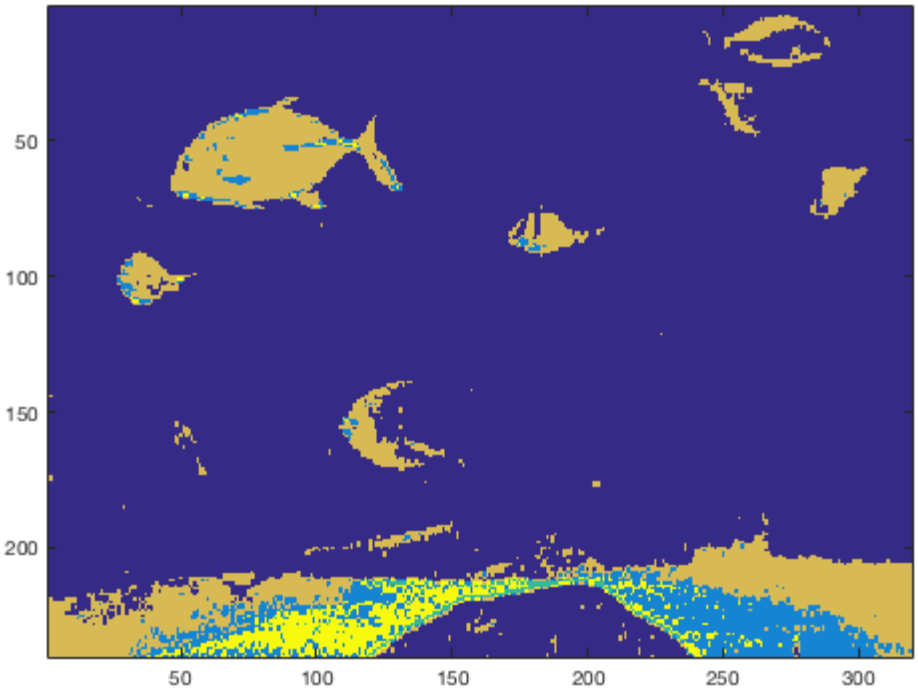


Assigning the mean values of the two background labels, however, show that they have similar values.

Original Fish04 Image

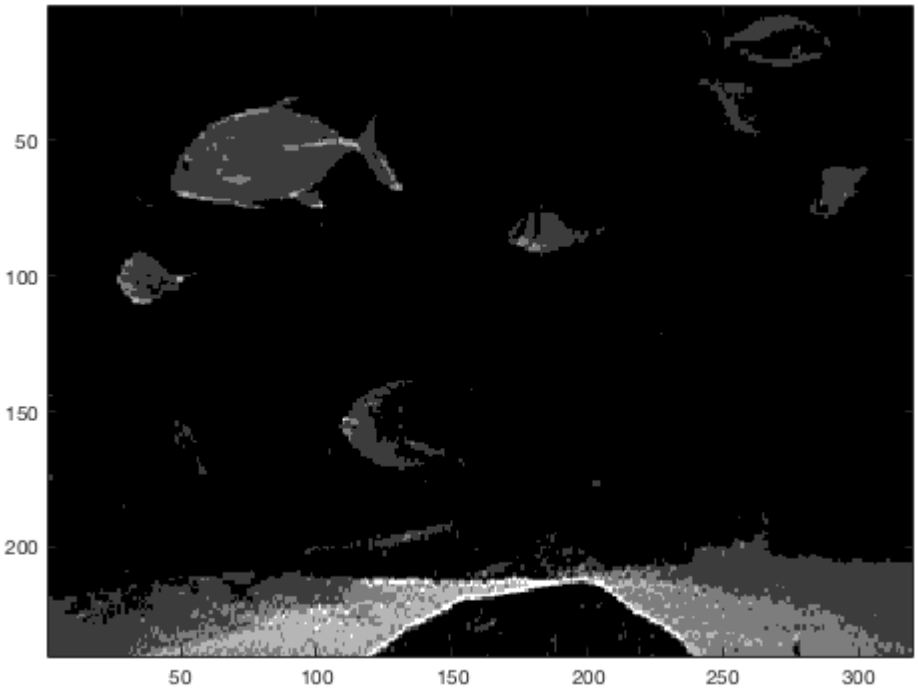


Segmented Fish04 Image Using kmeans Function

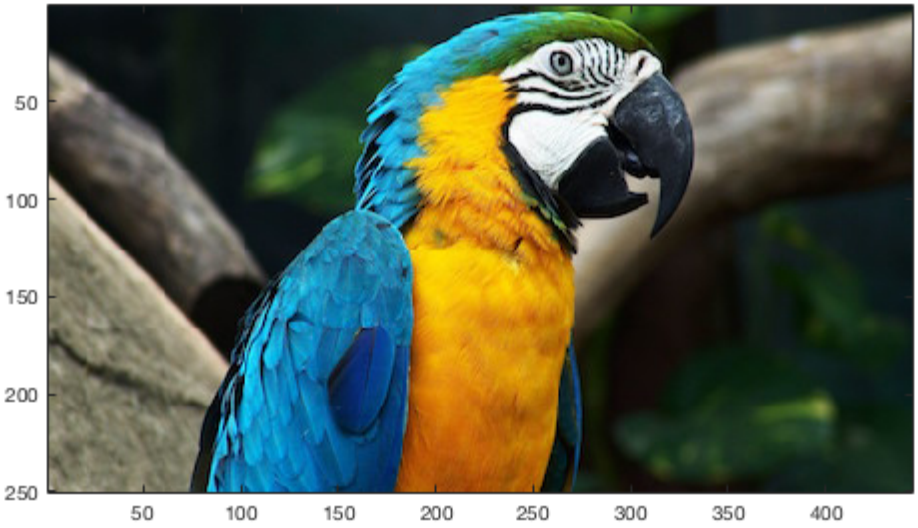


Max Iterations: 6
K Value: 5

Segmented Fish04 Average Value Image

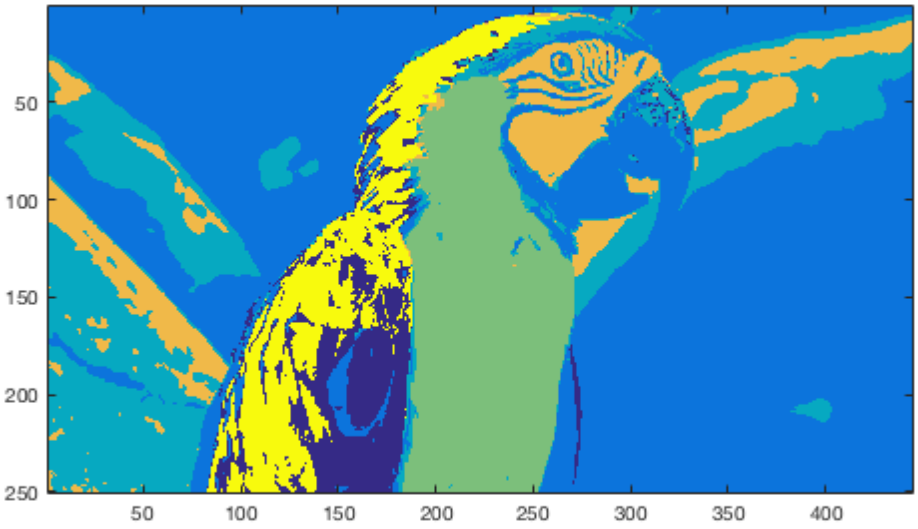


Original Bird RGB Image



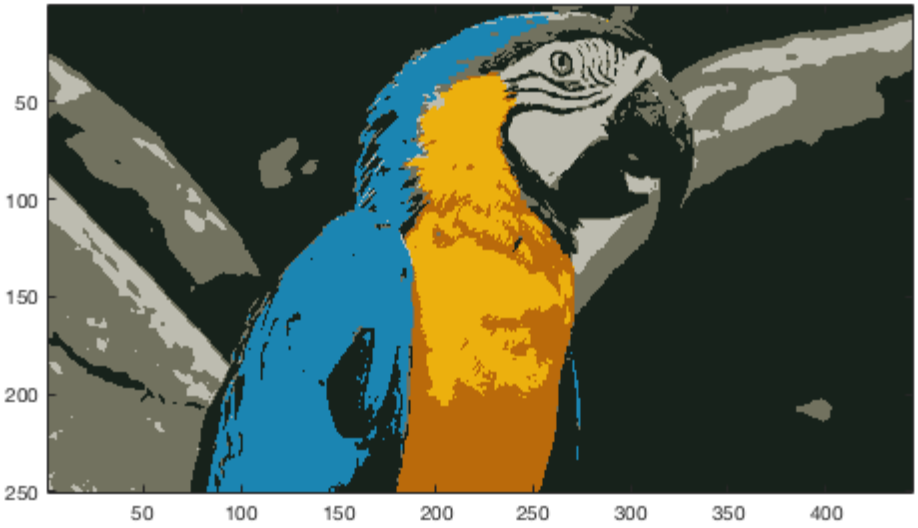
Max Iterations: 10
K Value: 6

Segmented Bird Image Using kmeans Function



A greater k value was used for this image, which was more complex.

Segmented Bird Image Average Image Value



Limiting to only 6 layers causes the green color to be lost from the bird's head, as it is assigned the same label as the brown tree color in the background.

Published with MATLAB® R2016b

```
%============================== segKmeans ==============================
%
%  function [J, K, means] = segKmeans(I, iter, means, ncov)
%
%  Perform k-means segmentation on the grayscale image I.
%
%  Input:
%    I          - image I ( from |R^2 -> |R ).
%    iter       - Maximum number of iterations (can stop earlier if no change).
%    means      - initial guess at means, each column is a mean value.
%    ncov       - covariance matrix to use in the scaling (use 1 if none).
%                      can be scalar for all means to use, or can be a unique
%                      value for each mean value.
%
%  Output:
%    J          - the segmentation map.
%    K          - the simplified image using the means and the segmentation.
%    means      - the final segmentation means.
%
%============================== segKmeans ==============================

%
%  Name:                segKmeans.m
%
%  Author:              Patricio A. Vela, pvela@gatech.edu
%
%  Created:             2010/01/05
%  Modified:    2012/04/07
%
%============================== segKmeans ==============================
function [J, K, means] = kmeans(I, iter, means, ncov)

%--[0] Parse the input arguments, set to defaults if needed.
if (nargout == 0)               % If nothing expected, then don't bother
  return;                               %   doing the computations.
end

if (nargin < 3)                 % If first three not given, can't do much.
  disp('ERROR: Need at least the first three arguments');
  error('BadArgs');
end

if ((nargin == 3) || isempty(ncov))
  ncov = 1;                     % If no covariance, default is 1.
end

if (isscalar(ncov))             % If scalar, copy for each mean value.
  ncov = repmat(ncov, [1, size(means,2)]);
end


%--[1] Prep workspace and variables.
imsize = size(I);

xi = 1:length(means);           % Generate set of class labels.
diff2 = zeros([imsize, xi(end)]);
                                                        % Pre-allocate memory for data energy.
J  = ones(imsize);                      % Instantiate the return variable.
oJ = zeros(imsize);                     % Want to keep track of old segmenation map.
```

```
%--[2] Perform the segmentation iterations.
while (any(oJ(:) ~= J(:)) && (iter > 0))
  iter = iter - 1;                      % Update number of iterations left.
  oJ = J;                                       % Set old copy to previous segmentation map.

  % YOUR CODE HERE.  USE THE HELPER FUNCTIONS BELOW.
  % Steps:
  %  (1) Compute the data energy.
  compEnergy();

  %  (2) Minimize energy to generate assignments (segmentation).
  [val, J] = min(diff2,[],3);

  %  (3) Update the means based on the segmentation
  compMeans();

  % Debugging code
  bins = [1];
  for vi=1:length(xi)
    bins(vi) = sum(J(J == vi));
  end
end

if (nargout >= 2)               % If image expected, then create it.
  % Map segmentation to it's mean color. hint: use interp1
  K = J;
  means = means .* ncov;
  for i=1:size(means,2)
      K(J == i) = means(i);
  end
end

%
%============================ Helper Functions ============================
%
%  These functions live within the scope of the segKmeans function.
%  What that means is that they can be invoked from the loop above
%  and they will have access to the variables above (think of them
%  as global variables in some sense).  Using functions within a
%  function is a clean way to do complex thing but have the main
%  code of the function look nice and clean.
%

  %---------------------------- compEnergy ----------------------------
  %
  %  Compute the k-means data matching energy.  The result is k data
  %  matching score image slices.  This function has access to all
  %  of the variables from the function scope above (I, means, ncov,
  %  diff2, etc.).
  %
  function compEnergy

  for ei=1:length(xi)                           % For each class label ...
    diff2(:,:,ei) = (I - means(ei)).^2;
  end
  end

  %---------------------------- compMeans ----------------------------
  %
  %  Given the segmentation, compute the class means.  This function
```

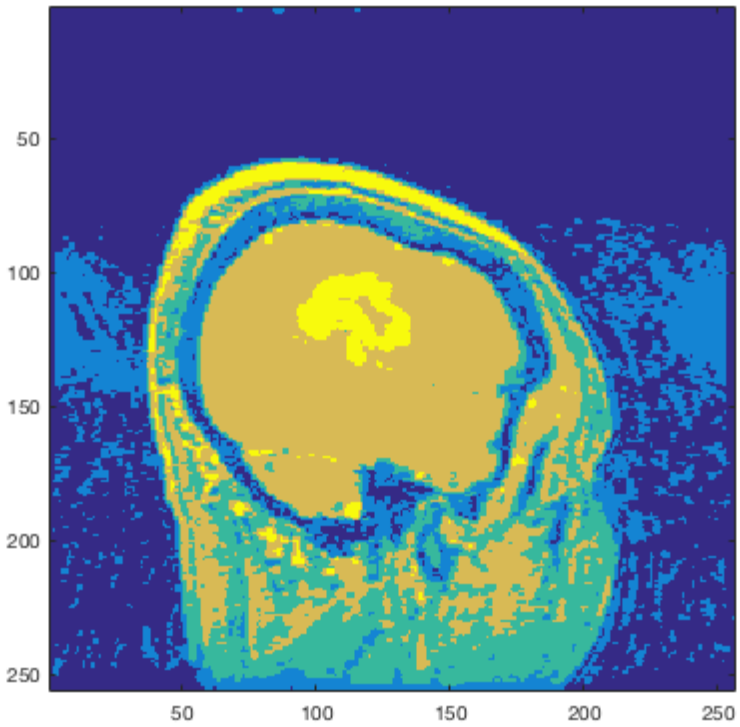
```
  %  also has access.
  %
  function compMeans

  for mi=1:length(xi)                          % For each class label ...
    pts_in_layer = sum(sum(J == mi));
    if (pts_in_layer > 0)
        means(mi) = sum(I(J == mi)) ./ pts_in_layer;
    else
        means(mi) = 0;
    end
  end

  end

end
%
%============================== segKmeans ==============================
```

Published with MATLAB® R2016b

```
%============================== segmentK ==============================
%
%  script segment1
%
%
%  This is a Matlab script that will run the simple k-means algorithm
%  for segmenting an image.   You should select two images from the
%  homework Matlab file and perform segmentation on them. The output will
%  be used by another script of mine for visualization.
%
%
%  This code uses some Matlab tricks to be somewhat generic.  First,
%  all arguments are encapsulated into a cell array.  This works as
%  follows.  The cell array belows consists of two arguments:
%
%  >> sampleCellArray = {40, 34};
%
%  that when expanded as an argument to a function, provides two
%  inputs to the function,
%
%  >> plus( sampleCellArray{:} )
%
%  The output should be the addition of the two arguments:
%
%  ans =
%
%     74
%
%  Anyhow, this function expectes the same, but the arguments are
%  consistent with what the segKmeans function expects.
%
%============================== segmentK ==============================
close all
clear
load('segment.mat');
picks = [1 2];

for i = 1:length(picks)
  switch (picks(i))
    case 1,
      images{i} = westin;
      iparms{i} = { 5 , [3, 10, 60, 140, 150] };
    case 2,
      images{i} = fish04;
      iparms{i} = { 5 , [1, 5 ,40, 80, 120] };
  end
end
% Add more cases if you want to do all of the images, and change
% "picks" accordingly.


for i = 1:length(images)
  [segimg{i}, K, nmeans{i}] = segKmeans(double(images{i}), iparms{i}{:});
  figure(i);

  figure(3 * i);
  imagesc(images{i});
  axis image;
  colormap('gray');
```

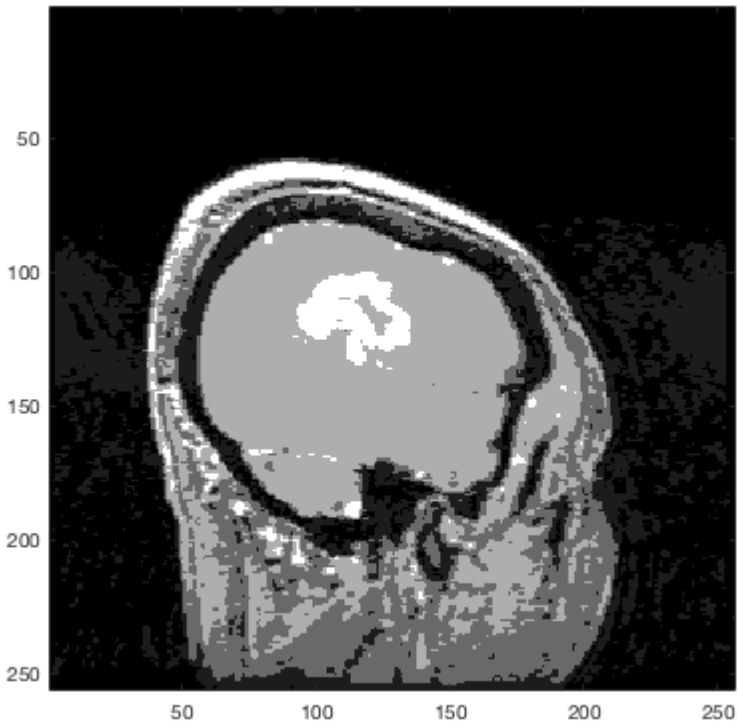
```
  figure(3 * i + 1);
  imagesc(uint8(segimg{i}));
  axis image;
  colormap('default');

  figure(3 * i + 2);
  imagesc(uint8(K));
  axis image;
  colormap('gray');

  nmeans{i};
end

%
%================================ segmentK ================================
```

Westin K-Means Image Segmentation



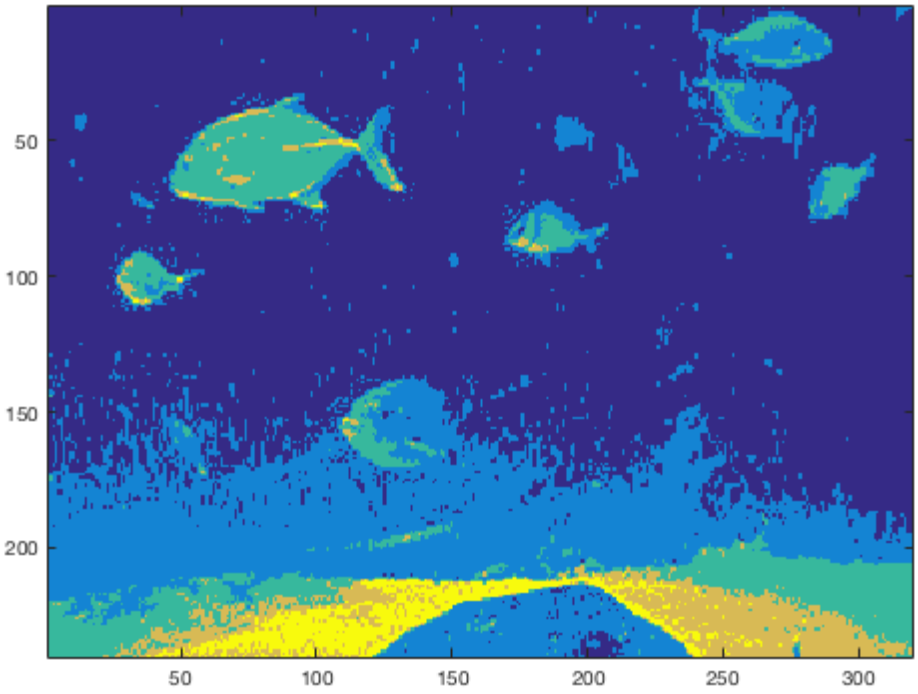
Max Iterations: 5
Initial Means:
[3, 10, 60, 140, 150]

Westin K-Means Image Segmentation
Using Average Values



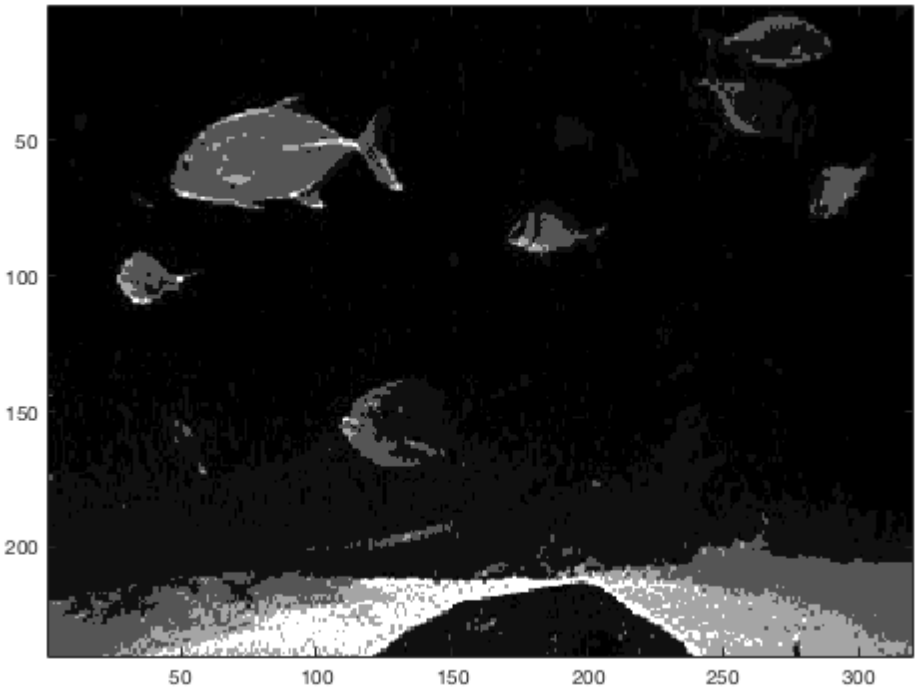
Compared to the built in kmeans, the image segmentation algorithm worked better, since it was able to pick out the small section inside the brain apart from the rest of the brain.

Fish04 K-Means Image Segmentation



Max Iterations: 5
Initial Means:
[1, 5 ,40, 80, 120]

Fish04 K-Means Image Segmentation Using Average Values



The image based segmentation algorithm could pick out more of the fish from the dark background. This was helped by choosing initial means that matched the values common in the picture.

Published with MATLAB® R2016b

```
%================================ segIMC ================================
%
%  function [J, K, means] = segIMC(I, iter, lambda, means, ncov)
%
%  Use the Iterative Conditioning Modes (ICM) algorithm for segmentation
%  of the grayscale image I.
%
%  Input:
%    I          - image I ( from |R2 -> |R ).
%    iter       - #iteration (iter = -1 for loop until no change)
%    lambda - the weighting of the neighborhood agreement part.
%    means      - initial guess at means, each column is a mean value.
%    ncov       - covariance matrix to use in the scaling (use 1 if none).
%                      can be scalar for all means to use, or can be a unique
%                      value for each mean value.
%
%  Output:
%    J          - the segmentation map.
%    K          - the simplified image using the means and the segmentation.
%    means      - the final segmentation means.
%
%================================ segIMC ================================

%
%  Name:                segIMC.m
%
%  Author:              Patricio A. Vela, pvela@gatech.edu
%
%  Created:             2010/01/05
%  Modified:    2012/04/07
%
%================================ segIMC ================================
function [J, K, means] = segIMC(I, iter, lambda, means, ncov)

%--[0] Parse the input arguments, set to defaults if needed.
if (nargout == 0)       % If nothing expected, then don't bother
  return;               %   doing the computations.
end

if (nargin < 4)         % If first three not given, can't do much.
  disp('ERROR: Need at least the first three arguments');
  error('BadArgs');
end

if ((nargin < 5) || isempty(ncov))
  ncov = 1;             % If no covariance, default is 1.
end

if (isscalar(ncov))     % If scalar, copy for each mean value.
  ncov = repmat(ncov, [1, size(means,2)]);
end

%--[1] Prep workspace and variables.
imsize = size(I);

xi = 1:length(means);           % Generate set of class labels.
J  = ones(imsize);                      % Instantiate the return variable.
oJ = zeros(imsize);                     % Want to keep track of old segmenation map.


% YOUR CODE HERE
% compute the data energy and set to "energy" variable (see function below).
% minimize energy to generate initial set of assignments.
```

```
% store the assignments as J.  Basically this is k-means with 1 iteration.
% Can simply code it here, or run k-means once.  Better to just code.
E = zeros([imsize, xi(end)]);
                                                    % Pre-allocate memory for data energy.


%--[2] Perform the segmentation iterations.

iter = round(iter);                     % Make integer if not.

% While segmentation does not change or number of iterations not yet met.
while (any(oJ(:) ~= J(:)) && (iter > 0))
  iter = iter - 1;              % Update number of iterations left.
  oJ = J;                               % Set old copy to previous segmentation map.

  % YOUR CODE HERE.  USE THE HELPER FUNCTIONS BELOW.
  % Steps:
  %  (1) Compute the data energy and set to "energy".
  E = compDataEnergy();

  %  (2) Compute and add the smoothing energy to "energy".
  E = E + compSmoothingEnergy();

  %  (3) Minimize "energy" to generate assignments (segmentation).
  [val, J] = min(E,[],3);

  %  (4) Update the means based on the segmentation
  compMeans();

end

if (nargout >= 2)               % If image expected, then create it.
  K = J;
  means = means .* ncov;
  for i=1:size(means,2)
      K(J == i) = means(i);
  end
end

%
%============================ Helper Functions ==========================
%
%  These functions live within the scope of the segICM function.
%  What that means is that they can be invoked from the loop above
%  and they will have access to the variables above (think of them
%  as global variables in some sense).  Using functions within a
%  function is a clean way to do complex thing but have the main
%  code of the function look nice and clean.
%

  %--------------------------- compDataEnergy -------------------------
  %
  %  Compute the ICM data matching energy.
  %
  function energyD = compDataEnergy

  energyD = zeros([imsize, xi(end)]);   % Initialize data energy.
  for ei=1:length(xi)                           % For each class label ...
    energyD(:,:,ei) = (I - means(ei)).^2;
  end

  end

  %----------------------- compSmoothingEnergy ----------------------
  %
```

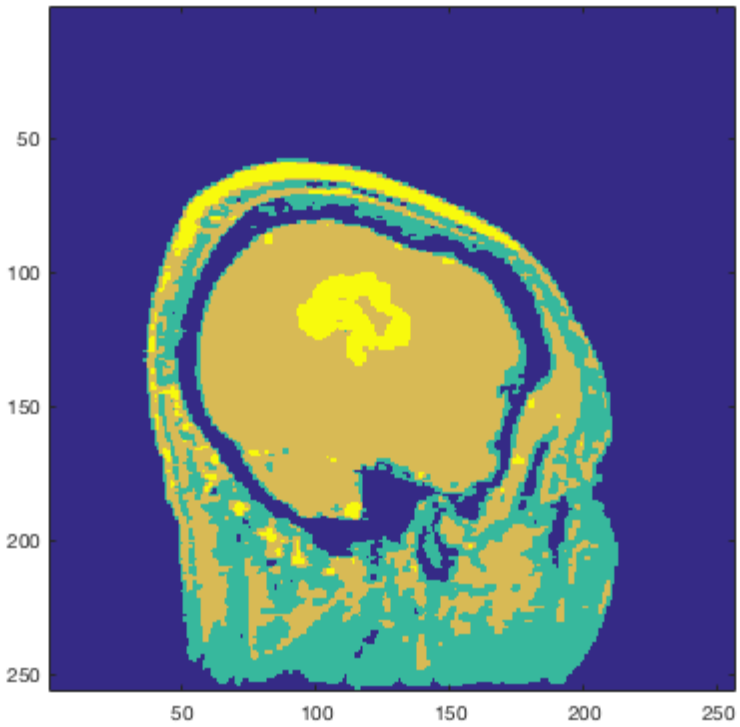
```
  %  Compute the smoothing energy, sometimes call the regularization energy.
  %  Uses the 8 neighbors to compute the energy.  For more aggressive
  %  smoothing, con try to use 24 neighbors or more.
  %
  function energyS = compSmoothingEnergy

  energyS = zeros([imsize, xi(end)]);   % Initialize smoothing energy.

  nhbrKern = [1 1 1; 1 0 1; 1 1 1];             % Look at all 8 neighbors.
  for ei=xi
    segImg = double(J ~= ei);           % Pixels NOT in current class give 1.
    energyS(:,:,ei) = lambda * imfilter(segImg, nhbrKern);
                                                                    % Count up disagreeing neighbors.
  end

  end

  %---------------------------- compMeans ----------------------------
  %
  %  Given the segmentation, compute the class means.  The mean should
  %  be updated only if there is more than 1 pixel in the class.
  %
  function compMeans

  for mi=1:length(xi)                           % For each class label ...
    pts_in_layer = sum(sum(J == mi));
    if (pts_in_layer > 0)
        means(mi) = sum(I(J == mi)) ./ pts_in_layer;
    else
        means(mi) = 0;
    end
  end

  end

end
%
%================================ segIMC ================================
```

Published with MATLAB® R2016b

```
%============================== segmentI ==============================
%
%  script segmentI
%
%
%  This is a Matlab script that will run the simple k-means algorithm
%  for segmenting an image.   You should select two images from the
%  homework Matlab file and perform segmentation on them. The output will
%  be used by another script of mine for visualization.
%
%
%  This code uses some Matlab tricks to be somewhat generic.  First,
%  all arguments are encapsulated into a cell array.  This works as
%  follows.  The cell array belows consists of two arguments:
%
%  >> sampleCellArray = {40, 34};
%
%  that when expanded as an argument to a function, provides two
%  inputs to the function,
%
%  >> plus( sampleCellArray{:} )
%
%  The output should be the addition of the two arguments:
%
%  ans =
%
%     74
%
%  Anyhow, this function expectes the same, but the arguments are
%  consistent with what the segKmeans function expects.
%
%============================== segmentI ==============================
close all
clear
load('segment.mat');
picks = [1 2];

for i = 1:length(picks)
  switch (picks(i))
    case 1,
      images{i} = westin;
      iparms{i} = { 5 , 100, [3, 10, 60, 140, 150] };
    case 2,
      images{i} = fish04;
      iparms{i} = { 5 , 40, [1, 5 ,40, 80, 120] };
  end
end
% Add more cases if you want to do all of the images, and change
% "picks" accordingly.


for i = 1:length(images)
  [segimg{i}, K, nmeans{i}] = segICM(double(images{i}), iparms{i}{:});
  figure(i);

  figure(3 * i);
  imagesc(images{i});
  axis image;
  colormap('gray');
```

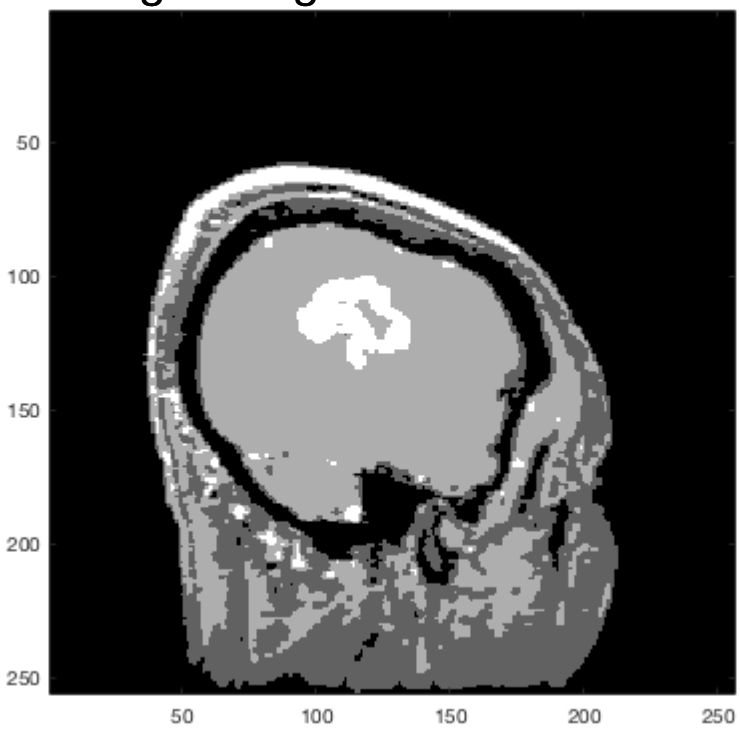
```
  figure(3 * i + 1);
  imagesc(uint8(segimg{i}));
  axis image;
  colormap('default');

  figure(3 * i + 2);
  imagesc(uint8(K));
  axis image;
  colormap('gray');

  nmeans{i};
end

%
%================================ segmentI ================================
```

Westin K-Means Image Segmentation
Iterative Conditioning Mode



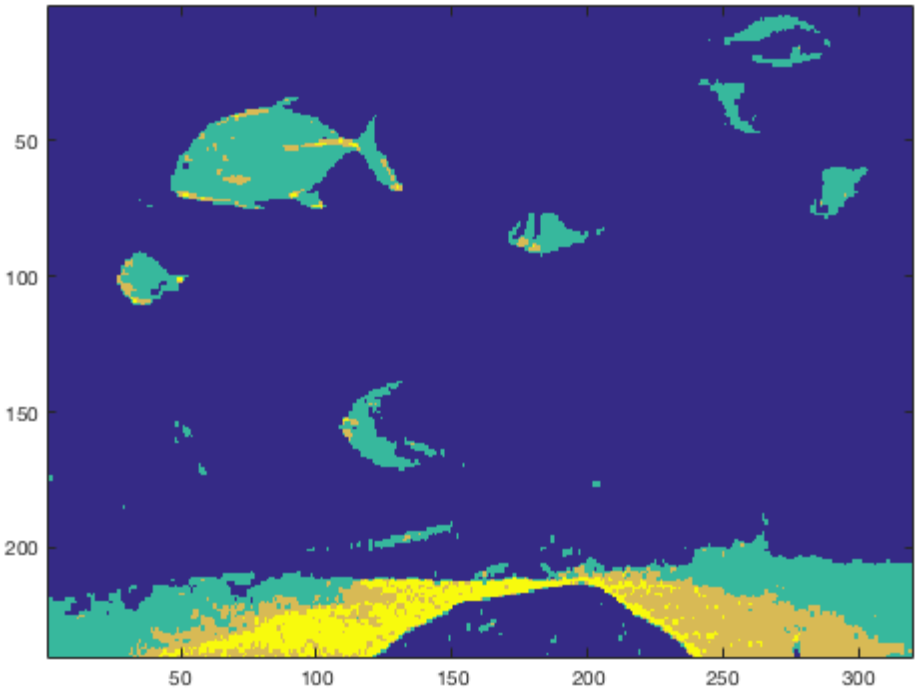
Max Iterations: 5
Lambda: 100
Initial Means:
[3, 10, 60, 140, 150]

Westin K-Means Image Segmentation
Iterative Conditioning Mode
Using Average Values



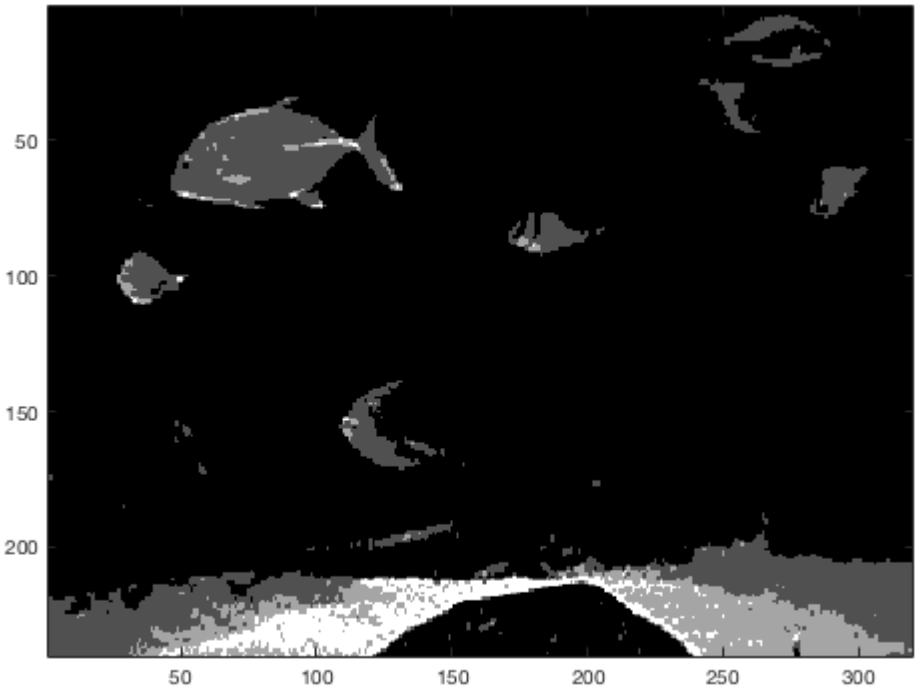
Adding the regularization term helped smoothen the image. This is apparent in the background. While the previous algorithms showed two different labels for the dark background, this algorithm labels the entire background as one.

Fish04 K-Means Image Segmentation
Iterative Conditioning Mode



Max Iterations: 5
Lambda: 40
Initial Means:
[3, 10, 60, 140, 150]

Fish04 K-Means Image Segmentation
Iterative Conditioning Mode
Using Average Values



The regularization term has a negative effect on this image. The smoothing removes parts of the fish that should be distinguished from the background.

Published with MATLAB® R2016b