```
%============================== 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