```
%================================== 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.
```

```matlab
% 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