```
function [coordinates,newElements,varargout] ...
          = refine(coordinates,elements,varargin)

%refine: local refinement of finite element mesh by red-green-blue
%        refinement, where marked elements are red-refined.
%
%Usage:
%
%[COORDINATES,ELEMENTS,DIRICHLET,NEUMANN] ...
%    = REFINE(COORDINATES,ELEMENTS,DIRICHLET,NEUMANN,MARKED)
%
%    refine expects as input a finite element mesh described by the
%    fields COORDINATES, ELEMENTS, DIRICHLET, and NEUMANN. The vector
%    MARKED contains the indices of elements which are refined by
%    red-refinement, i.e. which are split into 4 similar triangles.
%
%    The function returns the refined mesh in terms of the same data as
%    for the input.
%
%Remark:
%
%    This program is a supplement to the paper
%    >> Efficient Implementation of Adaptive P1-FEM in Matlab <<
%    by S. Funken, D. Praetorius, and P. Wissgott. The reader should
%    consult that paper for more information.
%
%Authors:
%
%    S. Funken, D. Praetorius, P. Wissgott  13-07-09

nE = size(elements,1);
markedElements = 1:nE;
%*** Sort elements such that first edge is longest
dx = coordinates(elements(:,[2,3,1]),1)-coordinates(elements,1);
dy = coordinates(elements(:,[2,3,1]),2)-coordinates(elements,2);
[hT,idxMax] = max(reshape(dx.^2+dy.^2,nE,3),[],2);
idx = ( idxMax==2 );
elements(idx,:) = elements(idx,[2,3,1]);
idx = ( idxMax==3 );
elements(idx,:) = elements(idx,[3,1,2]);
%*** Obtain geometric information on edges
[edge2nodes,element2edges,boundary2edges{1:nargin-2}] ...
    = provideGeometricData(elements,varargin{1:end});
%*** Mark edges for refinement
edge2newNode = ones(max(max(element2edges)),1);
%*** Generate new nodes
edge2newNode(find(edge2newNode)) = size(coordinates,1) + [1:nnz(edge2newNode)];
idx = find(edge2newNode);
coordinates(edge2newNode(idx),:) ...
    = (coordinates(edge2nodes(idx,1),:)+coordinates(edge2nodes(idx,2),:))/2;
%*** Refine boundary conditions
for j = 1:nargout-2
    boundary = varargin{j};
    if ~isempty(boundary)
        newNodes = edge2newNode(boundary2edges{j});
        markedEdges = find(newNodes);
        if ~isempty(markedEdges)
            boundary = [boundary(~newNodes,:); ...
                        boundary(markedEdges,1),newNodes(markedEdges); ...
                        newNodes(markedEdges),boundary(markedEdges,2)];
```

```
        end
    end
    varargout{j} = boundary;
end
%*** Provide new nodes for refinement of elements
newNodes = edge2newNode(element2edges);
%*** Determine type of refinement for each element
markedEdges = (newNodes~=0);
red     = ( markedEdges(:,1) &  markedEdges(:,2) &  markedEdges(:,3) );
%*** Generate element numbering for refined mesh
idx = 4*ones(nE,1);
idx = [1;1+cumsum(idx)];
newElements([idx(red),1+idx(red),2+idx(red),3+idx(red)],:) ...
    = [elements(red,1),newNodes(red,1),newNodes(red,3); ...
       newNodes(red,1),elements(red,2),newNodes(red,2); ...
       newNodes(red,3),newNodes(red,2),elements(red,3); ...
       newNodes(red,2),newNodes(red,3),newNodes(red,1)];
end

%-------------------------------------------------------------------------
function [edge2nodes,element2edges,varargout] ...
            = provideGeometricData(elements,varargin)

%provideGeometricData: returns geometric data for finite element mesh
%
%Usage:
%
%[EDGE2NODES,ELEMENT2EDGES,DIRICHLET2EDGES,NEUMANN2EDGES] ...
%    = PROVIDEGEOMETRICDATA(ELEMENTS,DIRICHLET,NEUMANN)
%
%Comments:
%
%    provideGeometricData expects as input a finite element mesh described
%    by the fields COORDINATES, ELEMENTS, DIRICHLET, and NEUMANN. The
%    function chooses a numbering of the edges and then return this numbering
%    related to nodes, edges, and the boundary conditions.
%
%    EDGE2NODES(K) returns the indices of the two nodes of the k-th edge.
%    ELEMENT2EDGES(J,K) provides the edge number of the edge between the two
%    nodes ELEMENTS(J,K) and ELEMENTS(J,K+1). DIRICHLET2EDGES(K) provides
%    the number of the k-th Dirichlet edge given by DIRICHLET(K,:). The same
%    applies for NEUMANN2EDGES
%
%Remark:
%
%    This program is a supplement to the paper
%    >> Efficient Implementation of Adaptive P1-FEM in Matlab <<
%    by S. Funken, D. Praetorius, and P. Wissgott. The reader should
%    consult that paper for more information.
%
%Authors:
%
%    S. Funken, D. Praetorius, P. Wissgott  10-07-08

nE = size(elements,1);
nB = nargin-1;
%*** Node vectors of all edges (interior edges appear twice)
I = elements(:);
J = reshape(elements(:,[2,3,1]),3*nE,1);
%*** Symmetrize I and J (so far boundary edges appear only once)
pointer = [1,3*nE,zeros(1,nB)];
```

```
for j = 1:nB
    boundary = varargin{j};
    if ~isempty(boundary)
        I = [I;boundary(:,2)];
        J = [J;boundary(:,1)];
    end
    pointer(j+2) = pointer(j+1) + size(boundary,1);
end
%*** Create numbering of edges
idxIJ = find(I < J);
edgeNumber = zeros(length(I),1);
edgeNumber(idxIJ) = 1:length(idxIJ);
idxJI = find(I > J);
number2edges = sparse(I(idxIJ),J(idxIJ),1:length(idxIJ));
[foo{1:2},numberingIJ] = find( number2edges );
[foo{1:2},idxJI2IJ] = find( sparse(J(idxJI),I(idxJI),idxJI) );
edgeNumber(idxJI2IJ) = numberingIJ;
%*** Provide element2edges and edge2nodes
element2edges = reshape(edgeNumber(1:3*nE),nE,3);
edge2nodes = [I(idxIJ),J(idxIJ)];
%*** Provide boundary2edges
for j = 1:nB
    varargout{j} = edgeNumber(pointer(j+1)+1:pointer(j+2));
end
end
```