```
function hh2fem1D(N)

gn=3;
a=0;
b=1;

% gauss rule for computation of error
[gx,gw] = gauss(gn);
leftHat = 0.5*[1-gx];
rightHat = 0.5*[1+gx];

% initial mesh
co0 = [0;0.5;1];
el0 = [1 2;2 3];

%-------------------------------------
% smooth solution
%-------------------------------------
%f = @(x) x.^2 + x - 3;
%u = @(x) x.*(x-1);
%uP = @(x) 2*x-1;

%-------------------------------------
% singular solution
%-------------------------------------
alpha = 0.9;
u = @(x) x.^(alpha).*(x-1);
uP = @(x) (alpha+1)*x.^(alpha) - alpha*x.^(alpha-1);
f = @(x) -(alpha+1)*alpha*x.^(alpha-1) + alpha*(alpha-1)*x.^(alpha-2) ...
    + (alpha+1)*x.^(alpha) - alpha*x.^(alpha-1) + ...
    x.^(alpha+1)-x.^(alpha);


energy = zeros(N,1);
nDofs = zeros(N,1);
H1error = zeros(N,1);
est = zeros(N,1);

co = co0;
el = el0;

for n=1:N
    nE = size(el,1);
    [x,energy(n)] = fem1D(co,el,f);
    x = [0 x' 0 ]';

    % uniform refinement
    [co_fine,el_fine,f2s] = refine(co,el,1:size(el,1));
    [x_fine] = fem1D(co_fine,el_fine,f);
    x_fine = [0 x_fine' 0 ]';

    % compute H1-error of refined solution
    tmp=0;
    for k=1:nE
        nodes = el_fine(k,:);
        c = co_fine(nodes);
        h=c(2)-c(1);
        uPx = uP(0.5*(c(2)+c(1) + gx*h));
        uhPx = (x_fine(nodes(2))-x_fine(nodes(1)))/h;
        tmp = tmp + 0.5*h*(gw*( (uPx-uhPx).^2 ))';
```

```
    end
    H1error(n) = sqrt(tmp);
    nDofs(n) = size(el_fine,1);

    % compute error estimator
    [rho] = hh2est(x,x_fine,co,el,co_fine,el_fine,f2s);
    est(n) = sqrt(sum(rho));

    % mark elements
    [marked] = doerfler_marking(rho,0.25);

    % refine marked elements
    [co,el] = refine(co,el,marked);
end

figure(1);
subplot(1,2,1);
loglog(nDofs,H1error,'r-',nDofs,est,'b',nDofs,nDofs.^(-1),'k--')
xlabel('N = Number of elements');
legend('H1-Error','h/h2 error estimator','N^{-1}');
subplot(1,2,2);
plot(co_fine,x_fine,[0:0.001:1],u([0:0.001:1]))
legend('fe-solution','true solution');

end

%-------------------------------------------------------------------------
function marked = doerfler_marking(indicators,theta)
% realization of doerfler marking
%
% INPUT: indicators ... (N x 1) vector of element indicators
%        theta ... parameter in dörfler marking
%
% OUTPUT: marked ... (1 x K)-vector of indices of marked elements

[indicators,idx] = sort(indicators,'descend');
sum_indicators = cumsum(indicators);
ell = find(sum_indicators >= sum_indicators(end)*theta,1);
marked = idx(1:ell)';

end

%-------------------------------------------------------------------------
function [co_new,el_new,f2s] = refine(co,el,marked)
%
% simple refinement routine for 1D-mesh
%
% INPUT: co ... ( (N+1) x 1 )-vector for coordinates
%        el ... (N x 2)-vector for elements
%        marked ... (1 x K)-vector containing the indices of marked elements
%
% OUTPUT: co_new ... ( (N+K+1) x 1 )-vector of new coordinates
%         el_new ... (N+K x 1)-vector of new elements
%         f2s ... (N x 2) array of father2son relation

nE = size(el,1);
nC = size(co,1);

% initial father2son array, nothing is refined yet!
f2s = [ [1:nE]' [1:nE]' ];
```

```
co_new = co;
el_new = el;

% run through indices of marked elements. find midpoint on element
% and update el_new and f2s
for k = marked
  left_node_idx = el(k,1);
  right_node_idx = el(k,2);
  left_node = co(left_node_idx);
  right_node = co(right_node_idx);
  new_node = 0.5*(left_node + right_node);
  co_new = [co_new; new_node];
  new_node_idx = size(co_new,1);
  el_new(k,:) = [left_node_idx new_node_idx];
  el_new = [el_new ; [ new_node_idx right_node_idx] ];
  new_el_idx = size(el_new,1);
  f2s(k,:) = [k new_el_idx];
end

% sort nodes
[co_new,idx] = sort(co_new);
v(idx) = 1:size(idx,1);
el_new(:,1) = v(el_new(:,1));
el_new(:,2) = v(el_new(:,2));

end
```