```
 1: function [K,F] = assemble_1D(x,lambda,f)
 2: N = length(x);
 3: K = zeros(N);
 4: F = zeros(N,1);
 5:
 6: for i = 1:N-1
 7:     h = x(i+1)-x(i);
 8:     xm = (x(i)+x(i+1))/2; %mid point evaluation
 9:     lam = lambda(xm);
10:     Ke = lam/h * [1 -1; -1 1]; %exact P1 stiffness matrix
11:     xm = (x(i)+x(i+1))/2;
12:     Fe = f(xm)*h/2 * [1;1]; %mid point quadrature for RHS
13:     %assembly into global system
14:     K(i:i+1,i:i+1) = K(i:i+1,i:i+1) + Ke;
15:     F(i:i+1) = F(i:i+1) + Fe;
16: end
17: end
```

midpoint rule is not accurate enough.

```
clear; clc; close all;
pList = [5, 10, 20, 100];
nIter = 9;

figure;
tiledlayout(2,2)
for ip = 1:length(pList)
    p = pList(ip);

    f = @(x) 2*p^3*x./(p^2*x.^2+1).^2; %RHS
    lambda = @(x) 1; %DIffusion coefficient

    u_exact = @(x) atan(p*x);

    x = linspace(-1,1,10); %initial uniform mesh (includes x=0)

    for it = 1:nIter %adaptive FEM loop
        [K,F] = assemble_1D(x,lambda,f); %assemble global stiffness matrix K and load vector F
        %Neumann boundary conditions at x=1
        F(end) = F(end) + p/(p^2+1);
        %Dirichlet boundary conditions at x=-1
        K(1,:) = 0;
        K(1,1) = 1;
        F(1) = -atan(p);

        u = K \ F;

        if it < nIter % h-adaptivity loop
            eta = flux_jump(x,u,lambda);
            x = h_adapt(x,eta,0.3);
        end
    end

    u_ex = u_exact(x(:));          % force column vector
    err_inf = max(abs(u - u_ex));
    nexttile
    plot(x,u,'-o','LineWidth',1.5); hold on
    plot(x,u_ex,'--','LineWidth',1.5)
    title(['p = ',num2str(p)])
    xlabel('x'), ylabel('u')
    legend('FEM','exact','Location','best')
    grid on

    fprintf('p = %d\n', p);
    fprintf('error = %.3e\n\n', err_inf);
end

sgtitle('Exercise 6A: h-adaptivity and exact solution comp
```

```
arison')
```

```
clear; clc;
lambda = @(x) (x < 1/sqrt(2)) + 10*(x >= 1/sqrt(2)); %diffusion coefficient
f = @(x) 0; %homogeneous RHS
x = linspace(0,1,6);   %coarse initial mesh, doesn't include x_m
nIter = 6;

for it = 1:nIter
    [K,F] = assemble_1D(x,lambda,f); %assemble
    K(1,:) = 0;  K(1,1) = 1;  F(1) = 0; % Dirichlet BC at x=0
    K(end,:) = 0; K(end,end) = 1; F(end) = 1; % Dirichlet BC at x=1
    u = K\F;

    if it < nIter %h-adaptivity loop
        eta = flux_jump(x, u, lambda);
        x = h_adapt(x,eta,0.4);
    end
end

figure
plot(x,u,'-o','LineWidth',1.5)
xlabel('x'), ylabel('u')
title('Ex6B, h-adaptivity with coefficient jump')
grid on
```

```
clear; clc; close all;
p = 70;
%p=-70;
lambda = @(x) 1; %constant diffusion
f = @(x) 0; %homogeneous RHS

N = 30;      % Initial mesh (uniform)
x = linspace(0,1,N)';

nIter = 8;

for it = 1:nIter
    [K,F] = assemble_1D(x,lambda,f); %assemble
    for i = 1:length(x)-1 %convection term added manually
        h = x(i+1) - x(i);
        Ke_conv = p/2 * [-1 1; -1 1];
        K(i:i+1,i:i+1) = K(i:i+1,i:i+1) + Ke_conv;
    end
    K(1,:) = 0; K(1,1) = 1; F(1) = 0; % Dirichlet BC at x=0
    K(end,:) = 0; K(end,end) = 1; F(end) = 1; % Dirichlet BC at x=1

    u = K\F;

    if it < nIter
        Nn = length(x); %initialize indicator
        eta = zeros(Nn,1);
        for j = 2:Nn-1 %interior nodes only
            ul = (u(j)-u(j-1))/(x(j)-x(j-1)); %left derivative
            ur = (u(j+1)-u(j))/(x(j+1)-x(j)); %right derivative
            %Jump in convection-diffusion flux
            Jl = -ul + p*u(j);
            Jr = -ur + p*u(j);
            eta(j) = abs(Jr - Jl);
        end
        x = r_adapt(x, eta);
    end
end

figure
plot(x,u,'-o','LineWidth',1.5)
xlabel('x'), ylabel('u')
title(['Ex6C, r-adaptivity, PÃ©clet p = ',num2str(p)])
grid on
```

```
function eta = flux_jump(x,u,lambda)
N = length(x);
eta = zeros(N,1);

for j = 2:N-1
    ul = (u(j)-u(j-1))/(x(j)-x(j-1)); %left derivative
    ur = (u(j+1)-u(j))/(x(j+1)-x(j)); %right derivative
    eta(j) = abs(lambda(x(j))*ur - lambda(x(j))*ul); %erro
r indicator
end
end
```

```
function xnew = h_adapt(x,eta,theta) %eta is local error
xnew = x(1);
for i = 1:length(x)-1
    if eta(i) > theta*max(eta) %if local error is large, i
nster a midpoint
        xnew = [xnew, (x(i)+x(i+1))/2];
    end
    xnew = [xnew, x(i+1)]; %if local error is small, keep
the element unchanged
end
xnew = unique(xnew);
end
```

```
 1: function xnew = r_adapt(x,eta_elem)
 2: N = length(x);
 3: w = abs(eta_elem) + 1e-10; %error based monitor function
 4: s = zeros(N,1);
 5: for i = 2:N
 6:     s(i) = s(i-1) + w(i-1); %cumulative sum
 7: end
 8: s = s / s(end);    % normalize to [0,1]
 9: s_new = linspace(0,1,N)';
10: xnew = interp1(s, x, s_new, 'linear');
11: end
```