

```
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.

```
1: clear; clc; close all;
2: pList = [5, 10, 20, 100];
3: nIter = 9;
4:
5: figure;
6: tiledlayout(2,2)
7: for ip = 1:length(pList)
8:     p = pList(ip);
9:
10:    f = @(x) 2*p^3*x./(p^2*x.^2+1).^2; %RHS
11:    lambda = @(x) 1; %Diffusion coefficient
12:
13:    u_exact = @(x) atan(p*x);
14:
15:    x = linspace(-1,1,10); %initial uniform mesh (includes
x=0)
16:
17:    for it = 1:nIter %adaptive FEM loop
18:        [K,F] = assemble_1D(x,lambda,f); %assemble global
stiffness matrix K and load vector F
19:        %Neumann boundary conditions at x=1
20:        F(end) = F(end) + p/(p^2+1);
21:        %Dirichlet boundary conditions at x=-1
22:        K(1,:) = 0;
23:        K(1,1) = 1;
24:        F(1) = -atan(p);
25:
26:        u = K \ F;
27:
28:        if it < nIter % h-adaptivity loop
29:            eta = flux_jump(x,u,lambda);
30:            x = h_adapt(x,eta,0.3);
31:        end
32:    end
33:
34:    u_ex = u_exact(x(:)); % force column vector
35:    err_inf = max(abs(u - u_ex));
36:    nexttile
37:    plot(x,u,'-o','LineWidth',1.5); hold on
38:    plot(x,u_ex,'--','LineWidth',1.5)
39:    title(['p = ',num2str(p)])
40:    xlabel('x'), ylabel('u')
41:    legend('FEM','exact','Location','best')
42:    grid on
43:
44:    fprintf('p = %d\n', p);
45:    fprintf('error = %.3e\n\n', err_inf);
46: end
47:
48: sgtitle('Exercise 6A: h-adaptivity and exact solution comp
```

arison')

```
1: clear; clc;
2: lambda = @(x) (x < 1/sqrt(2)) + 10*(x >= 1/sqrt(2)); %diff
usion coefficient
3: f = @(x) 0; %homogeneous RHS
4: x = linspace(0,1,6); %coarse initial mesh, doesn't inclu
de x_m
5: nIter = 6;
6:
7: for it = 1:nIter
8:     [K,F] = assemble_1D(x,lambda,f); %assemble
9:     K(1,:) = 0; K(1,1) = 1; F(1) = 0; % Dirichlet BC at
x=0
10:    K(end,:) = 0; K(end,end) = 1; F(end) = 1; % Dirichlet
BC at x=1
11:    u = K\F;
12:
13:    if it < nIter %h-adaptivity loop
14:        eta = flux_jump(x, u, lambda);
15:        x = h_adapt(x,eta,0.4);
16:    end
17: end
18:
19: figure
20: plot(x,u,'-o','LineWidth',1.5)
21: xlabel('x'), ylabel('u')
22: title('Ex6B, h-adaptivity with coefficient jump')
23: grid on
```

```

1: clear; clc; close all;
2: p = 70;
3: %p=-70;
4: lambda = @(x) 1; %constant diffusion
5: f = @(x) 0; %homogeneous RHS
6:
7: N = 30; % Initial mesh (uniform)
8: x = linspace(0,1,N)';
9:
10: nIter = 8;
11:
12: for it = 1:nIter
13:     [K,F] = assemble_1D(x,lambda,f); %assemble
14:     for i = 1:length(x)-1 %convection term added manually
15:         h = x(i+1) - x(i);
16:         Ke_conv = p/2 * [-1 1; -1 1]; ✓
17:         K(i:i+1,i:i+1) = K(i:i+1,i:i+1) + Ke_conv;
18:     end
19:     K(1,:) = 0; K(1,1) = 1; F(1) = 0; % Dirichlet BC at x=
0
20:     K(end,:) = 0; K(end,end) = 1; F(end) = 1; % Dirichlet
BC at x=1
21:
22:     u = K\F;
23:
24:     if it < nIter
25:         Nn = length(x); %initialize indicator
26:         eta = zeros(Nn,1);
27:         for j = 2:Nn-1 %interior nodes only
28:             ul = (u(j)-u(j-1))/(x(j)-x(j-1)); %left deriva
tive
29:             ur = (u(j+1)-u(j))/(x(j+1)-x(j)); %right deriv
ative
30:             %Jump in convection-diffusion flux
31:             Jl = -ul + p*u(j);
32:             Jr = -ur + p*u(j);
33:             eta(j) = abs(Jr - Jl);
34:         end
35:         x = r_adapt(x, eta);
36:     end
37: end
38:
39: figure
40: plot(x,u,'-o','LineWidth',1.5)
41: xlabel('x'), ylabel('u')
42: title(['Ex6C, r-adaptivity, Péclet p = ',num2str(p)])
43: grid on

```

λ_{right}

λ_{left}

why?

should be $u'(j)$

```
1: function eta = flux_jump(x,u,lambda)
2: N = length(x);
3: eta = zeros(N,1);
4:
5: for j = 2:N-1
6:     ul = (u(j)-u(j-1))/(x(j)-x(j-1)); %left derivative
7:     ur = (u(j+1)-u(j))/(x(j+1)-x(j)); %right derivative
8:     eta(j) = abs(lambda(x(j))*ur - lambda(x(j))*ul); %error indicator
9: end
10: end
```

↑
not correct of material change

↑
-0

↑
+0

```
1: function xnew = h_adapt(x,eta,theta) %eta is local error
2: xnew = x(1);
3: for i = 1:length(x)-1
4:     if eta(i) > theta*max(eta) %if local error is large, i
nster a midpoint
5:         xnew = [xnew, (x(i)+x(i+1))/2];
6:     end
7:     xnew = [xnew, x(i+1)]; %if local error is small, keep
the element unchanged
8: end
9: xnew = unique(xnew); ✓
10: 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
```