```
    1: % h-adaptivity
    2: % for a given mesh and solution we generate a new adapted
mesh by splitting
    3: % the elements with large errors into two new elements
    4:
    5: function mesh = adapt_h(nodes,u,lambda)
    6:     n_nodes = length(nodes);
    7:     flux_jumps = jumps_flux(nodes,u,lambda);
    8:     alpha = 0.5;     % parameter for choosing elements to r
efine
    9:     crit_error = alpha*max(abs(flux_jumps));
   10:     mesh = nodes;    % will be the new nodes/mesh
   11:     % finding elements to refine
   12:     for i=2:n_nodes-1
   13:         if abs(flux_jumps(i)) > crit_error
   14:             mesh = [mesh, nodes(i-1)+(nodes(i)-nodes(i-1))
/2, nodes(i)+(nodes(i+1)-nodes(i))/2];
   15:         end
   16:     end
   17:     mesh = unique(mesh);
   18: end
```

ind. sorting of nodes.

```
    1: % r-adaptivity
    2: % for a given mesh and solution we generate a new adapted
mesh by moving
    3: % the existing nodes within the mesh; they get moved to a
position in order
    4: % to equally distribute the error over the intervall
    5: % we use the De Boor's algorithm (Huang, Russell; Adaptive
 Moving Mesh
    6: % Methods; $ 2.2.1)
    7:
    8: function mesh = adapt_r(nodes,u,lambda)
    9:     n_nodes = length(nodes);
   10:     flux_jumps = abs(jumps_flux(nodes,u,lambda));
   11:     p = 1/2*(flux_jumps(1:end-1) + flux_jumps(2:end));  %
has values for each element
   12:     h_vec = nodes(2:end) - nodes(1:end-1);
   13:     P = cumsum(h_vec'.*p);
   14:     P = [0;P];
   15:     xi = linspace(0,1,n_nodes);
   16:     mesh = nodes;
   17:     for j=2:n_nodes-1
   18:         idx_k = find(xi(j)*P(end) <= P, 1, 'first');
   19:         if idx_k > 1
   20:             x_j = nodes(idx_k-1) + xi(j)*(P(end)-P(idx_k -
1))/p(idx_k -1);
   21:             mesh(j) = x_j;
   22:         end
   23:     end
   24: end
```

```
    1: % assembling of the stiffness matrix and the load vector f
or a given mesh
    2: % in 1D
    3:
    4: function [K,f_vec] = assembling(nodes,lambda,f)
    5:     n_nodes = length(nodes);
    6:     K = zeros(n_nodes);
    7:     f_vec = zeros(n_nodes,1);
    8:     h_diff_vec = nodes(2:end) - nodes(1:end-1); % step wid
th
    9:     for k = 1:n_nodes-1
   10:         % stiffness matrix
   11:         lambda_int = integral(lambda, nodes(k), nodes(k+1)
);
   12:         K_loc = lambda_int/h_diff_vec(k)^2*[1,-1;-1,1];
   13:         K(k:k+1,k:k+1) = K(k:k+1,k:k+1) + K_loc;
   14:         % right hand side
   15:         phi_left = @(x) (nodes(k+1)-x)/h_diff_vec(k).*f(x)
;
   16:         phi_right = @(x) (x-nodes(k))/h_diff_vec(k).*f(x);
   17:         f_loc = [integral(phi_left,nodes(k),nodes(k+1)); i
ntegral(phi_right,nodes(k),nodes(k+1))];
   18:         f_vec(k:k+1) = f_vec(k:k+1) + f_loc;
   19:     end
   20: end
```

```
% Sheet 6 / Ex A
clf, clear, close all

% you need to play around with the parameters (enough star
ting nodes,
% refinements etc)

%p_vec = [5,10,20,100];
p = 100; % choose your p
n_nodes = 10;     % number of nodes on the coarse mesh (eve
n - x=0 not included, odd - x=0 included)
nodes = linspace(-1,1,n_nodes);
f = @(x) 2*p^3.*x./(p^2.*x.^2+1).^2;
u_exact = @(x) atan(p*x);
lambda = @(x) 1+0*x;

% h-adaptivity
it_h = 4;    % number of refinements with h-adaptivity
for it = 1:it_h
    [K,f_vec] = assembling(nodes,lambda,f);
    % adaption for boundary (dirichlet left)
    f_vec(1) = f_vec(1) + K(1,1)*1e6*(-atan(p));
    K(1,1) = K(1,1)*(1+1e6);
    % adaption for boundary (neumann right)
    f_vec(end) = f_vec(end) + p/(p^2+1);

    % solving the system
    u = K\f_vec;

    % adapting the mesh
    if it < it_h
        nodes = adapt_h(nodes,u,lambda);
    end
end

figure(1)
hold on
plot(nodes,u,'-o')
fplot(u_exact,[-1,1])
title(['h-adapt: u_h, u for p=' num2str(p) ' after ' num2s
tr(it_h) ' refinements, nstart=' num2str(n_nodes) ', nend=' num2s
tr(length(nodes))]);
xlabel('x values');
ylabel('u');
legend('u_h', 'u');
grid on;
hold off
file_name_1 = ['ex_6_A_h-adap_p' num2str(p) '_ref' num2str
(it_h) '_n_nodes' num2str(n_nodes) '.jpg'];
saveas(figure(1), file_name_1);
```

```
46:
47:
48: % r-adaptivity
49: n_nodes_r = 20;     % number of nodes on the coarse mesh (even - x=0 not included, odd - x=0 included)
50: nodes_r = linspace(-1,1,n_nodes_r);
51: it_r = 3;    % number of refinements with r-adaptivity
52: for it = 1:it_r
53:     [K,f_vec] = assembling(nodes_r,lambda,f);
54:     % adaption for boundary (dirichlet left)
55:     f_vec(1) = f_vec(1) + K(1,1)*1e6*(-atan(p));
56:     K(1,1) = K(1,1)*(1+1e6);
57:     % adaption for boundary (neumann right)
58:     f_vec(end) = f_vec(end) + p/(p^2+1);
59:
60:     % solving the system
61:     u_r = K\f_vec;
62:
63:     % adapting the mesh
64:     if it < it_r
65:         nodes_r = adapt_r(nodes_r,u_r,lambda);
66:     end
67: end
68:
69: figure(2)
70: hold on
71: plot(nodes_r,u_r,'-o')
72: fplot(u_exact,[-1,1])
73: title(['r-adapt: u_h, u for p=' num2str(p) ' after ' num2str(it_r) ' refinements and n=' num2str(n_nodes_r)]);
74: xlabel('x values');
75: ylabel('u');
76: legend('u_h', 'u');
77: grid on;
78: hold off
79: file_name_2 = ['ex_6_A_r-adap_p' num2str(p) '_ref' num2str(it_r) '_n_nodes' num2str(n_nodes_r) '.jpg'];
80: saveas(figure(2), file_name_2);
```

```
% Sheet 6 / Ex B
clf, clear, close all

% you need to play around with the parameters (enough star
ting nodes,
% refinements etc)

n_nodes = 10;     % number of nodes on the coarse mesh
nodes = linspace(0,1,n_nodes);
f = @(x) 0;
lambda = @(x) (x<1/sqrt(2))*1 + (x>=1/sqrt(2))*10;

% h-adaptivity
it_h = 10;    % number of refinements with h-adaptivity
for it = 1:it_h
    [K,f_vec] = assembling(nodes,lambda,f);
    % adaption for dirichlet boundary
    K(1,1) = K(1,1)*(1 + 1e6);
    f_vec(end) = K(end,end)*1e6;
    K(end,end) = K(end,end)*(1 + 1e6);

    % solving the system
    u = K\f_vec;

    % adapting the mesh
    if it < it_h
        nodes = adapt_h(nodes,u,lambda);
    end
end

figure(1)
hold on
plot(nodes,u,'-o')
title(['h-adapt: u_h after ' num2str(it_h) ' refinements,
nstart=' num2str(n_nodes) ', nend=' num2str(length(nodes))]);
xlabel('x values');
ylabel('u');
grid on;
hold off
file_name_1 = ['ex_6_B_h-adap_ref' num2str(it_h) '_n_nodes
' num2str(n_nodes) '.jpg'];
saveas(figure(1), file_name_1);


% r-adaptivity
n_nodes_r = 10;     % number of nodes on the coarse mesh
nodes_r = linspace(0,1,n_nodes_r);
it_r = 3;    % number of refinements with r-adaptivity
for it = 1:it_r
    [K,f_vec] = assembling(nodes_r,lambda,f);
```

```
    % adaption for dirichlet boundary
    K(1,1) = K(1,1)*(1 + 1e6);
    f_vec(end) = K(end,end)*1e6;
    K(end,end) = K(end,end)*(1 + 1e6);

    % solving the system
    u_r = K\f_vec;

    % adapting the mesh
    if it < it_r
        nodes_r = adapt_r(nodes_r,u_r,lambda);
    end
end

figure(2)
hold on
plot(nodes_r,u_r,'-o')
title(['r-adapt: u_h after ' num2str(it_r) ' refinements and n=' num2str(n_nodes_r)]);
xlabel('x values');
ylabel('u');
grid on;
hold off
file_name_2 = ['ex_6_B_r-adap_ref' num2str(it_r) '_n_nodes' num2str(n_nodes_r) '.jpg'];
saveas(figure(2), file_name_2);
```

```
% Sheet 6 / Ex C
clf, clear, close all

% you need to play around with the parameters (enough starting nodes,
% refinements etc)

p = 70; % parameter
%p = -70;

n_nodes = 10;     % number of nodes on the coarse mesh
nodes = linspace(0,1,n_nodes);
f = @(x) 0;
lambda = @(x) 1+0*x;

% h-adaptivity
it_h = 10;    % number of refinements with h-adaptivity
for it = 1:it_h
    [K,f_vec] = assembling(nodes,lambda,f);
    % need to add the term for phi'*phi (convection term)
    conv_loc = p/2*[-1,-1;1,1];
    for k=1:length(nodes)-1
        K(k:k+1,k:k+1) = K(k:k+1,k:k+1) + conv_loc;
    end

    % adaption for dirichlet boundary
    K(1,1) = K(1,1)*(1 + 1e6);
    f_vec(end) = K(end,end)*1e6;
    K(end,end) = K(end,end)*(1 + 1e6);

    % solving the system
    u = K\f_vec;

    % adapting the mesh
    if it < it_h
        nodes = adapt_h(nodes,u,lambda);
    end
end

figure(1)
hold on
plot(nodes,u,'-o')
title(['h-adapt: u_h after ' num2str(it_h) ' refinements, nstart=' num2str(n_nodes) ', nend=' num2str(length(nodes))]);
xlabel('x values');
ylabel('u');
grid on;
hold off
file_name_1 = ['ex_6_C_h-adap_ref' num2str(it_h) '_n_nodes' num2str(n_nodes) '.jpg'];
```

```
48: saveas(figure(1), file_name_1);
49:
50:
51: % r-adaptivity
52: n_nodes_r = 20;     % number of nodes on the coarse mesh
53: nodes_r = linspace(0,1,n_nodes_r);
54: it_r = 3;    % number of refinements with r-adaptivity
55: for it = 1:it_r
56:     [K,f_vec] = assembling(nodes_r,lambda,f);
57:     % need to add the term for phi'*phi (convection term)
58:     conv_loc = p/2*[-1,-1;1,1];
59:     for k=1:length(nodes_r)-1
60:         K(k:k+1,k:k+1) = K(k:k+1,k:k+1) + conv_loc;
61:     end
62:
63:     % adaption for dirichlet boundary
64:     K(1,1) = K(1,1)*(1 + 1e6);
65:     f_vec(end) = K(end,end)*1e6;
66:     K(end,end) = K(end,end)*(1 + 1e6);
67:
68:     % solving the system
69:     u_r = K\f_vec;
70:
71:     % adapting the mesh
72:     if it < it_r
73:         nodes_r = adapt_r(nodes_r,u_r,lambda);
74:     end
75: end
76:
77: figure(2)
78: hold on
79: plot(nodes_r,u_r,'-o')
80: title(['r-adapt: u_h after ' num2str(it_r) ' refinements a
nd n=' num2str(n_nodes_r)]);
81: xlabel('x values');
82: ylabel('u');
83: grid on;
84: hold off
85: file_name_2 = ['ex_6_C_r-adap_ref' num2str(it_r) '_n_nodes
' num2str(n_nodes_r) '.jpg'];
86: saveas(figure(2), file_name_2);
```

```
    1: % calculation of the flux jumps according to the chosen me
sh
    2: % nodes - corresponding to the mesh
    3: % u - approximation
    4: % flux_jumps - jump of the flux in each node of the mesh i
ncluding the
    5: % paramter function lambda
    6:
    7: function flux_jumps = jumps_flux(nodes, u, lambda)
    8:     n_nodes = length(nodes);
    9:     flux_jumps = zeros(n_nodes,1);
   10:     diff_u = u(2:end) - u(1:end-1);
   11:     diff_x = nodes(2:end) - nodes(1:end-1);
   12:     eps = 1e-8;
   13:     for i = 2:n_nodes-1
   14:         flux_jumps(i) = diff_u(i)/diff_x(i)*lambda(nodes(i
)+eps) - diff_u(i-1)/diff_x(i-1)*lambda(nodes(i)-eps);
   15:     end
   16: end
```