

TMA373 Assignment 4

Edoardo Mangia
edoardom@student.chalmers.se

Bibiana Farinha
bibiana@student.chalmers.se

February 2025

Contents

1	Task 1	1
2	Task 2	5

1 Task 1

a)

In this task, we will implement the FEM for the stationary convection-diffusion problem

$$\begin{aligned} -Du''(x) + \frac{1}{2}u'(x) &= 1, \quad 0 < x < \pi \\ u(0) = u(\pi) &= 0 \end{aligned} \tag{1}$$

The first step of FEM is to find the Variational Form of the problem. For that purpose, consider the space

$$H_0^1(0, \pi) = \{u \in H^1(0, \pi) \mid u(0) = u(\pi) = 0\}$$

Let $v \in H_0^1(0, \pi)$, and calculate the integral of 1 multiplied by v . Since $v \in H_0^1(0, \pi)$, v and its first derivative are square-integrable.

$$\int_0^\pi v(x)dx = \int_0^\pi \frac{1}{2}v(x)u'(x)dx - \int_0^\pi Dv(x)u''(x)dx$$

The second term of the right hand side has a second-order derivative term u'' . To reduce the order, we integrate by parts, yielding:

$$\int_0^\pi v(x)dx = \int_0^\pi \frac{1}{2}v(x)u'(x)dx + \int_0^\pi Dv'(x)u'(x)dx - \left[Dv(x)u'(x) \right]_0^\pi$$

From the definition of $H_0^1(0, \pi)$, the values of $v(0)$ and $v(\pi)$ are known, so the expression above can be simplified further:

$$\int_0^\pi v(x)dx = \int_0^\pi \frac{1}{2}v(x)u'(x)dx + \int_0^\pi Dv'(x)u'(x)dx \quad (2)$$

The VF reads: Find $u \in H_0^1(0, \pi)$, such that 2 holds for all $v \in H_0^1(0, \pi)$.

The next step of FEM is the formulation of the Finite Elements problem. Consider the space $V_h^0 = \text{span}(\varphi_1, \varphi_2, \dots, \varphi_m)$, where $\{\varphi_j\}_{j=1}^m$ are hat functions defined $T_h : 0 = x_0 < x_1 < \dots < x_{m+1} = 1$.

The FE reads: Find $u_h \in V_h^0$ such that 3 and 4 hold for all $v_h \in V_h^0$.

$$\int_0^\pi v_h(x)dx = \int_0^\pi \frac{1}{2}v_h(x)u_h'(x)dx + \int_0^\pi Dv_h'(x)u_h'(x)dx \quad (3)$$

$$u_h(0) = u_h(\pi) = 0 \quad (4)$$

Finally, the last step of FEM is finding the linear system of equations. To do this, set $u_h(x) = \sum_{j=1}^m \xi_j \varphi_j(x)$, where ξ_j is unknown and $\varphi_j(x)$ is the basis, and take $v_h = \varphi_i$ for $i = 1 \dots m$. Now we insert this in the FE problem and get:

$$\int_0^\pi \varphi_i(x)dx = \int_0^\pi \frac{1}{2}\varphi_i(x) \sum_{j=1}^m \xi_j \varphi_j'(x)dx + \int_0^\pi D\varphi_i'(x) \sum_{j=1}^m \xi_j \varphi_j'(x)dx$$

Using linearity of integration, we move the summation and constants outside:

$$\int_0^\pi \varphi_i(x)dx = \frac{1}{2} \sum_{j=1}^m \xi_j \int_0^\pi \varphi_i(x) \varphi_j'(x)dx + D \sum_{j=1}^m \xi_j \int_0^\pi \varphi_i'(x) \varphi_j'(x)dx$$

Now, we can define the convection and stiffness matrices, and the load vector b :

$$C_{ij} = \int_0^\pi \varphi_i(x) \varphi_j'(x)dx, \quad S_{ij} = \int_0^\pi \varphi_i'(x) \varphi_j'(x)dx, \quad b_i = \int_0^\pi \varphi_i(x)dx$$

Thus, the system simplifies to:

$$\sum_{j=1}^m \xi_j \left(\frac{1}{2}C_{ij} + DS_{ij} \right) = b_i, \quad \forall i = 1, \dots, m$$

Or, in matrix form:

$$A\xi = b, \text{ where } A = DS + \frac{1}{2}C$$

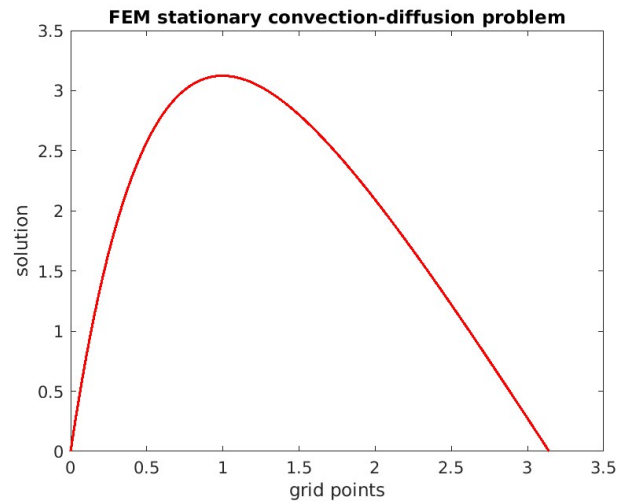
The values of the two matrices and the load vector follow:

$$S = \frac{1}{h} \begin{bmatrix} 2 & -1 & 0 & \dots \\ -1 & 2 & -1 & \dots \\ 0 & -1 & 2 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad C = \frac{1}{2} \begin{bmatrix} 0 & -1 & 0 & \dots \\ 1 & 0 & -1 & \dots \\ 0 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \quad b = h \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Implementation

```
1 function myfem(D, m)
2 h = pi/(m+1);
3 xv=0:h:pi;
4
5 S = zeros(m,m);
6 C = zeros(m,m);
7 b = zeros(m,1);
8 for j=1 : m
9     S(j,j) = 2/h;
10    C(j,j) = 0;
11    b(j) = h;
12    if j < m
13        S(j+1, j) = -1/h;
14        S(j, j+1) = -1/h;
15        C(j+1, j) = -1/2;
16        C(j, j+1) = 1/2;
17    end
18 end
19
20 A = S*D+(1/2)*C ;
21 xi = A\b ;
22 xi = [0; xi; 0];
```

Visualization



After implementing FEM for the stationary convection-diffusion problem, we ran the function with the arguments $D = 0.3$ and $m = 100$, the output is the visualization above.

b)

In this exercise we will study the influence that the value of D has on choosing a nice mesh size h . We will determine the exact solution, choose two values for D , one for each case in the instructions, and plot the previous FEM function along with the exact solution.

The exact solution of 1 follows:

$$u(x) = \frac{2 \left((e^{\pi/2D} - 1) x - \pi e^{x/2D} + \pi \right)}{e^{\pi/2D} - 1}$$

For case 1, we have chosen $D = 1$, and mesh sizes $h = 5$ and $h = 15$.

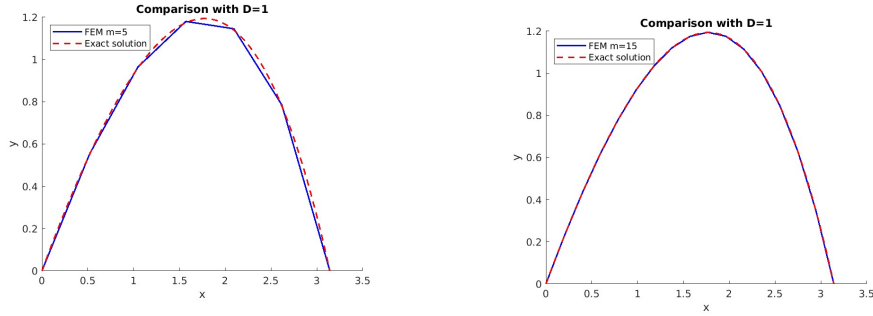
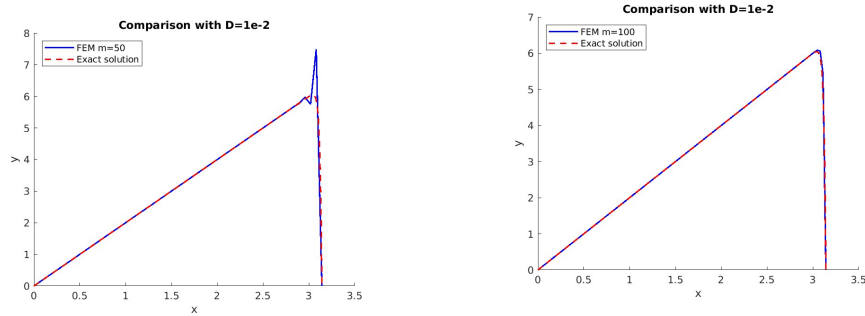


Figure 1: First figure

The figures above illustrate that, when $D \approx 1$, we don't need a lot of discretization points to have a good approximation. This is because the original function does not have any sharp angles.

For case 2, we have chosen $D = 1e - 2$, and mesh sizes $h = 50$ and $h = 100$.



The figures above illustrate that, when $D \ll 1$, we do need a lot of discretization points to have a good approximation. As D grows smaller, the function is follows

the linear function $2x$ until around the point $(3, 6)$, where it takes a sharp turn towards $y = 0$.

2 Task 2

Now, we'll try to apply cG(1) approximation (linear elements) and cG(2) approximation (with quadratic elements).

The boundary problem in consideration now is:

$$-u''(x) = f(x), \quad x \in (0, 1), \quad u(0) = u(1) = 0.$$

We'll test the methods with the function:

$$f(x) = \pi^2 \sin(\pi x),$$

For which we know the exact solution is:

$$u(x) = \sin(\pi x).$$

a)

With piecewise linear elements, for the standard assembly, we'll be using the local stiffness matrix $\frac{1}{h} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ and approximating the load vector via quadrature (here, with midpoint rule).

Implementation

```

1 a = 0; b = 1;
2 m = 8;
3 h = (b - a)/(m+1);
4 x_lin = linspace(a, b, m+2)';
5 N_lin = m;
6
7 I_lin = []; J_lin = []; S_lin = [];
8 b_lin = zeros(N_lin, 1);
9
10 for e = 1:(m+1)
11     nodes = [e, e+1];
12     x_e = x_lin(nodes);
13     A_loc = 1/h * [1 -1; -1 1];
14     % Midpoint quadrature for the load
15     xm = mean(x_e);
16     phi = [(x_e(2)-xm)/h, (xm-x_e(1))/h];
17     b_loc = pi^2*sin(pi*xm)*h * [phi(1); phi(2)];
18
19     for i_local = 1:2
20         I_global = nodes(i_local);
21         if I_global == 1 || I_global == m+2, continue; end
22         row = I_global - 1;

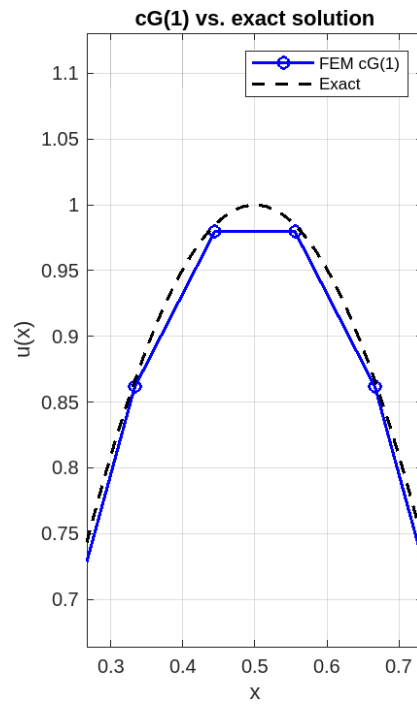
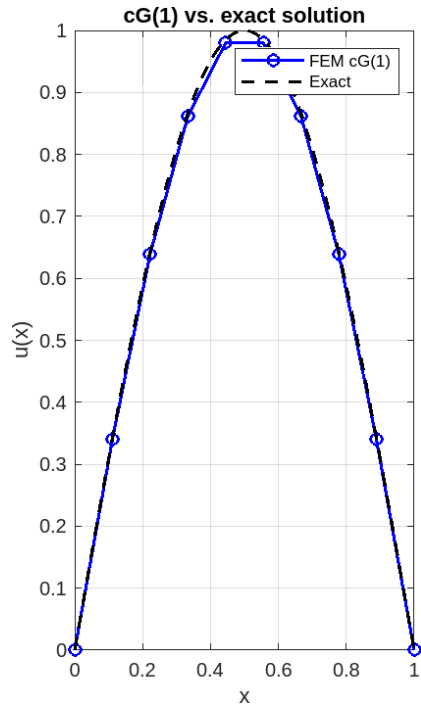
```

```

23     b_lin(row) = b_lin(row) + b_loc(i_local);
24     for j_local = 1:2
25         J_global = nodes(j_local);
26         if J_global == 1 || J_global == m+2, continue; end
27         col = J_global - 1;
28         I_lin(end+1,1) = row;
29         J_lin(end+1,1) = col;
30         S_lin(end+1,1) = A_loc(i_local,j_local);
31     end
32 end
33 end
34
35 A_lin = sparse(I_lin, J_lin, S_lin, N_lin, N_lin);
36 uh_lin = A_lin\b_lin;
37 u_lin = [0; uh_lin; 0];

```

Visualization



b)

With quadratic elements, each element has 3 nodes.

We define the quadratic shape functions on the reference element $[-1, 1]$:

$$\phi_1(r) = \frac{r(r-1)}{2}, \quad \phi_2(r) = 1 - r^2, \quad \phi_3(r) = \frac{r(r+1)}{2}.$$

The mapping to the physical element and the use of a 3-point Gauss-Legendre quadrature allows us to compute both the local stiffness matrix and load vector.

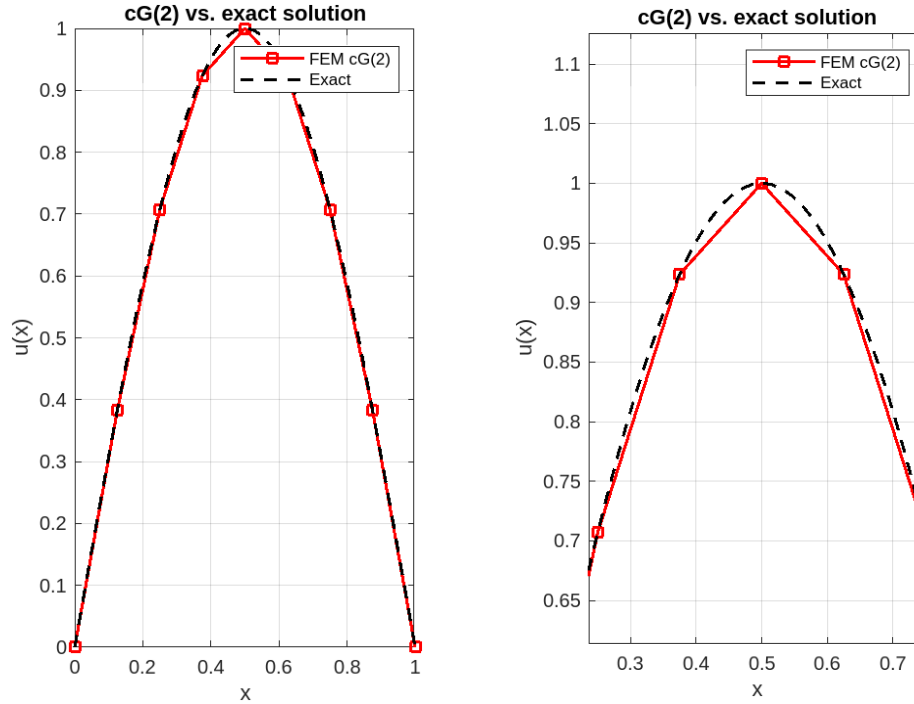
Implementation

```

1 a = 0; b = 1;
2 m_el = 4; % number of quadratic elements
3 x_quad = linspace(a, b, 2*m_el+1)';
4 N_quad = 2*m_el-1; % number of interior unknowns
5 h_quad = (b - a)/m_el;
6
7 % 3-point Gauss-Legendre quadrature on [-1,1]
8 r_gauss = [-sqrt(3/5); 0; sqrt(3/5)];
9 w_gauss = [5/9; 8/9; 5/9];
10
11 I_quad = []; J_quad = []; S_quad = [];
12 b_quad = zeros(N_quad,1);
13
14 for e = 1:m_el
15     nodes = [2*e-1, 2*e, 2*e+1];
16     x_e = x_quad(nodes);
17     A_loc = zeros(3,3);
18     b_loc = zeros(3,1);
19     for q = 1:length(r_gauss)
20         r = r_gauss(q);
21         w = w_gauss(q);
22         % Quadratic shape functions and derivatives on [-1,1]
23         phi = [ r*(r-1)/2; 1-r^2; r*(r+1)/2 ];
24         dphi_dr = [ (2*r-1)/2; -2*r; (2*r+1)/2 ];
25         dphi_dx = (2/h_quad)*dphi_dr;
26         x_val = (x_e(1)+x_e(3))/2 + (h_quad/2)*r;
27         A_loc = A_loc + (dphi_dx*dphi_dx')*(h_quad/2*w);
28         b_loc = b_loc + phi * (pi^2*sin(pi*x_val))*(h_quad/2*w);
29     end
30
31     for i_local = 1:3
32         I_global = nodes(i_local);
33         if I_global == 1 || I_global == 2*m_el+1, continue; end
34         row = I_global - 1;
35         b_quad(row) = b_quad(row) + b_loc(i_local);
36         for j_local = 1:3
37             J_global = nodes(j_local);
38             if J_global == 1 || J_global == 2*m_el+1, continue; end
39             col = J_global - 1;
40             I_quad(end+1,1) = row;
41             J_quad(end+1,1) = col;
42             S_quad(end+1,1) = A_loc(i_local,j_local);
43         end
44     end
45 end
46
47 A_quad = sparse(I_quad, J_quad, S_quad, N_quad, N_quad);
48 uh_quad = A_quad\b_quad;
49 u_quad = [0; uh_quad; 0];

```

Visualization



```

1 xx = linspace(a, b, 100);
2 u_exact = sin(pi*xx);
3
4 figure;
5 subplot(1,2,1)
6 plot(x_lin, u_lin, 'bo-', 'LineWidth',1.5); hold on;
7 plot(xx, u_exact, 'k--', 'LineWidth',1.5);
8 xlabel('x'); ylabel('u(x)');
9 title('cG(1) vs. Exact Solution');
10 legend('FEM cG(1)', 'Exact'); grid on;
11
12 subplot(1,2,2)
13 plot(x_quad, u_quad, 'rs-', 'LineWidth',1.5); hold on;
14 plot(xx, u_exact, 'k--', 'LineWidth',1.5);
15 xlabel('x'); ylabel('u(x)');
16 title('cG(2) vs. Exact Solution');
17 legend('FEM cG(2)', 'Exact'); grid on;

```

For the convergence study of the $cG(1)$ method, a loop over several mesh sizes is implemented, and the error (measured in the maximum norm) is plotted on a log-log scale to verify the expected order of convergence.

Implementation


```

1 a = 0; b = 1;
2 u_exact = @(x) sin(pi*x);
3 err = zeros(1,5);
4
5 for l = 1:5
6     m = 2^l; % number of interior subintervals; total
7             nodes = m+2
8     h = (b - a) / (m + 1);
9     x_nodes = linspace(a, b, m+2)'; % nodes including boundaries
10    N = m;
11
12    A = sparse(N, N);
13    bvec = zeros(N, 1);
14
15    for e = 1:(m+1)
16        localNodes = [e, e+1]; % indices of nodes on current
17                                element
18        x_e = x_nodes(localNodes);
19
20        % Local stiffness matrix for linear elements
21        A_loc = (1/h) * [1, -1; -1, 1];
22
23        phi1 = @(x) (x_e(2) - x) / h;
24        phi2 = @(x) (x - x_e(1)) / h;
25
26        f = @(x) pi^2 * sin(pi*x);
27
28        % Compute local load vector b_local using exact integration
29        b_local = zeros(2,1);
30        b_local(1) = integral(@(x) f(x) .* phi1(x), x_e(1),
31                               x_e(2));
32        b_local(2) = integral(@(x) f(x) .* phi2(x), x_e(1),
33                               x_e(2));
34
35        % Assemble local contributions into the global system
36        for i_local = 1:2
37            global_i = localNodes(i_local);
38
39            if global_i == 1 || global_i == m+2
40                continue;
41            end
42            row = global_i - 1; % shift index because first node
43                               is boundary
44            bvec(row) = bvec(row) + b_local(i_local);
45
46            for j_local = 1:2
47                global_j = localNodes(j_local);
48                if global_j == 1 || global_j == m+2
49                    continue;
50                end
51                col = global_j - 1;
52                A(row, col) = A(row, col) + A_loc(i_local,
53                                                       j_local);
54            end
55        end
56    end
57
58    % Solve the linear system for interior nodes

```

```

52 U_interior = A \ bvec;
53 U = [0; U_interior; 0]; % add boundary values u(0)=0 and
    u(1)=0
54
55 xx = linspace(a, b, 10 * m);
56 UU = interp1(x_nodes, U, xx, 'linear');
57
58 uu = u_exact(xx);
59
60 % Compute the error in the maximum norm
61 err(1) = norm(UU - uu, Inf);
62 end
63
64 h1 = (b - a) ./ ((2.^(1:5) + 1));
65 figure;
66 loglog(h1, err, 'b*-','LineWidth',1.5); hold on;
67 loglog(h1, h1, 'b.-', h1, h1.^2, 'r--','LineWidth',1.5);
68 xlabel('h'); ylabel('Error');
69 legend('Error','O(h)','O(h^2)','Location','southwest');
70 title('Convergence of cG(1) Method');
71 grid on;

```

Visualization

