Assignment 1

Finite Element Methods

R.G.A. Deckers

1. *introduction*

In this report we will discuss the effect on the error bounds of a finite element method that changing the underlying mesh has, using the Poisson Equation:

$$-u'' = f,\ -1 < x < 1, \tag{1.1}$$

$$u(-1) = u(1) = 0. \tag{1.2}$$

Here u is the function of interest, and f the driving force, which is taken to be equal to

$$f(x) = e^{-1000x^2} + 10^{-3} \tag{1.3}$$

Furthermore we will derive an a-posteriori error-bound on $\|u'\|$.

2. *a-posteriori error-estimate*

Using the notation $\|\cdot\|$ to indicate the L^2 norm over the whole domain and $\|\cdot\|_i$ to indicate the L^2 norm over the interval $[x_{i-1}, x_i]$:

$$\begin{aligned}
\|e'\|^2 &= \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} e'(e - \pi e)' \,\mathrm{d}x \\
&= \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} (-e'')(e - \pi e) \,\mathrm{d}x + [e'(e - \pi e)]_{x_i - 1}^{x_i} \\
&= \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} (-e'')(e - \pi e) \,\mathrm{d}x \\
&= \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} (-(u - u_h)'')(e - \pi e) \,\mathrm{d}x \\
&= \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} (f + u_h'')(e - \pi e) \,\mathrm{d}x \\
&\leq \sum_{i=1}^{N} \|f + u_h''\|_i \,\|e - \pi e\|_i \\
&\leq \sum_{i=1}^{N} h_i \,\|f + u_h''\|_i \,C\,\|e'\|_i \\
&\leq C \sqrt{\sum_{i=1}^{N} {h_i}^2 \,\|f + u_h''\|_i^2} \sqrt{\sum_{i=1}^{N} \|e'\|_i^2} \\
&\leq C \sqrt{\sum_{i=1}^{N} {\rho_i}^2}\, \|e'\| \\
\rightarrow \|e'\| &\leq C \sqrt{\sum_{i=1}^{N} \rho_i^2}
\end{aligned} \tag{2.1}$$

It is known that in the 1D case the constant C is unity.[1]

[1] *The Finite Element Method: Theory, Implementation, and Applications* page 7

3. *The Finite Element Implementation*

We modeled the problem using a finite element method and the standard hat-functions, defined by

$$\begin{aligned} \phi_i(x_{i-1} \leq x \leq x_i) &= (x - x_{i-1})/h, \\ \phi_i(x_i \leq x \leq x_{i+1}) &= (x_{i+1} - x)/h, \\ \phi_i(x \setminus [x_{i-1}, x_{i+1}]) &= 0. \end{aligned} \tag{3.1}$$

The boundary conditions were enforced by replacing the first and last row of the stiffness matrix A by their respective unit-vectors and the first and last element of the load vector b by the boundary values.

After we have computed our initial solution, we compute the Laplacian by solving

$$M\chi = -A\xi \tag{3.2}$$

for χ. Here M is the mass matrix, defined by

$$M_{ij} = \int_0^1 \phi_j(x)\phi_i(x)\ \mathrm{d}x \tag{3.3}$$

. We subsequent compute ρ_i using the trapezoidal rule for the interior points, that is:

$$\rho_i = \frac{1}{2}h_i^2\left((f(x_{i-1}) + \chi_{i-1}) + (f(x_i) + \chi_i)\right), \tag{3.4}$$

For the boundray intervals, we use a single pair of χ and f instead.

4. *Adaptive Mesh Refinement*

Having obtained a set of ρ_i (and thus an estimate of the error), we refine our mesh by subdividing each interval i for which

$$\rho_i > \lambda \max_i(\rho_i),\ \lambda \in [0, 1]. \tag{4.1}$$

We have tested $\lambda = 0.05, 0.25, 0.5, 1$, starting from 5 nodes and stopping once we have more than 1000 nodes.

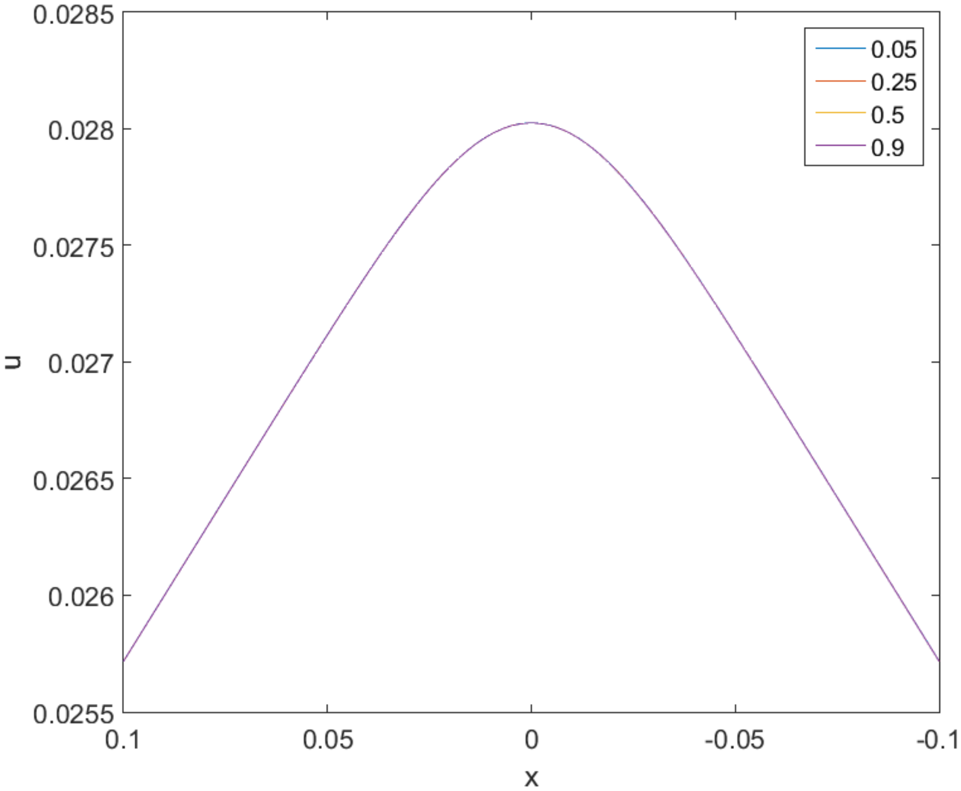
5. *Results*

Our results are contained in figure 1, 2, and 3.

Looking at figure 1 we note that the result becomes less smooth for increasing λ near its inflection point, as the algorithm puts less points there for higher λ. Visually, all results look similar.

Looking at the total error estimate in figure 2 we see that higher values of λ perform better by up to two orders of magnitude for the same amount of points N. It should be note however, that for high values of λ the mesh refinement algorithm takes significantlly longer to reach $N = 1000$ as it adds fewer points per iteration.

Finally, figure 3 shows how the density of points changes with λ. Surprisingly, larger values of λ give a more uniform distribution while $= 0$ will always give a perfectly uniform distribution.



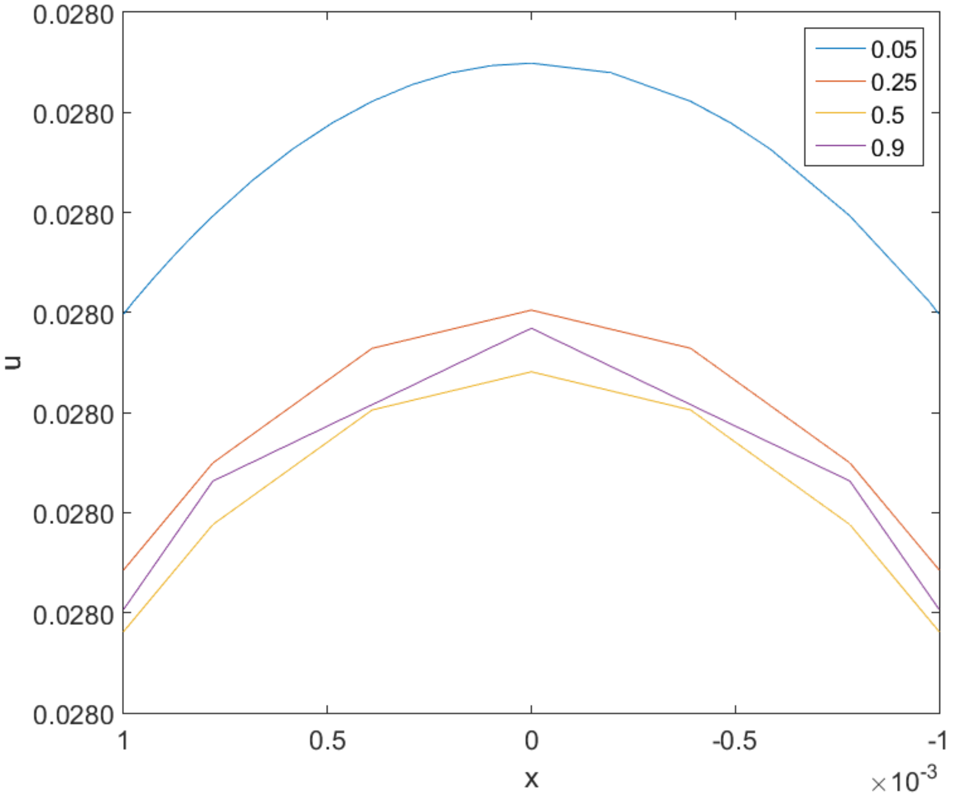


Figure 1: The result of our method for different values of λ.

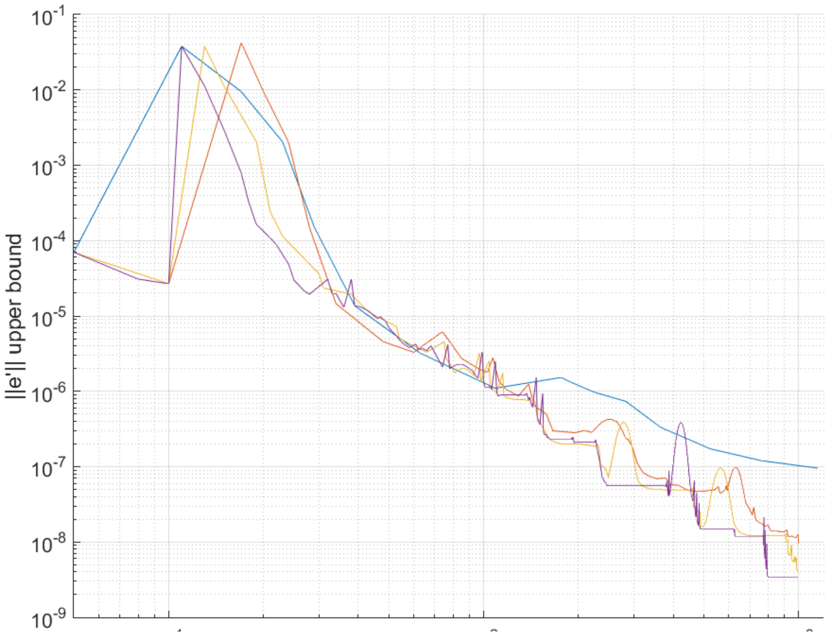


Figure 2: The total sum of ρ_i^2 for different values of λ.

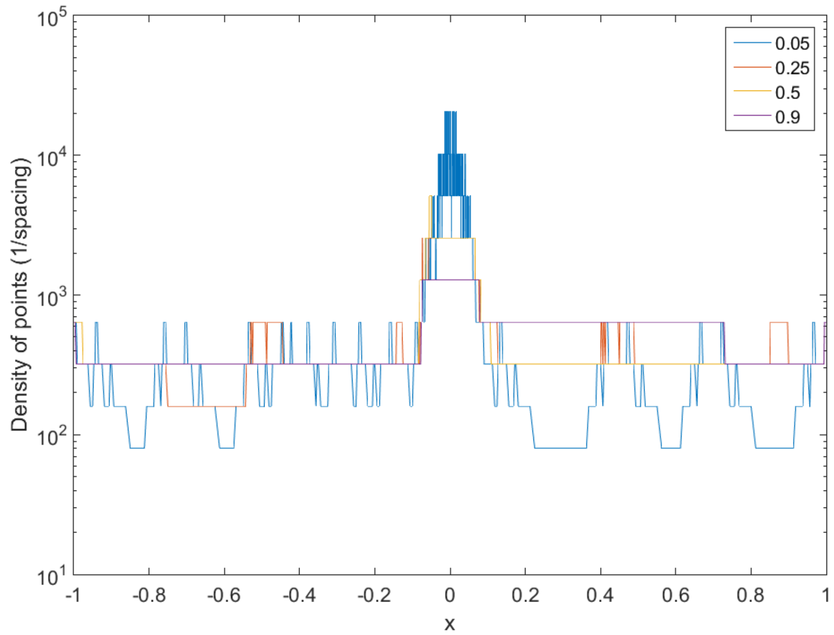


Figure 3: The density of points for different values of λ.

6. *Code*

6a) *Adaptive*

```
function adaptive(N, N_max, lambda)
a = -1; % left end point of interval
b = 1; % right
h = (b-a)/N; % mesh size
x = a:h:b; % node coords
low =  0;
high = 0.;
figure(4)
hold on;
figure(3)
hold on;
figure(2)
hold on;
figure(1)
hold on;
N_array = [];
total_residual_array =  [];

while N < N_max
    N_array = [N_array N];
    B=my_load_vector_assembler(x, low, high, @f);
    M = mass_matrix(x); %last points kept fixed
    A_fixed =stiffness_matrix_fixed(x,low, high);
    xi_fixed = A_fixed\B; % solve system of equations
    reversed = (-A_fixed*xi_fixed);
    lap_fixed = M \(reversed(2:end-1));
    F = arrayfun(@f, x(2:end-1)).';
    delta = abs(F+lap_fixed);
    rho = trapezoidal(x, delta);
    total_residual_array = [total_residual_array sqrt(sum(rho.*rho))];
    threshold = max(rho)*lambda;
    for i = 1:length(rho)
        if rho(i) > threshold
            x = [x (x(i+1)+x(i))/2];
        end
    end
    x = sort(x);
    N = size(x,2)
end
%total_residual_array(end)
%rho
N_array = [N_array N]
B=my_load_vector_assembler(x, low, high, @f);
```

```
M = mass_matrix(x); %last points kept fixed
A_fixed =stiffness_matrix_fixed(x,low, high);
xi_fixed = A_fixed\B; % solve system of equations
reversed = (-A_fixed*xi_fixed);
lap_fixed = M \(reversed(2:end-1));
F = arrayfun(@f, x(2:end-1)).';
delta = abs(F+lap_fixed);
rho = trapezoidal(x, delta);
total_residual_array = [total_residual_array sqrt(sum(rho.*rho))];
figure(1)
plot(x, xi_fixed)
figure(2)
semilogy(x(2:end), [1./diff(x)])
figure(3)
loglog(N_array, total_residual_array)
figure(4)
plot(x(1:end-1), rho)
%x
function y=f(x)
%y=2;
y=exp(-1000*x^2)+10^-3;
%y = pi^2*49*sin(x*pi*7);
%y = x*(x-1);
```

6b) *Mass Matrix*

```
function M = mass_matrix( x )
    N = length(x) - 2;
    diag = zeros(N,1);
    h = zeros(N,1);
    upper = zeros(N-1,1);
    for i = 1:N+1
        h(i) = x(i+1)-x(i);
    end

    %diag(1) = (1/3*(x(2)^3-x(1)^3)+x(2)^2*h(1)-x(2)*(x(2)^2-x(1)^2))/h(1)^2;
    for i = 2:N+1
        diag(i-1) =              (1/3*(x(i+1)^3-x(i)^3)+x(i+1)^2*h(i)-x(i+1)*(x(i+1)^2-x(i)
        diag(i-1) = diag(i-1) + (1/3*(x(i)^3-x(i-1)^3)+x(i-1)^2*h(i-1)-x(i-1)*(x(i)^2-x(
    end
    %diag(N+1) = (1/3*(x(N+1)^3-x(N)^3)+x(N)^2*h(N)-x(N)*(x(N+1)^2-x(N)^2))/h(N)^2;

    for i = 1:N-1
        upper(i) = (-1/3*(x(i+1)^3-x(i)^3)+1/2*(x(i)+x(i+1))*(x(i+1)^2-x(i)^2)-x(i)*x(i+
        %lower(i) = (-1/3*(x(i+1)^3-x(i)^3)+1/2*(x(i)+x(i+1))*(x(i+1)^2-x(i)^2)-x(i)*x(i
    end
    M = gallery('tridiag', upper, diag, upper);
```

```
end
```

6c) *Trapezoidal*

```
function [ rho ] = trapezoidal(x, delta)
    N = length(delta)+1;
    rho = zeros(N,1);
    rho(1) = (x(2)-x(1))^2*delta(1); %first interval, project backwards
    for i = 2:N-1
        h = x(i+1)-x(i);
        rho(i) = 0.5*h^2*(delta(i-1)+delta(i));
    end
    rho(N) = (x(N+1)-x(N))^2*delta(N-1); %last interval, project forwards
end
```