# Project 4: Error estimation and adaptive meshing-using the exact solution as a test

Zhipeng Yu

## 1. Introduction to the problem:

Solve the following boundary value problem, with domain (0, L), analytically, the conditions are given like this.

- Consider the boundary value problem $\frac{d}{dx}\left(E\frac{du}{dx}\right) = f(x)$, $E = 1$, with domain $\Omega = (0, L)$, $L = 1$, and solution $u(x) = \cos(10\pi x^5)$.
- Compute the finite element solution $u^N$ to this problem using linear equal-sized elements. Determine how many elements are needed in order to achieve

$$e^N \overset{\text{def}}{=} \frac{||u - u^N||_{E(\Omega)}}{||u||_{E(\Omega)}} \leq TOL = 0.05\,,$$

$$||u||_{E(\Omega)} \overset{\text{def}}{=} \sqrt{\int_\Omega \frac{du}{dx} E \frac{du}{dx} dx}$$

**What am I going to do** is to get a solution of "u" without direct integral satisfying the minimum error requirement..
**How am I going to do** is to combine some linear simple functions and add them up such that getting as closed as possible to the true solution. At the meantime, divide the L into N element and use 2N functions.
I begin with 20 elements and increase the number if there is any element that not satisfy error requirement.

## 2. Objective

**Goals:** Solve this with the finite element method using linear equal-sized elements. Begin with 20 elements and make the error ($A_I$) less than 0.05 in every element by increasing the number of element.

$$A_I^2 \stackrel{\text{def}}{=} \frac{\frac{1}{h_I}||u-u^N||^2_{E(\Omega_I)}}{\frac{1}{L}||u||^2_{E(\Omega)}}.$$

Here $I$ is the element index, $h_I$ is the length of element $I$, and

$$||u||^2_{E(\Omega_I)} \stackrel{\text{def}}{=} \int_{\Omega_I} \frac{du}{dx} E \frac{du}{dx} dx\,.$$

## 3. My procedure

First we decide to use the Weak Formulation to solve it.

Find $u \in H^1(\Omega)$ $u|_{\Gamma_u} = d$ such that $\forall \nu \in H^1(\Omega), \nu|_{\Gamma_u} = 0$

$$\int_\Omega \frac{d\nu}{dx} E \frac{du}{dx}\, dx = \int_\Omega f\nu\, dx + t\nu|_{\Gamma_t}.$$

We approximate "u" by:

$$u^h(x) = \sum_{j=1}^{N} a_j \phi_j(x).$$

If we choose "v" with the same approximation functions, but a different linear combination, we get "v" like this:

$$\nu^h(x) = \sum_{i=1}^{N} b_i \phi_i(x),$$

Since the "v" are arbitrary (formulation definition), the "bi" are arbitrary, therefore

$\sum_{i=1}^N b_i \left(\sum_{j=1}^N K_{ij} a_j - R_i\right) = 0 \Rightarrow [K]\{a\} = \{R\},$

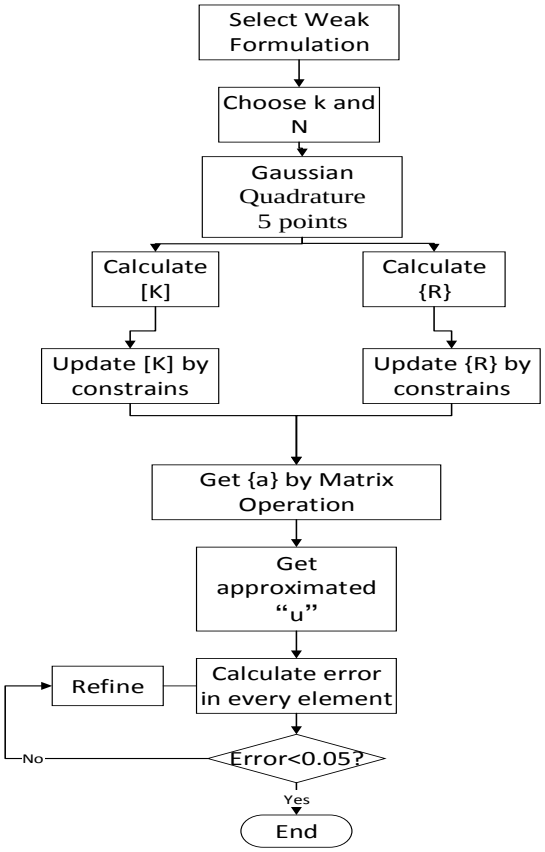
$K_{ij} \stackrel{\text{def}}{=} \int_\Omega \frac{d\phi_i}{dx} E \frac{d\phi_j}{dx}\, dx$ and

$R_i \stackrel{\text{def}}{=} \int_\Omega \phi_i f\, dx + \phi_i t|_{\Gamma_t},$

According to that, we will use some mathematical trick to simplify the

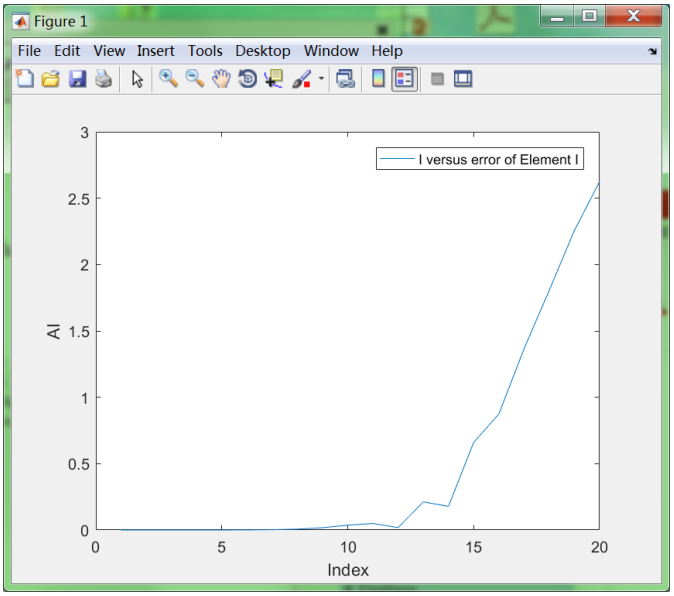
integral calculation such as Gaussian quadrature (ϕi represents the simple "function" that we build by ourselves). Then we will get [K] and {R} and add constrains to them. And then use matrix operations to get {a} directly. Next, calculate the error in every element and split the element if it doesn't satisfy the requirement until the error less than 0.05.

Refine the mesh (by dividing elements into two) until AI < TOLE for all I.



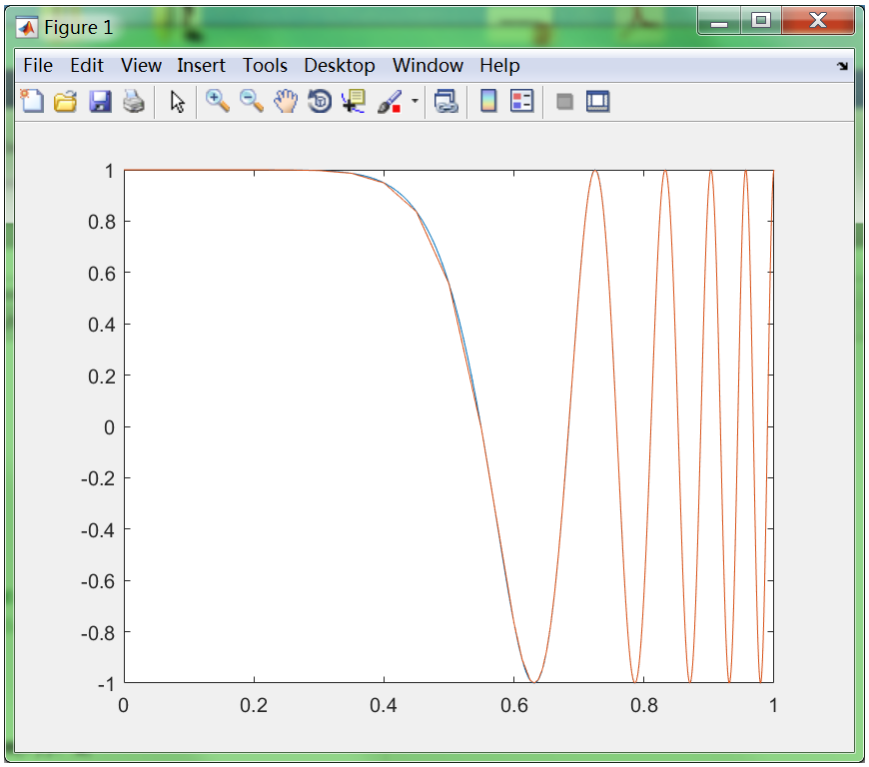

## 4. Findings

1) At beginning when N=20, error in every element.




2) Totally it needs 404 elements to make error less than 0.05 in every element.

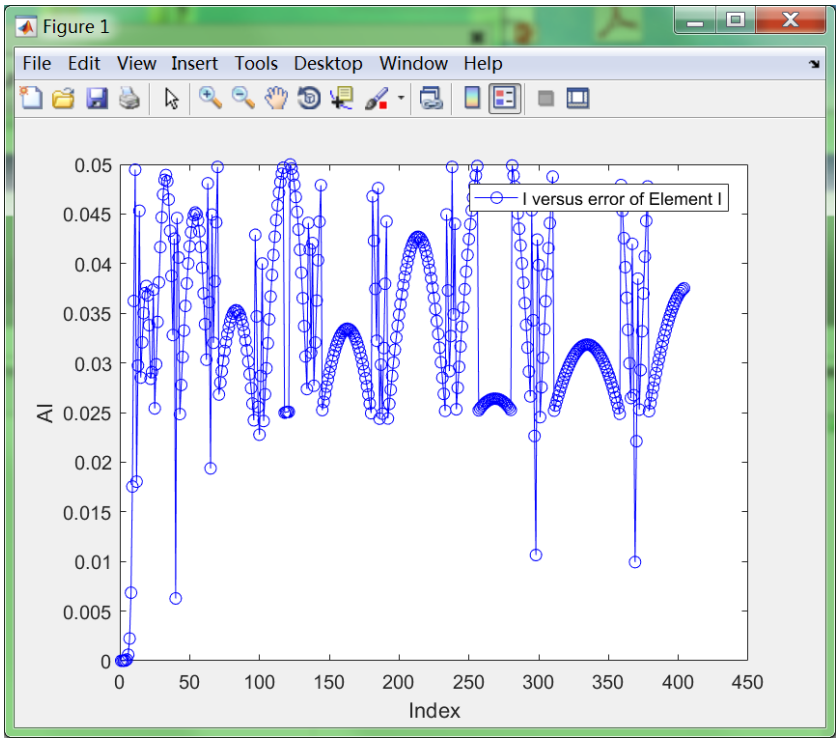
3) Exact U and my solution for 404 elements.




4) The final number of elements that fall into each of the initial 20 elements.

| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 6 | 5 | 14 | 22 | 40 | 52 | 93 | 159 |

5) XI versus AI for the final solution (XI = position of node I).




## 5. Observations and discussion

According to plot,

a) I tried two methods to get uN based on {a}. It was weird that they had distinct performance.

Figure 1 : plot (Xi , a) directly.

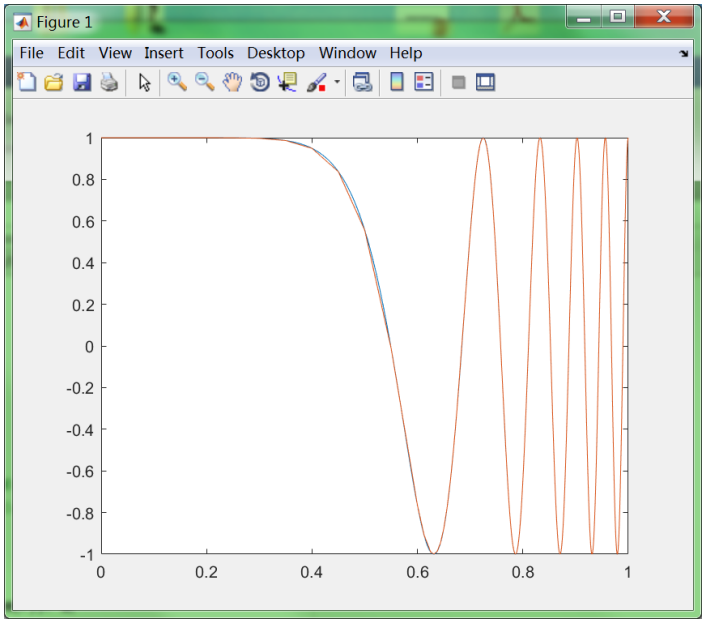
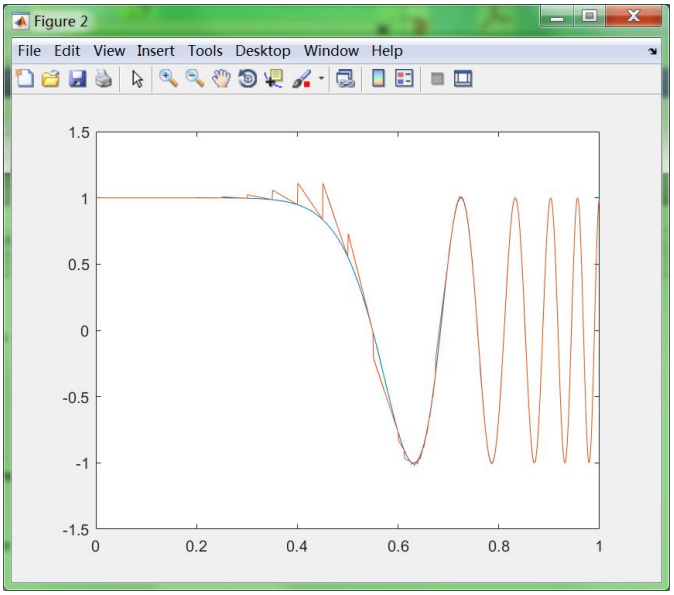


Figure 2: Use formulation in the notes to get uN.

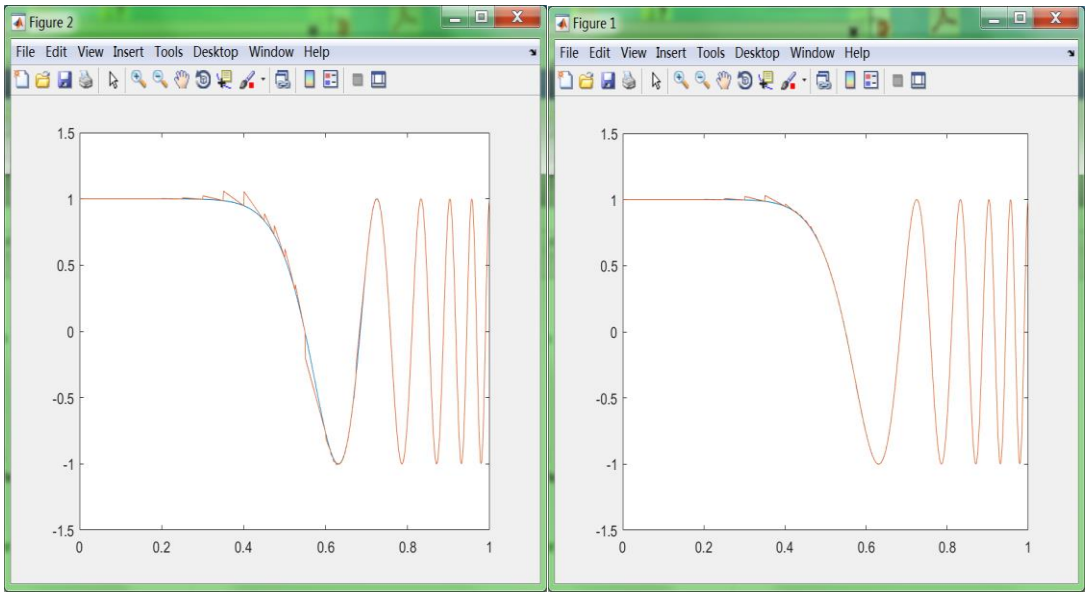
$$u^h(x) = \sum_{j=1}^{N} a_j \phi_j(x). \qquad \hat{\phi}_1 = \frac{1}{2}(1-\zeta) \quad \text{and} \quad \hat{\phi}_2 = \frac{1}{2}(1+\zeta).$$

```
he=[0,he];
theta1=@(x1)0.5*(1-(2*x1-sum(he(1:find(NodeX>=x1,1)))-
sum(he(1:find(NodeX>=x1,1)+1)))/he(find(NodeX>=x1,1)+1));
theta2=@(x1)0.5*(1+(2*x1-sum(he(1:find(NodeX>=x1,1)))-
sum(he(1:find(NodeX>=x1,1)+1)))/he(find(NodeX>=x1,1)+1));
funuN=@(x1)A(find(NodeX>=x1,1))*theta1(x1)+A(find(NodeX>=x1,1)+1)
*theta2(x1);
uN=[];
for i=1:size(x,2)-1
    uN=[uN,funuN(x(i))];
end
plot(x(1:5000),uN);
```

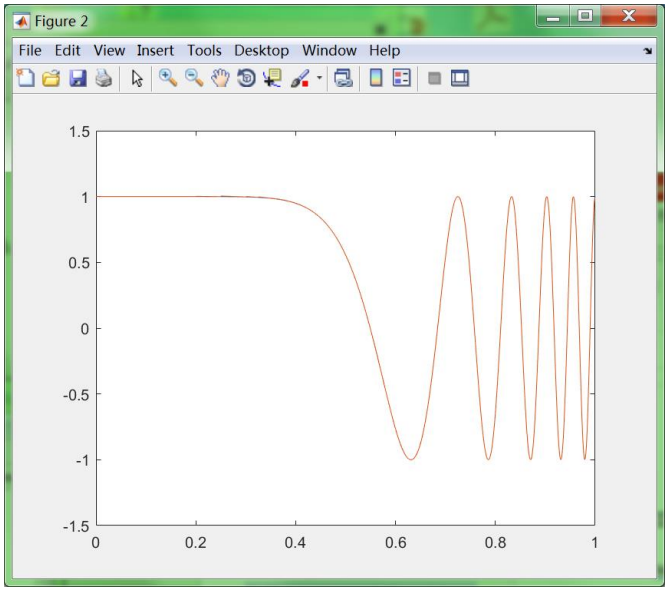



b) As we can see, actually the second method which got uN from definition showed some noise among x=(0.2:0.5) but first method didn't. I think the second method is correct because the first method only show the value of uN on the position of Node I and ignore the value between Node I and Node II. And actually the value between Node I and Node II is not such smooth.

c) When I change to tolerance to 0.02, 0.01. It was more accurate but not good enough like below.




Then I further changed the tol to 0.002. It looks good but took about 52 seconds to run (need 9820 elements). Time-Consuming. Show like below.




| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| AutoMesh | 1 | 52.224 s | 10.918 s | |

d) And I also found that when I dropped tol from 0.05 to 0.001, the element number in noisy area only changed from 1 to 5-10. But the number of

element in sin shape(from x=0.6 to x=1.0) increased a lot which consuming most of time.

e) So I wonder whether the criterion of splitting element is unfair to the smooth shape. And what makes it jagged?

# Appendix

**Mesh:**

```
while max(AI)>0.05
    numMesh=find(AI>0.05);
    nIndex=(nodeIndex(numMesh)+nodeIndex(numMesh+1))/2;
    nodeIndex=[nodeIndex,nIndex];
    nodeIndex=sort(nodeIndex);
    N=size(nodeIndex,2)-1;
    he=nodeIndex(2:N+1)-nodeIndex(1:N);
    recalculate
    for i=1:N
        errorfun=@(x)E*(du(x)-(A(i+1)-A(i))/he(i)).^2;
        AI(i)=sqrt(L/he(i)*integral(errorfun,nodeIndex(i),nodeIndex(i+1)))/uE;
    end
end
```

**plot:**

```
he=[0,he];
theta1=@(x1)0.5*(1-(2*x1-sum(he(1:find(nodeIndex>=x1,1)))-
sum(he(1:find(nodeIndex>=x1,1)+1)))/he(find(nodeIndex>=x1,1)+1));
theta2=@(x1)0.5*(1+(2*x1-sum(he(1:find(nodeIndex>=x1,1)))-
sum(he(1:find(nodeIndex>=x1,1)+1)))/he(find(nodeIndex>=x1,1)+1));
funuN=@(x1)A(find(nodeIndex>=x1,1))*theta1(x1)+A(find(nodeIndex>=x1,1)+1)*th
eta2(x1);
uN=[];
for i=1:size(x,2)-1
    uN=[uN,funuN(x(i))];
end
plot(x(1:5000),uN);
num=[];
for i=1:20
    num=[num ,size(find(nodeIndex<0.05*i),2)];
end
num(2:20)=num(2:20)-num(1:19);
```