Homework #1

Hojjat Anvari (734003280)

Prof. Strouboulis

MEMA 646

Spring 2024

Question 1: Compute the exact solution and finite element approximations of it for two, four, and six elements of the boundary-value problem.

$$-u''(x) = x \quad , \quad 0 < x < 1$$
$$u(0) = 0 \quad , \quad u(1) = 0$$

a) Compute the error measures $||e||_E$ and $||e||_0$ for each mesh and plot *log* $||e||$ versus *log* h. What are the rates of convergence for each choice of the norm?
b) Check the pointwise error in the solution at $x = 1/8$ for each mesh. What is the rate of convergence there?
c) Check the pointwise error at the central node $x = 1/2$. What remarkable result do you find?

Exact Solution:

First, we find the exact solution to the boundary value problem by taking the integral of both sides of the equation two times:

$$u''(x) = -x \rightarrow u'(x) = -\frac{x^2}{2} + C_1$$

$$u(x) = -\frac{x^3}{6} + C_1 x + C_2$$

$$u(0) = 0 \rightarrow 0 = -\frac{0^3}{6} + C_1.0 + C_2 \rightarrow C_2 = 0$$

$$u(1) = 0 \rightarrow 0 = -\frac{1^3}{6} + C_1.1 \rightarrow C_1 = \frac{1}{6}$$

$$u_{exact}(x) = \frac{1}{6}\left(x - x^3\right) \qquad \text{Eq. 1}$$

Finite Element Approximation:

We first write the weak form of the equation:

$$\int_0^1 (-u'')\, v\,dx = \int_0^1 xv\,dx \qquad \text{Eq. 2}$$

The function v (the weight function or test function) is any arbitrary function of x that is well-behaved for the integrals to exist. For the test function $v(x)$ we have $v(0) = 0$, $v(1) = 0$ in H. Therefore, we have from Eq. 2:

$$\int_0^1 (-u'' - x)\, v\,dx = 0 \qquad for\ all\ v \in H$$
$$u(0) = 0 \quad , \quad u(1) = 0 \qquad \text{Eq. 3}$$

Here *u(x)* belongs to the class of trial function while the class of test function is different. To make sure that both *u(x)* and *v(x)* are in the same class, we perform integration by parts based on the chain role of differentiation:

$$(u'v)' = u''v + u'v' \quad \Leftrightarrow \quad u''v = (u'v)' - u'v'$$
$$\int_0^1 -u''v\,dx = \int_0^1 u'v'dx - u'v\Big|_0^1 \qquad \text{Eq. 4}$$

Regarding Eq. 4, *v(x)* must be differentiable to ensure that the integral on the right side of this equation is available. In addition, as *v(0) = 0* and *v(1) = 0*, the second term is zero. As a result, the Eq. 4 will be in this form:

$$\int_0^1 (u'v' - xv)\,dx = 0 \qquad , \qquad for\ all\ v \in H_0^1 \qquad \text{Eq. 5}$$

The basic requirement is that each $v \in H_0^1$ be representable as a linear combination of such functions of the type:

$$v(x) = \sum_{i=1}^{\infty} \beta_i \phi_i(x) \qquad \text{Eq. 6}$$

If we take only a finite number *N* of terms in this series, then we will obtain only an approximation v_N of *v*:

$$v_N(x) = \sum_{i=1}^{N} \beta_i \phi_i(x) \qquad \text{Eq. 7}$$

Galerkin's method consists of seeking an approximate solution to Eq. 5 in a finite-dimensional subspace $H_0^{(N)}$ of the space H_0^1 of admissible functions rather than in the whole space H_0^1. Thus, instead of tackling the infinite-dimensional problem Eq. 5, we seek an approximate solution u_N in $H_0^{(N)}$ of the form:

$$u_N(x) = \sum_{i=1}^{N} \alpha_i \phi_i(x) \qquad \text{Eq. 8}$$

The variational statement of the approximate problem is this:

Find $u_N \in H_0^{(N)}$ such that:

$$\int_0^1 \left(u'_N v'_N\right) dx = \int_0^1 x v_N\,dx \qquad , \qquad for\ all\ v_N \in H_0^{(N)} \qquad \text{Eq. 9}$$

To determine the specific values, α_i, we introduce Eq. 7 and Eq. 8 into Eq. 9 to obtain the condition:

$$\int_0^1 \left\{ \frac{d}{dx}\left[\sum_{i=1}^{N} \beta_i \phi_i(x)\right] \frac{d}{dx}\left[\sum_{j=1}^{N} \alpha_j \phi_j(x)\right] - x \sum_{i=1}^{N} \beta_i \phi_i(x) \right\} dx = 0,$$
$$for\ all\ \beta_i,\ i = 1, 2, \ldots, N \qquad \text{Eq. 10}$$

Expanding Eq. 10 and factoring the coefficients β_i gives:

$$\sum_{i=1}^{N} \beta_i \left(\sum_{j=1}^{N} \left\{ \int_0^1 \left[\phi'_i(x)\, \phi'_j(x) \right] dx \right\} \alpha_j - \int_0^1 x \phi_i(x)\, dx \right) = 0,$$
$$for\ all\ \beta_i,\ i = 1, 2, \ldots, N \qquad \text{Eq. 11}$$

The structure of Eq. 11 can be written more compactly:

$$\sum_{i=1}^{N} \beta_i \left(\sum_{j=1}^{N} K_{ij} \alpha_j - F_i \right) = 0, \quad for\ all\ \beta_i,\ i = 1, 2, \ldots, N \qquad \text{Eq. 12}$$

Where:

$$K_{ij} = \int_0^1 \left[\phi'_i(x)\, \phi'_j(x) \right] dx \quad , \quad F_i = \int_0^1 x \phi_i dx$$
$$i,j = 1, 2, \ldots, N \qquad \text{Eq. 13}$$

Therefore, we have a system of *N* linear equations in the *N* unknown coefficients α_j:

$$\sum_{j=1}^{N} K_{ij} \alpha_j = F_i \qquad , \qquad i = 1, 2, \ldots, N \qquad \text{Eq. 14}$$

Since the functions ϕ_i have been chosen to be independent, Eq. 14 will be independent, and therefore the stiffness matrix *K* will be invertible. It follows that the coefficients α_j are uniquely determined by Eq. 14 and are of the form:

$$\alpha_j = \sum_{i=1}^{N} \left(K^{-1} \right)_{ji} F_i \qquad \text{Eq. 15}$$

If the coordinates of the nodes are denoted x_i ($i = 0, 1, 2, \ldots, N$), then the basis functions shown for $i = I, 2, \ldots, N$ are given by:

$$\phi_i(x) = \begin{cases} \dfrac{\left(x - x_{i-1}\right)}{h_i} & for\ x_{i-1} \le x \le x_i \\ \dfrac{\left(x_{i+1} - x\right)}{h_{i+1}} & for\ x_i \le x \le x_{i+1} \\ 0 & for\ x \le x_{i-1}\ and\ x \ge x_{i+1} \end{cases} \qquad \text{Eq. 16}$$

Where $h_i = x_i - x_{i-1}$ is the length of element Ω_i. Their first derivates are:

$$\phi'_i(x) = \begin{cases} \dfrac{1}{h_i} & for\ x_{i-1} < x < x_i \\ \dfrac{-1}{h_{i+1}} & for\ x_i < x < x_{i+1} \\ 0 & for\ x < x_{i-1}\ \ and\ \ x > x_{i+1} \end{cases} \qquad \text{Eq. 17}$$

Following Eq. 14, we define the global stiffness matrix and then the element stiffness matrix for finite element Ω_e:

$$K_{ij} = \sum_{e=1}^{N} K_{ij}^e \quad \rightarrow \quad K_{ij}^e = \int_{\Omega_e} \left(\phi'_i \phi'_j \right) dx \qquad \text{Eq. 18}$$

Similarly, for the global load vector and the element load vector, we have:

$$F_i = \sum_{e=1}^{N} F_i^e \quad \rightarrow \quad F_i^e = \int_{\Omega_e} x \phi_i dx \qquad \text{Eq. 19}$$

Shape functions ψ_A^e and ψ_B^e that are shown in Fig. 1 defined for element Ω_e. Since these are simply parts of ϕ_A and ϕ_B, these shape functions are given in terms of the local coordinate ξ by:

$$\psi_A^e(\xi) = 1 - \frac{\xi}{h} \quad , \quad \psi_B^e(\xi) = \frac{\xi}{h} \qquad \text{Eq. 20}$$

And their first derivatives are:

$$\psi_A^{e'}(\xi) = -\frac{1}{h} \quad , \quad \psi_B^{e'}(\xi) = \frac{1}{h} \qquad \text{Eq. 21}$$

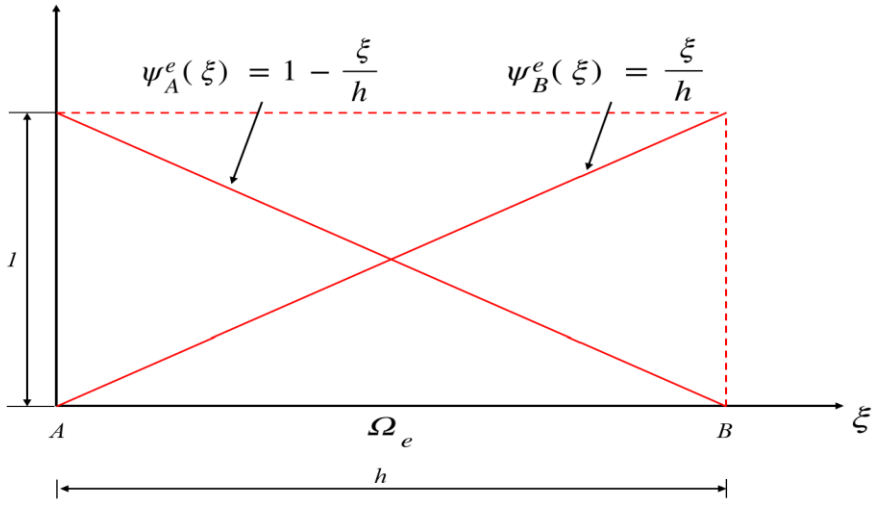

Figure 1: *Element Ω_e with linear shape functions*

According to Eq. 18, the element matrix coefficients for our generic element Ω_e, are:

$$K^e_{AA} = \int_0^h \left(\psi^{e'}_A(\xi)\right)^2 d\xi = \int_0^h \frac{1}{h^2} d\xi = \frac{1}{h}$$

$$K^e_{AB} = K^e_{BA} = \int_0^h \left(\psi^{e'}_A(\xi)\,\psi^{e'}_B(\xi)\right) d\xi = \int_0^h \left(-\frac{1}{h}\right)\frac{1}{h} d\xi = -\frac{1}{h}$$

$$K^e_{BB} = \int_0^h \left(\psi^{e'}_B(\xi)\right)^2 d\xi = \int_0^h \frac{1}{h^2} d\xi = \frac{1}{h} \qquad \text{Eq. 22}$$

Similarly, the components of the element load vector are:

$$F^e_A = \int_0^h \left(x_A + \xi\right)\left(1 - \frac{\xi}{h}\right) d\xi = \frac{h}{6}\left(2x_A + x_B\right)$$

$$F^e_B = \int_0^h \left(x_A + \xi\right)\left(\frac{\xi}{h}\right) d\xi = \frac{h}{6}\left(x_A + 2x_B\right) \qquad \text{Eq. 23}$$

These quantities are entries in local element stiffness and load vectors k^e and f^e for the generic element Ω_e:

$$k^e = \begin{bmatrix} \frac{1}{h} & -\frac{1}{h} \\ -\frac{1}{h} & \frac{1}{h} \end{bmatrix} \quad , \quad f^e = \frac{h}{6}\begin{bmatrix} 2x_A + x_B \\ x_A + 2x_B \end{bmatrix} \qquad \text{Eq. 24}$$

When we have the local element stiffness matrix and local load vector from Eq. 24, we can compute the global matrices from Eq. 18 and Eq. 19 respectively. Upon solving $\sum_{j=1}^{N} \left(K^{-1}\right)_{ji} F_i$, the finite element approximation of the equation is obtained.

Using the global load vector and the finite element approximation matrix, the value of energy in the finite element solution can be calculated:

$$U_h = \frac{1}{2}\int_0^1 \left[\left(u'_h\right)^2\right] dx = \frac{1}{2} u^T \int_0^1 \left(\phi'^T \phi'\right)\, dx\, u$$

$$= \frac{1}{2} u^T K u = \frac{1}{2} u^T F \qquad \text{Eq. 25}$$

We programmed all these steps in Python programming language for 2, 4, and 6 elements, and Tab. 1 shows the results. The corresponding plot is shown in Fig. 2.

	n = 2	n = 4	n = 6

K_g	`[4.]`	`[ 8. -4.  0.]`
		`[-4.  8. -4.]`
		`[ 0. -4.  8.]`
			`[12. -6.  0.  0.  0.]`
			`[-6. 12. -6.  0.  0.]`
			`[ 0. -6. 12. -6.  0.]`
			`[ 0.  0. -6. 12. -6.]`
			`[ 0.  0.  0. -6. 12.]`

F_g	`[0.25]`	`[0.0625]`
		`[0.125 ]`
		`[0.1875]`
			`[0.02777778]`
			`[0.05555556]`
			`[0.08333333]`
			`[0.11111111]`
			`[0.13888889]`

u_h	`[0.0625]`	`[0.0390625]`
		`[0.0625   ]`
		`[0.0546875]`
			`[0.02700617]`
			`[0.04938272]`
			`[0.0625   ]`
			`[0.0617284 ]`
			`[0.04243827]`

U_h	`[0.0078125]`	`[0.01025391]`	`[0.01072745]`

Table 1: *Results from the Python code for* $-u''(x) = x \;,\; 0 < x < 1 \quad | \quad u(0) = 0,\; u(1) = 0$

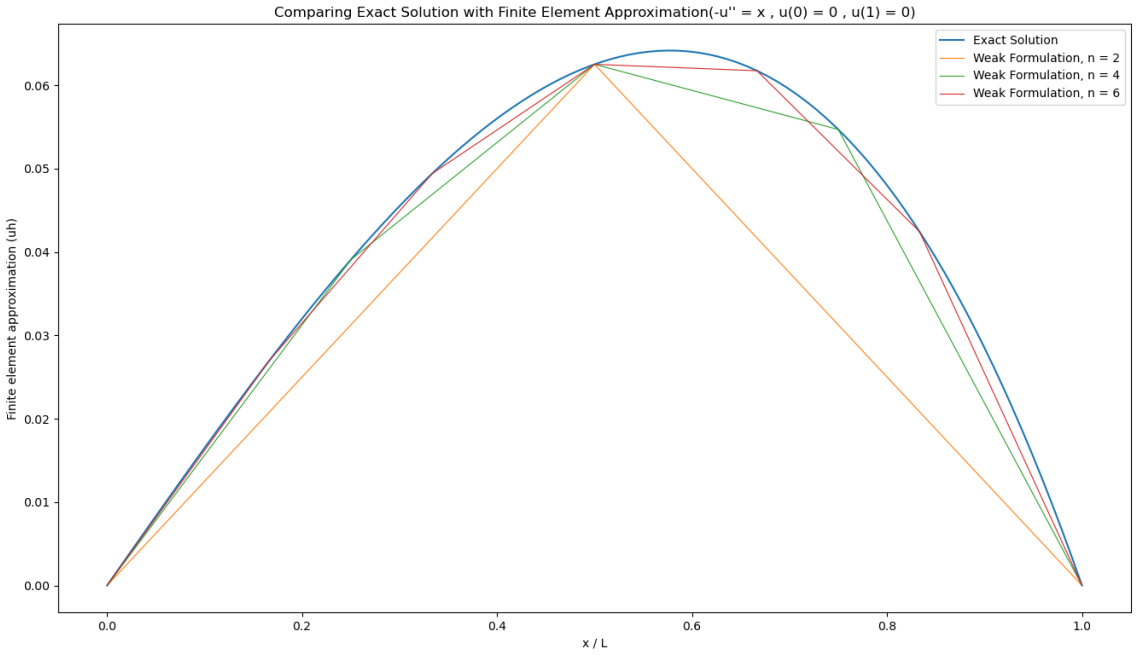

Figure 2: *Exact solution and the weak formulation results for* $-u''(x) = x \;,\; 0 < x < 1 \quad | \quad u(0) = 0,\; u(1) = 0$

The exact value of the differential equation at $x = ½$ is:

$$u_{exact}\left(\frac{1}{2}\right) = \frac{1}{6}\left(\frac{1}{2} - \left(\frac{1}{2}\right)^3\right) = \frac{1}{16} = 0.0626 \qquad \text{Eq. 26}$$

As we can see from the exact value in Eq. 26 and comparing that with the u_h value of the midpoint in Tab. 1, the exact solution and the finite element solution for different numbers of elements are the same.

Finally, we plot the *log(||error||)* vs *-log(h)* in Fig. 3:

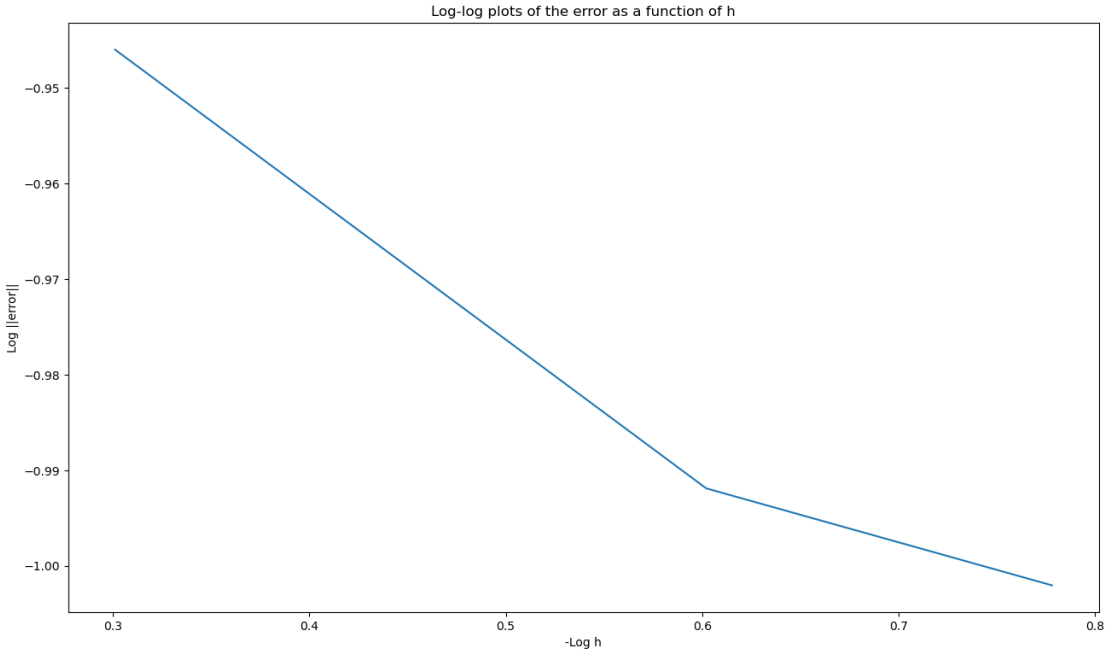

Figure 3: *Logarithmic plot of error as a function of the element length (h)*

$$-u''(x) = x \;,\; 0 < x < 1 \quad | \quad u(0) = 0,\; u(1) = 0$$

Question 2: Replace the differential equation in question 1 by:

$$-u''(x) + u(x) = x \;,\; 0 < x < 1$$
$$u(0) = 0 \;,\; u(1) = 0$$

Exact Solution:

The exact solution to this differential equation is:

$$u_{exact}(x) = x - \frac{\sinh(x)}{\sinh(1)} \qquad \text{Eq. 27}$$

Finite Element Approximation:

The overall steps are similar to the first question but there are some differences in generating the stiffness matrix. The variational statement of the approximate problem is this:

Find $u_N \in H_0^{(N)}$ such that:

$$\int_0^1 \left(u'_N v'_N + u_N v_N \right) dx = \int_0^1 x v_N \, dx \qquad , \qquad for\ all\ v_N \in H_0^{(N)} \qquad \text{Eq. 28}$$

Therefore, Eq. 10, Eq. 11, and Eq. 13 need to be modified in the following way:

$$\int_0^1 \left\{ \frac{d}{dx}\left[\sum_{i=1}^N \beta_i \phi_i(x)\right] \frac{d}{dx}\left[\sum_{j=1}^N \alpha_j \phi_j(x)\right] + \left[\sum_{i=1}^N \beta_i \phi_i(x)\right]\left[\sum_{j=1}^N \alpha_j \phi_j(x)\right] - x \sum_{i=1}^N \beta_i \phi_i(x) \right\} dx = 0,$$
$$for\ all\ \beta_i,\ i = 1, 2, \ldots, N \qquad \text{Eq. 29}$$

.

$$\sum_{i=1}^N \beta_i \left(\sum_{j=1}^N \left\{ \int_0^1 \left[\phi'_i(x) \phi'_j(x) + \phi_i(x) \phi_j(x) \right] dx \right\} \alpha_j - \int_0^1 x \phi_i(x) \, dx \right) = 0,$$
$$for\ all\ \beta_i,\ i = 1, 2, \ldots, N \qquad \text{Eq. 30}$$

.

$$K_{ij} = \int_0^1 \left[\phi'_i(x) \phi'_j(x) + \phi_i(x) \phi_j(x) \right] dx \quad , \quad F_i = \int_0^1 x \phi_i dx$$
$$i,j = 1, 2, \ldots, N \qquad \text{Eq. 31}$$

Applying similar basis functions, we modify the global and element stiffness matrices in Eq. 18:

$$K_{ij} = \sum_{e=1}^N K_{ij}^e \quad \rightarrow \quad K_{ij}^e = \int_{\Omega_e} \left(\phi'_i \phi'_j + \phi_i \phi_j \right) dx \qquad \text{Eq. 32}$$

Following Eq. 32, the element matrix coefficients for our generic element Ω_e, are:

$$K_{AA}^e = \int_0^h \{ [\psi_A^{e'}(\xi)]^2 + [\psi_A^e(\xi)]^2 \} d\xi = \int_0^h \left[\frac{1}{h^2} + \left(1 - \frac{\xi}{h}\right)^2 \right] d\xi = \frac{1}{h} + \frac{h}{3}$$
$$K_{AB}^e = K_{BA}^e = \int_0^h \left[\psi_A^{e'}(\xi) \psi_B^{e'}(\xi) + \psi_A^e(\xi) \psi_B^e(\xi) \right] d\xi = \int_0^h \left[\left(-\frac{1}{h}\right)\frac{1}{h} + \left(1 - \frac{\xi}{h}\right)\frac{\xi}{h} \right] d\xi = -\frac{1}{h} + \frac{h}{6}$$
$$K_{BB}^e = \int_0^h \{ [\psi_B^{e'}(\xi)]^2 + [\psi_B^e(\xi)]^2 \} d\xi = \frac{1}{h} + \frac{h}{3} \qquad \text{Eq. 33}$$

The components of the element load vector remain unchanged, but the local element stiffness matrix is modified based on Eq. 33:

$$k^e = \begin{bmatrix} \frac{1}{h} + \frac{h}{3} & -\frac{1}{h} + \frac{h}{6} \\ -\frac{1}{h} + \frac{h}{6} & \frac{1}{h} + \frac{h}{3} \end{bmatrix} \qquad \text{Eq. 34}$$

Considering the new local stiffness matrix for the differential equation in Python codes, here are the results for 2, 4, and 6 elements:

	n = 2	n = 4	n = 6
K_g	[4.33333333]	[8.16666667 -3.95833333 0.] [-3.95833333 8.16666667 -3.95833333] [0. -3.95833333 8.16666667]	[12.11111111 -5.97222222 0. 0. 0.] [-5.97222222 12.11111111 -5.97222222 0. 0.] [0. -5.97222222 12.11111111 -5.97222222 0.] [0. 0. -5.97222222 12.11111111 -5.97222222] [0. 0. 0. -5.97222222 12.11111111]
F_g	[0.25]	[0.0625] [0.125] [0.1875]	[0.02777778] [0.05555556] [0.08333333] [0.11111111] [0.13888889]
u_h	[0.05769231]	[0.0352125] [0.05685947] [0.05051862]	[0.02423976] [0.04450482] [0.05670955] [0.05654339] [0.03935052]
U_h	[0.00721154]	[0.00939023]	[0.00980978]

Table 2: *Results from the Python code for* $-u''(x) + u(x) = x\ ,\ 0 < x < 1 \quad | \quad u(0) = 0,\ \ u(1) = 0$

Similarly, we plotted the exact solution and finite element approximations in Fig. 4. The main important point is that the exact value and the finite element value at the midpoint (x = ½) are different:

$$u_{exact}\left(\frac{1}{2}\right) = \frac{1}{2} - \frac{\sinh\left(\frac{1}{2}\right)}{\sinh(1)} = 0.05659 \qquad \text{Eq. 35}$$

The exact value is 0.05659 while Tab. 2 shows that the result in finite element solution is 0.05769, 0.05686, and 0.05671 for 2, 4, and 6 elements respectively. If we calculate the error of the finite element solution compared to the exact solution for different numbers of elements, we find that the error will decrease by increasing the number of elements. The *log(||error||)* vs *-log(h)* plot in Fig. 5 shows this claim more clearly.

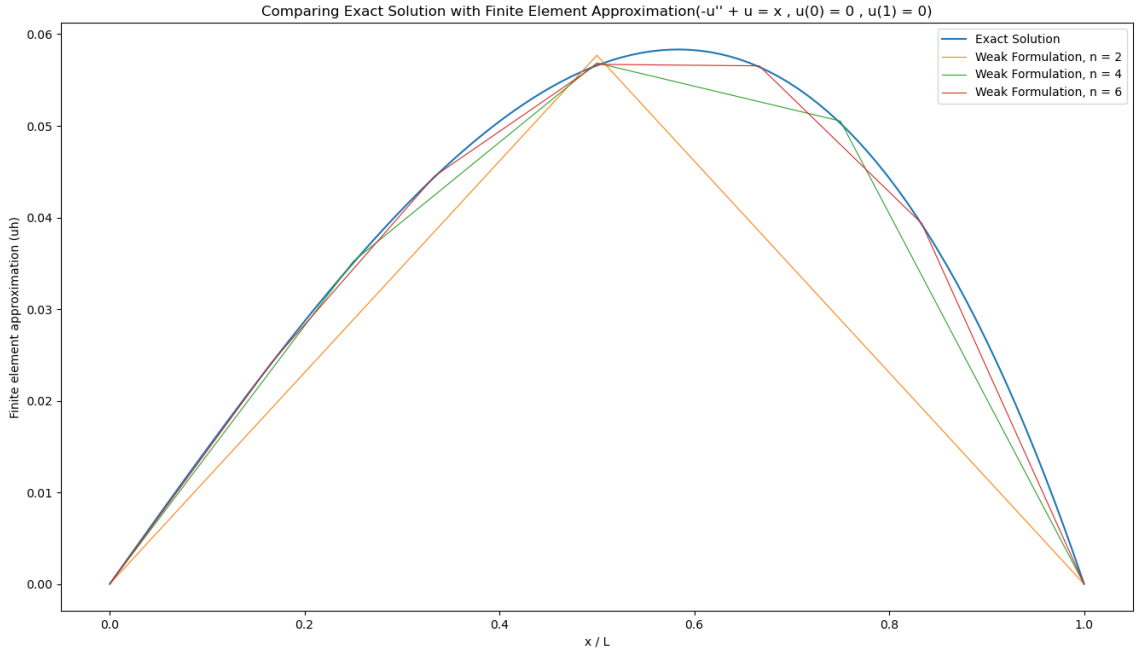

Figure 4: *Exact solution and the weak formulation results for*

$$-u''(x) + u(x) = x \ , \ 0 < x < 1 \quad | \quad u(0) = 0, \ \ u(1) = 0$$

.

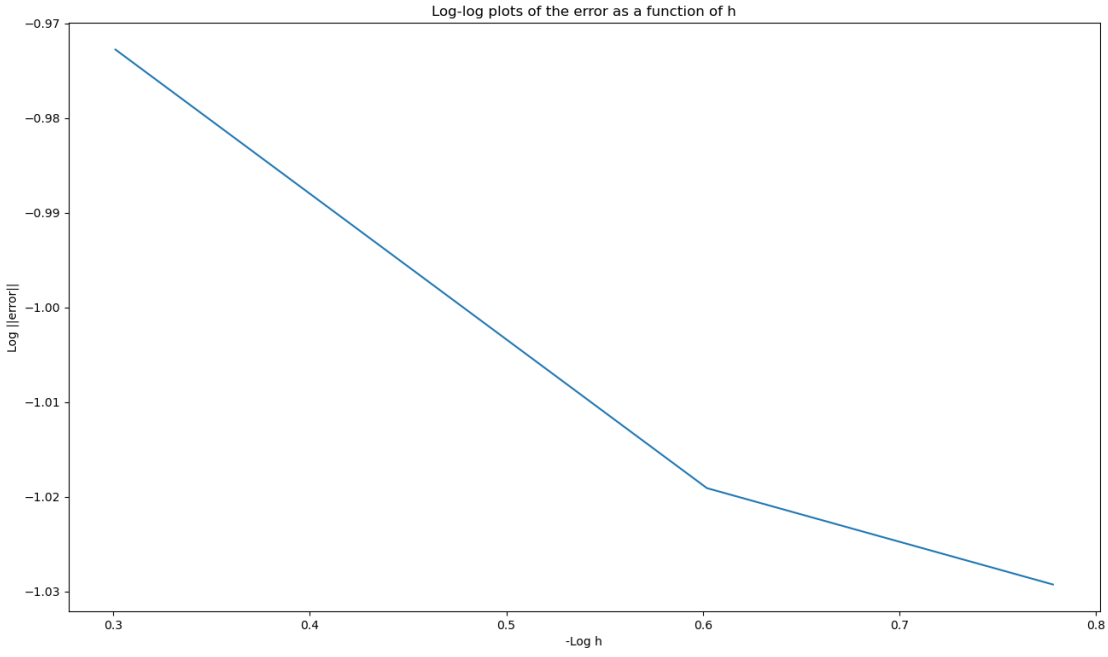

Figure 3: *Logarithmic plot of error as a function of the element length (h)*

$$-u''(x) + u(x) = x \ , \ 0 < x < 1 \quad | \quad u(0) = 0, \ \ u(1) = 0$$

Python codes:

```
#HW 1
#MEMA 646
#Hojjat Anvari
#Prof. Strouboulis
#Texas A&M University
#-------------------------------------------------------------------------------
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as spi
###----------- Question 1: -u'' = x     |     u(0) = 0 , u(1) = 0 ------------###

#----- Exact Solution -----#
x = np.linspace(0,1,100)
u_exact = 1/6 * (x - x**3)
L = 1
#----- Weak Formulation -----#
# Global K Matrix
def K_global(K_g, n):
    for row in range (1,n):
        if (row == 1):
            K_g[0][0] = Ke[1][1]
        else:
            K_g[row -2][row - 2] += Ke[0][0]
            K_g[row-1][row-1] = Ke[1][1]
            K_g[row -2][row-1] = Ke[0][1]
            K_g[row-1][row-2] = Ke[1][0]
        if(row == n-1):
            K_g[row-1][row-1] += Ke[0][0]

# Global F vector
def F_global(F_g, n):
    for row in range (1,n):
        if (row == 1):
            Xa = 0
            Xb = h
            fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
            F_g[0][0] = fe[1][0]
        else:
            Xa += h
            Xb += h
            fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
            F_g[row-2][0] += fe[0][0]
            F_g[row-1][0] += fe[1][0]
```

```
        if(row == n-1):
            Xa += h
            Xb += h
            fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
            F_g[row-1][0] += fe[0][0]
#-------------------------------------------------------------------------------#
print("------------------------------------ Question 1: -u'' =
x     |     u(0) = 0 , u(1) = 0 ---------------------------------------")
#---------------------------------||Uex||^2 ---------------------------------#
print("\n")
def d_uex_2(xx):
    return (1/6 - (xx - xx**3))**2
uex_E_2, _ = spi.quad(d_uex_2, 0, 1)
print("\n||Uex||^2:")
print(uex_E_2)
# ---------------------------------- n = 2 ---------------------------------- #
n = 2
h = L/n
# Local element stiffness and load vectors
Xa = 0
Xb = h
Ke = np.array([[1/h, -1/h], [-1/h, 1/h]])
fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
#--------------------------------------
K_g2 = np.zeros((n-1,n-1))
F_g2 = np.zeros((n-1,1))
K_global(K_g2, n)
F_global(F_g2, n)
U2 = np.linalg.inv(K_g2) @ F_g2
U2 = np.insert(U2, 0, 0, axis = 0) # Boundary condition
U2 = np.insert(U2, n, 0, axis = 0)
F_g2 = np.insert(F_g2, 0, 0, axis = 0)
F_g2 = np.insert(F_g2, n, 0, axis = 0)
Uh2 = 1/2 * np.transpose(U2) @ F_g2
error_2_2 = abs(uex_E_2 - Uh2)
print("\n")
print("------------------------- Number of elements: 2 ------------------------
-")
print("Global stiffness matrix(K_g):")
print(K_g2)
print("\nGlobal load vector(F_g):")
print(F_g2)
print("\nThe finite element approximation (uh) coefficients:")
print(U2)
print("\nEnergy in the finite element solution (1/2*uT*F):")
```

```
print(Uh2)
print("\n||eE||^2:")
print(error_2_2)
# ---------------------------------- n = 4 ---------------------------------- #
n = 4
h = L/n
# Local element stiffness and load vectors
Xa = 0
Xb = h
Ke = np.array([[1/h, -1/h], [-1/h, 1/h]])
fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
#----------------------------------
K_g4 = np.zeros((n-1,n-1))
F_g4 = np.zeros((n-1,1))
K_global(K_g4, n)
F_global(F_g4, n)
U4 = np.linalg.inv(K_g4) @ F_g4
U4 = np.insert(U4, 0, 0, axis = 0) # Boundary condition
U4 = np.insert(U4, n, 0, axis = 0)
F_g4 = np.insert(F_g4, 0, 0, axis = 0)
F_g4 = np.insert(F_g4, n, 0, axis = 0)
Uh4 = 1/2 * np.transpose(U4) @ F_g4
error_2_4 = abs(uex_E_2 - Uh4)
print("\n")
print("-------------------------- Number of elements: 4 -------------------------
-")
print("Global stiffness matrix(K_g):")
print(K_g4)
print("\nGlobal load vector(F_g):")
print(F_g4)
print("\nThe finite element approximation (uh) coefficients:")
print(U4)
print("\nEnergy in the finite element solution (1/2*uT*F):")
print(Uh4)
print("\n||eE||^2:")
print(error_2_4)
# ---------------------------------- n = 6 ---------------------------------- #
n = 6
h = L/n
# Local element stiffness and load vectors
Xa = 0
Xb = h
Ke = np.array([[1/h, -1/h], [-1/h, 1/h]])
fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
#------------------------------
```

```
K_g6 = np.zeros((n-1,n-1))
F_g6 = np.zeros((n-1,1))
K_global(K_g6, n)
F_global(F_g6, n)
U6 = np.linalg.inv(K_g6) @ F_g6
U6 = np.insert(U6, 0, 0, axis = 0) # Boundary condition
U6 = np.insert(U6, n, 0, axis = 0)
F_g6 = np.insert(F_g6, 0, 0, axis = 0)
F_g6 = np.insert(F_g6, n, 0, axis = 0)
Uh6 = 1/2 * np.transpose(U6) @ F_g6
error_2_6 = abs(uex_E_2 - Uh6)
print("\n")
print("------------------------- Number of elements: 6 ------------------------
-")
print("Global stiffness matrix(K_g):")
print(K_g6)
print("\nGlobal load vector(F_g):")
print(F_g6)
print("\nThe finite element approximation (uh) coefficients:")
print(U6)
print("\nEnergy in the finite element solution (1/2*uT*F):")
print(Uh6)
print("\n||eE||^2:")
print(error_2_6)
#------------------------------------ Error ----------------------------------#
error = [error_2_2**0.5, error_2_4**0.5, error_2_6**0.5]
logerror = np.log10(error)
logerror = logerror.ravel()   # Convert to 1D array
dx = [L/2, L/4, L/6]
logdx = -np.log10(dx)
#------------------------------------ Plot -----------------------------------#
plt.figure(1)
plt.plot(x,u_exact,label="Exact Solution",linewidth=1.5)
x2 = np.linspace(0,1,3)
plt.plot(x2,U2,label="Weak Formulation, n = 2",linewidth=0.8)
x4 = np.linspace(0,1,5)
plt.plot(x4,U4,label="Weak Formulation, n = 4",linewidth=0.8)
x6 = np.linspace(0,1,7)
plt.plot(x6,U6,label="Weak Formulation, n = 6",linewidth=0.8)
plt.xlabel("x / L")
plt.ylabel("Finite element approximation (uh)")
plt.title("Comparing Exact Solution with Finite Element Approximation(-u'' = x ,
u(0) = 0 , u(1) = 0)")
plt.legend()
plt.tight_layout()
```

```
plt.show()
#-------------------------------------------------------------
plt.figure(2)
plt.plot(logdx, logerror,linewidth=1.5)
plt.xlabel("-Log h")
plt.ylabel("Log ||error||")
plt.title("Log-log plots of the error as a function of h")
plt.tight_layout()
plt.show()
#-------------------------------------------------------------------------------
###-------- Question 2: -u'' + u = x     |     u(0) = 0 , u(1) = 0 ----------###
#-------------------------------------------------------------------------------
#----- Exact Solution -----#
x = np.linspace(0,1,100)
u_exact = x - np.sinh(x)/np.sinh(1)
L = 1
#----- Weak Formulation -----#
# Global K Matrix
def K_global(K_g, n):
    for row in range (1,n):
        if (row == 1):
            K_g[0][0] = Ke[1][1]
        else:
            K_g[row -2][row - 2] += Ke[0][0]
            K_g[row-1][row-1] = Ke[1][1]
            K_g[row -2][row-1] = Ke[0][1]
            K_g[row-1][row-2] = Ke[1][0]
        if(row == n-1):
            K_g[row-1][row-1] += Ke[0][0]
# Global F vector
def F_global(F_g, n):
    for row in range (1,n):
        if (row == 1):
            Xa = 0
            Xb = h
            fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
            F_g[0][0] = fe[1][0]
        else:
            Xa += h
            Xb += h
            fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
            F_g[row-2][0] += fe[0][0]
            F_g[row-1][0] += fe[1][0]
        if(row == n-1):
            Xa += h
```

```
            Xb += h
            fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
            F_g[row-1][0] += fe[0][0]
#-------------------------------------------------------------------------------#
print("------------------------------------ Question 2: -u'' + u =
x     |     u(0) = 0 , u(1) = 0 ------------------------------------")
#---------------------------------||Uex||^2 ---------------------------------#
print("\n")
def d_uex_2(xx):
    return (1 - np.cosh(xx) / np.sinh(1))**2
uex_E_2, _ = spi.quad(d_uex_2, 0, 1)
print("\n||Uex||^2:")
print(uex_E_2)
# ---------------------------------- n = 2 ---------------------------------- #
n = 2
h = L/n
# Local element stiffness and load vectors
Xa = 0
Xb = h
Ke = np.array([[1/h + h/3, -1/h + h/6], [-1/h + h/6, 1/h + h/3]])
fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
#--------------------------------------
K_g2 = np.zeros((n-1,n-1))
F_g2 = np.zeros((n-1,1))
K_global(K_g2, n)
F_global(F_g2, n)
U2 = np.linalg.inv(K_g2) @ F_g2
U2 = np.insert(U2, 0, 0, axis = 0) # Boundary condition
U2 = np.insert(U2, n, 0, axis = 0)
F_g2 = np.insert(F_g2, 0, 0, axis = 0)
F_g2 = np.insert(F_g2, n, 0, axis = 0)
Uh2 = 1/2 * np.transpose(U2) @ F_g2
error_2_2 = abs(uex_E_2 - Uh2)
print("\n")
print("------------------------- Number of elements: 2 -------------------------
-")
print("Global stiffness matrix(K_g):")
print(K_g2)
print("\nGlobal load vector(F_g):")
print(F_g2)
print("\nThe finite element approximation (uh) coefficients:")
print(U2)
print("\nEnergy in the finite element solution (1/2*uT*F):")
print(Uh2)
print("\n||eE||^2:")
```

```
print(error_2_2)
# ---------------------------------- n = 4 ---------------------------------- #
n = 4
h = L/n
# Local element stiffness and load vectors
Xa = 0
Xb = h
Ke = np.array([[1/h + h/3, -1/h + h/6], [-1/h + h/6, 1/h + h/3]])
fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
#-----------------------------------
K_g4 = np.zeros((n-1,n-1))
F_g4 = np.zeros((n-1,1))
K_global(K_g4, n)
F_global(F_g4, n)
U4 = np.linalg.inv(K_g4) @ F_g4
U4 = np.insert(U4, 0, 0, axis = 0) # Boundary condition
U4 = np.insert(U4, n, 0, axis = 0)
F_g4 = np.insert(F_g4, 0, 0, axis = 0)
F_g4 = np.insert(F_g4, n, 0, axis = 0)
Uh4 = 1/2 * np.transpose(U4) @ F_g4
error_2_4 = abs(uex_E_2 - Uh4)
print("\n")
print("------------------------- Number of elements: 4 ------------------------
-")
print("Global stiffness matrix(K_g):")
print(K_g4)
print("\nGlobal load vector(F_g):")
print(F_g4)
print("\nThe finite element approximation (uh) coefficients:")
print(U4)
print("\nEnergy in the finite element solution (1/2*uT*F):")
print(Uh4)
print("\n||eE||^2:")
print(error_2_4)
# ---------------------------------- n = 6 ---------------------------------- #
n = 6
h = L/n
# Local element stiffness and load vectors
Xa = 0
Xb = h
Ke = np.array([[1/h + h/3, -1/h + h/6], [-1/h + h/6, 1/h + h/3]])
fe = h/6 * np.array([[2*Xa + Xb], [Xa + 2*Xb]])
#------------------------------
K_g6 = np.zeros((n-1,n-1))
F_g6 = np.zeros((n-1,1))
```

```
K_global(K_g6, n)
F_global(F_g6, n)
U6 = np.linalg.inv(K_g6) @ F_g6
U6 = np.insert(U6, 0, 0, axis = 0) # Boundary condition
U6 = np.insert(U6, n, 0, axis = 0)
F_g6 = np.insert(F_g6, 0, 0, axis = 0)
F_g6 = np.insert(F_g6, n, 0, axis = 0)
Uh6 = 1/2 * np.transpose(U6) @ F_g6
error_2_6 = abs(uex_E_2 - Uh6)
print("\n")
print("------------------------- Number of elements: 6 -------------------------
-")
print("Global stiffness matrix(K_g):")
print(K_g6)
print("\nGlobal load vector(F_g):")
print(F_g6)
print("\nThe finite element approximation (uh) coefficients:")
print(U6)
print("\nEnergy in the finite element solution (1/2*uT*F):")
print(Uh6)
print("\n||eE||^2:")
print(error_2_6)
#---------------------------------- Error ----------------------------------#
error = [error_2_2**0.5, error_2_4**0.5, error_2_6**0.5]
logerror = np.log10(error)
# Convert to 1D array
logerror = logerror.ravel()
dx = [L/2, L/4, L/6]
logdx = -np.log10(dx)
#----------------------------------- Plot -----------------------------------#
plt.figure(1)
plt.plot(x,u_exact,label="Exact Solution",linewidth=1.5)
x2 = np.linspace(0,1,3)
plt.plot(x2,U2,label="Weak Formulation, n = 2",linewidth=0.8)
x4 = np.linspace(0,1,5)
plt.plot(x4,U4,label="Weak Formulation, n = 4",linewidth=0.8)
x6 = np.linspace(0,1,7)
plt.plot(x6,U6,label="Weak Formulation, n = 6",linewidth=0.8)
plt.xlabel("x / L")
plt.ylabel("Finite element approximation (uh)")
plt.title("Comparing Exact Solution with Finite Element Approximation(-u'' + u =
x , u(0) = 0 , u(1) = 0)")
plt.legend()
plt.tight_layout()
plt.show()
```

```
#-------------------------------------------------------------
plt.figure(2)
plt.plot(logdx, logerror,linewidth=1.5)
plt.xlabel("-Log h")
plt.ylabel("Log ||error||")
plt.title("Log-log plots of the error as a function of h")
plt.tight_layout()
plt.show()
```

References:

1- *Becker,, E. B. G. F. Carey, and J. T. Oden, Finite Elements an Introduction, Texas Institute for Computational Mechanics, UT Austin,1981.*