MODELING AND USE OF FEM ON STATIC STRUCTURE

A SECOND YEAR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF B.Sc. IN COMPUTATIONAL MATHEMATICS

BY

1. Samrajya Raj Acharya(Exam Roll No.)
2. Bishesh Kafle (Exam Roll No.)
3. Priyanka Panta (Exam Roll No.)
4. Mukesh Tiwari (Exam Roll No.)

SCHOOL OF SCIENCE
KATHMANDU UNIVERSITY
DHULIKHEL, NEPAL

April, 2022

CERTIFICATION

This project entitled "Modeling and use of FEM on static structure" is carried out under my supervision for the specified entire period satisfactorily, and is hereby certified as a work done by following students

1. Samrajya Raj Acharya (Exam Roll No.)
2. Bishesh Kafle (Exam Roll No.)
3. Priyanka Panta (Exam Roll No.)
4. Mukesh Tiwari (Exam Roll No.)

in partial fulfillment of the requirements for the degree of B.Sc. in Computational Mathematics, Department of Mathematics, Kathmandu University, Dhulikhel, Nepal.

Dr. Gokul KC
Assistant Professor
Department of Mathematics,
School of Science, Kathmandu University,
Dhulikhel, Kavre, Nepal
Date:April 17, 2022

APPROVED BY:

I hereby declare that the candidate qualifies to submit this report of the Math Project (MATH-252) to the Department of Mathematics.

Head of the Department
Department of Natural Sciences
School of Science
Kathmandu University
Date:April 17, 2022

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Your parameters here.

CHAPTER 1

MOTIVATION/INTRODUCTION

1.1 Context/Rationale/Background

In this section, the author(s) shall discuss the historical background related to the work along with the introduction of the topic in brief.

Finite Element Method (FEM) is a numerical method for solving a differential or integral euation. It is a numeral method for finding approximate solutions to partial differential equations in two or three space variables. In the finite element method, a given domain is viewed as a collection of sub-domains, and over each subdomain the govening equation is approximated by any of the traditional variational methods.

1.2 History

The concept of FEM first originated with the need to solve complex elasticity and structural analysis problems in civil and aeronautical engineering. A. Hrennikoff, R. Courant, Ioannis Argyris were pioneers from early 1940s. It was again rediscovered in China by Feng Kong in the later 1950s and early 1960s as Finite Difference Method based on variation principle. All of these works were based on mesh descritization of a continuos domain into a set of discrete sub-domains. Olgied Cecil Ziekiewicz published his first paperin 1947 dealing with numerical approximation to the stress analysis of dams. He first recognised the general potential for using the finite element method to resolve problems in area outside of solid mechanics. Proper in-depth development of FEM began in the middle to late 1950s. By late 1950s, key concepts of stiffness matric and element assembly existed essentially in the form used today. In 1965, NASA proposed for the development of FEM

software NASTRAN and sponsered the original version of it, and UC Berkeley made the finite element programSAP IV widely available. In 1976, Strang and Fix published 'An Analysis Of The Finite Element Method' which provided a rigorous mathematical asis to FEM. FEM has since been generalized into a branch of applied mathematics fornumerical modeling of physical systems in different disciplines like electromagnetism and fluid dynamics.

1.3 Objectives

1. **To set up differential equations with variable boundary conditions.**
2. **To learn to perform numericals manually and then implement those in a high level programming language**
3. **To learn about FEM, its variations, and its application to various mechanical problems.**
4. **Visualization and Project Writing.**

1.4 Significance/Scope

The use of numerical methods is prevalent in all fields of science and technology today. Our project focuses on one powerful numerical tool known as FEM which is theoretically sound and computationally efficient. The basic ideas of such tools which one is sure to encounter later will be very beneficial. This project also opens up the world of variational calculus and its applications which are generally not covered in undergraduate mathematics.

The computer implementation of a package that implements the algorithm for solving problem using FEM while handling inputs and visualizing the output is a stark contrast from the dummy math problem that are usually used in computer programming classes to teach concepts of general programming rather than mathematical programming. This project imparts the skill to convert mathematical knowledge into efficient and all around packages that solve problems that are based on real world applications and are similar to ones encountered later at work in industries or academia.

1.5 Limitations

The project only focuses on application of FEM to a single differential equation and thus despite being a great starting point for diving into the world of finite element method, it lacks behind in providing full demonstration of its potential. We have applied FEM to solve 2D truss structures. The computer implementation is also limited to this subset of problems and can work with only 2D trusses.

CHAPTER 2

METHODOLOGY/MODEL EQUATION

2.1 Theoretical/Conceptual Framework

In this section, the author(s) should describe the theoretical/mathematical principles behind the whole work relative to the project. The information collected from literature review shall be relevant in this section.

Consider

$$-\frac{d}{dx}\left[a(x)\frac{du}{dx}\right] = f(x) \qquad \text{for} \qquad 0 < x < L. \tag{2.1}$$

$$\tag{2.2}$$

for $u(x)$ subject to the boundary conditions

$$u(0) = u_0 \tag{2.3}$$

$$a(x)\frac{du}{dx}\bigg|_{x=L} = Q_L \tag{2.4}$$

where $a(x)$ and $f(x)$ are known functions and u_0 and Q_L are known values. In case of bar (which is a axially loaded structure)

u = displacement

$a(x)$ = EA (stiffness)

f = distributed axial force

Q_L = axial load

Now, we want an approximation of u(x) in the form

$$u(x) \approx u_N(x) = \sum_{j=1}^{N} c_j \phi_j + (x) + \phi_0(x) \tag{2.5}$$

Substituting U_N(x) in our Differential Equation,

$$-\frac{d}{dx}\left[a(x)\frac{dU_N}{dx}\right] = f(x) \qquad \text{for} \qquad 0 < x < L. \tag{2.6}$$

If this equally holds for all $x \in [0, L]$ the solution is exact. Since, we assume it only as an approximation. we define Residual Function $R(x, c_1, c_2, c_3, ..., c_N)$ as

$$R = -\frac{d}{dx}\left[a(x)\frac{dU_N}{dx}\right] - f(x) \tag{2.7}$$

We have R $\neq 0$ $\forall x \in [0, L]$ as a U_N is only a approximation to solution.

Now we have various ways to minimize R in some senses over the domain to make this approximation close to the actual solution

1. Collocation Method

   It forces that R is zero at selected N points of the domain.

   i.e;

$$R(x, c_1, c_2, c_3, ..., c_N) = 0 \qquad \forall x = x_i, i = 1, 2, ..., N \tag{2.8}$$

2. Least Square Method

$$\frac{\partial}{\partial c_j}\int_0^L R^2 dx = 0 \qquad \forall i = 1, 2, ..., N \tag{2.9}$$

3. Weighted Residual Method we desire that

$$\int_0^L w_i(x) R dx = 0 \qquad \forall i = 1, 2, ..., N \tag{2.10}$$

where $w_i(x)$ are N linearly independent functions called weight functions.

Now we convert our differential equation and boundary condition to weak form.

- Step 1 :

  We write the weighted integral statement, i.e;

$$\int_0^L w \left[-\frac{d}{dx}\left(a(x)\frac{du}{dx} \right) - f \right] dx = 0 \tag{2.11}$$

  It is equivalent to differential equations and does not include any boundary conditions and thus the variable u must be differentiable to as many order as is required by the differential equation.

- Step 2

  Weakening Differentiability conditions

$$\int_0^L \left(w \left[-\frac{d}{dx}\left(a(x)\frac{du}{dx} \right) \right] - wf \right) dx = 0 \tag{2.12}$$

  Integrating by parts, we get,

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf \right] dx - \left[wa\frac{du}{dx} \right]_0^L = 0 \tag{2.13}$$

  we have used the fact that

$$\int_a^b \left(w\frac{dv}{dx} \right) dx = -\int_a^b v dw + [wv]_0^L = 0 \tag{2.14}$$

  by considering,

$$v = -a\frac{du}{dx} \tag{2.15}$$

  we can see that now we require that w be differentiable at least once but this also made that u needs to be differentiable only once even though the equation is of second order.// Now, we take care of the boundary conditions which are of two types

– Natural Boundary condition

– Essential Boundary Condition

Before defining these conditions we define primary and secondary variables.

Definition 1 *Primary Variable pass*

Definition 2 *Secondary Variable pass*

In our case , $a\frac{du}{dx}$ is secondary variable.

If secondary variable(SV) is specified in the boundary, then such conditions are called natural boundary conditions (NBC). If primary variable(PV) is specified in the boundary, then such conditions are called essential boundary conditions (NBC).

let us now define secondary variable as Q.

$$Q = a\frac{du}{dx}n_x \tag{2.16}$$

where n_x is ???

For 1D problems,

$$n_x = -1 \qquad \text{at left} \tag{2.17}$$

$$n_x = 1 \qquad \text{at right} \tag{2.18}$$

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right] dx - \left[wa\frac{du}{dx}\right]_0^L = 0 \tag{2.19}$$

$\Longrightarrow$

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right] dx + wa\frac{du}{dx}\bigg|_{x=L} - wa\frac{du}{dx}\bigg|_{x=0} = 0 \tag{2.20}$$

since $n_x = -1 \text{at} x = 0$ and $n_x = 1 \text{at} x = L$

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right] dx - n_x wa\frac{du}{dx}\bigg|_{x=L} - n_x wa\frac{du}{dx}\bigg|_{x=0} = 0 \tag{2.21}$$

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right] dx - (wQ)_0 - (wQ)_L = 0 \tag{2.22}$$

Finally we require that weight functions vanishes at boundaries where essential boundary conditions are specified.

$$u(0) = u_0 \implies w(0) = 0 \tag{2.23}$$

Thus, we get

$$\int_0^L \left[a \frac{dw}{dx} \frac{du}{dx} - wf \right] dx - w(L) Q_L = 0 \tag{2.24}$$

$$B(w,u) = \int_0^L a \frac{dw}{dx} \frac{du}{dx} dx \tag{2.25}$$

$$l(w) = \int_0^L wf dx + w(L) Q_L \tag{2.26}$$

Hence, our weak form can be written as

$$0 = B(w,u) - l(w) \tag{2.27}$$

or ,

$$B(w,u) = l(w) \tag{2.28}$$

This is the variational form of the problem that is associated with our ODE and its boundary conditions.

When the differential equation is linear and of even order, the resulting weak form will have symmetric bi-linear form in u and w.

Let us now choose the approximate solutions that satisfy the two conditions for primary variables as other conditions are already included in the weak form. i.e,

$$u_h^e(x_a) = u_1^e \tag{2.29}$$

$$u_h^e(x_b) = u_2^e \tag{2.30}$$

let,

$$u_h^e(x) = c_1 + c_2 x \tag{2.31}$$

Then by the conditions,

$$u_h^e(x_a) = c_1 + c_2 x_a = u_1^e \tag{2.32}$$

$$u_h^e(x_b) = c_1 + c_2 x_b = u_2^e \tag{2.33}$$

writing in matrix form, we obtain

$$\begin{bmatrix} u_1^e \\ u_2^e \end{bmatrix} = \begin{bmatrix} 1 & x_a \\ 1 & x_b \end{bmatrix} \begin{bmatrix} c_1^e \\ c_2^e \end{bmatrix} \tag{2.34}$$

We can rewrite the equations as

$$\begin{bmatrix} c_1^e \\ c_2^e \end{bmatrix} = \frac{1}{x_b - x_a} \begin{bmatrix} x_b & -x_a \\ -1 & 1 \end{bmatrix} \begin{bmatrix} u_1^e \\ u_2^e \end{bmatrix} \tag{2.35}$$

Hence we get,

$$c_1^e = \frac{1}{h_e}(x_b u_1^e - x_b u_2^e) \tag{2.36}$$

$$c_2^e = \frac{1}{h_e}(-u_1^e + u_2^e) \tag{2.37}$$

If we let,

$$\alpha_1^e = x_b$$

$$\alpha_2^e = -x_a$$

$$\beta_1^e = -1$$

$$\beta_2^e = 1$$

$$c_1^e = \frac{1}{h_e}(\alpha_1^e u_1^e + \alpha_2^e u_2^e) \tag{2.38}$$

$$c_2^e = \frac{1}{h_e}(\beta_1^e u_1^e + \beta_2^e u_2^e) \tag{2.39}$$

Hence,

$$U_h^e(x) = \frac{1}{h_e}[\alpha_1^e u_1^e + \alpha_2^e u_2^e + (\beta_1^e u_1^e + \beta_2^e u_2^e)x]$$

$$U_h^e(x) = \frac{1}{h_e}[\alpha_1^e + \beta_1^e x]u_1^e + \frac{1}{h_e}[\alpha_2^e + \beta_2^e x]u_2^e$$

$$U_h^e(x) = \sum_{j=1}^{2} \frac{1}{h_e}[\alpha_j^e + \beta_j^e x]u_j^e \tag{2.40}$$

$$U_h^e(x) = \sum_{j=1}^{2} \psi_j^e(x) u_j^e \tag{2.41}$$

where $\psi_j^e(x)$ are known as interpolation functions.

2.2 Python Implementation

2.2.1 Input/Output

To feed the data to our program, we came up with the idea of using csv files so as to make the program available for general problems and for a easy to use input interface.

Node	x_pos	y_pos	x_dis	y_dis	x_force	y_force
1	0	0	0	0	0	0
2	2	0	0	0	0	0
3	4	0	0	0	0	0
4	6	0	0	0	0	0
5	8	0	0	0	0	0
6	8	2.918	nan	nan	0.2	-0.1
7	6	2.1838	nan	nan	0	-0.1
8	4	1.4559	nan	nan	0	-0.1
9	2	0.7279	nan	nan	0	-0.1

Table 2.1: Example of a CSV file used to input node data

Table 2.1shows the input format of nodes where each row represents a node and its various attributes in the specified columns. Going from left to right we get node number, position , displacements and external force applied along the local x and y-axis.

The values that are unknown and thus to be computed will be denoted by nan and any geometry and boundary conditions can be specified in this table.

Element	Length	CS-Area	Young's Modulus	Global Angle	Start node	End node
1	2	1	1	pi*0	1	2
2	2	1	1	pi*0	2	3
3	2	1	1	pi*0	3	4
4	2	1	1	pi*0	4	5
5	2.9118	1	1	pi*0.5	5	6
6	2.1284	1	1	pi*0.11111	7	6

Table 2.2: Example of a CSV file used to input element data

Table 2.2 shows the input format of the elements where each row represents an element and its attributes: element number, length , cross-section area , young's modulus , angle with the global x-axis and the nodes that it connects. We can use this table to supply material properties and specify the problem geometry.

The output is similarly given as a CSV file.

2.2.2 Element and Node classes

The input from csv files are then used to define the objects of two classes namely node and ele. These class have all the attributes needed and methods to be able to easily manipulate and visualize them.

Formation of global stiffness matrix was a challenge and reducing the code to faster execution and simplicity was even bigger of a task. Finally the method applied is as follows.

```
def globalstiff(eles , num_nodes):
  #dimension of global matrix
  dim = 2 * num_nodes
  #generate and store the element-wise stiffness matrices
  SMs = [e.stiff() for e in eles]

  GK = np.zeros((dim,dim))

  for e in eles:
    i = 2 * e.node_a.num -2
    j = 2 * e.node_a.num -1
    k = 2 * e.node_b.num -2
    l = 2 * e.node_b.num -1
    e_stiff = e.stiff()

    index = [i,j,k,l]
    index2d = [(a,b) for a in index for b in index]
    d = {i:0, j:1, k:2, l:3}

    for p,q in index2d:
        GK[p][q] = GK[p][q] + e_stiff[d[p]][d[q]]

  return GK
```

The list of elements is iterated throughout for accessing each element. Then with the logic of how the values in local stiffness matrix is appended to the global matrix, we came up with an efficient algorithm that fits well to our purpose as well as works on all such general problems with slightest of tweaks.

After generating global stiffness matrix we now had 3 variables: the stiffness matrix , the displacement and the force vector. For simplifying the calculations and faster computation, for n^{th} element which was specified in the displacement vector, we reduced the global matrix by removing corresponding n^{th} row and columns by changing the secondary

variable accordingly. After reduction simple linear algebra has been applied to find out the unknown/missing values of displacement of each node.

```
#check for undetermined values in dis and create linear eqns

GK_dis =  copy.deepcopy(GK)
f_dis =  copy.deepcopy(f)

#get a list of rows to remove
del_row = []

index_list = list(range(0,2*len(nodes)))
for i in range(0,2*len(nodes)):
    if not np.isnan(dis_list[i]):
        del_row.append(i)
        index_list.remove(i)

#remove the rows that have displacement given
GK_dis = np.delete(GK_dis,del_row,0)
f_dis = np.delete(f_dis,del_row,0)

#before deleting the columns we subratct these from force
   vector
for i in del_row:
        f_dis = f_dis - dis_list[i] * GK_dis[:,i]

#delete the columns that are due to the displacements that
   are determined
GK_dis = np.delete(GK_dis,del_row,1)
```

2.2.3 Visualization

For the last and final part of visualizing our problem and the visualizing the solution so as to make each bit of data understandable we used the Turtle module of Python. Initially we draw our problem provided the position of the nodes and elements connecting them.

```
  for ele in self.ele_list:
  t.goto(ele.node_a.pos_x, ele.node_a.pos_y)
  t.pendown()
  t.goto(ele.node_b.pos_x, ele.node_b.pos_y)
  t.penup()

  for node in self.node_list:
  t.penup()
  t.goto(node.pos_x, node.pos_y)
  t.pendown()
  t.dot(15, "red")

  t.penup()
  t.goto((ele.node_a.pos_x + ele.node_a.dis_x), (ele.node_a.
   pos_y + ele.node_a.dis_y))
  t.pendown()

  # after solving
  t.pencolor("green")
  for ele in self.ele_list:
  t.goto((ele.node_a.pos_x + ele.node_a.dis_x), (ele.node_a.
   pos_y + ele.node_a.dis_y))
```

```
21      t.pendown()
22      t.goto((ele.node_b.pos_x + ele.node_b.dis_x), (ele.node_b.
        pos_y + ele.node_b.dis_y))
23      t.penup()
24
25      t.speed(0)
26      for node in self.node_list:
27      t.penup()
28      t.goto((node.pos_x + node.dis_x), (node.pos_y + node.dis_y)
        )
29      t.pendown()
30      t.dot(15, "yellow")
```

Then using the displacement data of each node that we get after the computation, the same algorithm is applied to draw the final result/ shape of the truss. This time the position of node is the sum of its original position and its displacement in x and y-axis. Different tools have been used to make the visualized problem and result understandable.

CHAPTER 3

RESULTS AND DISCUSSIONS

3.1 Result

In this chapter, the author(s) shall present the results and simulations (in the form of numerical data or graphical form) of the theory described in CHAPTER-2 and finally discuss the results.

CHAPTER 4

CONCLUSIONS

In this section, the author(s) shall summarize the main points and the results of the work. Include key facts from the background research to help explain your results as needed.