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(026592-19)
2. Bishesh Kafle (026600-19)
3. Priyanka Panta (026605-19)
4. Mukesh Tiwari (026615-19)

DEPARTMENT OF MATHEMATICS

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 (026592-19)
2. Bishesh Kafle (026600-19)
3. Priyanka Panta (026605-19)
4. Mukesh Tiwari (026615-19)

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 23, 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 Mathematics
School of Science
Kathmandu University
Date:April 23, 2022

ACKNOWLEDGMENTS

This research was carried out under the supervision of ‘Dr. Gokul K.C, assistant professor, Department of Mathematics’. We would like to express our sincere gratitude towards our supervisor for his excellent supervision, guidance and suggestion for accomplishing this work. And to the entire faculty of “Department of Mathematics” for encouraging, supporting and providing this opportunity.

We are indebted to all our classmates and friends for helping and guiding us over the related software.

Lastly, we would like to thank everyone who helped us directly and indirectly during the duration of completing our project work.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

MOTIVATION/INTRODUCTION

1.1 Context/Rationale/Background

Finite Element Method (FEM) is a numerical method for solving a differential or integral equation. 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 governing 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 [1], R. Courant [2], Ioannis Argyris [3] 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[4] published his first paper in 1947 dealing with numerical approximation to the stress analysis of dams. He first recognized 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 matrix and element assembly existed essentially in the form used today. In 1965, NASA proposed for the development of FEM software NASTRAN and sponsored the original version of it, and UC Berkeley made the finite element program SAP IV widely available. In 1976, Strang and Fix[5] published 'An Analysis Of The Finite Element Method' which provided a rigorous mathematical basis to FEM. FEM has since been generalized into a branch of applied mathematics for numerical modeling of physical systems in different disciplines like electromagnetism and fluid dynamics[6][7].

1.3 Objectives

1. **To set up differential equations with variable boundary conditions.**
   Differential equations are the backbone for the mathematical modeling of physical systems. To translate physical problems into the mathematical ones, it is crucial to correctly specify the boundary conditions. So, the foremost objective is learning to correctly apply the differential equations and specify proper boundary conditions by working of various kinds of truss.

2. **To learn about FEM, its variations, and its application to various mechanical problems.**
   It is important to have all the theoretical knowledge about the finite element method, its different types and application in different engineering disciplines such as thermal and fluid dynamics.

3. **To learn to perform computations manually and then implement those in a high level programming language**
   To understand the physical systems properly, we have to learn the solving techniques and processes manually first,through pen and paper, and then finally implement it into the computer so that the solutions can be verified.

4. **Visualization and Project Writing.**
   Moreover, as the results created through FEM is a huge set of data. It is very difficult to gain information from a large data set and numbers. Visualization fills the gap, so that, we can explore solution through our eyes. Thus, we also aim to be able to visualize the raw data into information so that it is easy to grasp. Visualization is an important skill in today's world and we plan to learn problem specific visualization using python.

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.Thus, other physical structures might not be solved through this same approach. 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 Conceptual Framework - Truss

Truss is a structure consisting of organized objects in a shape of connecting triangles in such a way that it behaves as a single unit object. In order to enable the distribution of weight and grasp up the the fluctuating tension and compression without bending and shearing, it is made up of a web of triangles which is most stable form of geometry. In general, truss is used since it has a long life span and has least weight to the possible which in turn supports large loads and reduces deflections.

Commonly used in bridges, roofs and high rise buildings, it gives high value for mega constructions like the Eiffel Tower, construction of a stadium and all. We can think of truss as a beam where the web consists of series of separate members instead of a con-

Figure 2.1: Truss in bridge

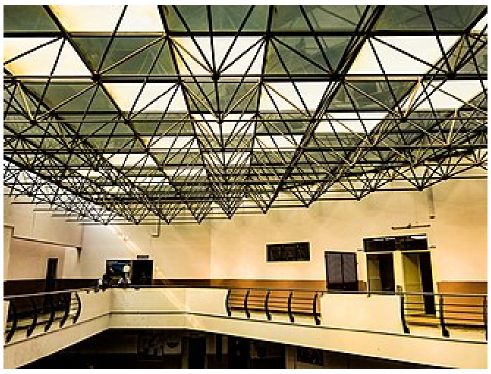


Figure 2.2: Truss in roofing

Figure 2.3: Three Dimensional Truss

tinuous plate. In engineering perspective, a truss is a structure that comprises of one or more triangular units constructed in a linear pattern such that its ends are connected to joints which is called as external nodes. The external forces and reaction to the forces are considered to act only at the nodes and the results in forces in the members are either tensile or compressive forces. In truss, the lower horizontal structure and the upper horizontal structure carry tension and compression. The diagonal and vertical structure form the truss web and it carries shear stress. Technically, they are also in tension and compression where the exact arrangement of forces depends on the type of truss and on the direction of bending. The structures serves in order to stabilize each other in order to prevent buckling. The structures that are under compression are to be designed to be safe against buckling. After knowing the force on each member, we determine the cross sectional area of the individual truss members. The weight of truss structure depends directly on its cross section area. The effect of weight of individual structures in a large truss is generally insignificant compared to the force exerted by external loads.

Later, after determining the minimum area of cross section of individual structures, it is dealt with bolted joints that involves shear stress of the bolt and connections used in the joints. The joints of truss can be designed as rigid, semi-rigid, or hinged.

Equation of truss:

The strain-displacement and stress-strain relationship is:

$$\epsilon = \frac{du}{dx} \tag{2.1}$$

$$\sigma = E\epsilon \tag{2.2}$$

where ϵ is strain , σ is strain and u is the displacement.

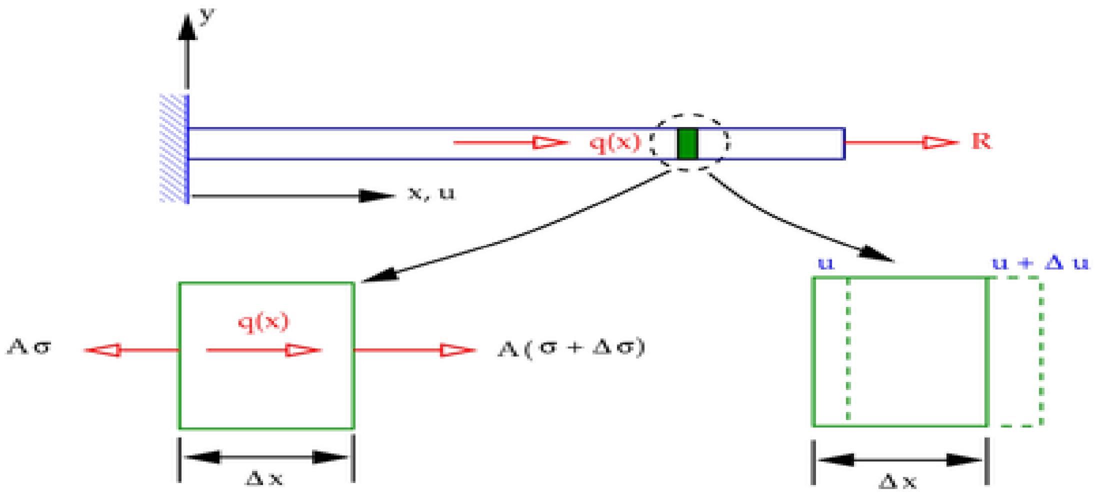


Figure 2.4: derivation of equation for a rod

From the law of equilibrium, we get,

$$A\sigma = A(\sigma + \Delta\sigma) + q(x)\Delta x \tag{2.3}$$

where $q(x)\Delta x$ is the force acting on the slice of the bar under consideration.

$$A\frac{(\sigma_{x+\Delta x} - \sigma_x)}{\Delta x} + q(x) = 0 \tag{2.4}$$

Taking limit as $\Delta x \to 0$

$$A\frac{d\sigma}{dx} + q(x) = 0 \tag{2.5}$$

Using 2.1 , we get:

$$-AE\frac{d^2u}{dx^2} = q(x) \tag{2.6}$$

which is the equation of truss.

2.2 Mathematical Model and solution methods

Consider

$$-\frac{d}{dx}\left[a(x)\frac{du}{dx}\right] = f(x) \qquad \text{for} \qquad 0 < x < L. \tag{2.7}$$

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

$$u(0) = u_0 \tag{2.8}$$

$$a(x)\frac{du}{dx}\bigg|_{x=L} = Q_L \tag{2.9}$$

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) , the variables denote these quantities:

$$u = \text{displacement}$$

$$a(x) = \text{EA (stiffness)}$$

$$f = \text{distributed axial force}$$

$$Q_L = \text{axial load}$$

Our goal is to determine the function $u(x)$. Since analytic or exact solution is not feasible, we want an approximation of u(x) in the form

$$u(x) \approx u_N(x) = \sum_{j=1}^{N} c_j\phi_j + (x) + \phi_0(x) \tag{2.10}$$

Substituting U_N(x) in our Differential Equation,

$$-\frac{d}{dx}\left[a(x)\frac{dU_N}{dx}\right] = f(x) \qquad \text{for} \qquad 0 < x < L. \tag{2.11}$$

If this equally holds for all $x \in [0, L]$ the solution is exact. Since, it is only an approximation , we define residual function $R(x, c_1, c_2, c_3, ..., c_N)$ to check the how close the approximation is to our original problem.

$$R = -\frac{d}{dx}\left[a(x)\frac{dU_N}{dx}\right] - f(x) \tag{2.12}$$

We have R $\neq$0 $\forall x \in [0, L]$ as 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. At this point all we want to do is to get N conditions to impose on the R in such a way that we obtain N linear equations in the unknown coefficients $c_1, c_2, c_3, ..., c_N$. This is allow us to determine the coefficients in our approximation as in equation 2.10

1. Collocation Method

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

   i.e;

$$R(x, c_1, c_2, c_3, ..., c_N) = 0 \qquad \forall x = x_i, i = 1, 2, ..., N \tag{2.13}$$

2. Least Square Method

$$\frac{\partial}{\partial c_j} \int_0^L R^2 dx = 0 \qquad \forall i = 1, 2, ..., N \tag{2.14}$$

3. Weighted Residual Method we desire that

$$\int_0^L w_i(x) R dx = 0 \qquad \forall i = 1, 2, ..., N \tag{2.15}$$

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

   Choice of weight functions is totally arbitrary but there are some standard ways to choose the weight function which have been assigned special names. If we let $w_i = \phi_i$, then the method is known as **galerkin's method.**

Now we convert our differential equation and boundary condition to weak form to make it easier to choose approximation functions.

- Step 1 : We write the weighted integral statement from eqn 2.15, i.e;

$$\int_0^L w \left[-\frac{d}{dx} \left(a(x) \frac{du}{dx} \right) - f \right] dx = 0 \tag{2.16}$$

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

- Step 2:

  to weaken differentiability conditions for $u(x)$, let us rewrite our eqn 2.16 as,

$$\int_0^L \left(w \left[-\frac{d}{dx} \left(a(x) \frac{du}{dx} \right) \right] - wf \right) dx = 0 \tag{2.17}$$

Integrating by parts, we get,

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right] dx - \left[wa\frac{du}{dx}\right]_0^L = 0 \tag{2.18}$$

we have used the fact that

$$\int_a^b \left(w\frac{dv}{dx}\right) dx = -\int_a^b vdw + [wv]_0^L = 0 \tag{2.19}$$

by considering,

$$v = -a\frac{du}{dx} \tag{2.20}$$

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.

- Step 3:

  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) *The coefficients of the weight function (and its derivatives) in the boundary expressions is called primary variable.*

  Definition 2 (Secondary Variable) *The dependent variable of the problem(u), expressed in the same form as the weight function (w) appearing in the boundary tern is called primary variable.*

  In our case , $u(x)$ is the primary variable and $a\frac{du}{dx}$ 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.

$$Q = a\frac{du}{dx}n_x \tag{2.21}$$

where n_x is the cosine of the cosine of the angle between the positive x-axis and the normal to the boundary.

For 1D problems,

$$n_x = -1 \qquad \text{at left} \tag{2.22}$$

$$n_x = 1 \qquad \text{at right} \tag{2.23}$$

To utilize these new facts , we use the equation 2.18

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right]dx - \left[wa\frac{du}{dx}\right]_0^L = 0 \tag{2.24}$$

$$\implies \int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right]dx + wa\frac{du}{dx}\bigg|_{x=L} - wa\frac{du}{dx}\bigg|_{x=0} = 0 \tag{2.25}$$

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

$$\implies \int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right]dx - n_x wa\frac{du}{dx}\bigg|_{x=L} - n_x wa\frac{du}{dx}\bigg|_{x=0} = 0 \tag{2.26}$$

$$\implies \int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right]dx - (wQ)_0 - (wQ)_L = 0 \tag{2.27}$$

Weight functions can be interpreted as virtual change of the primary variable ($w \approx \delta u$) thus we require that weight functions vanishes at boundaries where essential boundary conditions are specified as at such points the value of u is known exactly.

$$u(0) = u_0 \implies w(0) = 0 \tag{2.28}$$

Thus, we get

$$\int_0^L \left[a\frac{dw}{dx}\frac{du}{dx} - wf\right]dx - w(L)Q_L = 0 \tag{2.29}$$

Let us define two functionals B and l as,

$$B(w,u) = \int_0^L a\frac{dw}{dx}\frac{du}{dx}dx \tag{2.30}$$

$$l(w) = \int_0^L wf dx + w(L)Q_L \tag{2.31}$$

Hence, our weak form can be written as

$$0 = B(w, u) - l(w) \tag{2.32}$$

or ,

$$B(w, u) = l(w) \tag{2.33}$$

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.

We will now use ritz-galerkin , we will obtain the linear equations to obtain the coefficients which we used to define the 2.10 by using the above vairational form.

for Ritz method , we substitute $w_i = \phi_i$ hence eqn 2.45 becomes

$$B\left(\phi_i, \sum_{1}^{N} c_j\phi_j + \phi_0\right) = l(\phi_i) \qquad i = 1, 2, ..., N \tag{2.34}$$

Since $B(\cdot, \cdot)$ is bilinear , it is linear in u. hence,

$$\sum_{1}^{N} B\left(\phi_i, \phi_j\right) c_j = l(\phi_i) - B\left(\phi_i, \phi_0\right) \qquad i = 1, 2, ..., N \tag{2.35}$$

Let ,

$$K_{ij} = B(\phi_i, \phi_j) \tag{2.36}$$

$$F_i = l(\phi_i) - B(\phi_i, \phi_0) \tag{2.37}$$

Then eqn 2.35 can be written in matrix form as

$$Kc = F \tag{2.38}$$

Since K and f are completely determined ,we can easily compute the coefficients.

The last hurdle towards writing down the approximate solution exactly is choosing the approximate functions ϕ_i appropriately.

Firstly we must make sure that these functions satisfy the essential boundary conditions. Since, the natural boundary conditions are already included in the weak form, this is the last step to properly apply all the given boundary conditions.

We can demand that

$$\phi_0(x_0) = u_0 \tag{2.39}$$

$$\sum_1^N c_j \phi_j(x_0) = 0 \tag{2.40}$$

We will obtain

$$U_N(x_0) = u_0 \tag{2.41}$$

and hence the essential boundary conditions are satisfied.

It is obvious that these functions must be as differentiable and integrable as the weak form demands. In addition, we want these to be complete set of **linearly independent functions**.

Thus variational method provides us with a powerful method to solve equations. provided we can compute the integrals in case of complex geometry as the discussion encompasses the entire domain of the problem at once. Since, this is not always possible for complex geometry or material property, we discritize our domain into line elements. A typical element between points A and B with coordinates x_a and x_b respectively, is denoted Ω_e. The length of element is $h_e = x_b - x_a$.

Now we follow the above steps to obtain the approximate solution (u_h^e) over this element. The approximate solution is assumed to be opf the form

$$u_h^e = \sum_{j=0}^N u_j^e \psi_j^e(x) \tag{2.42}$$

Here u_j^e and $\psi_j^e(x)$ are the coefficients and the approximation functions respectively.

Writing eqn 2.16 for our element's domain,

$$\int_{x_a}^{x_b} w \left[-\frac{d}{dx}\left(a(x)\frac{du}{dx} \right) - f \right] dx = 0 \tag{2.43}$$

Similarly, the final weak form can be written from eqn 2.27 as

$$\int_{x_a}^{x_b} \left[a\frac{dw}{dx}\frac{du}{dx} - wf \right] dx - (wQ)_a - (wQ)_b = 0 \tag{2.44}$$

let us also write the corresponding variational form,

$$B^e(w,u) = l^e(w) \tag{2.45}$$

where,

$$B^e(w,u) = \int_{x_a}^{x_b} a\frac{dw}{dx}\frac{du}{dx}dx \tag{2.46}$$

$$l^e(w) = \int_{x_a}^{x_b} wf dx + w(x_a)Q_a + w(x_b)Q_b \tag{2.47}$$

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,

$$u_h^e(x_a) = u_1^e \tag{2.48}$$

$$u_h^e(x_b) = u_2^e \tag{2.49}$$

let,

$$u_h^e(x) = c_1 + c_2 x \tag{2.50}$$

Then by the conditions,

$$u_h^e(x_a) = c_1 + c_2 x_a = u_1^e \tag{2.51}$$

$$u_h^e(x_b) = c_1 + c_2 x_b = u_2^e \tag{2.52}$$

writing in matrix form, we obtain

$$\begin{bmatrix} u_1^e \\ u_2^e \end{bmatrix} = \begin{bmatrix} 1 & x_a \\ 1 & x_b \end{bmatrix} \begin{bmatrix} c_1^e \\ c_2^e \end{bmatrix} \tag{2.53}$$

We can rewrite the equations as

$$\begin{bmatrix} c_1^e \\ c_2^e \end{bmatrix} = \frac{1}{x_b - x_a} \begin{bmatrix} x_b & -x_a \\ -1 & 1 \end{bmatrix} \begin{bmatrix} u_1^e \\ u_2^e \end{bmatrix} \tag{2.54}$$

Hence we get,

$$c_1^e = \frac{1}{h_e}(x_b u_1^e - x_b u_2^e) \tag{2.55}$$

$$c_2^e = \frac{1}{h_e}(-u_1^e + u_2^e) \tag{2.56}$$

If we let,

$$\alpha_1^e = x_b$$
$$\alpha_2^e = -x_a$$
$$\beta_1^e = -1$$
$$\beta_2^e = 1$$

$$c_1^e = \frac{1}{h_e}(\alpha_1^e u_1^e + \alpha_2^e u_2^e) \tag{2.57}$$

$$c_2^e = \frac{1}{h_e}(\beta_1^e u_1^e + \beta_2^e u_2^e) \tag{2.58}$$

Hence,

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

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

$$U_h^e(x) = \sum_{j=1}^{2} \frac{1}{h_e}[\alpha_j^e + \beta_j^e x]u_j^e \tag{2.59}$$

$$U_h^e(x) = \sum_{j=1}^{2} \psi_j^e(x)u_j^e \tag{2.60}$$

where $\psi_j^e(x)$ are known as interpolation functions. It can be seen that $\psi_j^e(x)$ are simply Lagrange interpolation functions and can be written quite succinctly as,

$$\psi_1^e(\bar{x}) = 1 - \frac{\bar{x}}{h_e}, \qquad \psi_2^e(\bar{x}) = \frac{\bar{x}}{h_e} \tag{2.61}$$

where $\bar{x} = x - x_a$

From eqn 2.36 we obtain

$$K_{ij}^e = B(\psi_i^e, \psi_j^e) = \int_{x_a}^{x_b} a_e \frac{d\psi_j^e}{dx}\frac{d\psi_i^e}{dx}dx \tag{2.62}$$

$$F_i^e = l(\psi_i^e) = \int_{x_a}^{x_b} f\psi_i^e dx \tag{2.63}$$

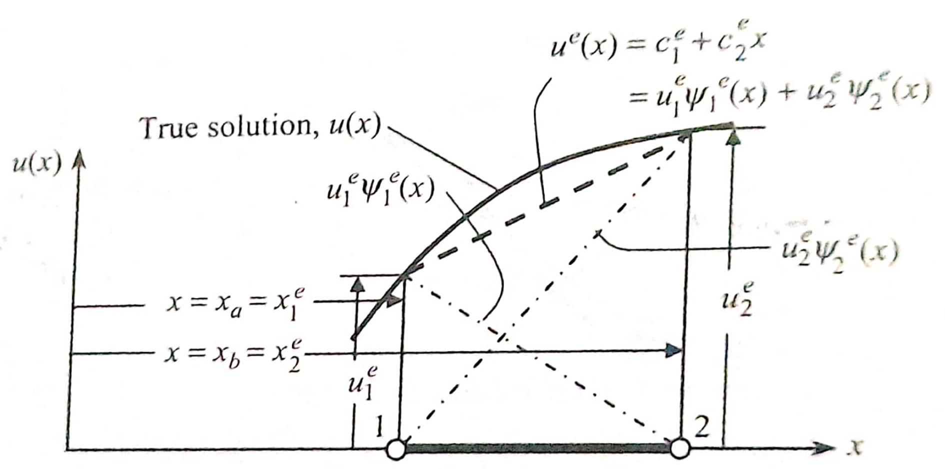


Figure 2.5: Linear Lagrange Interpolation Functions

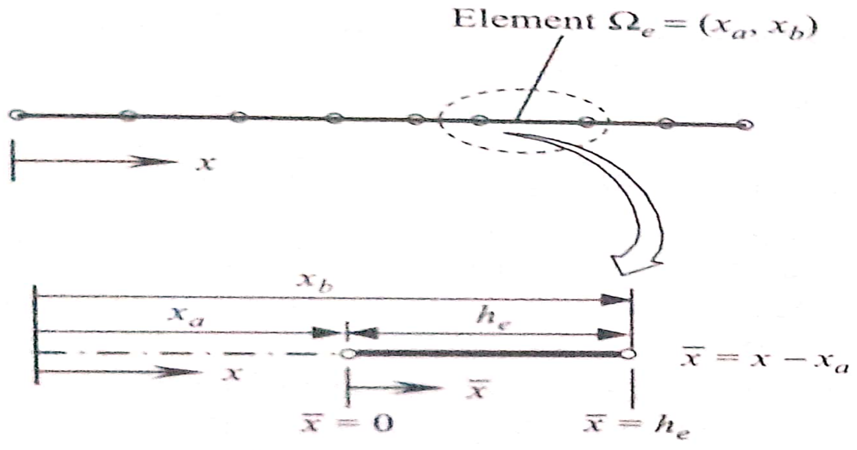


Figure 2.6: Local Coordinate system for elements

Taking local coordinate $\bar{x}$,

$$K_{ij}^e = \int_0^{h_e} aE \frac{d\psi_j^e}{d\bar{x}} \frac{d\psi_i^e}{d\bar{x}} d\bar{x} \tag{2.64}$$

$$F_i^e = \int_0^{h_e} f\psi_i^e d\bar{x} \tag{2.65}$$

subsituting eqn 2.61 ,

$$K_{11}^e = \int_0^{h_e} a_e \left(-\frac{1}{h_e}\right)\left(-\frac{1}{h_e}\right) d\bar{x} \tag{2.66}$$

$$\implies K_{11}^e = \frac{a_e}{h_e} \tag{2.67}$$

similarly,

$$K_{12}^e = K_{21}^e = -\frac{a_e}{h_e} \tag{2.68}$$

$$K_{22}^e = \frac{a_e}{h_e} \tag{2.69}$$

we can also assume constant f_e to obtain ,

$$f_1^e = f_2^e = \frac{f_e h_e}{2} \tag{2.70}$$

Hence,

$$[K^e] = \frac{a_e}{h_e}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \qquad [f^e] = \frac{f_e h_e}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{2.71}$$

Summarizing we have obtained the following system,

$$\frac{a_e}{h_e}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} u_1^e \\ u_2^e \end{bmatrix} = \frac{f_e h_e}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} Q_1^e \\ Q_2^e \end{bmatrix} \tag{2.72}$$

The above linear system, cannot be solved for each element as it has more variables than there are equations. Thus, we will now combine back our elements to obtain the entire problem domain.

We will be putting additional constraints that $u(x)$ is continuous and source terms are balanced.

Mathematically, If node i of Ω_a , node j of Ω_b and node k of Ω_c , then

$$u_i^a = u_j^b = u_k^c \tag{2.73}$$

$$Q_i^a + Q_j^b + Q_k^c = Q_I \tag{2.74}$$

where , Q_I is externally applied point source

The corresponding matrices of each element must also be added to ensure consistency when external point source are added. For example, In the figure below,

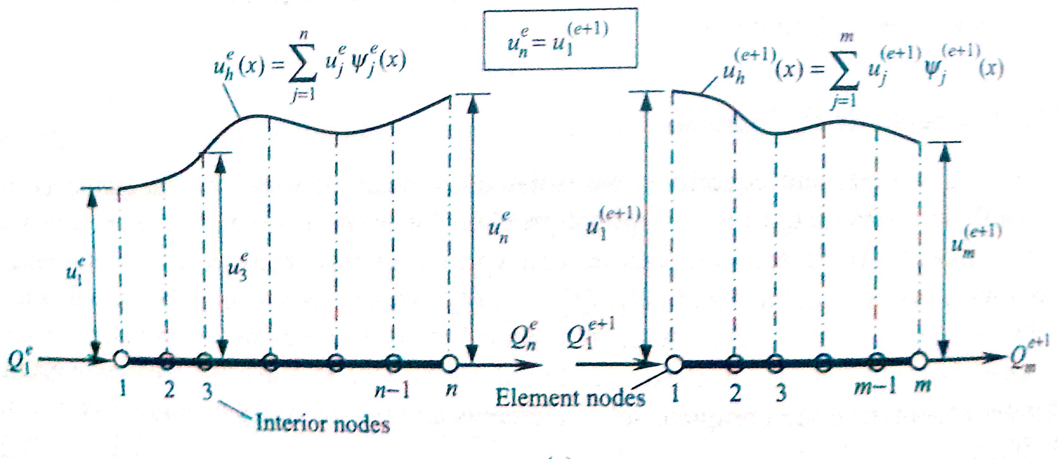
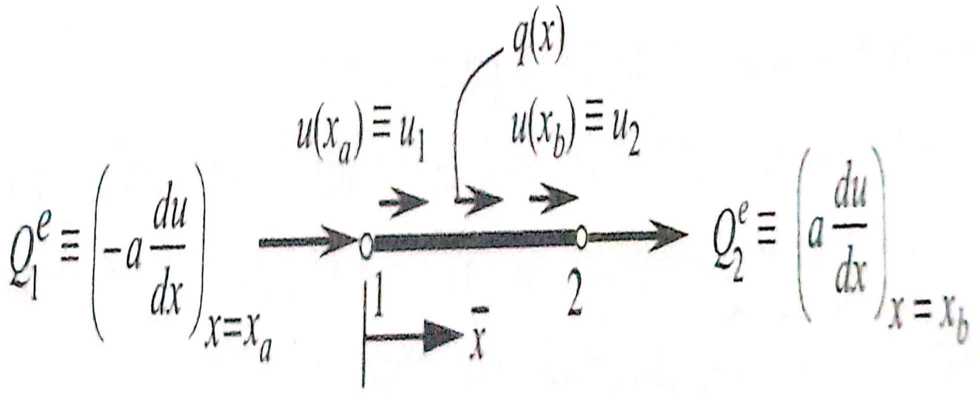


Figure 2.7: Example of basic Assembly of elements



$q(x)$
$u(x_a) \equiv u_1$
$u(x_b) \equiv u_2$
$Q_1^e \equiv \left(-a\frac{du}{dx}\right)_{x=x_a}$
1
2
$\bar{x}$
$Q_2^e \equiv \left(a\frac{du}{dx}\right)_{x=x_b}$

Figure 2.8: Assembly process for Secondary Variables

$$u_1^1 = U_1, \quad u_2^1 = u_1^2 = U_2, \dots u_2^i = u_1^{i+1} = U_i, \dots u_2^N = U_{N+1} \tag{2.75}$$

$$Q_2^i + Q_1^{i+1} = 0 \quad \text{if no external point source is applied.} \tag{2.76}$$

$$\text{otherwise} \quad Q_2^i + Q_1^{i+1} = Q_I \tag{2.77}$$

where Q_I is the magnitude of the external point source applied.

Adding eqn 2.72 for sucessive elements, we get

$$\begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 & \dots & 0 & 0 \\ K_{21}^1 & K_{22}^1 + K_{11}^2 & K_{12}^2 & \dots & 0 & 0 \\ 0 & K_{21}^2 & K_{22}^2 + K_{11}^3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & K_{22}^{N-1} + K_{22}^N & K_{12}^N \\ 0 & 0 & 0 & \dots & K_{21}^N & K_{22}^N \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{N-1} \\ U_N \end{bmatrix}$$
$$= \begin{bmatrix} f_1^1 \\ f_2^1 + f_1^2 \\ f_2^2 + f_1^3 \\ \vdots \\ f_2^{N-1} + f_1^N \\ f_2^N \end{bmatrix} + \begin{bmatrix} Q_1^1 \\ Q_2^1 + Q_1^2 \\ Q_2^2 + Q_1^3 \\ \vdots \\ Q_2^{N-1} + Q_1^N \\ Q_2^N \end{bmatrix}$$

Now we can impose the boundary conditions on the by providing the values of U_i and Q_i and solve the equation to obtain the solution at the nodal values. The final solution

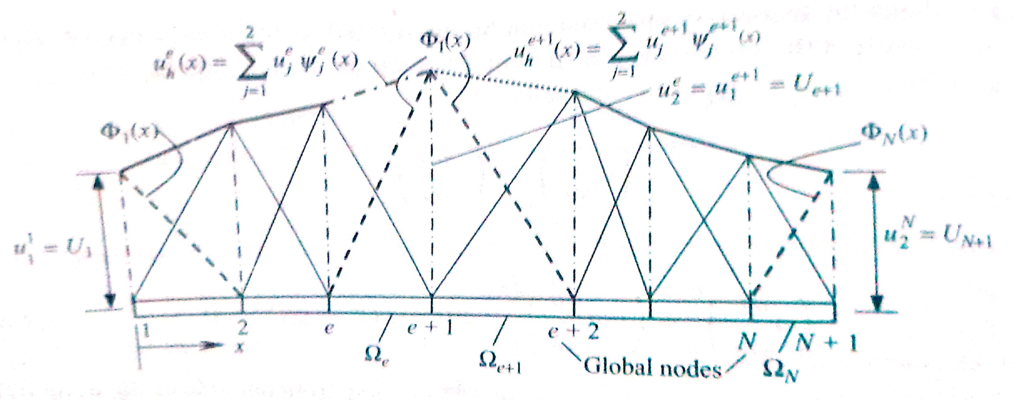


Figure 2.9: Obtaining Global solution from element solutions

can be written as

$$u(x) = \begin{cases} u_h^1(x) = \sum_{j=1}^{n} u_j^1 \psi_j^1(x) & x \in \Omega^1 \\ u_h^2(x) = \sum_{j=1}^{n} u_j^2 \psi_j^2(x) & x \in \Omega^2 \\ \vdots \\ u_h^N(x) = \sum_{j=1}^{n} u_j^N \psi_j^N(x) & x \in \Omega^N \end{cases}$$

In case of Truss, 2.71 can be written as

$$[K^e] = \frac{E_e A_e}{h_e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \tag{2.78}$$

as $a_e = A_e E_e$ for a bar

We can embed this truss element along x-axis by placing it in 2D plane. then,

$$\frac{E_e A_e}{h_e} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \bar{u_1^e} \\ \bar{v_2^e} \\ \bar{u_1^e} \\ \bar{v_2^e} \end{bmatrix} = \begin{bmatrix} \bar{F_1^e} \\ 0 \\ \bar{F_1^e} \\ 0 \end{bmatrix} \tag{2.79}$$

This can be expressed as

$$[\bar{K^e}]\{\bar{\Delta^e}\} = \{\bar{F^e}\} \tag{2.80}$$

Here we have used the $\bar{x}$ to emphasize that these are local coordinates of the element. also $\bar{F_1^e} = \bar{f_1^e} + \bar{Q_1^e}$. $\bar{u_1^e}$ and $\bar{v_2^e}$ denote the displacement along local x and y -axis respectively. Now let us suppose that the element is placed making angle θ_e with the x-axis. Then The transformation matrix to convert from local to global are,

$$\begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} \cos\theta_e & -\sin\theta_e \\ \sin\theta_e & \cos\theta_e \end{bmatrix} \begin{bmatrix} \bar{x_e} \\ \bar{y_e} \end{bmatrix} \tag{2.81}$$

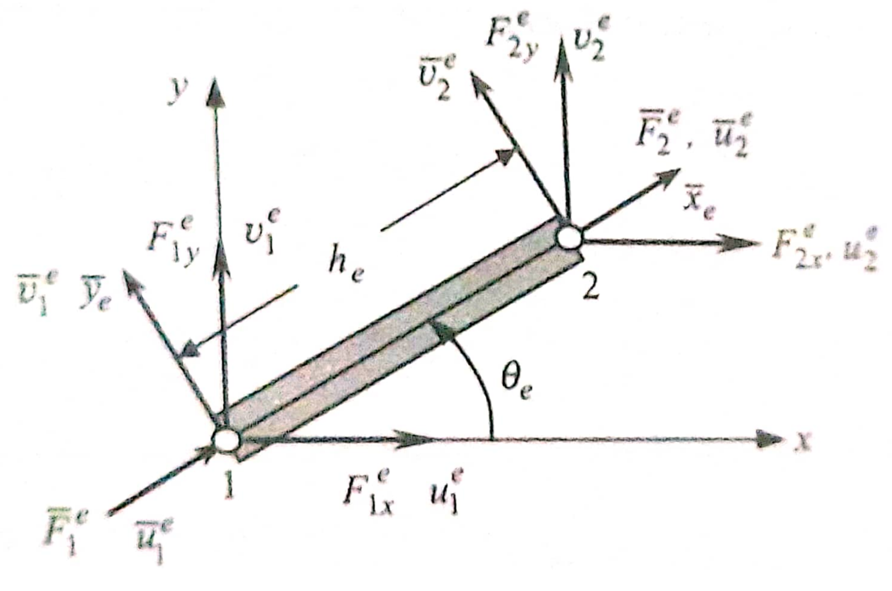


Figure 2.10: A typical Truss Element

Using the above transformation for displacements and forces

$$\begin{bmatrix} \bar{u_1^e} \\ \bar{v_2^e} \\ \bar{u_1^e} \\ \bar{v_2^e} \end{bmatrix} = \begin{bmatrix} \cos\theta_e & -\sin\theta_e & 0 & 0 \\ \sin\theta_e & \cos\theta_e & 0 & 0 \\ 0 & 0 & \cos\theta_e & -\sin\theta_e \\ 0 & 0 & \sin\theta_e & \cos\theta_e \end{bmatrix} \begin{bmatrix} u_1^e \\ v_2^e \\ u_1^e \\ v_2^e \end{bmatrix} \tag{2.82}$$

$$\{\bar{\Delta^e}\} = [T^e]\{\Delta^e\} \tag{2.83}$$

$$\{\bar{F^e}\} = [T^e]\{F^e\} \tag{2.84}$$

$$[\bar{K^e}][T^e]\{\Delta^e\} = [T^e]\{F^e\} \tag{2.85}$$

$$\implies [T^e]^T[\bar{K^e}][T^e]\bar{\Delta}^e = [T^e]^T\{\bar{F^e}\} \tag{2.86}$$

$$\tag{2.87}$$

let

$$\{F^e\} = [T^e]^T\{\bar{F^e}\} \tag{2.88}$$

$$[K^e] = [T^e]^T[\bar{K^e}][T^e] \tag{2.89}$$

So finally we can write,

$$[K^e]\{\Delta^e\} = \{F^e\} \tag{2.90}$$

where,

$$[K^e] = \frac{E_e A_e}{h_e} \begin{bmatrix} \cos^2\theta_e & \frac{1}{2}\sin 2\theta_e & -\cos^2\theta_e & -\frac{1}{2}\sin 2\theta_e \\ \frac{1}{2}\sin 2\theta_e & \sin^2\theta_e & -\frac{1}{2}\sin 2\theta_e & -\sin^2\theta_e \\ -\cos^2\theta_e & -\frac{1}{2}\sin 2\theta_e & \cos^2\theta_e & \frac{1}{2}\sin 2\theta_e \\ -\frac{1}{2}\sin 2\theta_e & -\sin^2\theta_e & \frac{1}{2}\sin 2\theta_e & \sin^2\theta_e \end{bmatrix} \tag{2.91}$$

Further process is identical to the assembly and solving linear equations as we explained above.

More details can be obtained from [8].

2.3 Python Implementation

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

num	pos_x	pos_y	dis_x	dis_y	f_x	f_y
1	0.0	0.0	0.0	0.0	0.0	0.0
2	2.0	0.0	0.0	0.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	0.0
6	8.0	2.918	1.2484	-0.60462	0.2	-0.1
7	6.0	2.1838	0.70085	-0.21838	0.0	-0.1
num	length	area	ym	theta	node_a	node_b
1	2.0	1.0	1.0	0.0	1	2
2	2.0	1.0	1.0	0.0	2	3
14	2.1838	1.0	1.0	1.5707963267948966	4	7
15	3.5325	1.0	1.0	0.9773840320509742	4	6

An sample of output csv file generated is as follows

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.

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. We took inspiration of ideas from [9] 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)
```

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.

```python
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))
t.pendown()
t.goto((ele.node_b.pos_x + ele.node_b.dis_x), (ele.node_b.
 pos_y + ele.node_b.dis_y))
t.penup()

t.speed(0)
for node in self.node_list:
t.penup()
t.goto((node.pos_x + node.dis_x), (node.pos_y + node.dis_y)
 )
t.pendown()
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

For the purpose and verification by a concrete example and demonstration of potential application of the method, we designed a custom truss and studied its deformations under the prescribed forces . These forces and the overall structure of truss is as below.

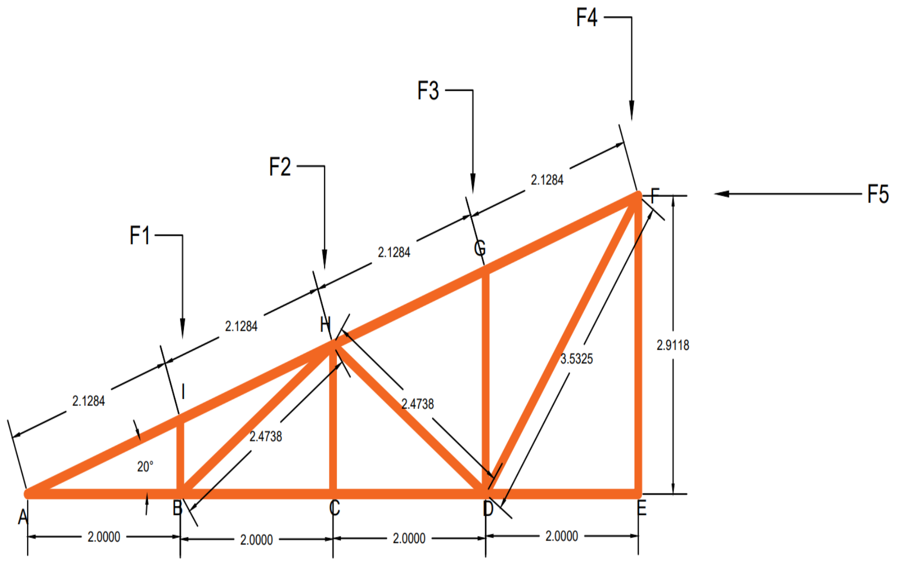


Figure 3.1: Truss

Our module drew the structure of truss as

Implementing the problem in python gave the following structure after deformation.

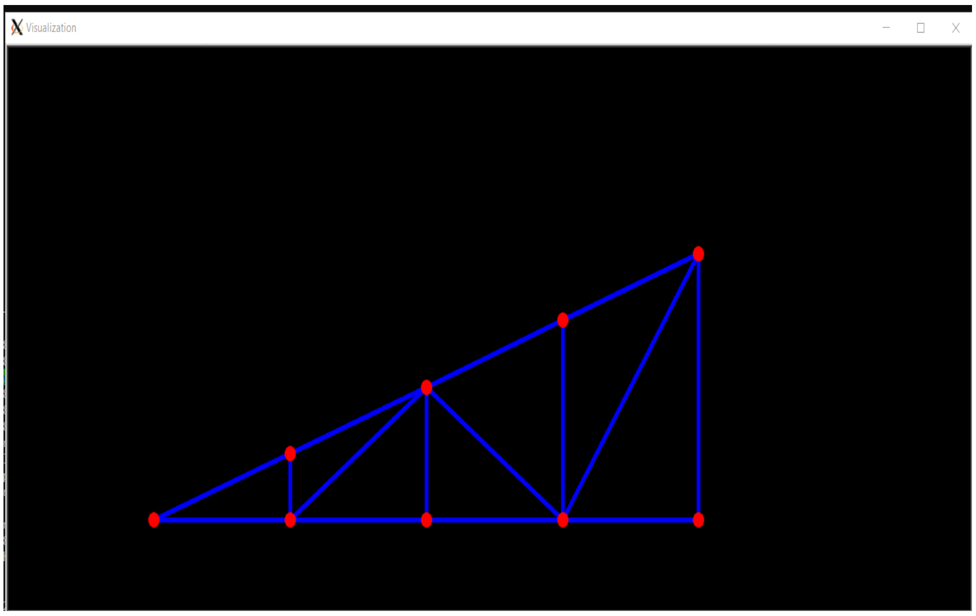


Figure 3.2: Above Truss as drawn by truss_solve module

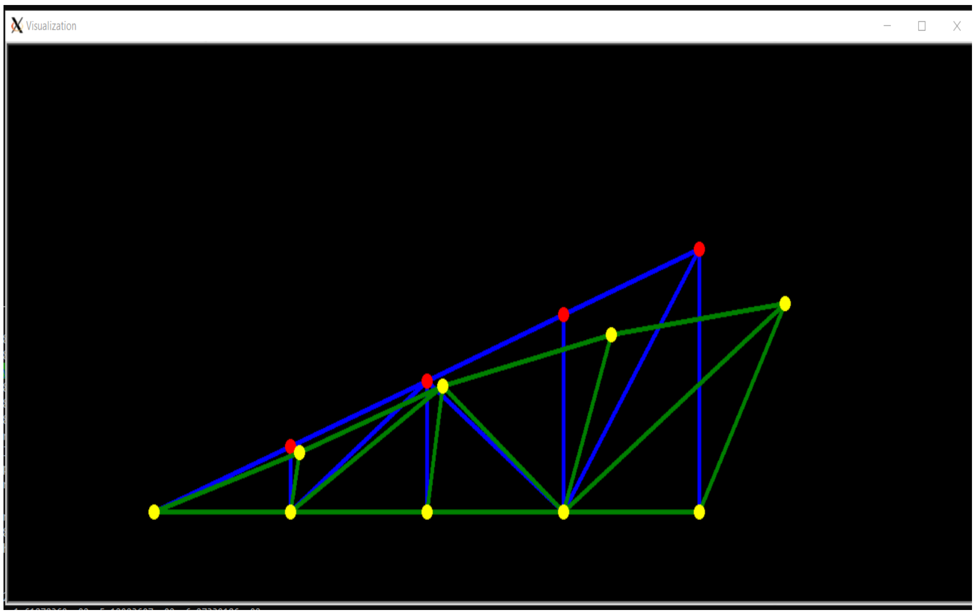


Figure 3.3: Above Truss along with deformation

We also verified our solution in Frame3dd. Frame3dd is a free open-source software for static and dynamic structural analysis of 2D and 3D frames and trusses with elastic and geometric stiffness. It is a copyright of

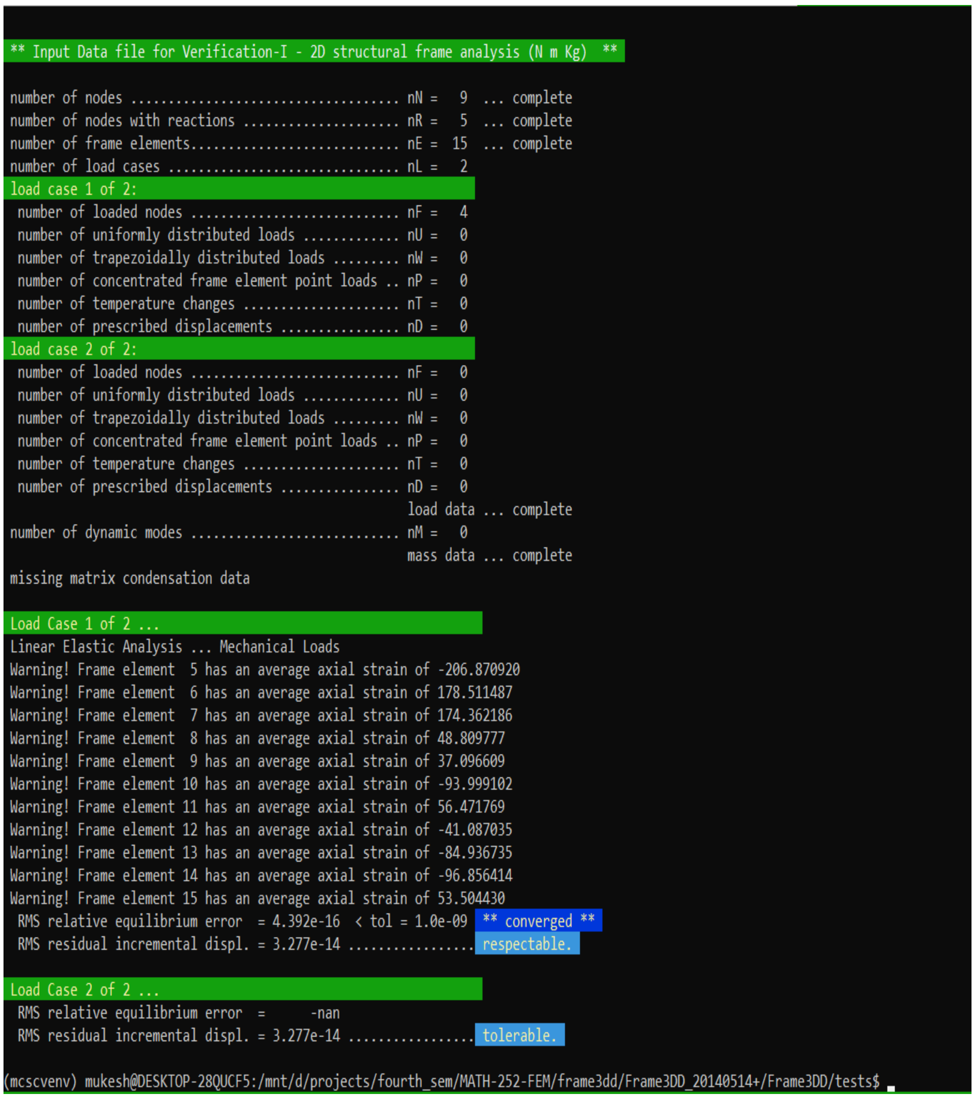
Department of Civil and Environmental Engineering

Edmund T. Pratt School of Engineering

Duke University - Box 90287, Durham, NC 27708-0287

Henri P. Gavin, Ph.D., P.E.,

After verification in the software, the following structure was obtained.



```
** Input Data file for Verification-I - 2D structural frame analysis (N m Kg)  **

number of nodes .................................. nN =   9  ... complete
number of nodes with reactions ................... nR =   5  ... complete
number of frame elements.......................... nE =  15  ... complete
number of load cases ............................. nL =   2
load case 1 of 2:
 number of loaded nodes .......................... nF =   4
 number of uniformly distributed loads ............ nU =   0
 number of trapezoidally distributed loads ........ nW =   0
 number of concentrated frame element point loads .. nP =   0
 number of temperature changes .................... nT =   0
 number of prescribed displacements ............... nD =   0
load case 2 of 2:
 number of loaded nodes .......................... nF =   0
 number of uniformly distributed loads ............ nU =   0
 number of trapezoidally distributed loads ........ nW =   0
 number of concentrated frame element point loads .. nP =   0
 number of temperature changes .................... nT =   0
 number of prescribed displacements ............... nD =   0
                                                     load data ... complete
number of dynamic modes .......................... nM =   0
                                                     mass data ... complete
missing matrix condensation data

Load Case 1 of 2 ...
Linear Elastic Analysis ... Mechanical Loads
Warning! Frame element  5 has an average axial strain of -206.870920
Warning! Frame element  6 has an average axial strain of 178.511487
Warning! Frame element  7 has an average axial strain of 174.362186
Warning! Frame element  8 has an average axial strain of 48.809777
Warning! Frame element  9 has an average axial strain of 37.096609
Warning! Frame element 10 has an average axial strain of -93.999102
Warning! Frame element 11 has an average axial strain of 56.471769
Warning! Frame element 12 has an average axial strain of -41.087035
Warning! Frame element 13 has an average axial strain of -84.936735
Warning! Frame element 14 has an average axial strain of -96.856414
Warning! Frame element 15 has an average axial strain of 53.504430
 RMS relative equilibrium error  = 4.392e-16  < tol = 1.0e-09  ** converged **
 RMS residual incremental displ. = 3.277e-14 ................ respectable.

Load Case 2 of 2 ...
 RMS relative equilibrium error  =      -nan
 RMS residual incremental displ. = 3.277e-14 ................ tolerable.

(mcscvenv) mukesh@DESKTOP-28QUCF5:/mnt/d/projects/fourth_sem/MATH-252-FEM/frame3dd/Frame3DD_20140514+/Frame3DD/tests$
```

Figure 3.4: Solving using Frame3DD

Results in python	Results in frame3dd
12.48407187	12.1552098
-6.04631381	-6.03649343
7.00856832	6.66504758
-2.1838	-2.11534409
2.35025907	2.16352748
-0.56650909	-0.59822723
-0.7279	-0.68431346

Table 3.1: Comparision of results obtained in python to that in frame3dd for displacement.

Table 3.1 shows the percentage error in each node for displacement. Going from left to right we get displacement a as obtained in python, displacement as obtained in frame3dd and percentage error in each displacement.

The average percentage error in displacement is 1.73%.

CHAPTER 4

CONCLUSIONS

Finally, We learned about the FEM, its variations and applications to various fields. We chose a truss structure and applied FEM to solve a 2D truss with straight bar elements that only undergo axial deformation. Initially we solved various examples manually and then implemented the same problems on a computer in python and verified results.We then verified results for the truss structure and were able to write a general code that would solve any other truss structures given essential parameters .Finally, we were able to visualize the initial problem along with the deformed structure.

This journey made us appreciate the logical and theoretical framework behind Finite Element analysis. Although we barely utilized a fraction of its potential, we gained experience in the basic ideas and the general approach in coding such problems in computer. We are confident that we can extend this idea in future by expanding the domain and complexity of the problems we tackle with this method.

CONTRIBUTIONS

1. Samrajya Raj Acharya:
    - worked on engineering aspects of theory.
    - helped find bugs in computer implementation.
    - worked on writing report of the project.
2. Bishesh Kafle:
    - worked on aspects of theory
    - implemented the visualization module
    - helped implement various parts of solution
    - worked on writing report of the project.
3. Priyanka Panta:
    - worked on manual solution of examples.
    - worked on verification of solution obtained.
    - worked on writing report of the project.
4. Mukesh Tiwari
    - worked on theory
    - implemented the computer implementation of truss module.
    - worked on writing theory portion of the report.

Bibliography

[1] Alexander Hrennikoff. "Solution of problems of elasticity by the framework method". In: (1941).

[2] Richard Courant. "Variational methods for the solution of problems of equilibrium and vibrations". In: *Bulletin of the American mathematical Society* 49.1 (1943), pp. 1–23.

[3] Vinod Bandela and Saraswathi Kanaparthi. "Finite Element Analysis and Its Applications in Dentistry". In: *Finite Element Methods and Their Applications*. IntechOpen London, UK, 2020.

[4] Erwin Stein. "Olgierd C. Zienkiewicz, a pioneer in the development of the finite element method in engineering science". In: *Steel Construction: Design and Research* 2.4 (2009), pp. 264–272.

[5] Gilbert Strang and George J Fix. "An analysis of the finite element method(Book-An analysis of the finite element method.)" In: *Englewood Cliffs, N. J., Prentice-Hall, Inc., 1973. 318 p* (1973).

[6] Olek C Zienkiewicz, Robert L Taylor, and Jian Z Zhu. *The finite element method: its basis and fundamentals*. Elsevier, 2005.

[7] Ludmila Banakh. "Oscillations properties of the dynamic fractal structures". In: *Journal of Sound and Vibration* 520 (2022), p. 116541.

[8] John Tinsley Oden and Junuthula Narasimha Reddy. *An introduction to the mathematical theory of finite elements*. Courier Corporation, 2012.

[9] Peter I Kattan. *MATLAB guide to finite elements: an interactive approach*. Springer Science & Business Media, 2010.

APPENDIX

We will attach all the code created for the module of this packet in this chapter. The updated version can be found at `https://github.com/mukeshdroid/trussfem`

node.py

```python
#class for nodes
import math
import numpy as np

class node:

# init method or constructor
#nan values are used to indicate if the value needs to be determined.
def __init__(self, num, pos_x , pos_y , dis_x , dis_y, f_x , f_y):
self.num = num
self.pos_x = pos_x
self.pos_y = pos_y
self.dis_x = dis_x
self.dis_y = dis_y
self.f_x = f_x
self.f_y = f_y

def print(self):
print('num = ', self.num , '\n' , 'pos_x =', self.pos_x , '\n pos_y = ' , self.pos_y , '\n dis_x = ' , self.dis_x , '\n dis_y = ', self.dis_y , '\n f_x = ', self.f_x, '\n f_y = ', self.f_y)
return 0

def value(self):

return [self.num , self.pos_x , self.pos_y , self.dis_x , self.dis_y, self.f_x, self.f_y]
```

ele.py

```python
#class for elements
import math
import numpy as np
import node

class ele:

# init method or constructor
```

```
#nan values are used to indicate if the value needs to be
   determined.
def __init__(self, num, length , area , ym , theta, node_a ,
   node_b):
self.num = num
self.length = length
self.area = area
self.ym = ym
self.theta = theta
self.node_a = node_a
self.node_b = node_b

def print(self):
print('num = ', self.num , '\n' , 'length=', self.length , '\
   n area = ' , self.area , '\n youngs modulus = ' , self.ym
   , '\n theta = ', self.theta , '\n node_a = ', self.node_a,
    '\n node_b = ', self.node_b)
return 0

def value(self):
return [self.num, self.length , self.area , self.ym , self.
   theta , self.node_a.num, self.node_b.num]

def stiff(self):
ang = self.theta
c2 = (math.cos(ang))**2
cs = math.sin(ang) * math.cos(ang)
s2 = (math.sin(ang))**2
mat = np.array([[c2 , cs , -c2 , -cs],
[cs , s2 , -cs , -s2],
[-c2 , -cs , c2 , cs],
[-cs , -s2 , cs , s2]])
return ((self.ym * self.area) / self.length) * mat
```

node.py

```
#class for nodes
import math
import numpy as np

class node:

# init method or constructor
#nan values are used to indicate if the value needs to be
   determined.
def __init__(self, num, pos_x , pos_y , dis_x , dis_y, f_x ,
   f_y):
self.num = num
self.pos_x = pos_x
self.pos_y = pos_y
self.dis_x = dis_x
self.dis_y = dis_y
self.f_x = f_x
self.f_y = f_y

def print(self):
print('num = ', self.num , '\n' , 'pos_x =', self.pos_x , '\n
    pos_y = ' , self.pos_y , '\n dis_x = ' , self.dis_x , '\n
    dis_y = ', self.dis_y , '\n f_x = ', self.f_x, '\n f_y =
   ', self.f_y)
return 0

```

```
def value(self):

return [self.num , self.pos_x , self.pos_y , self.dis_x ,
   self.dis_y, self.f_x, self.f_y]
```

trusssolve.py

```
import sys
from node import node
from ele import ele
import turtle
import numpy as np
import math
import copy
import pyautogui
import time
import csv
import os

node_file = sys.argv[1]
ele_file = sys.argv[2]

class truss:
node_list = []
ele_list = []
dis_list = []
f_list = []
GK = []

def __init__(self):

with open(node_file) as csvfile1:
reader = csv.reader(csvfile1)
for row in reader:
num = int(row[0])
pos_x = float(row[1])
pos_y = float(row[2])
dis_x = np.nan if row[3] == 'nan' else float(row[3])
dis_y = np.nan if row[4] == 'nan' else float(row[4])
f_x = float(row[5])
f_y = float(row[6])
temp1 = node(num, pos_x, pos_y, dis_x, dis_y, f_x, f_y)
self.node_list.append(temp1)

with open(ele_file) as csvfile2:
reader = csv.reader(csvfile2)
for row in reader:
num = int(row[0])
length = float(row[1])
area = float(row[2])
ym = float(row[3])
theta = math.pi * float(row[4][3:])
node_a = self.node_list[int(row[5]) - 1]
node_b = self.node_list[int(row[6]) - 1]
temp2 = ele(num, length, area, ym, theta, node_a, node_b)
self.ele_list.append(temp2)

def globalstiff(self):
# dimension of global matrix
dim = 2 * len(self.node_list)
```

```
eles = self.ele_list
# generate and store the element-wise stiffness matrices

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]]
self.GK = GK

def solve(self):
# generate global matrix by calling globalstiff method
self.globalstiff()

dis_list = []
for n in self.node_list:
dis_list.append(n.dis_x)
dis_list.append(n.dis_y)

dis = np.array(dis_list)

f_list = []
for n in self.node_list:
f_list.append(n.f_x)
f_list.append(n.f_y)

f = np.array(f_list)

#check for undetermined values in dis and create linear eqns

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

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

index_list = list(range(0,2*len(self.node_list)))
for i in range(0,2*len(self.node_list)):
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
```

```python
GK_dis = np.delete(GK_dis,del_row,1)

ans_dis = np.linalg.solve(GK_dis, f_dis)

for i in range(0, len(ans_dis)):
dis[index_list[i]] = ans_dis[i]
self.dis_list = dis

ans_force = np.dot(self.GK, dis)
self.force_list = ans_force

k = 0
for i in self.node_list:
i.dis_x = self.dis_list[k]
k = k + 1
i.dis_y = self.dis_list[k]
k = k + 1

#print(self.dis_list)
#print(self.force_list)

def visualize(self, height=560, width=1300, grid=12, speed=7, delay=2):
try:
width, height = pyautogui.size()
except:
None

turtle.title("Visualization")
turtle.setup(width, height)
turtle.bgcolor("black")
ratio = height / width
turtle.setworldcoordinates(-2, -2 * ratio, grid, ratio * grid)

t = turtle.Turtle()
t.speed(speed)
t.hideturtle()
t.pensize(5)
t.pencolor("blue")

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()

t.speed(0)
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()

# def fun():
#     return None
# turtle.onclick(fun, btn=1, add=None)
time.sleep(delay)

t.speed(speed)
# 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))
t.pendown()
t.goto((ele.node_b.pos_x + ele.node_b.dis_x), (ele.node_b.
    pos_y + ele.node_b.dis_y))
t.penup()

t.speed(0)
for node in self.node_list:
t.penup()
t.goto((node.pos_x + node.dis_x), (node.pos_y + node.dis_y))
t.pendown()
t.dot(15, "yellow")
turtle.Screen().exitonclick()

def writeoutput(self):
with open('./csv_files/output.csv' , 'w') as csvfile3:
writer = csv.writer(csvfile3)
writer.writerow(['num','pos_x','pos_y', 'dis_x','dis_y',  '
    f_x', 'f_y'])
for node in self.node_list:
writer.writerow(node.value())
writer.writerow(['num', 'length' , 'area' , 'ym', 'theta', '
    node_a' , 'node_b'])
for ele in self.ele_list:
writer.writerow(ele.value())

truss1 = truss()
truss1.solve()
truss1.writeoutput()
truss1.visualize()
```