# FM132 : Foundations of Physical World
# Trusses on Computers

Arnav Rustagi U20220021

February 18, 2023

# Contents

**1 Introduction** **3**

**2 How to use** **4**
2.1 Programming the truss . . . . . 4
2.2 Plotting the truss . . . . . 5
2.3 Visualising the forces on the truss . . . . . 6
2.4 Additional Features . . . . . 8
2.5 Verifying the program . . . . . 9

**3 Core logic** **10**
3.1 Core classes . . . . . 10
3.1.1 Truss . . . . . 10
3.1.2 Point . . . . . 11
3.1.3 Line . . . . . 11
3.1.4 Force . . . . . 12
3.1.5 Equation . . . . . 13
3.2 Core Logic . . . . . 14

**4 Output** **15**
4.1 Sample 6.4 . . . . . 15
4.2 6.1.17 . . . . . 16
4.3 6.1.20 . . . . . 17

# 1 Introduction

Trusses are extremely strong and rigid structures made from beams, connecting the nodes, usually it is only on a two-dimensional plane, containing forces in the X and Y axes. Trusses are often assembled in complex manners, and to measure the structural integrity of the truss we need to assess the ammount of stress which is being applied on the different members of the truss and use that information to reinforce the weak points of the truss, to ensure that the strain experienced by the truss is minimum.

As the logic of solving a truss can be abstracted and generalised for each point, it is an ideal system which can be solved using computation, thus justifying the need for a computational analysis of trusses.

My implementation is written in python3, using numpy (for mathematical computation), pygame (for rendering), it is written in small modules each building upon each other and assisting each other and solving the truss, all the modules are written with coherence to the way we solve trusses, and have very straightforward naming

# 2 How to use

## 2.1 Programming the truss

I have created my own general purpose language to create trusses, one can simply program it in a `.truss` file, and it is as follows

```
% defining the points for the given truss
A (7,8)
B (9,6)
C (5,4)

% joining the points with beams
A-B
B-C
C-A

% Now we put the reactions in at the pin and roller joints
% defining a pin joint
RY1 A (0,1)
RX1 A (1,0)
% defining a roller joint
RY2 B (0,1)

% Now for an external force which represents all the forces which behave as loads
~ F1 A (0,1000)
~ F2 B (1000/sqrt(2),1000/sqrt(2))
```

In this the joints can be programmed using simple and verbose syntax

```
<Point Name> (X,Y)
```

The beams can be joined with

```
<Point Name 1>-<Point Name 2>
```

Now to define implicit reactions we use the following code

```
<Force Name> <Point Name> (X,Y)
```

where the Force Name is where name of the force, the Point Name is the point on which the force is acting and X and Y are the unit vector of the direction of the reaction forces
And finally for external forces which represent the load, they can be represented as

```
~ <Force Name> <Point Name> (X,Y)
```

Here the (X,Y) represent the components of the force, not the unit vector of the direction

**Please do not put any spaces in the components or the names as that will cause an error**

### 2.2 Plotting the truss

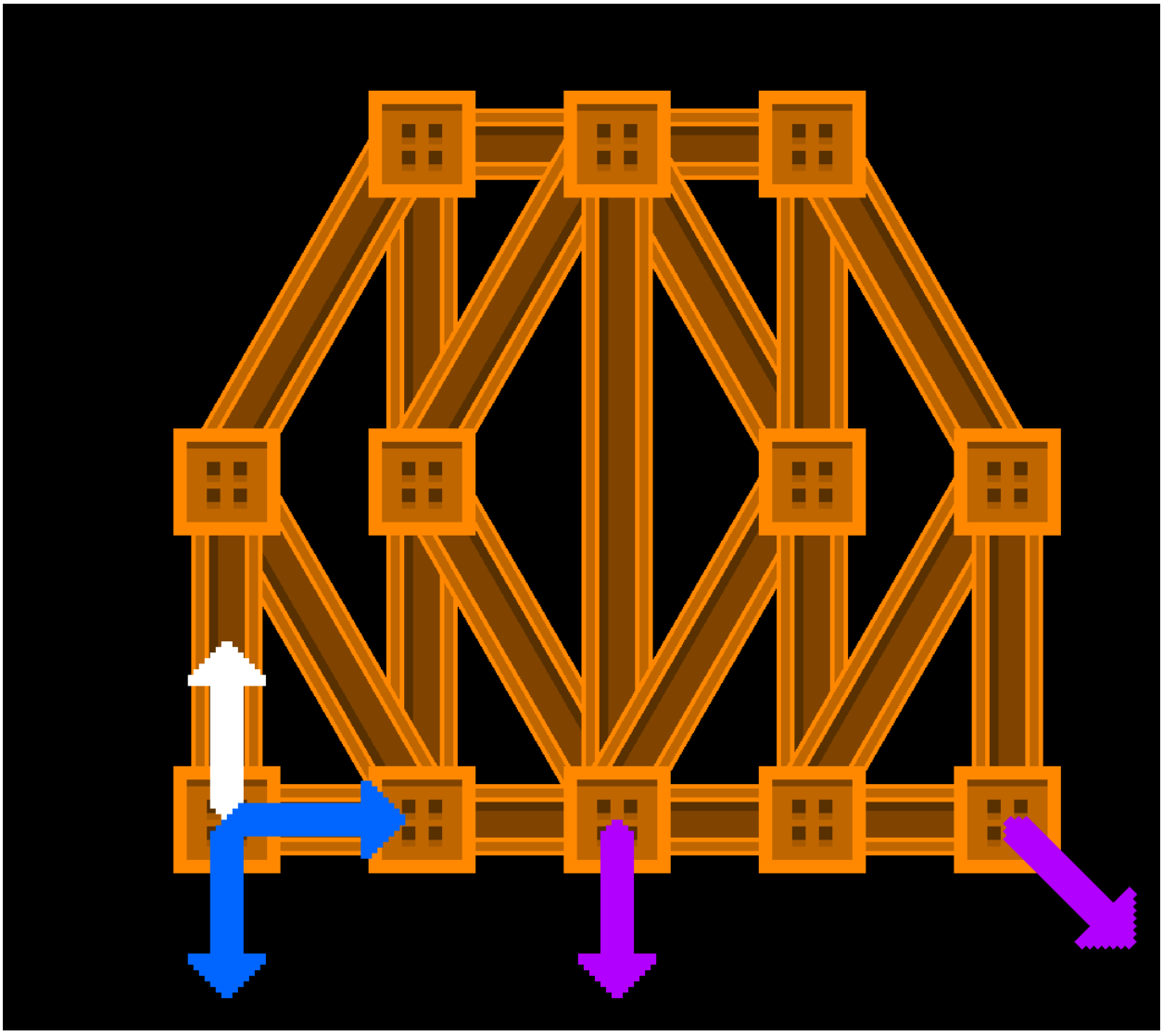
To plot the truss, first install the dependencies with the following line

```
$ pip install -r requirements.txt
```

Then to plot the truss we can execute the following program with the given flags

```
$ python3 main.py -p <file>.truss
```

the output will be like

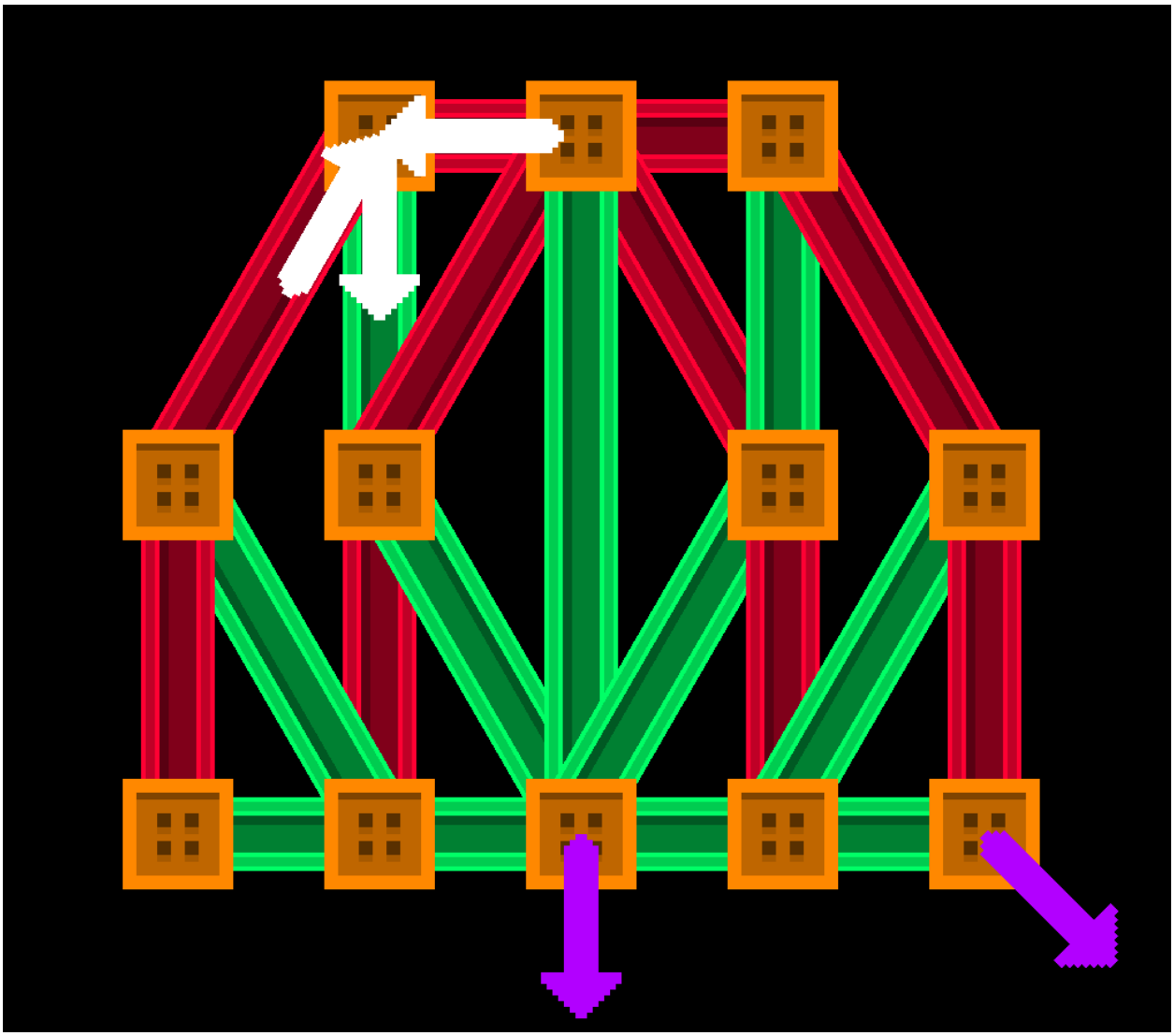


### 2.3 Visualising the forces on the truss

To visualise the results of solving the truss we can use the given command

```
$ python3 main.py <file>.truss
```

the output will be like



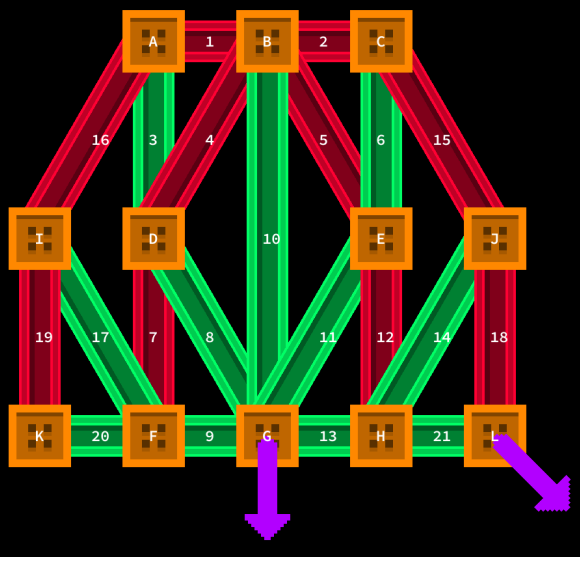
To get the values of the tensions on each of the rods we can execute the given command

```
$ python3 main.py <file>.truss -v
```

The output will be like

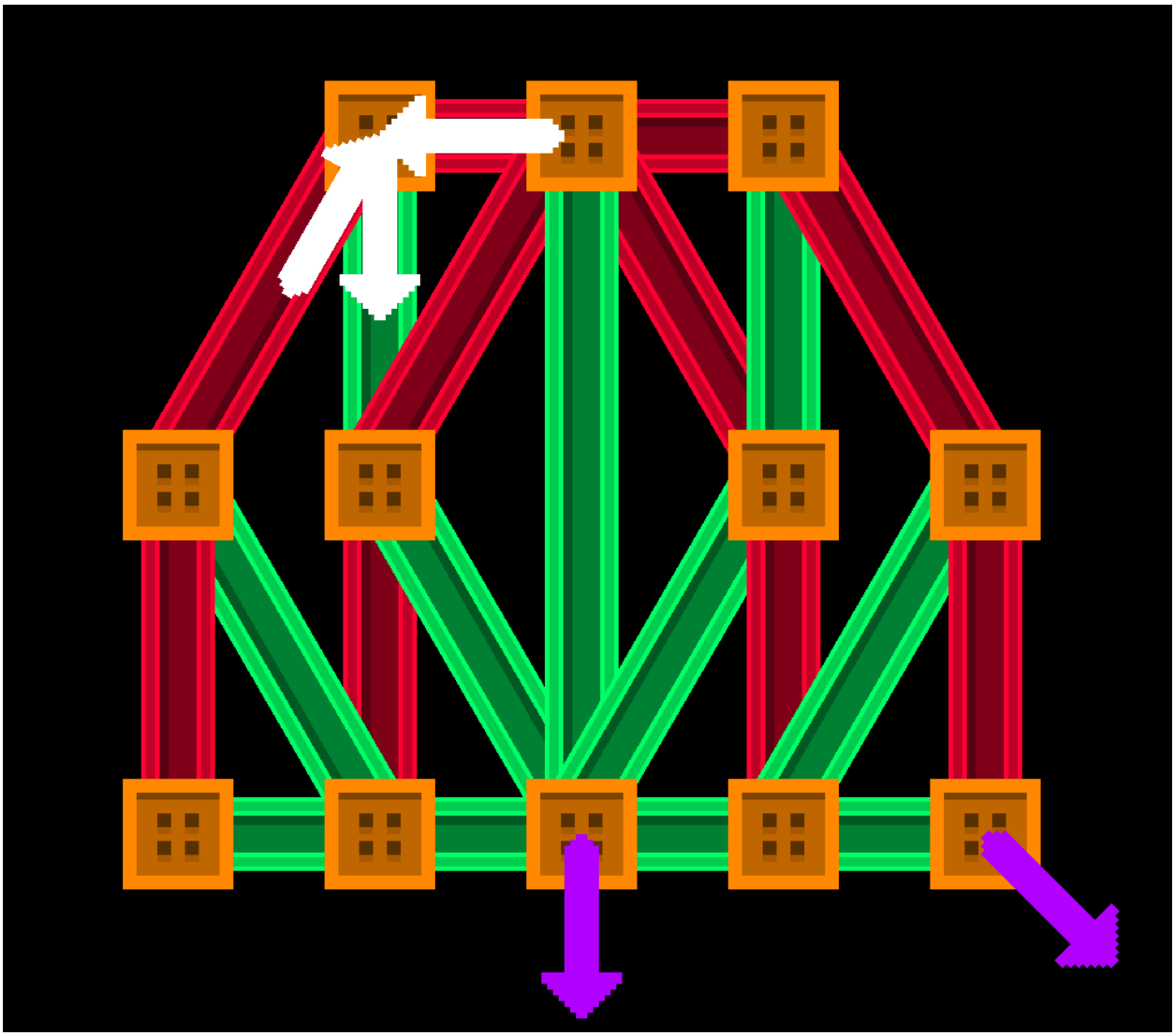
External Force : 3000 N

Tension in rod 1 : 433 N
Tension in rod 2 : 433 N
Tension in rod 3 : 750 N
Tension in rod 4 : 866 N
Tension in rod 5 : 866 N
Tension in rod 6 : 749 N
Tension in rod 7 : 750 N
Tension in rod 8 : 866 N
Tension in rod 9 : 2554 N
Tension in rod 10 : 1500 N
Tension in rod 11 : 866 N
Tension in rod 12 : 750 N
Tension in rod 13 : 2554 N
Tension in rod 14 : 866 N
Tension in rod 15 : 866 N
Tension in rod 16 : 866 N
Tension in rod 17 : 866 N
Tension in rod 18 : 1500 N
Tension in rod 19 : 1500 N
Tension in rod 20 : 2121 N
Tension in rod 21 : 2121 N



### 2.4 Additional Features

One can hover on each of the nodes to see the direction of the forces in the truss, tensions are represented by white arrows, reactions are represented by blue arrows, and the external forces are shown by purple arrows



Moreover my program can solve any `N` dimensional truss given that you give it the right constraints

### 2.5 Verifying the program

I have included the example programs mentioned in the project brief in the which include the following problems, these are present in the director `./verfication/`

- Sample 6.4
- 6.1.17
- 6.1.20
- 6.2.8

Moreover in the directory `./trusses/` I have the programs to plot several famous trusses like the `k-truss, james-webb telescope's truss, warren truss, pratt truss, howe truss` which you can run simulations of to check the program

# 3 Core logic

## 3.1 Core classes

As I said in my introduction my program was based on how I solved trusses, using the joint method, so first of all I modeled my main variables present in the problem solving in several classes which are as follows

- Truss (in truss.py)
- Point (in geometry.py)
- Line (in geometry.py)
- Force (in physics.py)
- Equation (in algebra.py)

So now delving into these classes in depth

### 3.1.1 Truss

The Truss class has the following constructor

```
def __init__(self, path: str, scale: int = 125, dimension:int = 2):
self.fpath = path
self.dimension = dimension
self.rods = []
self.points = {}
self.forces = {}
self.external_forces = {}
self.scale = scale
```

It contains the following instance variables

- `self.fpath`: This basically contains the path to the file containing the truss code
- `self.dimension`: This contains the dimension of the truss, by default it is 2
- `self.rods`: This is a list of all the beams/rods connecting each joint in the truss
- `self.points`: This is a dictionary, linking all the names of points to the Point object corresponding the name of the point
- `self.forces`: This is a dictionary linking all the names of the forces to the Force object corresponding to the name of the force, this includes the 2 tensions in every rod, the reactions and the external forces
- `self.external_forces`: This is a dictionary which contains the load forces acting on the system
- `self.scale`: This is the scale of the model, usually used for rendering

The truss object is the mother object to solve a truss all you need to call is the function

```
solved_truss = truss.balance()
```

Which will form the matrix, on itself and solve it, returning a solved_truss which you can render. The interpretter is called by the private method `__interpret(self, line)` where it is given each line from the `.truss` file from the function `__initialise_from_specfile(self)`, after interpretting the truss needs to internally link all the dependencies, i.e. link all the points to all the lines, initialise all the tensions, linke all the lines to the points etc, for this the function `__link_all` is a called, to finally prepare the truss to be used

### 3.1.2 Point

The constructor for the a Point object is as follows

```
def __init__(self, name: str, location: tuple, dimension:int):
self.name = name
self.dimension = dimension
assert(dimension==len(location))
self.location = location
self.lines = []
self.equation = []
self.forces = []
```

- `self.name` This is the name of the point
- `self.dimension` This is the dimension of the point, this is asserted to ensure it does not cause problems during the formation of the matrix
- `self.location` This is the location of the point
- `self.lines` This is a list of all the lines connected to the point, this helps us access the tensions
- `self.equation` This is a list of all the equations in `N` dimensions pertaining to all the forces acting on the point
- `self.forces` This is a list of all the forces acting on the point, mainly contains of all the external forces

A point object is the first breakpoint of solving the truss, we iterate over all points of the trusses and form equations in them, then use the said equations to form the matrix to solve to find all the values of the tensions

### 3.1.3 Line

The constructor for the said class is

```
def __init__(self, a: Point, b: Point, dimension:int,number:int=-1):
self.number = number
assert(len(a.location)==len(b.location) and len(a.location)==dimension)
self.edges = (a, b)
self.center = tuple(map(lambda x, y: (x + y) / 2, a.location, b.location))
self.length = np.linalg.norm(
```

```
    tuple(map(lambda x, y: x - y, a.location, b.location))
)
"""
these forces act relative to the
center of the line
"""
self.tensions = []
self.tension_equation = Equation()
```

- `self.number` This is the number of the rod, as all the rods are denoted by numbers
- `self.edges` This is a tuple containing the 2 points which denote the 2 ends of the line
- `self.center` This is the center of the rod, denoted by a tuple of floats
- `self.length` This is a tuple containing the length of the line
- `self.tensions` This contains the 2 tensions acting in the rod
- `self.tension_equation` I treat both the tensions as seperate entities, so this equation binds both the tensions together in one equation relating the opposing tensions with each other

Each line has 2 tensions acting on its ends, these tensions are treated as independent forces, and are related with the tension equation that the constructor has, which is later added in the matrix

#### 3.1.4 Force

The constructor for the said class is given here

```
def __init__ (self, name:str, point:Point, dimension:int, magnitude:float=1, from_components:bo
self.name = name
self.dimension = dimension
assert(dimension == len(point.location))
self.acting_on = point

if from_components:
    self.components = point.location
    self.magnitude = np.linalg.norm(point.location)
    self.calculate_direction()
    self.INITIAL_DIRECTION = self.direction
    print(self)
else:
    self.magnitude = magnitude
    self.direction = point.location
    self.INITIAL_DIRECTION = self.direction
    self.calculate_components()
```

- `self.name` This is the name of the force
- `self.dimension` This is the dimension of the force
- `self.acting_on` This is the point on which the force is acting on

- `self.components` This is a tuple of all the components the force
- `self.magnitude` This is the magnitude of the force
- `self.direction` This is a unit vector depicting the direction of the force
- `self.INITIAL_DIRECTION` this is a copy of the direction made during initialisation so one can compare if the truss is being compressed or elongated

### 3.1.5 Equation

This is a simple class depicting mathematical equation

```
def __init__(self, coeff: list = list(), var: list = list()):
self.coeffecients = coeff[:]
self.variables = var[:]
```

- `self.coeffecients` This contains a list of float coeffecients
- `self.variables` This contains a list of string denoting the variables

The Equation class is used to find the matrix, as it is an easy way to save the data in a way which we use the data, thus making it intuitive

### 3.2 Core Logic

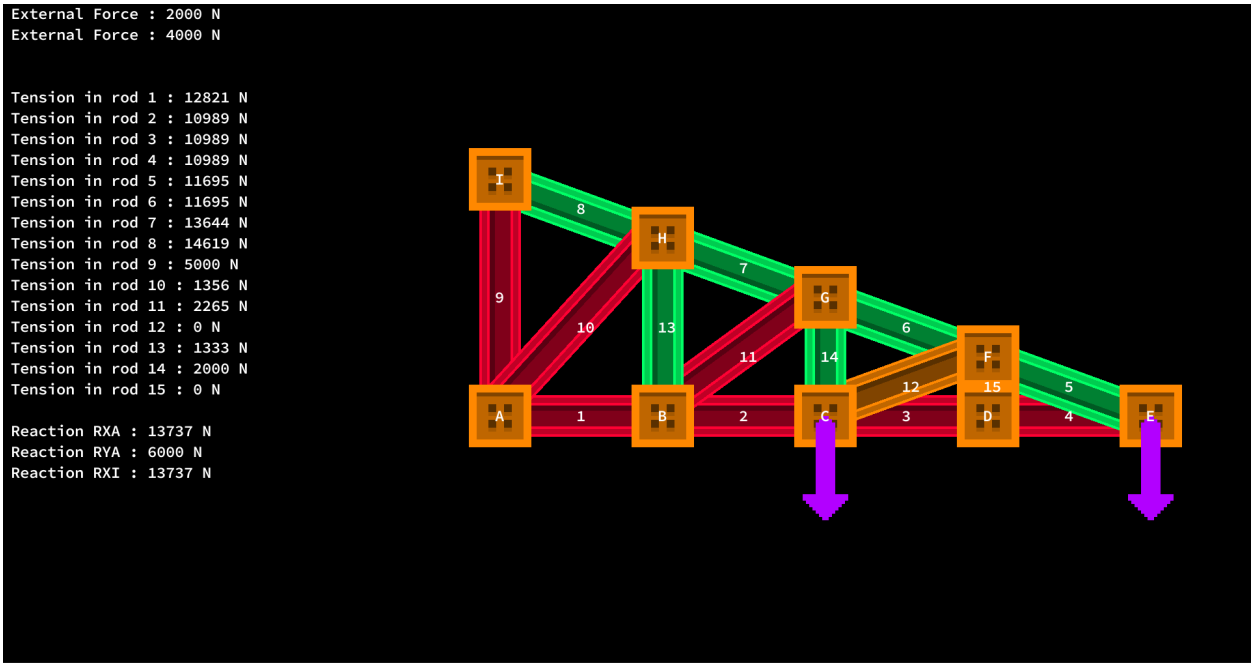
The core logic follows the following steps

1. First of all interpret the truss file inputted by the user
2. Link all the Joints, Beams and generate Tensions
3. Iterate through each point and form equations at each point, and store the N equations in a list
4. Use this list of equations to form the coeffecient matrix
5. Use the dictionary of the external forces to form the RHS matrix B
6. Use `np.linalg.solve` to solve the said matrix
7. Visualise the said solutions

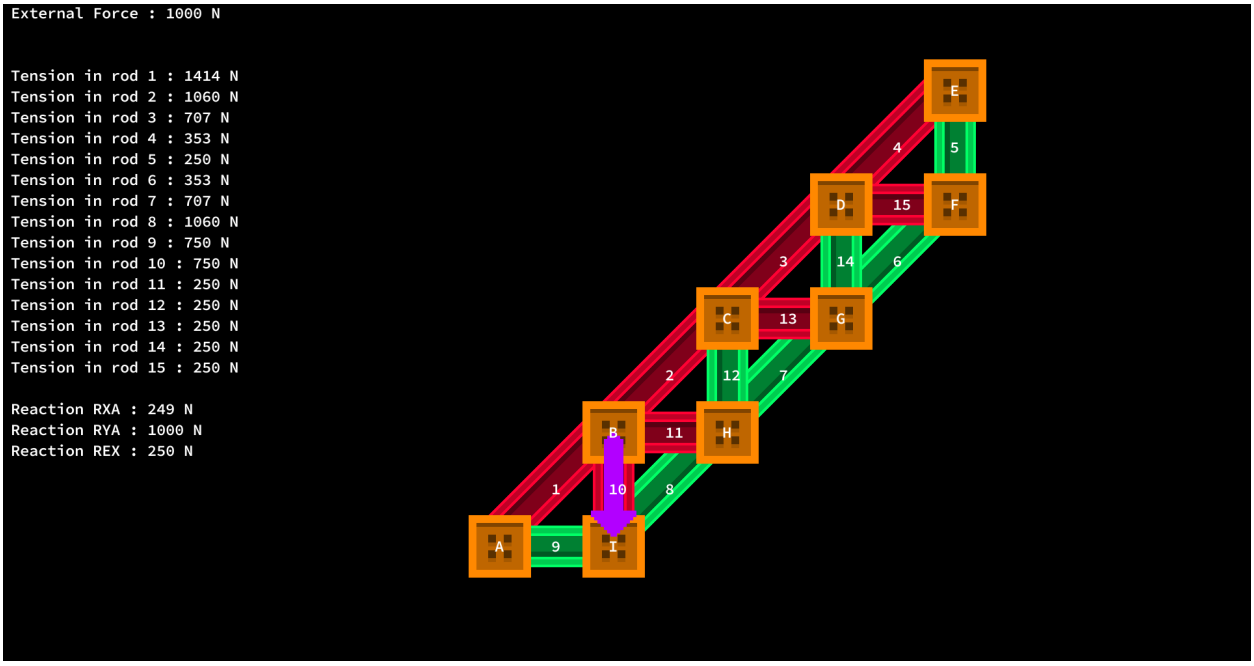
# 4 Output

The proof for solution is the fact that all the reaction forces are equal to the total net load applied by the given system as all external forces should sum up to 0, thus all the equations formed by my program must be correct

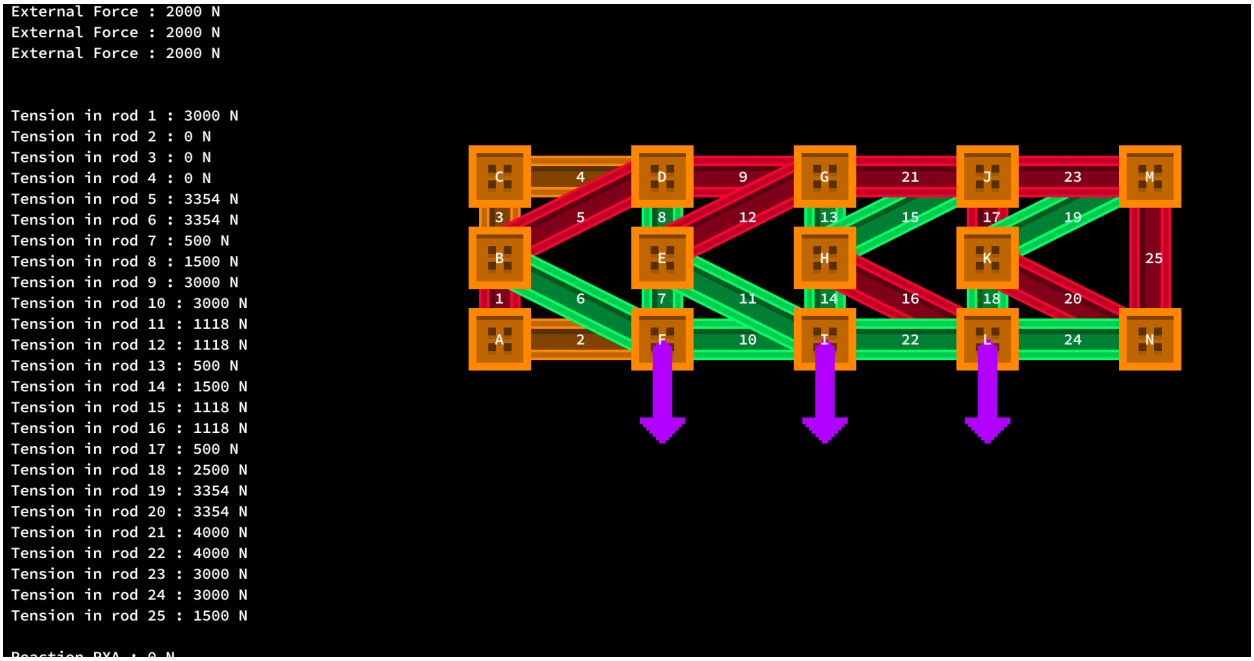
## 4.1 Sample 6.4



## 4.2 6.1.17


External Force : 1000 N
Tension in rod 1 : 1414 N
Tension in rod 2 : 1060 N
Tension in rod 3 : 707 N
Tension in rod 4 : 353 N
Tension in rod 5 : 250 N
Tension in rod 6 : 353 N
Tension in rod 7 : 707 N
Tension in rod 8 : 1060 N
Tension in rod 9 : 750 N
Tension in rod 10 : 750 N
Tension in rod 11 : 250 N
Tension in rod 12 : 250 N
Tension in rod 13 : 250 N
Tension in rod 14 : 250 N
Tension in rod 15 : 250 N
Reaction RXA : 249 N
Reaction RYA : 1000 N
Reaction REX : 250 N
A
B
C
D
E
F
G
H
I
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

## 4.3 6.1.20


External Force : 2000 N
External Force : 2000 N
External Force : 2000 N
Tension in rod 1 : 3000 N
Tension in rod 2 : 0 N
Tension in rod 3 : 0 N
Tension in rod 4 : 0 N
Tension in rod 5 : 3354 N
Tension in rod 6 : 3354 N
Tension in rod 7 : 500 N
Tension in rod 8 : 1500 N
Tension in rod 9 : 3000 N
Tension in rod 10 : 3000 N
Tension in rod 11 : 1118 N
Tension in rod 12 : 1118 N
Tension in rod 13 : 500 N
Tension in rod 14 : 1500 N
Tension in rod 15 : 1118 N
Tension in rod 16 : 1118 N
Tension in rod 17 : 500 N
Tension in rod 18 : 2500 N
Tension in rod 19 : 3354 N
Tension in rod 20 : 3354 N
Tension in rod 21 : 4000 N
Tension in rod 22 : 4000 N
Tension in rod 23 : 3000 N
Tension in rod 24 : 3000 N
Tension in rod 25 : 1500 N
C
D
G
J
M
B
E
H
K
A
F
I
L
N
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25