BTECH PROJECT

GEOMETRY OPTIMIZATION USING ML TECHNIQUES

Team Members:
MANAS SACHIN DESHMUKH(2022102040)
ADITI SINGH (2022101075)

Faculty Advisor: Jofin George

PRIMARY OBJECTIVES

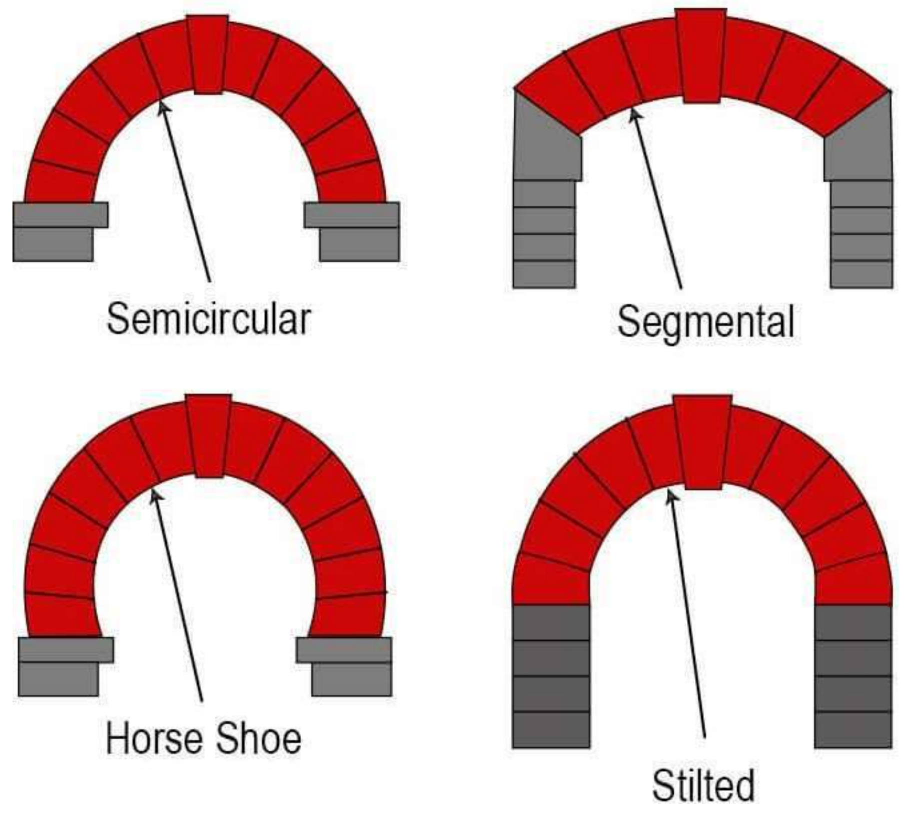
Develop an machine learning algorithm to find the optimum thickness (*t*) of a ***masonry arch*** by minimizing the collapse probability considering horizontal seismic forces and vertical gravity loads, and externally applied load.

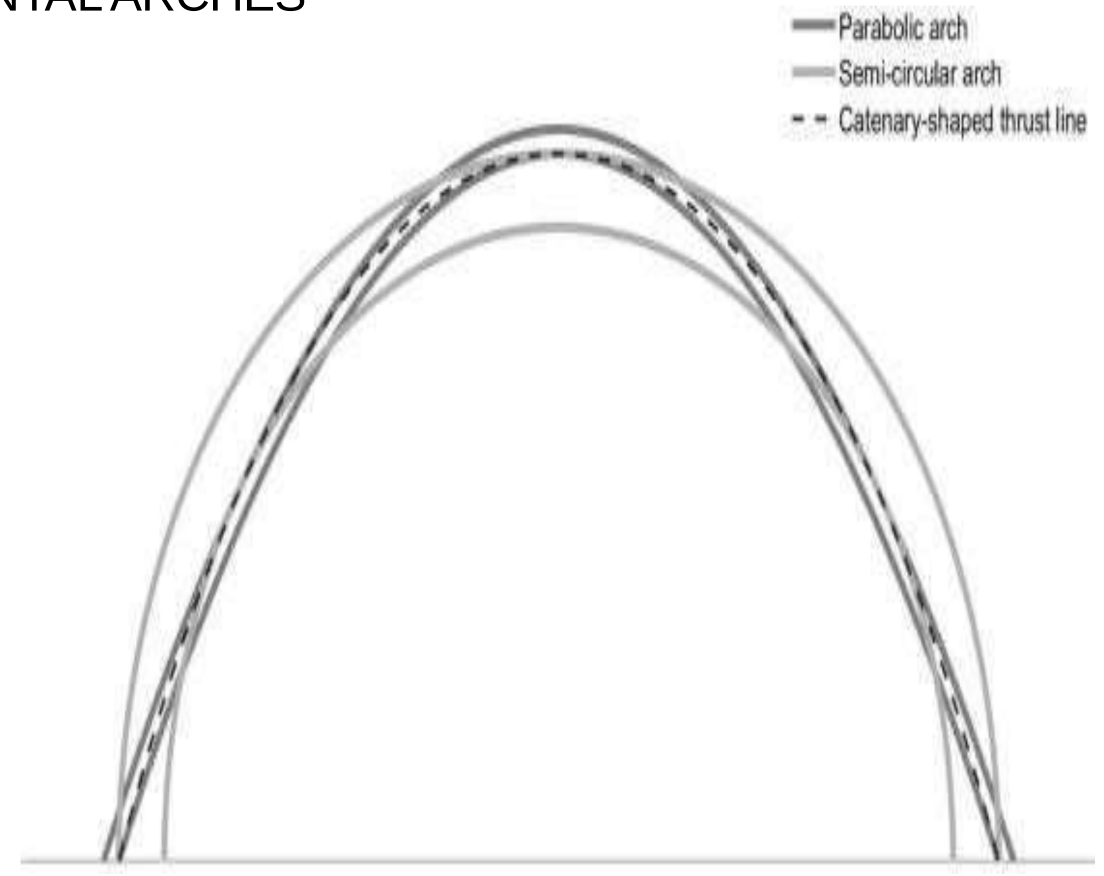
SECONDARY OBJECTIVES

- Identify the failure pattern of the arch considering vertical loads, horizontal forces and a combination of both.
- To quantify the optimum **uniform** arch thickness considering all possible load combinations of horizontal forces and vertical loads.
- To quantify the optimum **non-uniform** arch thickness considering all possible load combinations of horizontal forces and vertical loads

TYPES OF STANDARD ARCHES

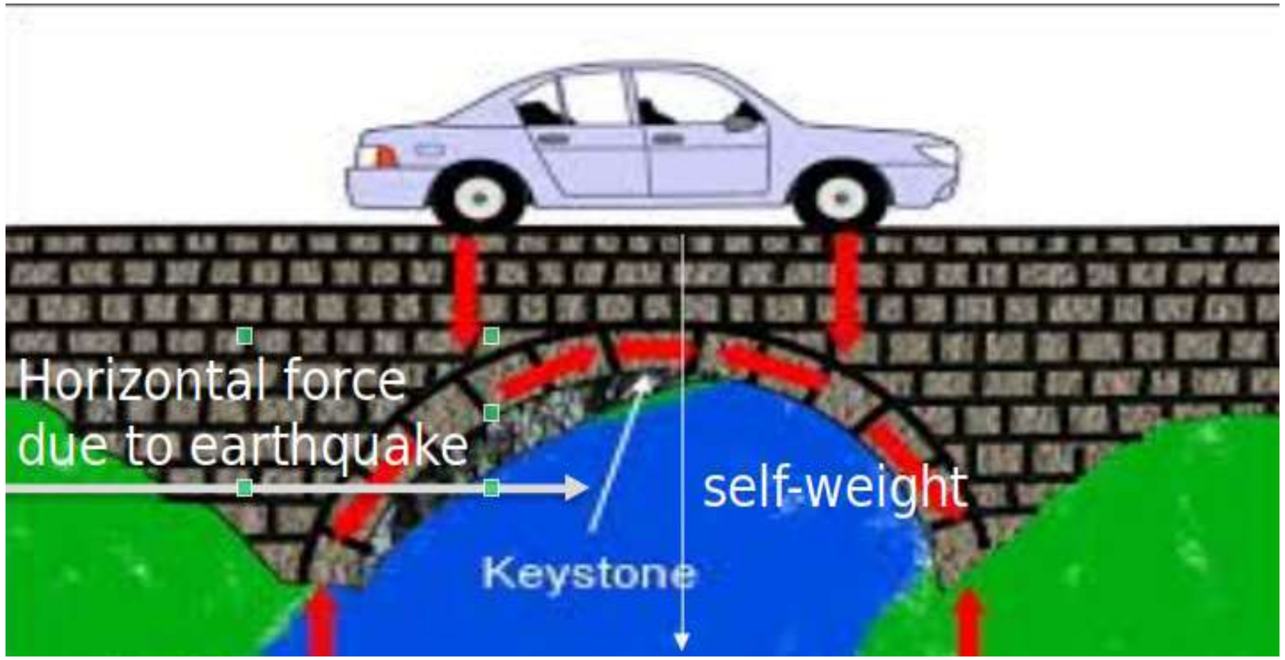
INITIALLY FOCUS ON SEMICIRCULAR AND SEGMENTAL ARCHES





CASE 1 DETAILS

•In this case, the main force acting on the arch is the **horizontal seismic force** due to ground motion. The only vertical force present is the self-weight of the arch due to gravity. There are **no additional vertical loads** from vehicles or construction materials.



Net Force (F_1) = Gravity of the Arch (F_2) + Horizontal Seismic Force (F_3)

$F1=F2+F3$

and

$F2 = m*g$

CASE 2 DETAILS

•In this case, the forces acting on the arch are the **vertical self-weight** of the arch due to gravity, **vertical loads** from construction materials, and **vertical vehicle loads**. There are **no horizontal forces** (seismic forces) acting on the arch.

Net Force (F_1) = Force due to Gravity (F_2) + Force due to External Construction Material (F_3) + Force due to Vehicle (F_4)

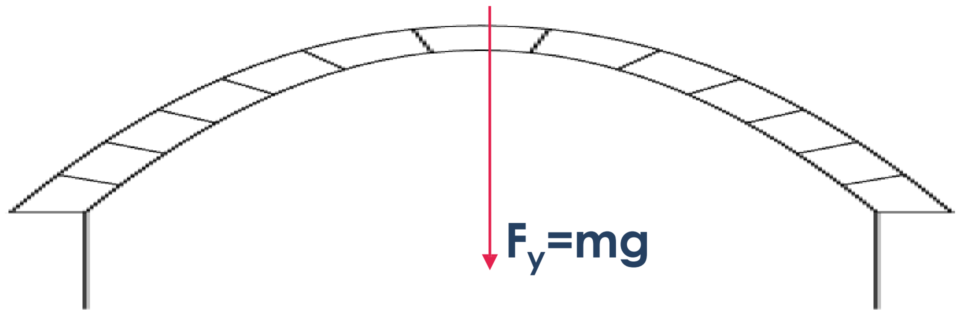
F1=F2+F3+F4

and

F2 = m*g

METHODOLOGY TO BE USED:-

- Research papers based on this topic were analyzed for ML techniques to get more knowledge about optimizations.
- We will consider three cases about the forces acting on an arch and we will try to compute using ML the optimum thickness in each of three cases.
- The first case considered is when only the vertical gravity force of the arch itself is considered so only m*g(mass of the arch * acceleration due to gravity) acts vertically and there are no horizontal forces on the arch.



INTRODUCTION TO MASONRY ANALYSIS

Objective: The primary goal is to analyze and predict optimal hinge positions and collapse factors for arches based on architectural parameters.

Focus Area: Given architectural inputs like span, thickness, block dimensions, and unit weight, the code calculates the required internal forces and identifies critical points (hinges) for structural integrity.

Approach:

1. Using geometry of the semi-circular arch.

2. Calculating block centroids and weights.

3. Determining internal work for hinge position optimization.

CORE FUNCTION:MASONRY ANALYSIS

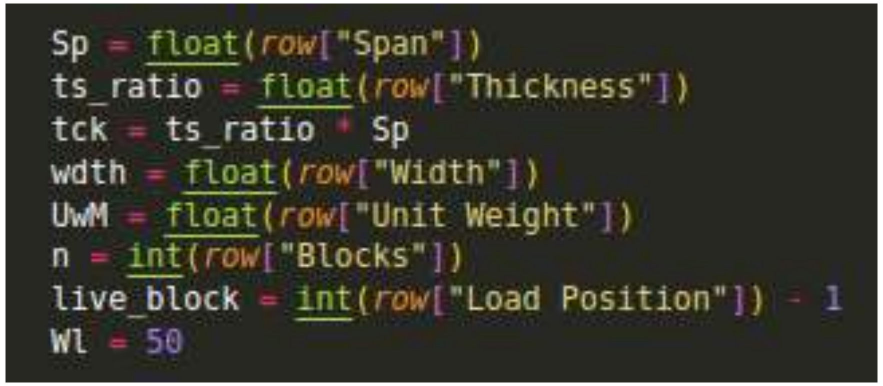
Inputs: Parameters like span, thickness, width, unit weight, number of blocks, and live load position are extracted from the input dataset.

Key Operations

Geometry Calculation: Calculating radii (ri, re, rc) and block area.

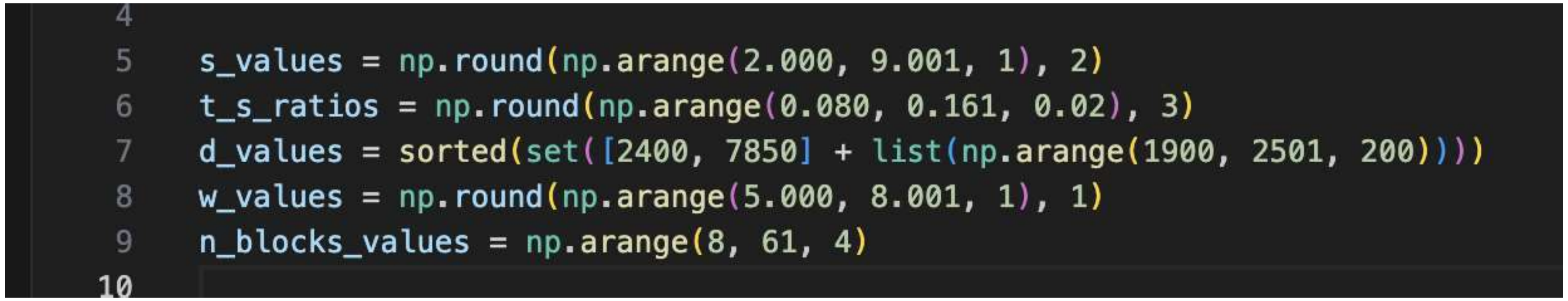
Block Weight: Weight of each block calculated using width and unit weight.

Block Centroids: Using polar coordinates to find centroids of each block.



```
Sp = float(row["Span"])
ts_ratio = float(row["Thickness"])
tck = ts_ratio * Sp
wdth = float(row["Width"])
UwM = float(row["Unit Weight"])
n = int(row["Blocks"])
live_block = int(row["Load Position"]) - 1
Wl = 50
```

DATASET GENERATION:-



```
s_values = np.round(np.arange(2.000, 9.001, 1), 2)
t_s_ratios = np.round(np.arange(0.080, 0.161, 0.02), 3)
d_values = sorted(set([2400, 7850] + list(np.arange(1900, 2501, 200))))
w_values = np.round(np.arange(5.000, 8.001, 1), 1)
n_blocks_values = np.arange(8, 61, 4)
```

The code generates parameter ranges for analysis or simulation. s_values (2.00–9.00) may represent size; t_s_ratios(0.080–0.160) are likely thickness-to-span ratios. d_values include common densities (e.g., 2400, 7850) and a range from 1900 to 2500. w_values (5.0–8.0) might represent widths or loads, and n_blocks_values (8–60) likely indicate the number of segments or blocks in a system.

INTERNAL WORK & HINGE POSITION CALCULATION

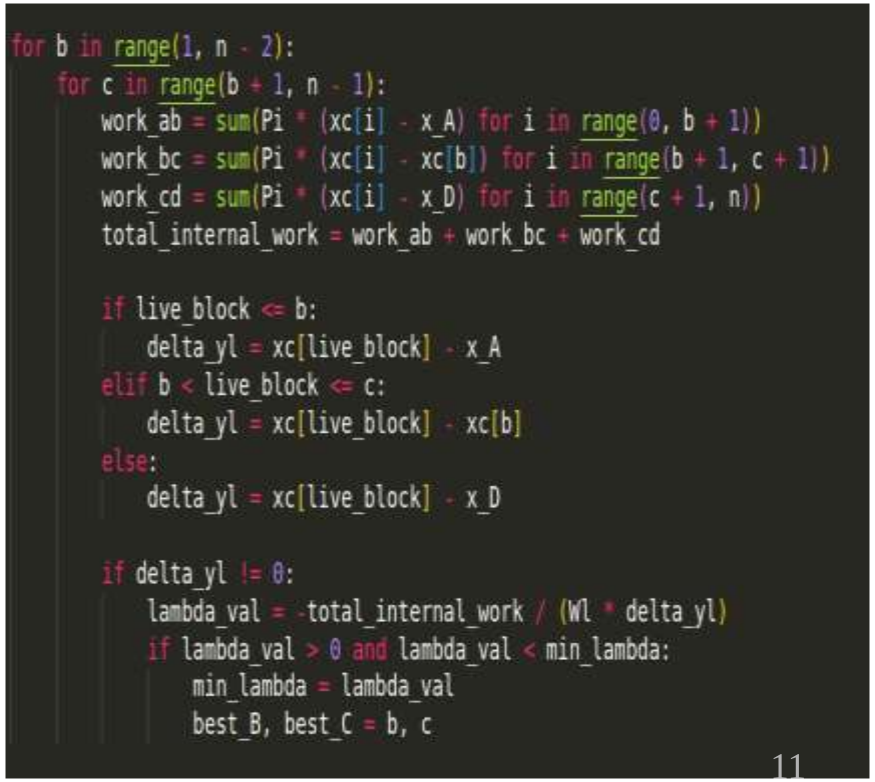
Work Calculation: Calculates internal work for three segments: A-B, B-C, and C-D, based on block centroids.

Lambda Calculation: Determines λ (collapse factor) for different hinge positions and selects the optimal one.

Core Logic:

1. Iterates over possible hinge positions.
2. Computes internal work and evaluates the resulting lambda for each confi-

   -guration.
3. Optimizes for the minimum λ value.
4. Lamda multiplied by g will give the horizontal acceleration required to trigger collapse of arch



```
for b in range(1, n - 2):
    for c in range(b + 1, n - 1):
        work_ab = sum(Pi * (xc[i] - x_A) for i in range(0, b + 1))
        work_bc = sum(Pi * (xc[i] - xc[b]) for i in range(b + 1, c + 1))
        work_cd = sum(Pi * (xc[i] - x_D) for i in range(c + 1, n))
        total_internal_work = work_ab + work_bc + work_cd

        if live_block <= b:
            delta_yl = xc[live_block] - x_A
        elif b < live_block <= c:
            delta_yl = xc[live_block] - xc[b]
        else:
            delta_yl = xc[live_block] - x_D

        if delta_yl != 0:
            lambda_val = -total_internal_work / (Wl * delta_yl)
            if lambda_val > 0 and lambda_val < min_lambda:
                min_lambda = lambda_val
                best_B, best_C = b, c
```

RESULT PREPARATION AND OUTPUT

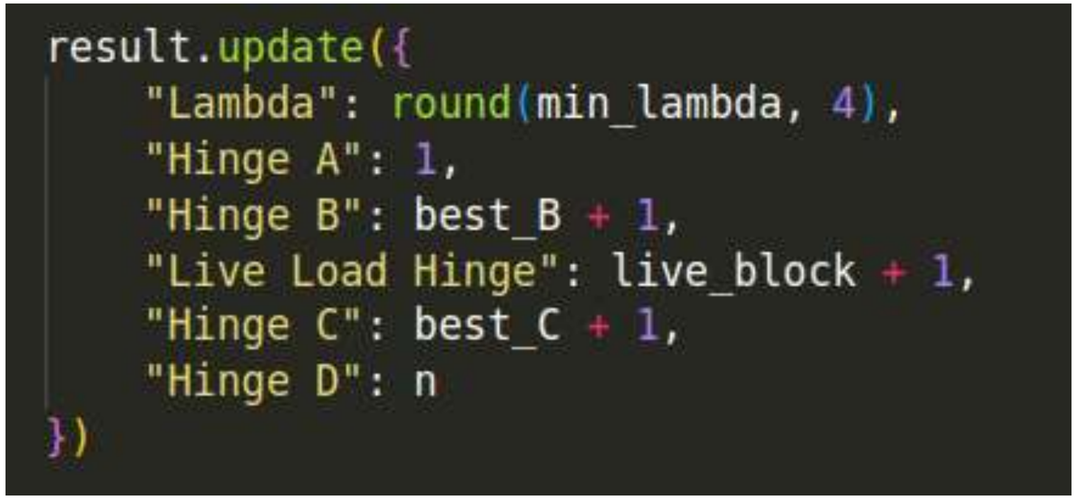
Hinge Positions: After computing the minimum lambda value, the best hinge positions are selected and recorded.

Output: Results include the lambda value, hinge positions (A, B, C, D), and load position.

Example Output:

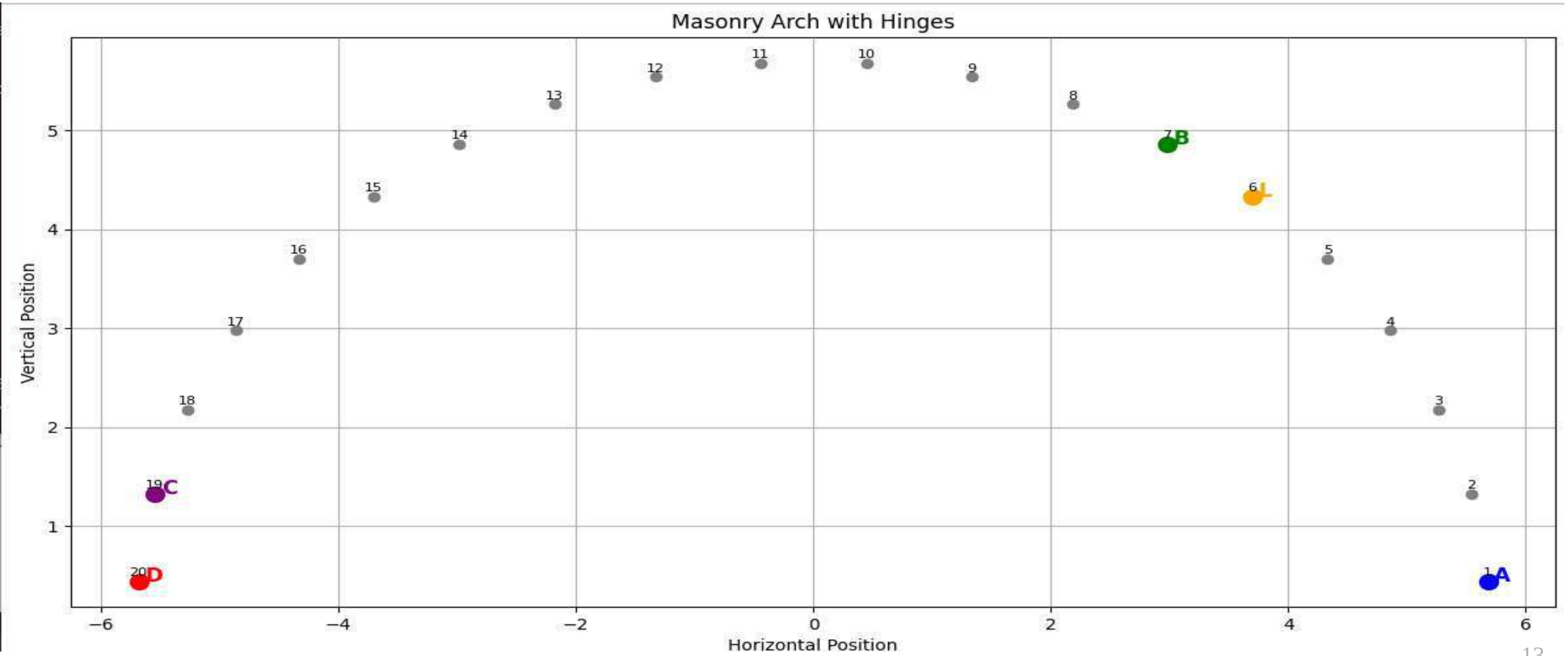
1. λ : 0.5274
2. Hinge Positions:

   A: 1, B: 4, C: 7, D: 10

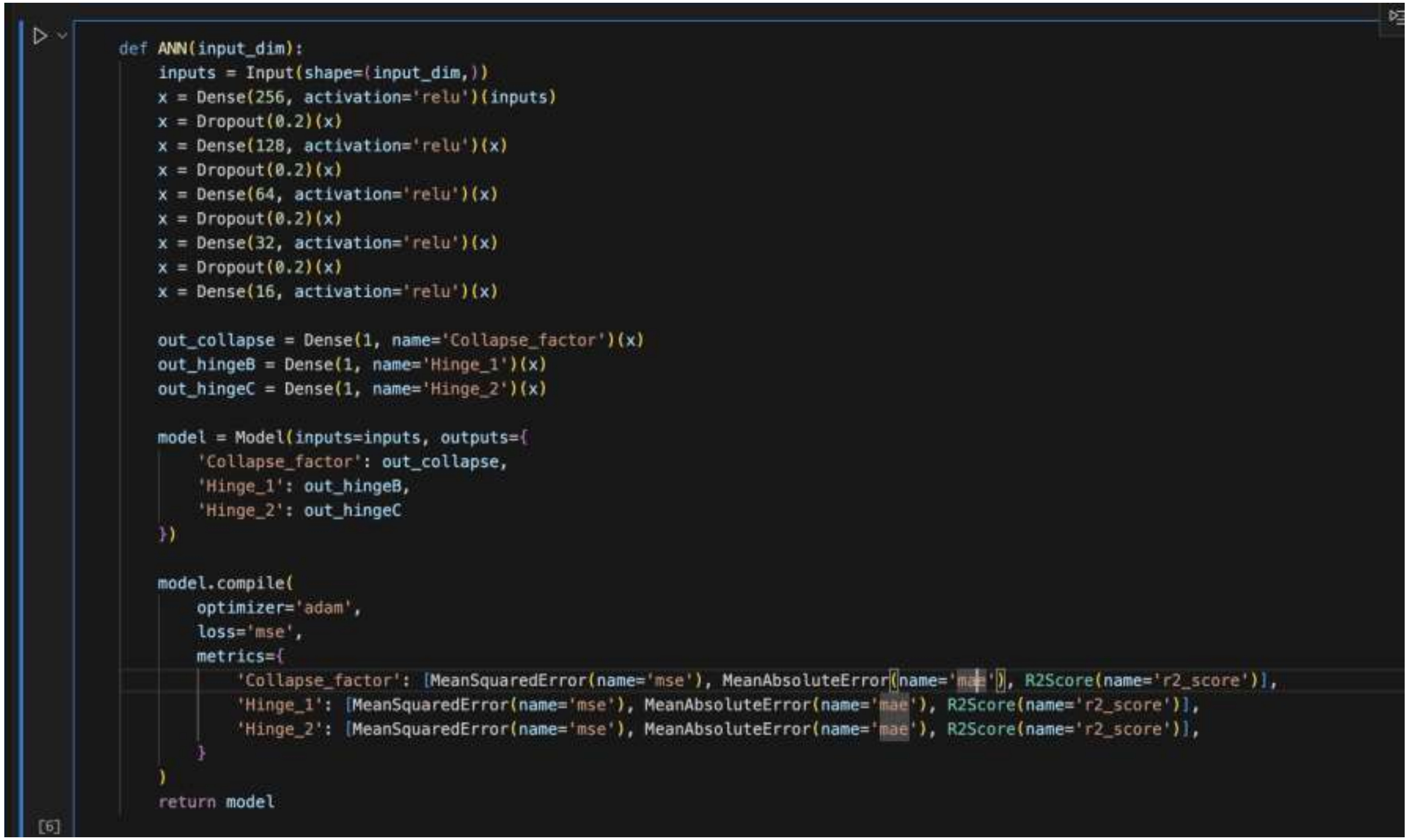


```
result.update({
    "Lambda": round(min_lambda, 4),
    "Hinge A": 1,
    "Hinge B": best_B + 1,
    "Live Load Hinge": live_block + 1,
    "Hinge C": best_C + 1,
    "Hinge D": n
})
```

ARCH POSITION CONSIDERED:-



CODE USED :-



```
def ANN(input_dim):
    inputs = Input(shape=(input_dim,))
    x = Dense(256, activation='relu')(inputs)
    x = Dropout(0.2)(x)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(32, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(16, activation='relu')(x)

    out_collapse = Dense(1, name='Collapse_factor')(x)
    out_hingeB = Dense(1, name='Hinge_1')(x)
    out_hingeC = Dense(1, name='Hinge_2')(x)

    model = Model(inputs=inputs, outputs={
        'Collapse_factor': out_collapse,
        'Hinge_1': out_hingeB,
        'Hinge_2': out_hingeC
    })

    model.compile(
        optimizer='adam',
        loss='mse',
        metrics={
            'Collapse_factor': [MeanSquaredError(name='mse'), MeanAbsoluteError(name='mae'), R2Score(name='r2_score')],
            'Hinge_1': [MeanSquaredError(name='mse'), MeanAbsoluteError(name='mae'), R2Score(name='r2_score')],
            'Hinge_2': [MeanSquaredError(name='mse'), MeanAbsoluteError(name='mae'), R2Score(name='r2_score')],
        }
    )
    return model
```

[6]

CODE EXPLANATION AND APPROACH USED

Multi-Output Modeling Approach

- Instead of training separate models for each target, this approach uses **a shared base network** followed by **three separate heads** for each regression target. This has the benefit of learning shared representations from the data, which is especially useful when outputs are related.

The model is compiled with:

- **Optimizer:** 'adam' — an adaptive learning rate method.
- **Loss:** 'mse' (Mean Squared Error) — applied uniformly across all outputs.
- **Metrics:** Different metrics for each output, including MSE, MAE (Mean Absolute Error), and R^2 Score. These help monitor the model's performance from multiple perspectives: average error, absolute error, and goodness-of-fit.

This setup is ideal for problems like structural predictions or physical simulations where multiple interdependent quantities need to be estimated from the same input features.

CODE CONCEPT AND APPROACH

- This code implements a **multi-output regression model** using a feedforward neural network in TensorFlow/Keras to predict three continuous targets: Alpha1, B, and C. The core machine learning pipeline includes **data preprocessing**, **model building**, **training**, and **evaluation**.

Data Preprocessing:

- The dataset is read from a CSV file, with missing values dropped. The input features

(n, Diameter, Thickness) and output targets are normalized using StandardScaler to improve model convergence. The data is split into training, validation, and test sets in a 50:20:30 ratio.

Model Architecture:

- The model is a deep artificial neural network with five hidden layers (128 → 64 → 64 → 32 → 16 neurons) using ReLU activation. The final layer has 3 neurons for predicting the three target variables simultaneously. The network is trained using the **Adam optimizer** and **mean squared error (MSE)** as the loss function, with **mean absolute error (MAE)** as an additional metric.

CODE CONCEPT AND APPROACH

Training and Evaluation:

- The training loop runs for 300 epochs, displaying metrics like MAE, MSE, and R^2 for each target every epoch. After training, the model's performance is assessed on the test set using the same metrics, helping evaluate generalization.
- This approach is efficient for related multi-target regression tasks and emphasizes consistent scaling, shared representation learning, and interpretability via evaluation metrics.

FUTURE WORK AND OPTIMIZATION

Current Approach: The current model identifies hinge positions using iterative methods.

Next Steps:

- **Optimization of Hinge Positions:** Transition to machine learning algorithms to predict optimal hinge configurations and collapse factors based on input parameters, removing the need for iterative calculations.
- ***Non-Uniform Thickness:*** Expand the model to consider non-uniform thickness along the arch span, optimizing the structure for material efficiency and stability.
- **Machine Learning Integration:** Apply optimization techniques or other ML-based methods for faster and more accurate predictions of structural integrity.

Goal: Optimize the arch design for minimum thickness while maintaining stability and performance, using advanced optimization techniques.

FUTURE WORKS

- We've developed a machine learning model to predict the relationship between input parameters (like material properties and load conditions) and outputs such as hinge positions and collapse factor. The next step is **optimization** to refine the design, aiming for the minimum thickness required for safety while ensuring efficiency.

Optimization for Minimum Thickness:

- Currently, we predict the arch **failure** using machine learning, but the next goal is to optimize to find the minimum thickness that ensures structural safety. This is vital for reducing material use while maintaining stability against vertical and horizontal forces, including gravity, external loads, and seismic forces.
- Future Approach: We'll use techniques like Gradient Descent, Genetic Algorithms, or Particle Swarm Optimization to iteratively adjust parameters, minimizing material usage without compromising safety.

Optimizing Hinges and Collapse Factor:

- Right now, we find hinge positions and collapse factor using an iterative method, which can be time-consuming and inefficient.
- Future Approach: We'll optimize hinge positions and collapse factor directly through machine learning, removing the need for iterative trial and error. This will ensure faster, more accurate designs with better performance.

Non-Uniform Thickness Optimization:

- The current design assumes uniform thickness throughout the arch, which simplifies the problem but may not reflect real-world conditions where thickness varies.
- Future Approach: We will optimize non-uniform thickness, allowing the arch to adjust thickness based on the load distribution. Areas under higher load will have greater thickness, while less stressed areas will be thinner. This will improve material efficiency while maintaining structural integrity.

KEY OUTCOMES

- **Optimization Tool: Which gives optimal building configurations when given details about the construction.**
- **Improved Structural Efficiency**: Optimization leads to more efficient use of materials, reducing construction costs and environmental impact while maintaining or improving structural integrity.
- **Enhanced Safety**: By predicting stress points and load distribution more accurately, ML optimizes arch designs to better handle extreme conditions, reducing the risk of failure.
- **Faster Design Processes**: Machine learning algorithms can rapidly simulate and test thousands of design variations, significantly speeding up the design and decision-making process in construction projects.
- **Cost Reduction**: Geometry optimization through ML helps identify the most cost-effective design and material choices, leading to lower overall expenses in construction and restoration projects.

APPLICATIONS

- **Structural Design Optimization**: ML models can optimize arch designs by predicting the most efficient shape and material distribution to minimize weight while maintaining strength, used in architecture and civil engineering.
- **Construction Material Selection**: Machine learning can analyze large datasets to recommend the best materials for arches, balancing cost, sustainability, and durability, used in construction projects.
- **Load Distribution Prediction**: ML algorithms can predict load distributions across arches, helping engineers optimize their form and structure to withstand specific loads, enhancing safety in bridges and buildings.
- **Historical Restoration**: ML-based geometry optimization can be applied in the restoration of historical arches, identifying the most accurate reconstruction methods and materials based on structural analysis and old designs.

THANK YOU!