GEOMETRY OPTIMIZATION USING ML TECHNIQUES

TEAM MEMBERS:
Aditi Singh (2022101075)
Manas Deshmukh (2022102040)

Faculty Advisor: Jofin George

Problem Statement & Objectives

Masonry arches → need minimum thickness for safety & efficiency.

Traditional iterative method → computationally expensive.

Objective:

- Predict collapse factor & hinge positions using ML.
- Optimize for minimum feasible thickness under **both horizontal and vertical loads , individually and combination as well** (future work).

Work Done in Previous Semester

- Conducted literature survey & dataset generation for **semicircular arches**.
- Considered **both vertical loads** and **horizontal seismic forces** in dataset creation.
- Implemented **multi-output regression neural network (Keras)**:
    - **Inputs**: (n, diameter, density, span, thickness)
    - **Outputs**: (hinge B, hinge C, collapse factor)
- Achieved predictions consistent with iterative method for both **vertical and horizontal cases**.
- Model achieved an **accuracy of 97–98%** on test data.

Work Done in This Semester

Horizontal Case (Seismic Loads)+Vertical Case(Gravity force)

Theory:

- Arch subjected to **horizontal seismic forces** + self-weight.
- Stability depends on hinge formation and collapse factor (λ).
- Goal: find **minimum feasible thickness** ensuring structural safety.

Implementation:

- Developed **optimization framework** (`optim_horiz.py`).
- **Objective function** → returns thickness if feasible, else penalty (1e6).
- Feasibility check based on ANN-predicted hinges (**α, b, c**) lying within valid range.

Key Features:

- **Objective Function:**
  - Returns thickness if ANN-predicted hinges are valid.
  - Applies **penalty (1e6)** if infeasible.
- **Feasibility Check:** $\alpha \in [0,1]$, $1 \leq b < c \leq n$ after rounding/clamping
- **Scalers:** Input/output scaling with `StandardScaler` and `MinMaxScaler` for ANN compatibility.
- **Flexible bounds:** Thickness search in `[0.16, 1.44] m`.

Workflow (Technical Details):

- **Load model & scalers:** `tf.keras.models.load_model()` + `joblib.load()`.
- **Select design cases:** From dataset or user-defined examples.
- **Objective evaluation:** Input → scale → ANN → inverse transform → clamp → **check feasibility**.
  - **Penalty:** If thickness/hinges infeasible, return `1e6 + violation magnitude`.
- **Run optimizers:** Sequentially call each method (details in next slide).
- **Store results:** Save thickness, α, b, c, time → `optim_results_summary.csv`

Integrated ANN model with multiple optimizers:

- **Grid Search** – brute-force search over defined parameter grid.
- **Random Search** – random sampling of thickness values within range.
- **Simulated Annealing** – probabilistic search that avoids local minima by allowing “uphill” moves.
- **Genetic Algorithm (GA)** – evolves candidate solutions using crossover and mutation.
- **Particle Swarm Optimization (PSO)** – particles move in search space influenced by personal & global best.
- **Differential Evolution (DE)** – combines population-based mutation and crossover for global optimization.
- **Bayesian Optimization (Optuna)** – models objective function probabilistically, selects promising samples.
- **Nelder–Mead** – simplex-based search, efficient for continuous, smooth functions

Results:

- Generated comparative results across optimizers.
- Exported outputs (`optim_results_summary.csv`) for analysis.
- Achieved **consistent convergence towards minimum thickness** across methods.

(b) Vertical Case :

Algorithms Used

1. **Gradient Descent (GD)**
   - Batch method: updates weights using all training samples.
   - Slower but stable.
   - Loss function includes penalty:

$$L(W) = \frac{1}{N}\sum(y_{\text{pred}} - y_{\text{true}})^2 + \lambda P$$

2. **Stochastic Gradient Descent (SGD)**
   - Updates weights **per sample (or mini-batch)** instead of full dataset.
   - Faster convergence, good for large datasets.
   - More variance in updates, but often avoids local minima.
3. **Newton's Method (Closed Form)**
   - Direct solution using normal equation:

$$W = (X^\top X)^{-1} X^\top Y$$

   - Exact solution (if invertible), but computationally heavy for large feature sets.
4. **Penalty Term for 3 Outputs**
   - Extra penalty ensures balanced prediction of **Alpha1, Bcon1, Ccon1**.
   - Prevents the model from favoring one output at the expense of others.
5. **Optimization for Minimum Thickness**
   - After training, the model is used to **search for minimum feasible thickness**.
   - Ensures structural design efficiency while keeping prediction errors low.

Challenges Faced (Horizontal Case)

- **Local minima trapping** and **premature convergence** in GA/PSO.
- **Penalty sensitivity**: feasible near-boundary solutions often discarded.
- **Computational bottleneck**: repeated ANN evaluations → high runtime for large *n*.
- **Search inefficiency**: optimizers overshoot feasible region, wasting iterations.
- **Hyperparameter tuning** (GA, PSO, DE) needed for stable convergence.

Challenges Faced (Vertical Case)

Multi-Output Prediction – Balancing accuracy across 3 targets (Alpha1, Bcon1, Ccon1).

Penalty Integration – Designing an effective penalty term to handle poor predictions.

Convergence Issues – Gradient Descent/SGD needing careful tuning of learning rate.

Computational Cost – Newton’s method becoming heavy with large matrices.

Design Constraints – Ensuring minimum thickness optimization without violating other design constraints

Future Works

1. Extend study to **combined Horizontal + Vertical force cases**.
2. Move from **uniform → non-uniform thickness arches** for more realistic modeling.
3. Refine optimization using **hybrid approaches** (e.g., GA + local search,GD+PSO).
4. Enhance dataset coverage for **better generalization & robustness**.
5. Benchmark **ML-based optimizers vs. classical iterative methods**.