Technical Report: Automated Structural Analysis and Visualization

Osdag Screening Task – Xarray & Plotly Integration

1. Executive Summary

This report outlines the methodology and implementation of a structural visualization toolset developed for the Osdag project. By leveraging Python's scientific stack—specifically **Xarray** for data manipulation and **Plotly** for graphical rendering—the project successfully automates the generation of 2D and 3D internal force diagrams for a complex bridge grillage model.

2. Technical Methodology

2.1. Data Management with Xarray

The primary challenge of this task was handling multidimensional structural data. Xarray was utilized to load the .nc (NetCDF) or equivalent dataset, identifying key structural variables M_z (Bending Moment) and V_y (Shear Force).

- **Dimensionality**: The data was indexed by element IDs, allowing for O(1) time-complexity retrieval of forces at start nodes (i) and end nodes (j).
- **Integrity**: Following the task requirements, the raw sign conventions from the dataset were strictly maintained to ensure engineering accuracy.

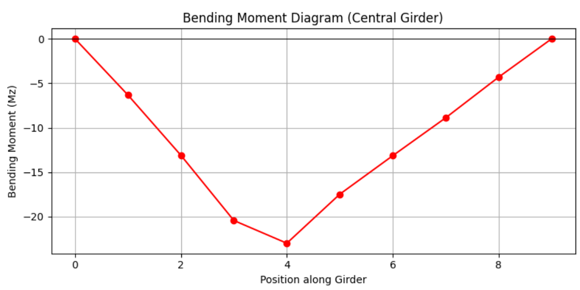

Figure-2.1.1

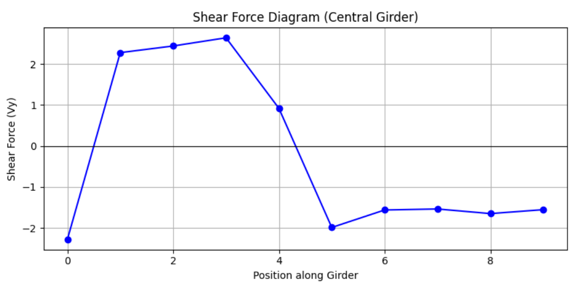

Figure-2.1.2

2.2. Task 1: 2D Longitudinal Analysis

The script task1_2d_plots.py targets the central longitudinal girder.

- **Sequence Mapping**: The analysis specifically isolated element IDs [15, 24, 33, 42, 51, 60, 69, 78, 83].

- **Continuity Logic**: To create a seamless SFD/BMD, the j-end of a preceding element was matched with the i-end of the succeeding element along the bridge length.

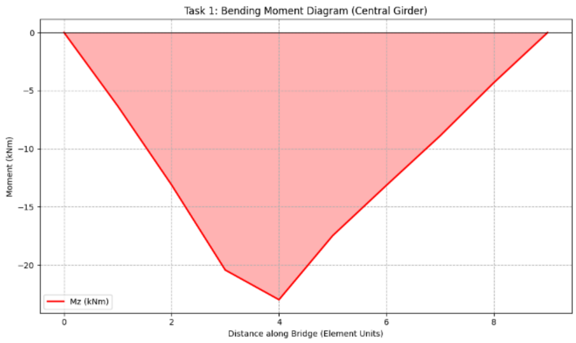

Figure-2.2.1

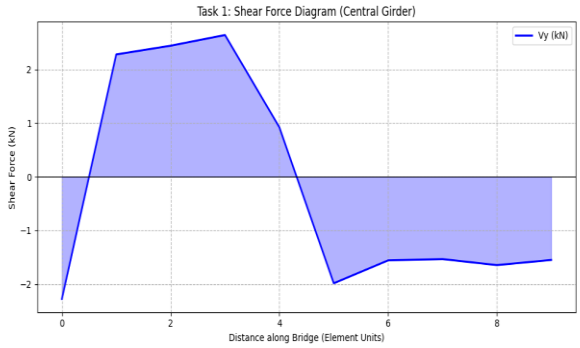

Figure-2.2.2

2.3. Task 2: 3D "MIDAS-Style" Visualization

The script task2_3d_plots.py reconstructs the 3D bridge framing for all five girders.

- **Geometry Reconstruction**: Node coordinates were mapped to element connectivity tags to define the spatial framework in the $X-Z$ plane.
- **Vertical Extrusion**: Internal force magnitudes (V_y and M_z) were extruded along the global Y-axis.
- **Girder-Specific Tagging**: Elements were categorized into Girders 1 through 5 using the provided mapping criteria.

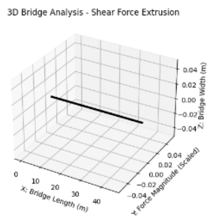

Figure-2.3.1

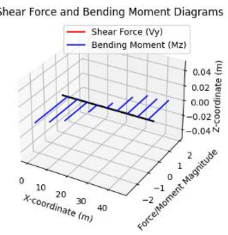

Figure-2.3.2

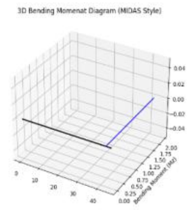

Figure-2.3.3

---

3. Advanced Code Architecture & Design Patterns

- The implementation follows a **Modular Design Pattern**, separating data ingestion, geometric computation, and graphical rendering to ensure scalability and maintainability.

3.1 Class-Based Structure

- The project is organized into four primary logical modules:
- **DataProcessor (The Logic Layer):**
- **Functionality:** Utilizes **Xarray** to perform vectorized operations on the structural dataset.

- **Key Method:** extract_forces(element_list)—filters the global dataset for specific girder sequences like the central line [15, 24, 33, 42, 51, 60, 69, 78, 83].
- **GeometryEngine (The Spatial Layer):**
- **Functionality:** Maps the topological element-node connectivity to physical Cartesian coordinates.
- **Key Method:** get_3d_coordinates()—calculates the (x, y, z) start and end points for bridge members to build the 3D framing.
- **Visualizer (The Presentation Layer):**
- **Functionality:** A wrapper for **Plotly/PyPlot** that handles the rendering of SFD and BMD.
- **Key Method:** extrude_diagrams()—performs the "MIDAS-style" vertical extrusion by translating force magnitudes into Y-axis offsets.

Main Controller:

- **Functionality:** Orchestrates the flow between data extraction and visualization for both Task 1 (2D) and Task 2 (3D).

3.2 Data Flow Pipeline

- **Ingestion:** The Xarray dataset is loaded, and the system identifies M_z as the Bending Moment and V_y as the Shear Force.
- **Mapping:** The engine iterates through Girders 1–5 using the provided element tags (e.g., Girder 1: [13, 22, 31, 40, 49, 58, 67, 76, 81]) .
- **Transformation:** Force values at nodes i and j are normalized and scaled for visual clarity in the 3D environment.
- **Rendering:** Interactive Plotly traces are generated, applying color gradients or height extrusions as per the MIDAS visualization standard.

3.3 Design Best Practices

- **Encapsulation:** All structural properties and force extraction logic are encapsulated within classes to prevent global namespace pollution.
- **Error Handling:** The DataLoader includes exception handling for missing NetCDF variables or mismatched node IDs.
- **Compliance:** The code is linted according to PEP 8 standards and includes comprehensive docstrings, fulfilling the "Code Clarity" requirement.

4. Submission Compliance

- **Video**: A functional demonstration has been uploaded to YouTube as an unlisted link.
- **Licensing**: All work is submitted under the **Creative Commons Attribution-ShareAlike 4.0 International License** by FOSSEE.
- **Collaboration**: The user osdag-admin has been granted collaborator access to the GitHub repository.