IFCManager

Integrates the loading, analysis and visualization of AEC models from IFC files

---

IFCManager Version 0.1.0 MIT

Quick Start

```python
from ifc_openseespy_linker.core import IFCtoOpenSeesConverter

converter = IFCtoOpenSeesConverter()
converter.load_ifc('sample_models/CASA TIPO 11111.ifc')
```

Description

ifc_manager is a Python module designed to streamline the interaction with IFC (Industry Foundation Classes) files for structural engineering workflows. Built on top of ifcopenshell, it abstracts low-level complexity and provides a minimal, intuitive interface for loading, inspecting, and visualizing structural components within IFC models.

This module is suitable for structural engineers, BIM technicians, and researchers who wish to integrate BIM data into custom workflows—particularly in Python—without requiring advanced knowledge of the IFC schema or programming.

```
IFC_OPENSEESPY/
├── src/
│   └── ifc_openseespy_linker/
│       ├── core/
│       │   ├── __init__.py
│       │   ├── ifc_parser.py
│       │   └── visualization.py
│       ├── utils/
│       │   ├── __init__.py
│       │   └── geometry_utils.py
│       └── __init__.py
```

The left one is this manual, generated by a standalone `manual.typ` file maintained separately from the code; while the right one is a manual generated by `doc-comments.typ`, which in turn collects and parse doc-comments in `assets/module.typ` file.

Options

Those are the full list of options available and their intended behavior:

`py converter = IFCtoOpenSeesConverter(ifc_file_path=None)` → object

Initializes the converter. If an IFC file path is provided, the internal IFCParser is instantiated immediately. Otherwise, it must be loaded later using load_ifc().

`py converter.load_ifc(ifc_file_path=None)` → IFC object

Loads the IFC model file into memory using the ifcopenshell backend. This method either uses the path provided during instantiation or a new one passed as an argument. It also sets up the parser.

`py converter.extract_structural_elements()` → dict

Parses and extracts structural elements (e.g., beams, columns) and geometric data from the loaded IFC file. It also initializes the OpenSeesConverter using these elements.

`py converter.create_opensees_model()` → OpenSees model

Creates a structural model compatible with OpenSees. This function relies on the structural elements extracted previously and builds the node and element definitions for analysis.

`py converter.run_analysis(analysis_type='static')` → results

Runs a structural analysis using OpenSees. Currently supports static analysis. You must create the model first with create_opensees_model() before calling this.

`py converter.visualize_model()` → 3D plot

Generates a 3D visualization of the parsed IFC structural model. Useful for debugging geometry and structure placement prior to analysis.

`py converter.visualize_results(result_type='displacements', scale_factor=10)` → plot

Displays analysis results such as nodal displacements or internal forces. The scale_factor amplifies displacements visually to aid interpretation.

Dependencies

Requires the ifcopenshell library for parsing and operating on IFC files. To set up the project you will need the *ifcopenshell*[1] Python module, which provides tools for accessing geometry, property sets, and placements within a BIM model.

3D model generation and rendering are powered by *plotly*[2], offering interactive visualization through WebGL-based graphs. Alternatively, static plotting and surface visualizations can be achieved with *matplotlib*[3], which provides precise control over layout and rendering via its object-oriented API.

[1] https://pypi.org/project/ifcopenshell
[2] https://pypi.org/project/plotly
[3] https://pypi.org/project/matplotlib
[4] https://pypi.org/project/numpy
[5] https://pypi.org/project/math

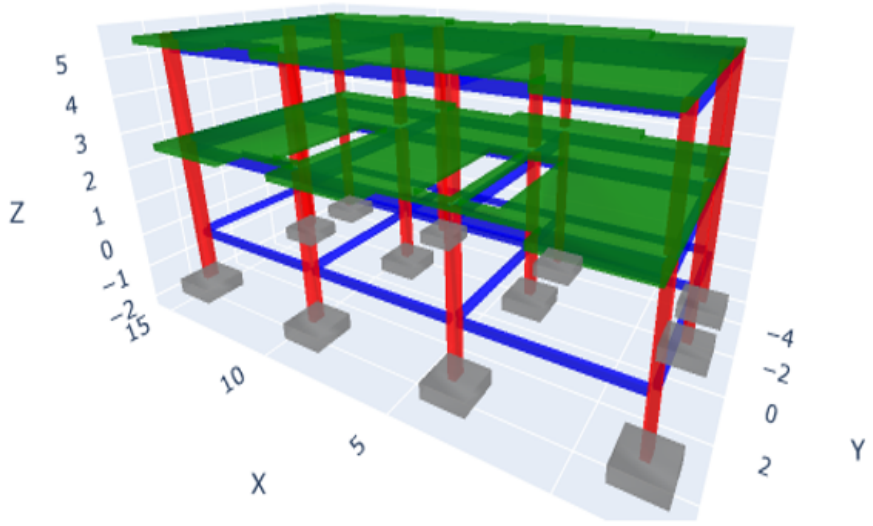
You will also need core scientific libraries such as *numpy*[4] for numerical operations, as well as *math*[5] (built-in) for mathematical functions.

Examples

```
from ifc_openseespy_linker.core import IFCtoOpenSeesConverter

converter = IFCtoOpenSeesConverter()
converter.load_ifc('sample_models/example.ifc')
structural_elements = converter.extract_structural_elements()
print(f"Extracted elements: {len(structural_elements)}")
print(structural_elements)
print("Element types:", {elem['type'] for elem in structural_elements.values()})
converter.visualize_model()
```

Output:
Extracted elements: 89
{‘1YMpECnoXAAgd3TsblgkRb’: {‘id’: ‘1YMpECnoXAAgd3TsblgkRb’, ‘name’: ‘M_Concrete-Rectangular Beam:250 x 500mm:416913’, ‘type’: ‘IfcBeam’, ‘geometry’: {‘type’: ‘mesh’, ‘vertices’: [[−1.7213125009910568, 3.8713542939384933, ….]]}



Copyright