Load SEL Program

Technical Report

1. Executive Summary

1.1. Purpose

This document provides a comprehensive technical overview of the **Load SEL Program** (also referred to as WE MechLoad Viewer). The program is a specialized desktop software application developed to address the needs of engineers in the field of mechanical analysis. Its core purpose is to provide an integrated, graphical environment for the loading, visualization, processing, and subsequent exporting of mechanical load data derived from either numerical simulations or physical testing. It is designed to significantly reduce the time and effort required to interpret complex datasets and prepare them for further analysis in Finite Element Analysis (FEA) software.

1.2. Scope

The scope of the Load SEL Program encompasses the entire workflow from raw data ingestion to final analysis-ready output. The application's capabilities are confined to handling specific pipe-delimited data formats (.pld files) containing either time-domain or frequency-domain mechanical load information. Key functionalities within this scope include:

- Multi-dataset loading and comparative visualization.
- Targeted plotting of individual data channels, grouped interfaces, and component-level loads.
- On-the-fly signal processing, including filtering, windowing, and data sectioning.
- Automated generation of input files (.dat and .inp scripts) for direct use in Ansys Mechanical for Transient Structural or Harmonic Response analyses.

The program is intended as a post-processing and data preparation tool, not as a solver or a data acquisition system.

2. Introduction

2.1. Background

In modern mechanical engineering, product design and validation cycles rely heavily on data. This data originates from two primary sources: computer-aided engineering (CAE) simulations, such as multi-body dynamics (MBD) and finite element analysis (FEA), and physical prototype testing using data acquisition (DAQ) systems. Both sources typically produce large, multi-channel data files that can be challenging to manage, compare, and interpret efficiently. Engineers often resort to writing custom scripts or performing manual data manipulation in spreadsheet software, which can be time-consuming, prone to error, and

lacks standardization.

2.2. Problem Definition

The central problem is the **lack of a streamlined, interactive workflow** for engineers to handle mechanical load data. The key challenges include:

1. **Inefficient Comparison:** Quickly comparing results from different design iterations or validating simulation data against test data is a cumbersome manual process.
2. **Lack of Integrated Tools:** Visualization, signal processing (e.g., filtering), and data formatting for subsequent analyses are typically performed in separate software environments, breaking the workflow.
3. **High Barrier to Entry:** Scripting-based solutions require programming knowledge, creating a barrier for many design and test engineers.
4. **Error-Prone Data Preparation:** Manually creating formatted load tables for FEA software like Ansys is a tedious task where errors can be easily introduced.

2.3. Proposed Solution

The Load SEL Program is the proposed solution to these challenges. It is a standalone, GUI-based desktop application that consolidates all the necessary steps into a single, intuitive tool. By providing an interactive environment for data exploration and a dedicated module for automated Ansys file generation, the program aims to:

- **Accelerate** the data analysis cycle.
- **Improve** engineering productivity.
- **Reduce** the likelihood of manual errors in data preparation.
- **Democratize** data analysis by making powerful tools accessible without requiring programming skills.

3. Analysis

3.1. System Architecture

The application is built using Python and the PyQt5 framework for the graphical user interface. It follows an architecture inspired by the **Model-View-Controller (MVC)** design pattern to ensure a clear separation of concerns, which enhances maintainability and scalability.

[a Model-View-Controller diagram resmi]

- **Model (DataManager):** This component is the data engine. It is solely responsible for all file system interactions, parsing the .pld data and header files, and structuring the data into a Pandas DataFrame. It has no knowledge of the user interface.
- **View (UI Tabs & Widgets):** This comprises all the graphical elements the user interacts with—the main window, the tabbed interface (SingleDataTab, PartLoadsTab, etc.), buttons, and dropdowns. Its role is to display data and capture user input.
- **Controller (PlotController, ActionHandler):** This layer acts as the brain of the

application. It listens for user actions from the View (via Qt signals), requests the necessary data from the Model, performs logic, and tells the View how to update. This keeps complex logic out of the UI code.

<!-- end list -->

```
Application Architecture
|
|-- main.py (Entry Point)
|
|-- Model (data_manager.py)
|   `-- Manages file I/O and data parsing into Pandas DataFrame.
|
|-- View (main_window.py, ui/*)
|   |-- QMainWindow (Container)
|   |-- QTabWidget
|   |   |-- SingleDataTab, PartLoadsTab, etc.
|   `-- DirectoryTreeDock (for multi-data selection)
|
`-- Controller (controllers/*)
    |-- PlotController
    |   `-- Handles all visualization logic.
    `-- ActionHandler
        `-- Handles export actions and complex UI events.
```

3.2. Data Flow

The application's interactivity is managed by **Qt's signal and slot mechanism**. Data flows through the system in a clear, event-driven pattern.

Example: Plotting a Single Channel

1. **User Action (View):** The user selects a channel from the column_selector dropdown in the SingleDataTab.
2. **Signal Emission (View):** The currentIndexChanged signal of the dropdown is emitted.
3. **Slot Activation (Controller):** This signal is connected to the update_single_data_plots() slot in the PlotController.
4. **Data Request (Controller -> Model/State):** The controller accesses the primary DataFrame (held as state in MainWindow) and reads the current settings from the View (the selected channel name).
5. **Processing (Utility):** The controller sends the data and settings to the Plotter utility, which generates a Plotly figure.
6. **View Update (Controller -> View):** The controller passes the generated figure back to the SingleDataTab, which renders it in its QWebEngineView.

This uni-directional data flow (View -> Controller -> Model -> View) is fundamental to the application's stability.

3.3. GUI Design

The GUI is built with PyQt5 and is designed for engineering workflows.

- **Tabbed Interface:** The primary workspace is a QTabWidget. This design choice logically separates different analysis tasks (single channel, interface, comparison, etc.), preventing UI clutter and guiding the user through different stages of analysis.
- **Dockable Widget:** The DirectoryTreeDock provides an intuitive way to manage and compare multiple datasets without navigating complex file menus.
- **Interactive Plots:** By integrating Plotly figures within a QWebEngineView, the plots are not static images. Users can zoom, pan, and hover over data points to get precise values, which is essential for detailed analysis.

3.4. Key Algorithms

The program relies on several key algorithms, primarily for signal processing, provided by the SciPy library.

- **Low-Pass Filtering:** A **Butterworth filter** (scipy.signal.butter and scipy.signal.filtfilt) is used. This is a common type of infinite impulse response (IIR) filter known for its maximally flat frequency response in the passband, making it ideal for removing high-frequency noise without distorting the underlying signal.
- **Windowing:** A **Tukey window** (scipy.signal.windows.tukey), also known as a tapered cosine window, is applied. This algorithm is crucial for preparing time-domain data for Fourier analysis. It tapers the signal towards zero at its edges, minimizing spectral leakage and ensuring more accurate frequency-domain representations.
- **Time-Domain Reconstruction:** For the "Time Domain Rep." tab, the algorithm uses basic trigonometric principles. For a given frequency f, it reads the magnitude A and phase phi (in degrees) from the data and reconstructs the time-domain signal y(t) for each channel using the formula:
- **Time-Domain Reconstruction:** For the "Time Domain Rep." tab, the algorithm uses basic trigonometric principles. For a given frequency f, it reads the magnitude A and phase phi (in degrees) from the data and reconstructs the time-domain signal y(t) for each channel using the formula:$y(t)=A\cdot\sin(2\pi ft+\phi 180\pi)$

4. Results

4.1. Visualization Capabilities

The program delivers a suite of specialized plots, enabling comprehensive data interrogation. Users can generate plots of magnitude vs. time/frequency, phase vs. frequency, and time-frequency spectrograms to identify critical events and their frequency content. The ability to overlay multiple datasets from the directory tree provides immediate visual feedback

on the effects of design changes.

4.2. Data Processing Features

The implemented on-the-fly data processing features allow users to instantly clean and condition their data. The low-pass filter is effective at removing sensor noise, while the data sectioning and Tukey windowing tools are critical for preparing signals for accurate export and use in FEA frequency-domain analyses.

4.3. Ansys Integration

The Ansys export functionality is a key result. The program successfully generates well-formatted APDL scripts and associated data files that can be directly consumed by Ansys Mechanical. This automates a previously manual and error-prone process, creating a seamless link between the data analysis and simulation stages of a project.

4.4. Performance

Through the use of the Pandas library for backend data manipulation, the application handles large datasets efficiently. UI responsiveness is maintained even with hundreds of thousands of data points. For the initial file I/O, a QTimer is used to ensure the GUI is fully rendered and interactive before the potentially blocking data-loading process begins, resulting in a smooth user experience.

5. Conclusion

5.1. Summary of Achievements

The Load SEL Program successfully achieves its objective of providing a robust, user-friendly tool for mechanical load data analysis. It effectively bridges the gap between raw data and engineering insight by integrating visualization, processing, and exporting into a single, cohesive workflow. The application demonstrably accelerates the analysis cycle and enhances the reliability of preparing data for FEA simulations.

5.2. Future Work

While the current version is fully functional, several enhancements could be implemented in the future:

- **Multi-threading:** For exceptionally large datasets (millions of points), moving the data loading and processing functions to a separate QThread would prevent any possibility of GUI freezing.
- **Advanced Analysis Tools:** Incorporate more signal processing features, such as on-the-fly Fast Fourier Transform (FFT) calculation, power spectral density (PSD) plots, and fatigue analysis tools.
- **Expanded File Support:** Add parsers for other common engineering data formats (e.g., RPC, CSV, MATLAB .mat files).

- **User-Defined Channels:** Implement a feature that allows users to create new data channels through mathematical operations on existing ones.

6. References

1. *PyQt5 Documentation*. Riverbank Computing. https://www.riverbankcomputing.com/static/Docs/PyQt5/
2. *Pandas Documentation*. The Pandas Development Team. https://pandas.pydata.org/docs/
3. *SciPy Documentation*. The SciPy Community. https://docs.scipy.org/
4. *Ansys Mechanical APDL Command Reference*. Ansys, Inc.

7. Nomenclature

- **APDL:** Ansys Parametric Design Language. The scripting language used to control Ansys Mechanical.
- **FEA:** Finite Element Analysis. A numerical method for solving engineering and mathematical physics problems.
- **GUI:** Graphical User Interface.
- **MVC:** Model-View-Controller. A software design pattern.
- **Signal:** In Qt, an event emitted by an object when its state changes.
- **Slot:** In Qt, a function that is called in response to a specific signal.
- **Time Domain:** A representation of a signal as a function of time.
- **Frequency Domain:** A representation of a signal as a function of frequency.
- **Interface:** A specific connection point in a mechanical assembly where loads are measured or applied (e.g., a bolt location).
- **Part Load:** A collection of loads associated with a larger component or side of an assembly.