Python Implementation for OpenDSS Analysis
IEEE 118-Bus System

Power System Analysis Documentation

February 24, 2025

1 Introduction

This document explains the Python implementation for analyzing the IEEE 118-bus system using OpenDSS. The implementation consists of several key components: initialization, data collection, analysis, and visualization.

2 Core Implementation

2.1 OpenDSS Interface

The primary interface with OpenDSS is established through the OpenDSSDirect.py package:

Listing 1: OpenDSS Interface Setup

```
import opendssdirect as dss
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

2.2 System Initialization

The initialization process sets up the OpenDSS environment and loads the circuit components:

Listing 2: System Initialization Function

```
def initialize_opendss():
    # Clear any existing circuit
    dss.run_command('Clear')

    # Set the frequency
    dss.run_command('Set DefaultBaseFrequency=50')

    # Create new circuit
    dss.run_command('New Circuit.ieee118bus basekv=138.0 phases=3 bus1=89
        _clinchrv')

    # Load components
    dss.run_command('Redirect generators.dss')
    dss.run_command('Redirect lines.dss')
```

```
    dss.run_command('Redirect transformers.dss')
    dss.run_command('Redirect loads.dss')
    dss.run_command('Redirect shunts.dss')

    # Set voltage bases
    dss.run_command('Set VoltageBases=[138.0]')
    dss.run_command('Calcv')

    # Solution parameters
    dss.run_command('set algorithm=NCIM')
    dss.run_command('set maxiterations=100')
    dss.run_command('set tolerance=0.0001')
    dss.run_command('set loadmodel=1')
```

3 Data Collection and Analysis

3.1 Voltage Data Collection

The implementation includes functions to collect voltage data from all buses:

Listing 3: Voltage Data Collection Function

```
def get_voltage_data():
    voltages = []
    bus_names = []

    dss.Circuit.SetActiveBus('')
    for bus in dss.Circuit.AllBusNames():
        dss.Circuit.SetActiveBus(bus)
        v_mag = dss.Bus.puVmagAngle()[0]
        voltages.append(v_mag)
        bus_names.append(bus)

    return pd.DataFrame({
        'Bus': bus_names,
        'Voltage (pu)': voltages
    })
```

3.2 Voltage Profile Visualization

The visualization component creates comprehensive plots of the voltage profile:

Listing 4: Voltage Profile Visualization Function

```
def plot_voltage_profile(df):
    plt.style.use('default')
    plt.rcParams['figure.figsize'] = [15, 10]
    plt.rcParams['figure.dpi'] = 300

    fig = plt.figure()

    # Bar plot of voltage magnitudes
    plt.subplot(2, 1, 1)
```

```
    plt.bar(range(len(df)), df['Voltage (pu)'],
            alpha=0.6, color='skyblue')
    plt.axhline(y=1.05, color='r', linestyle='--',
                label='Upper Limit (1.05 pu)')
    plt.axhline(y=0.95, color='r', linestyle='--',
                label='Lower Limit (0.95 pu)')
    plt.title('Voltage Profile of IEEE 118-Bus System')
    plt.xlabel('Bus Number')
    plt.ylabel('Voltage (pu)')
    plt.legend()

    # Voltage distribution histogram
    plt.subplot(2, 1, 2)
    sns.histplot(data=df['Voltage (pu)'],
                 bins=30, kde=True)
    plt.axvline(x=1.05, color='r', linestyle='--')
    plt.axvline(x=0.95, color='r', linestyle='--')
    plt.title('Voltage Distribution')
    plt.xlabel('Voltage (pu)')
    plt.ylabel('Count')

    plt.savefig('latex_report/voltage_profile.png',
                bbox_inches='tight')
    plt.close()
```

4 Statistical Analysis

The implementation includes comprehensive statistical analysis:

Listing 5: Statistical Analysis Implementation

```
def analyze_voltage_statistics(df):
    voltage_status = pd.cut(
        df['Voltage (pu)'],
        bins=[-np.inf, 0.95, 1.05, np.inf],
        labels=['Low', 'Normal', 'High']
    )

    status_counts = voltage_status.value_counts()
    total = len(df)
    percentages = (status_counts / total * 100).round(2)

    summary_stats = {
        'Mean': df['Voltage (pu)'].mean(),
        'Maximum': df['Voltage (pu)'].max(),
        'Minimum': df['Voltage (pu)'].min(),
        'Std Dev': df['Voltage (pu)'].std()
    }

    return status_counts, percentages, summary_stats
```

5 Results Generation

The implementation includes functions to generate comprehensive reports:

Listing 6: Results Generation

```
def generate_voltage_summary(df, counts, percentages, stats):
    with open('latex_report/voltage_summary.txt', 'w') as f:
        f.write("Voltage Profile Summary\n")
        f.write("======================\n\n")

        # Write classification results
        summary_df = pd.DataFrame({
            'Count': counts,
            'Percentage (%)': percentages
        })
        f.write(str(summary_df))

        # Write statistics
        f.write("\n\nVoltage Statistics:\n")
        f.write("-------------------\n")
        for key, value in stats.items():
            f.write(f"{key}: {value:.4f} pu\n")
```

6 Main Execution Flow

The main execution flow coordinates all components:

Listing 7: Main Execution Function

```
def main():
    # Initialize OpenDSS
    initialize_opendss()

    # Get voltage data
    voltage_data = get_voltage_data()

    # Create visualizations
    plot_voltage_profile(voltage_data)

    # Perform statistical analysis
    counts, percentages, stats = analyze_voltage_statistics(voltage_data)

    # Generate summary
    generate_voltage_summary(voltage_data, counts, percentages, stats)

    print("Analysis completed. Check latex_report directory for outputs.")

if __name__ == "__main__":
    main()
```

7 Key Features

The implementation provides several key features:

1. **Modularity**: Separate functions for different aspects of analysis
2. **Data Management**: Efficient handling of voltage data using pandas
3. **Visualization**: Comprehensive plotting using matplotlib and seaborn
4. **Statistical Analysis**: Detailed voltage profile statistics
5. **Report Generation**: Automated generation of analysis reports

8 Error Handling

The implementation includes basic error handling:

Listing 8: Error Handling Example

```
try:
    initialize_opendss()
except Exception as e:
    print(f"Error initializing OpenDSS: {str(e)}")
    error = dss.Error.Description()
    if error:
        print(f"OpenDSS Error: {error}")
    sys.exit(1)
```

9 Conclusion

This Python implementation provides a comprehensive framework for analyzing power systems using OpenDSS. It combines efficient data processing, detailed analysis, and clear visualization to provide insights into system behavior.