Technical Documentation: Mathematical Models and Algorithms for the Advanced Parkinson’s Movement Analyzer

Generated by Github.com

July 7, 2025

Contents

1 Introduction

This document provides a detailed explanation of the mathematical models, signal processing algorithms, and core libraries used in the *Advanced Parkinson's Movement Analyzer* application. The system is designed to acquire, analyze, and interpret accelerometer data from wearable sensors to provide quantitative insights into motor symptoms, particularly hand tremors.

The workflow is divided into four primary stages:

1. **Data Acquisition:** Fetching raw 3-axis accelerometer data from a Firebase Realtime Database, either in real-time or from stored recordings.
2. **Signal Processing:** Pre-processing the raw time-series data to prepare it for analysis. This includes time normalization and calculating key signal properties.
3. **Feature Extraction & Analysis:** Applying a suite of time-domain, frequency-domain, and statistical algorithms to quantify tremor characteristics like intensity, frequency, regularity, and smoothness.
4. **Interpretation & Visualization:** Synthesizing the extracted features into clinically relevant metrics, including a composite severity index, and presenting the results through an interactive dashboard and a comprehensive, downloadable HTML report.

2 Core Libraries and Roles

The application leverages several powerful open-source Python libraries for scientific computing and data visualization. The primary libraries are summarized below.

3 Mathematical Models and Algorithms

3.1 Data Preparation and Pre-processing

3.1.1 Time Normalization and Sampling Frequency

Before analysis, the absolute timestamps are converted into a relative time vector starting from zero. The sampling frequency (f_s), a critical parameter for frequency analysis, is then derived.
Algorithm: The sampling frequency is the total number of samples divided by the total duration of the signal in seconds.

$$f_s = \frac{N}{\Delta t} = \frac{\text{Number of Samples}}{\text{Total Time (s)}} \tag{1}$$

Implementation:

```python
# Calculate duration in seconds
df['time_s'] = (df['t'] - df['t'].iloc[0]) / 1000.0
duration = df['time_s'].iloc[-1] - df['time_s'].iloc[0]
# Calculate sampling frequency
fs = len(df) / duration
```

Libraries Used: `pandas`, `numpy`.

3.2 Time-Domain Analysis

3.2.1 Signal Magnitude Vector (SMV)

To create a rotation-invariant measure of tremor intensity, the three separate accelerometer axes (a_x, a_y, a_z) are combined into a single magnitude vector.

Table 1: Key Python Libraries and Their Functions

Library	Primary Role in the Application
`numpy`	The cornerstone for all numerical operations. Used for array manipulation, fundamental mathematical functions (`sqrt`, `mean`, `gradient`), and serves as the computational engine for most other scientific libraries.
`pandas`	Provides high-performance, easy-to-use data structures, primarily the `DataFrame`. Used to structure the incoming time-series data, handle missing values, and perform efficient data selection and manipulation.
`scipy`	The core library for scientific and technical computing. Specific modules used include: • `scipy.fft`: For performing the Fast Fourier Transform (FFT) to analyze the frequency content of the signal. • `scipy.signal`: For generating the spectrogram via the Short-Time Fourier Transform (STFT). • `scipy.stats`: For calculating statistical measures like spectral entropy.
`plotly`	The primary library for creating rich, interactive, and aesthetically pleasing data visualizations. Used to generate all charts, gauges, and heatmaps on the Streamlit dashboard.
`streamlit`	The web application framework used to build the entire user interface, manage application state, and handle user interactions through widgets like sliders, buttons, and tabs.
`requests`	Handles all network communication. Used to make HTTP requests (`GET`, `PUT`, `DELETE`) to the Firebase REST API for data retrieval and storage.

Model: The Euclidean norm of the acceleration vector at each time point.

$$\text{Magnitude} = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{2}$$

Implementation:

```
df['total_mag'] = np.sqrt(df['ax1']**2 + df['ay1']**2 + df['az1']**2)
```

Library Used: numpy.

3.2.2 Root Mean Square (RMS)

RMS quantifies the overall intensity or "power" of the tremor signal.

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \text{signal}(i)^2} \tag{3}$$

Implementation:

```
rms_tremor = np.sqrt(np.mean(df['total_mag'] ** 2))
```

Library Used: numpy.

3.2.3 Jerk Analysis

Jerk (the time derivative of acceleration) is used as a proxy for movement smoothness. A higher RMS of jerk indicates a more erratic, less smooth movement.

$$\text{Jerk}(t) = \frac{d}{dt}\text{Acceleration}(t) \tag{4}$$

Implementation: The numerical gradient is used to approximate the derivative.

```
df['jerk'] = np.gradient(df['total_mag'], df['time_s'])
rms_jerk = np.sqrt(np.mean(df['jerk'] ** 2))
```

Library Used: numpy.

3.3 Frequency-Domain Analysis

3.3.1 Fast Fourier Transform (FFT) and Power Spectrum

The FFT is used to decompose the time-domain signal into its constituent frequency components. The power spectrum shows the distribution of the signal's power across these frequencies.
Algorithm: The Real FFT (rfft) is used for efficiency. The power spectrum is the squared magnitude of the FFT output.

```
# N is the number of samples, fs is the sampling frequency
yf = rfft(df['total_mag'].to_numpy())
xf = rfftfreq(N, 1 / fs)
power_spectrum = np.abs(yf)**2
```

Library Used: scipy.fft.

3.3.2 Power in Parkinsonian Band (3-7 Hz)

This clinically relevant biomarker measures the percentage of the tremor's total energy that is concentrated in the typical Parkinsonian frequency range.

$$\text{Power Ratio}(\%) = \frac{\sum_{f=3\text{Hz}}^{7\text{Hz}} \text{Power}(f)}{\sum_{f=0}^{f_s/2} \text{Power}(f)} \times 100 \tag{5}$$

Implementation:

```
power_in_band_3_7_mask = (xf >= 3) & (xf <= 7)
power_in_band_3_7 = np.sum(power_spectrum[power_in_band_3_7_mask])
total_power = np.sum(power_spectrum)
power_in_band_ratio_3_7 = power_in_band_3_7 / total_power
```

Library Used: numpy.

3.3.3 Spectral Entropy

This measures the regularity of the tremor signal. A lower entropy value indicates a highly regular, predictable tremor (concentrated in a few frequencies), while a higher value indicates a more random, unpredictable tremor.
Model: Shannon Entropy applied to the normalized power spectrum (p_i).

$$H(X) = -\sum_i p_i \log_2(p_i) \quad \text{where} \quad p_i = \frac{\text{Power}(f_i)}{\sum_j \text{Power}(f_j)} \tag{6}$$

Implementation:

```
spectral_entropy_val = entropy(power_spectrum / total_power)
```

Library Used: scipy.stats.

3.4 Time-Frequency Analysis

3.4.1 Spectrogram

The spectrogram visualizes how the frequency content of the tremor changes over time. It is generated using the Short-Time Fourier Transform (STFT), which computes FFTs on short, overlapping segments of the signal.

```
f_spec, t_spec, Sxx = spectrogram(df['total_mag'], fs)
```

Library Used: `scipy.signal`.

3.5 Composite Models and Clinical Features

3.5.1 Composite Severity Index

A custom weighted model that combines three normalized biomarkers into a single score (0-1) representing overall tremor severity.

$$\text{Index} = \frac{w_{rms} \cdot \text{norm}_{rms} + w_{freq} \cdot \text{norm}_{freq} + w_{jerk} \cdot \text{norm}_{jerk}}{w_{rms} + w_{freq} + w_{jerk}} \tag{7}$$

Where each 'norm' term is the corresponding metric scaled to a [0, 1] range and 'w' terms are user-adjustable weights. **Implementation:**

```
# Normalization (example for RMS)
norm_rms = min(rms_tremor / 4000, 1.0)
# Composite Index Calculation
composite_index = (w_rms * norm_rms + w_freq * norm_power_ratio + w_jerk *
    norm_jerk) / weight_sum
```

Library Used: `streamlit` (to get weights from UI), standard Python.

3.5.2 Stabilizer Effectiveness

This metric quantifies the percentage reduction in tremor intensity achieved by the stabilizing device.

$$\text{Effectiveness}(\%) = \left(1 - \frac{\text{RMS}_{\text{stabilized}}}{\text{RMS}_{\text{raw}}}\right) \times 100 \tag{8}$$

Implementation:

```
effectiveness = (1 - (np.sqrt(np.mean(df['total_mag_stable']**2)) / rms_tremor)
    ) * 100
```

Library Used: `numpy`.

4 Visualization and Reporting

4.1 Interactive Dashboard (Plotly)

The application dashboard is built entirely with `plotly`. It translates the numerical results of the analysis into interactive visualizations, including:

- **Gauges:** For displaying the Composite Index.
- **Line Charts:** For showing time-series data of raw and stabilized movement.
- **Bar Charts:** For the FFT power spectrum and per-axis standard deviation.
- **Heatmaps:** For the spectrogram and cross-axis correlation matrix.
- **Scatter Plots:** For the Poincaré plot and 3D movement trajectory.

4.2 Automated HTML/SVG Report Generation

A key feature is the ability to generate a self-contained, downloadable HTML report. Instead of exporting static image files, the application programmatically generates Scalable Vector Graphics (SVG) code for the charts directly within the HTML.
Algorithm:

1. **Data Down-sampling:** For large datasets, the data is sampled to a maximum of 500 points to keep the report file size manageable.
2. **Robust Scaling:** The Y-axis maximum is determined using a high quantile (e.g., 99th percentile) to prevent a single outlier from compressing the visual scale of the chart.
3. **Coordinate Mapping:** Data values (e.g., time, acceleration) are linearly scaled to fit the pixel-based coordinate system of the SVG 'viewBox'.
4. **SVG Path Generation:** The scaled coordinates are used to construct the 'points' attribute for an SVG '¡polyline¿' (for the line) and the 'd' attribute for an SVG '¡path¿' (for the filled area), creating a complete chart from text strings.

This method ensures the final report is a single, portable file with no external dependencies, making it ideal for sharing and archiving.