jSciPy: A Java Scientific Computing Library

jSciPy is a Java library designed for scientific computing, offering functionalities inspired by popular scientific computing libraries. It currently provides modules for signal processing, including Butterworth filters, peak finding algorithms, and an RK4 solver for ordinary differential equations.

Features

- **Butterworth Filters**:
  - Implement various types of Butterworth filters: low-pass, high-pass, band-pass, and band-stop.
  - Supports zero-phase filtering (`filtfilt`) for applications where phase distortion is critical.
  - Provides standard filtering (`filter`) for causal applications.
- **Peak Finding**:
  - Efficiently detect peaks in one-dimensional signals.
  - Filter peaks based on properties like height, prominence, and minimum distance between peaks.
- **RK4 Solver**:
  - Solve ordinary differential equations using the Runge-Kutta 4th order method.
  - Flexible interface for defining custom differential equations.

Getting Started

Prerequisites

- Java Development Kit (JDK) 8 or higher
- Gradle (for building the project)

How to Include as a Dependency (JitPack)

JitPack is a novel package repository for JVM projects. It builds GitHub projects on demand and provides ready-to-use artifacts (jar, javadoc, sources).

To use this library in your Gradle project, add the JitPack repository and the dependency to your `build.gradle` file:

```
// In your root build.gradle (or settings.gradle for repository
definition)
allprojects {
    repositories {
        mavenCentral()
        maven { url 'https://jitpack.io' }
    }
}

// In your app's build.gradle
dependencies {
```

```
    implementation 'com.github.hissain:jSciPy:1.0.1' // Replace 1.0.1 with
the desired version or commit hash
}
```

Validation

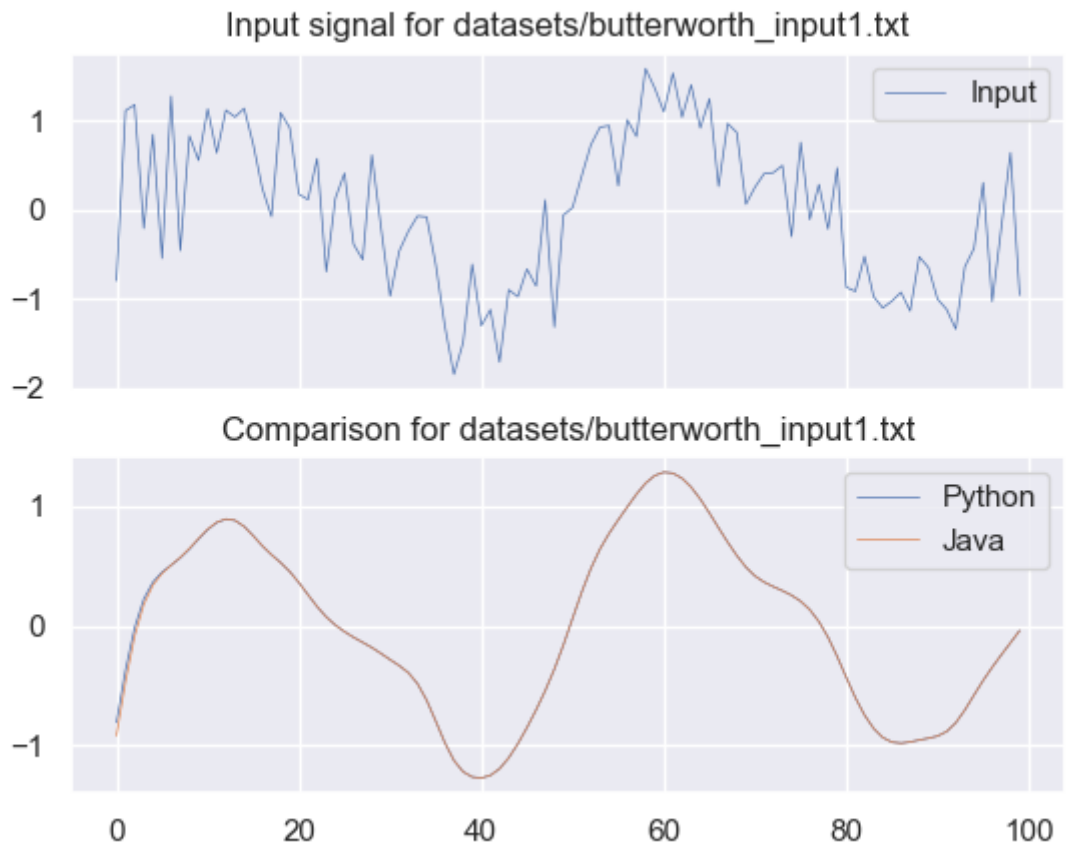
API Usage

For detailed examples on how to use the jSciPy library's public API, refer to the API Usage Examples document.

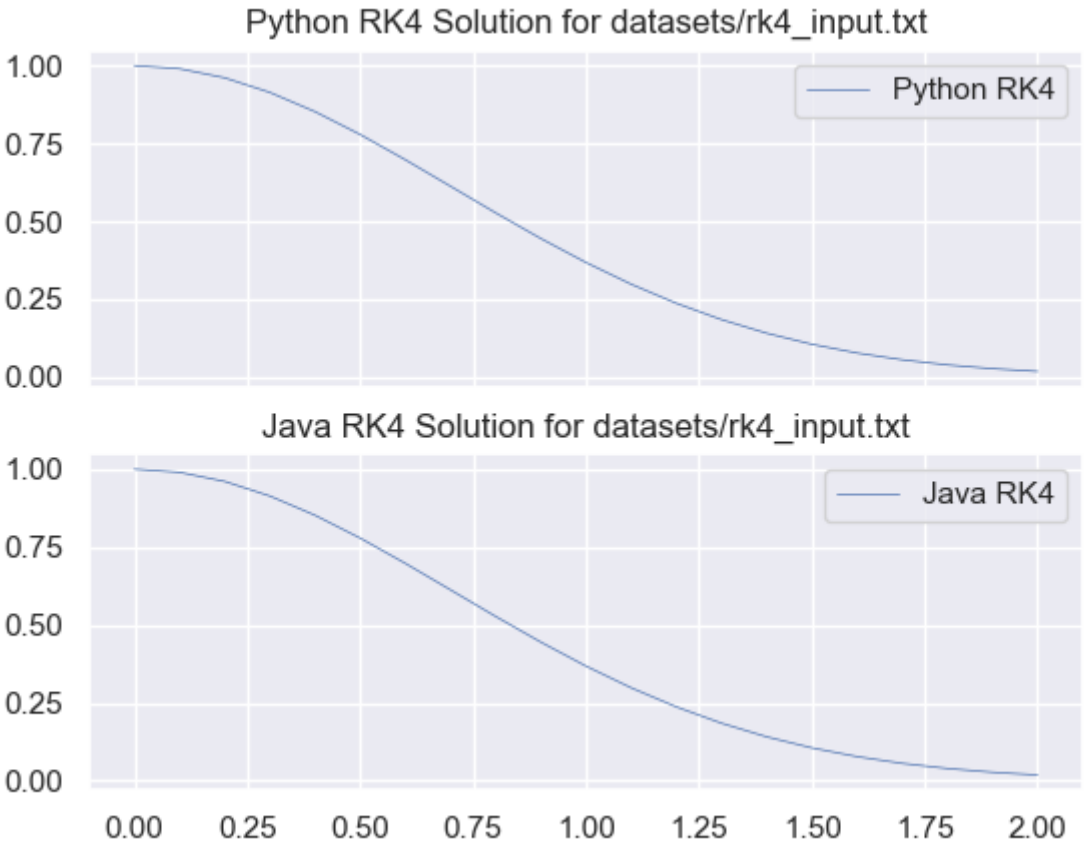
Validation

The Java implementations have been validated against scipy's implementation.

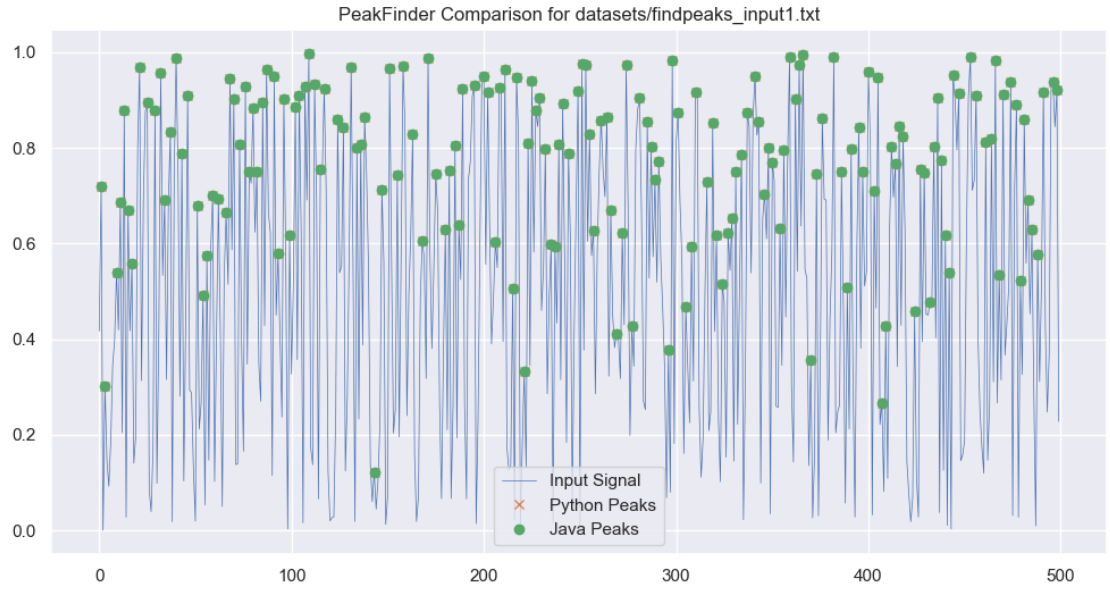
Butterworth Filter Comparison



RK4 Solver Comparison



PeakFinder Comparison



Usage Examples

Butterworth Filter

```java
import com.hissain.jscipy.signal.ButterworthFilter;

public class FilterExample {
    public static void main(String[] args) {
        double[] signal = {1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.4, 1.3, 1.2,
1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5};
        double sampleRate = 100.0; // Hz
        double cutoffFrequency = 10.0; // Hz
        int order = 4;

        ButterworthFilter filter = new ButterworthFilter();

        // Apply a low-pass filter (zero-phase)
        double[] filteredSignal = filter.filtfilt(signal, sampleRate,
cutoffFrequency, order);

        System.out.println("Filtered Signal (filtfilt):");
        for (double value : filteredSignal) {
            System.out.printf("%.2f ", value);
        }
        System.out.println();

        // Apply a low-pass filter (causal)
        double[] causalFilteredSignal = filter.filter(signal, sampleRate,
cutoffFrequency, order);

        System.out.println("Filtered Signal (causal filter):");
        for (double value : causalFilteredSignal) {
            System.out.printf("%.2f ", value);
        }
        System.out.println();
    }
}
```

Peak Finder

```java
import com.hissain.jscipy.signal.PeakFinder;
import java.util.Map;

public class PeakFinderExample {
    public static void main(String[] args) {
        double[] signal = {0.0, 1.0, 0.5, 2.0, 0.3, 1.5, 0.8, 3.0, 0.2,
1.0};

        PeakFinder peakFinder = new PeakFinder();
        PeakFinder.PeakParams params = new PeakFinder.PeakParams();
        params.distance = 2; // Minimum distance between peaks
        params.height = 0.5; // Minimum height of peaks
        params.prominence = 0.5; // Minimum prominence of peaks
```

```
        PeakFinder.PeakResult result = peakFinder.findPeaks(signal,
params);

        System.out.println("Detected Peaks at indices:");
        for (int peakIndex : result.peaks) {
            System.out.print(peakIndex + " ");
        }
        System.out.println();

        System.out.println("Peak Properties:");
        for (Map.Entry<String, double[]> entry :
result.properties.entrySet()) {
            System.out.println(entry.getKey() + ": " +
java.util.Arrays.toString(entry.getValue()));
        }
    }
}
```

RK4 Solver

```
import com.hissain.jscipy.signal.RK4Solver;
import com.hissain.jscipy.signal.api.IRK4Solver;

public class RK4SolverExample {
    public static void main(String[] args) {
        // Define a simple differential equation: dy/dt = -y
        IRK4Solver.DifferentialEquation equation = (t, y) -> -y;

        double y0 = 1.0; // Initial value of y
        double t0 = 0.0; // Initial time
        double tf = 5.0; // Final time
        double h = 0.1;  // Step size

        RK4Solver solver = new RK4Solver();
        RK4Solver.Solution solution = solver.solve(equation, y0, t0, tf,
h);

        System.out.println("RK4 Solver Solution:");
        System.out.println("Time (t)\tValue (y)");
        for (int i = 0; i < solution.t.length; i++) {
            System.out.printf("%.2f\t\t%.4f\n", solution.t[i],
solution.y[i]);
        }
    }
}
```

Contributing

Contributions are welcome! Please feel free to submit issues or pull requests.

License

This project is licensed under the MIT License.