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.
- **Find Peaks**:
  - 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.
- **Interpolation**:
  - Perform linear interpolation between data points.
  - Perform cubic spline interpolation for smoother curves.
- **FFT and RFFT**:
  - Compute the Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) of a signal.
  - Compute the Real Fast Fourier Transform (RFFT) and Inverse Real Fast Fourier Transform (IRFFT) for real-valued signals, which are more efficient.

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)
```

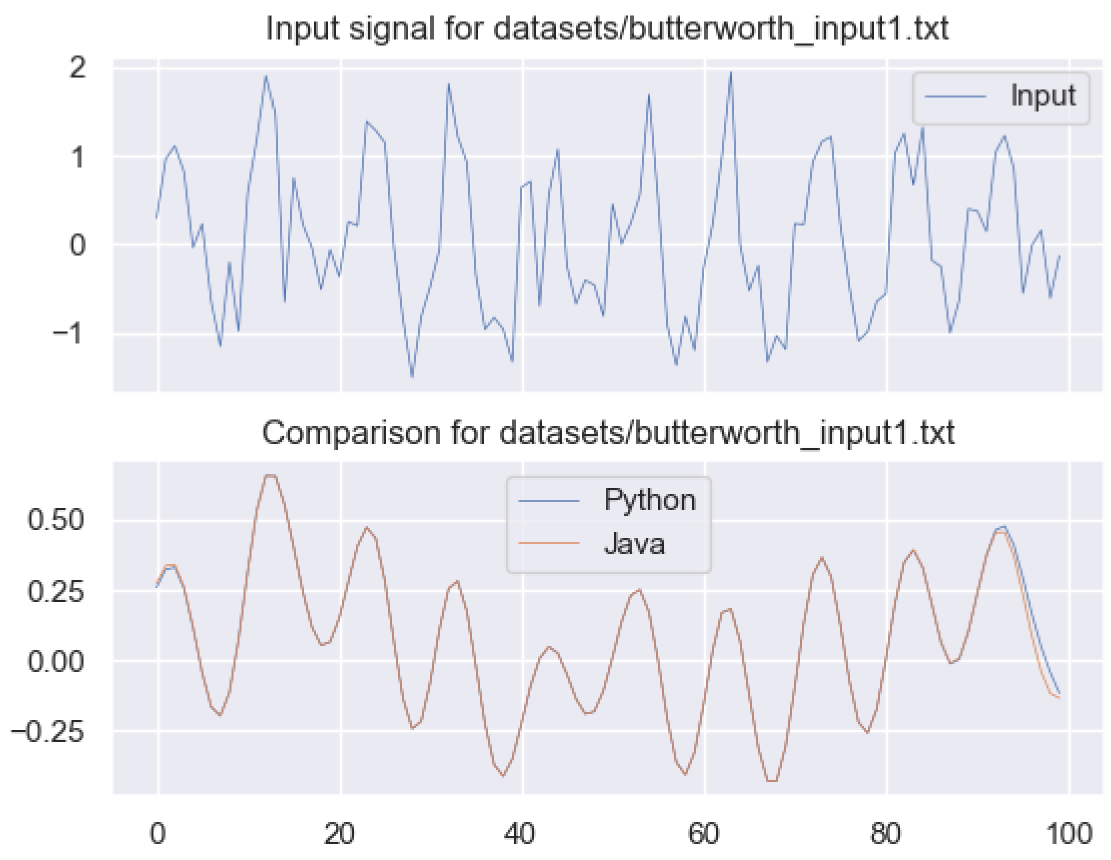
```
allprojects {
    repositories {
        mavenCentral()
        maven { url 'https://jitpack.io' }
    }
}

// In your app's build.gradle
dependencies {
    implementation 'com.github.hissain:jSciPy:1.0.2' // Replace 1.0.2 with
the desired version or commit hash
}
```

Exmple Demo Android Application

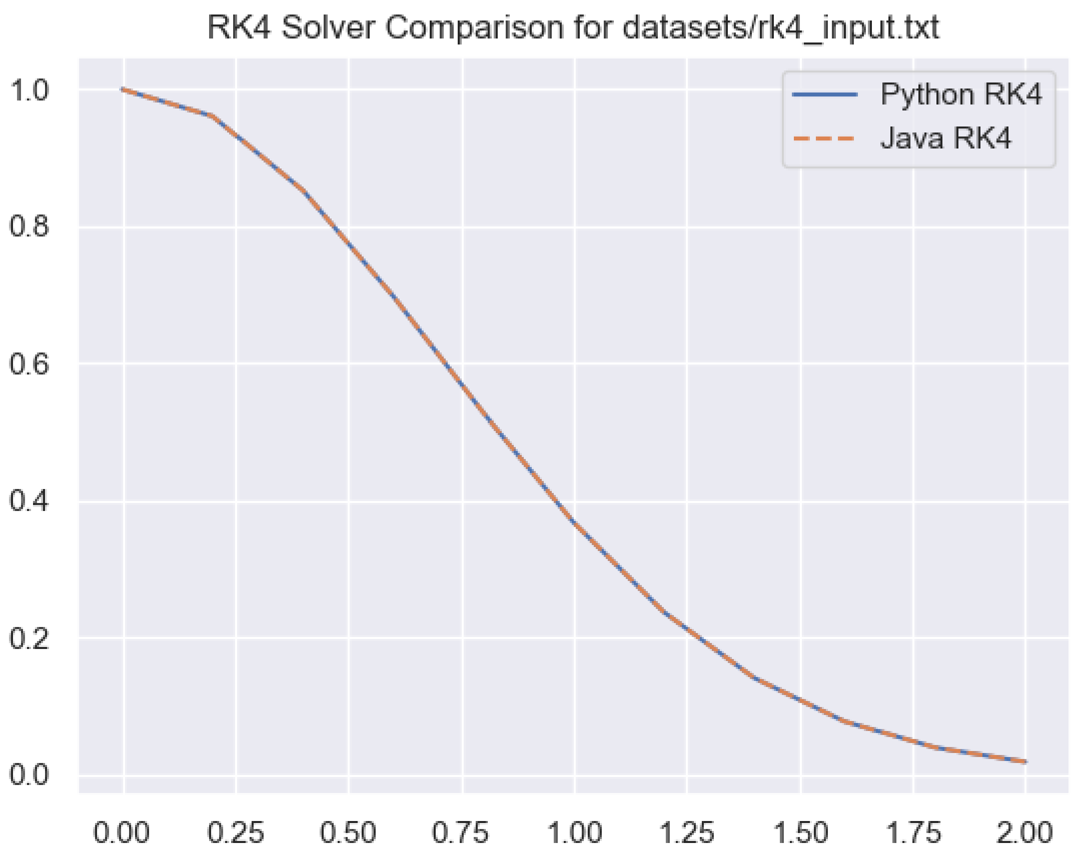
A seperate demo android application is built on this library that might be helpful to understand how to consume this library. The application can be accessed here.

Comparison Graphs

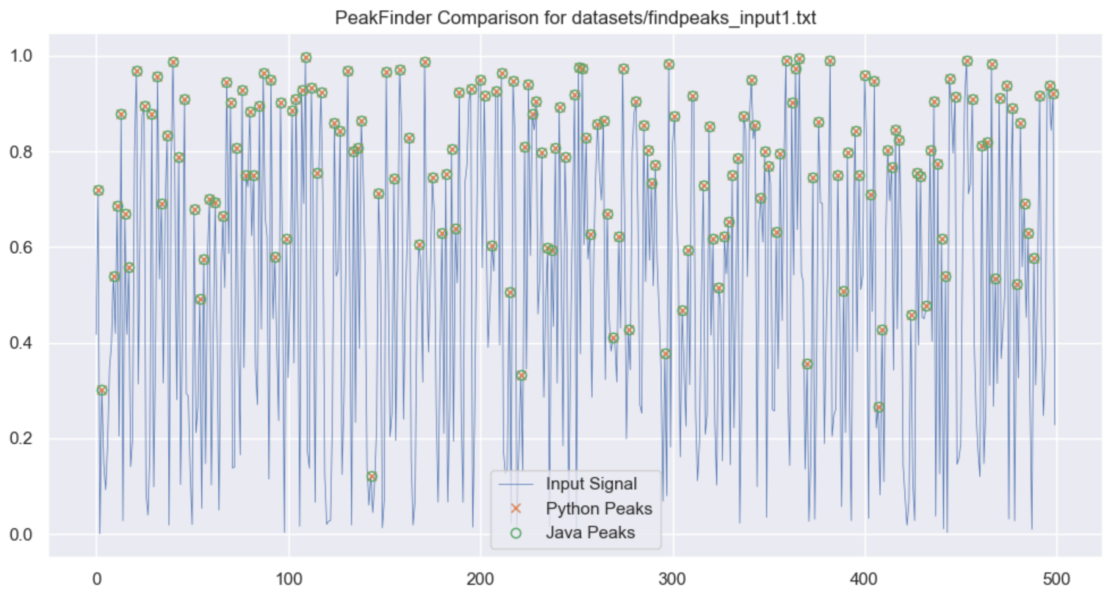
Butterworth Filter Comparison



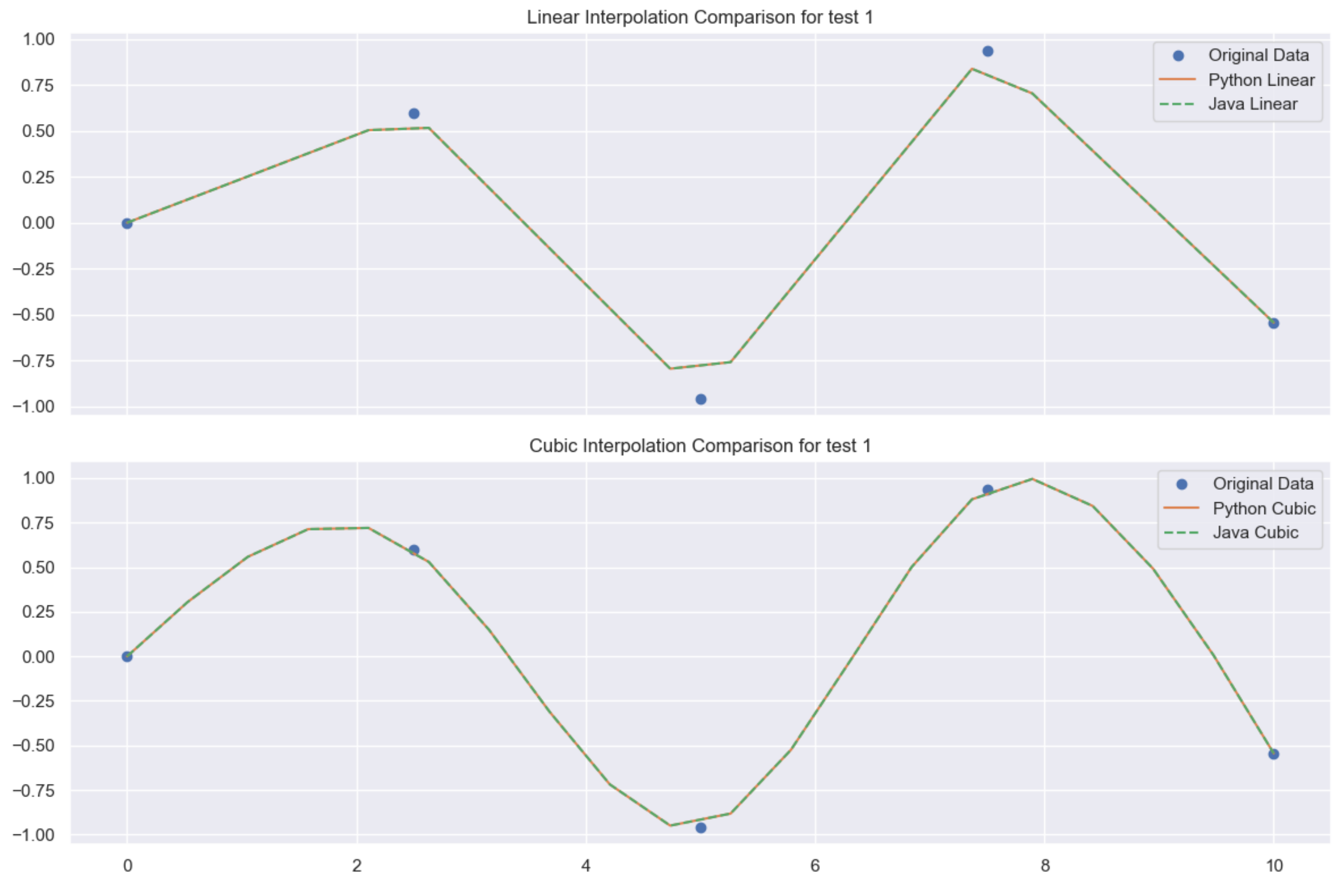
RK4 Solver Comparison



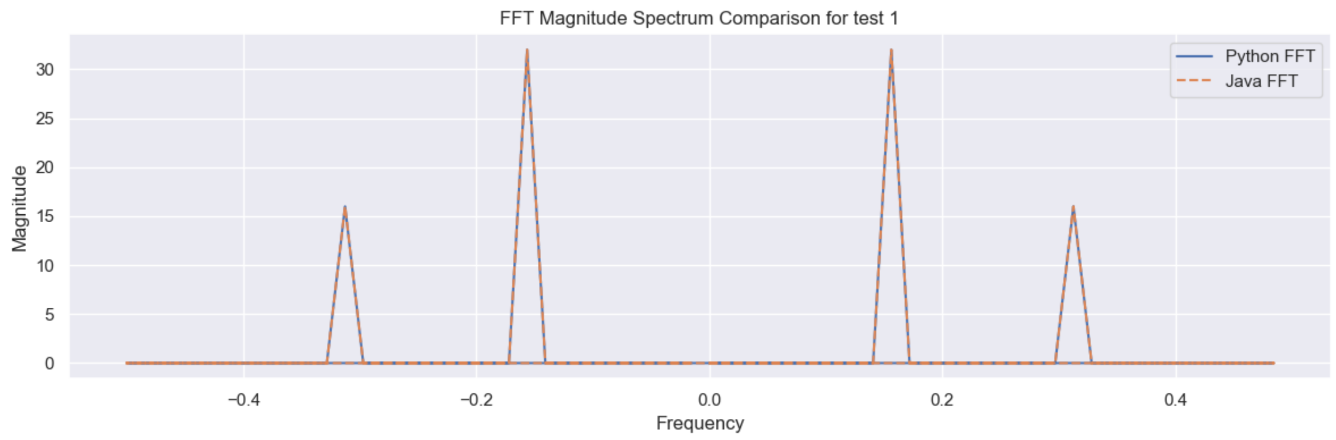
FindPeaks Comparison

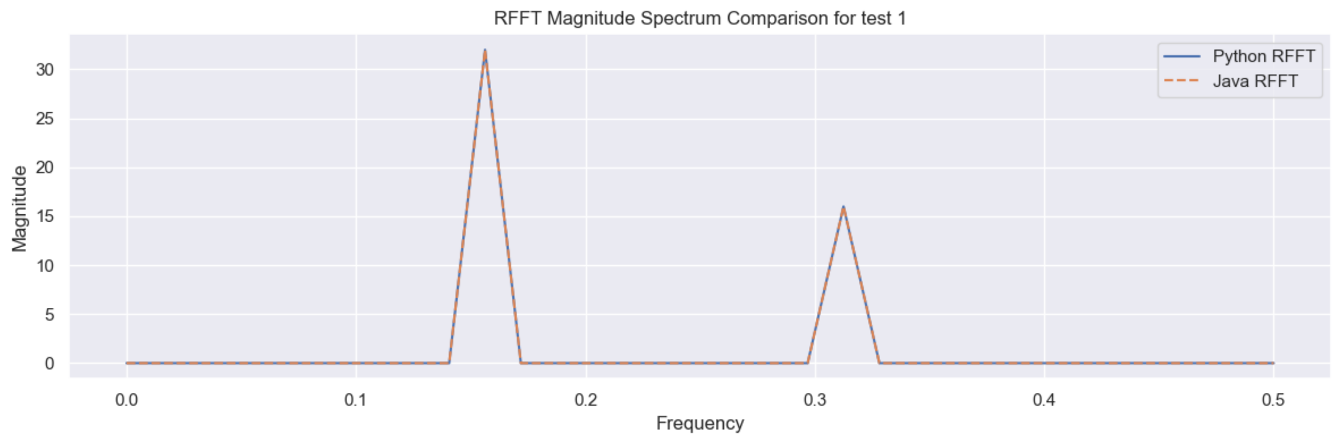


Interpolation Comparison



FFT Comparison





Usage Examples

Butterworth Filter

```
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();
    }
}
```

Find Peaks

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

public class FindPeaksExample {
    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};

        FindPeaks findPeaks = new FindPeaks();
        FindPeaks.PeakParams params = new FindPeaks.PeakParams();
        params.distance = 2; // Minimum distance between peaks
        params.height = 0.5; // Minimum height of peaks
        params.prominence = 0.5; // Minimum prominence of peaks

        FindPeaks.PeakResult result = findPeaks.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]);
        }
    }
}
```

Interpolation

```
import com.hissain.jscipy.signal.interpolate.Interpolation;

public class InterpolationExample {
    public static void main(String[] args) {
        double[] x = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0};
        double[] y = {0.0, 0.8, 0.9, 0.1, -0.8, -1.0};
        double[] newX = {0.5, 1.5, 2.5, 3.5, 4.5};

        Interpolation interpolation = new Interpolation();

        // Perform linear interpolation
        double[] linearY = interpolation.linear(x, y, newX);
        System.out.println("Linear Interpolation:");
        for (int i = 0; i < newX.length; i++) {
            System.out.printf("x = %.1f, y = %.4f\n", newX[i],
linearY[i]);
        }
        System.out.println();

        // Perform cubic interpolation
        double[] cubicY = interpolation.cubic(x, y, newX);
        System.out.println("Cubic Interpolation:");
        for (int i = 0; i < newX.length; i++) {
            System.out.printf("x = %.1f, y = %.4f\n", newX[i], cubicY[i]);
        }
    }
}
```

FFT and RFFT

```
import com.hissain.jscipy.signal.fft.FFT;
import org.apache.commons.math3.complex.Complex;

public class FFTExample {
    public static void main(String[] args) {
        double[] signal = {0.0, 1.0, 0.0, -1.0, 0.0, 1.0, 0.0, -1.0};

        FFT fft = new FFT();

        // Compute the FFT
        Complex[] fftResult = fft.fft(signal);
        System.out.println("FFT Result:");
        for (Complex c : fftResult) {
            System.out.printf("(%.2f, %.2f) ", c.getReal(),
c.getImaginary());
        }
        System.out.println();

        // Compute the RFFT
        Complex[] rfftResult = fft.rfft(signal);
        System.out.println("RFFT Result:");
        for (Complex c : rfftResult) {
            System.out.printf("(%.2f, %.2f) ", c.getReal(),
c.getImaginary());
        }
        System.out.println();

        // Compute the IFFT
        Complex[] ifftResult = fft.ifft(fftResult);
        System.out.println("IFFT Result:");
        for (Complex c : ifftResult) {
            System.out.printf("(%.2f, %.2f) ", c.getReal(),
c.getImaginary());
        }
        System.out.println();

        // Compute the IRFFT
        double[] irfftResult = fft.irfft(rfftResult, signal.length);
        System.out.println("IRFFT Result:");
        for (double d : irfftResult) {
            System.out.printf("%.2f ", d);
        }
        System.out.println();
    }
}
```

Contributing

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

License