**Numerical Methods Package: Ordinary Differential Equations**

Overview

This package provides a set of numerical methods for solving ordinary differential equations (ODEs), including Euler's Method, Runge-Kutta methods, Adams-Bashforth methods, and Taylor's Method. Each method is designed to approximate solutions to ODEs under different conditions and requirements.

Methods

1. **Euler's Method**
   * **Function:** **eulers\_method(f, h, l, r, y0, t0=0)**
   * **Parameters:**
     + **f**: The ODE as a function of **t** and **y**.
     + **h**: The step size.
     + **l**: The left (start) bound of the interval.
     + **r**: The right (end) bound of the interval.
     + **y0**: The initial value of **y**.
     + **t0**: The initial value of **t** (default is 0).
   * **Returns:** A tuple **(ts, ys)** of time values **ts** and corresponding solution values **ys**.
2. **Runge-Kutta 2nd Order (Midpoint and Heun's Method)**
   * **Function:** **runge\_kutta\_2\_midpoint(f, h, l, r, y0, t0=0)** for the Midpoint method and **runge\_kutta\_2\_heun(f, h, l, r, y0, t0=0)** for Heun's method.
   * Similar parameters and returns as Euler's Method.
3. **Runge-Kutta 4th Order**
   * **Function:** **runge\_kutta\_4(f, h, l, r, y0, t0=0)**
   * Similar parameters and returns as Euler's Method.
4. **Adams-Bashforth Methods (2nd and 4th Order)**
   * **Function:** **adams\_bashforth\_2(f, h, l, r, y0)** for 2nd order and **adams\_bashforth\_4(f, h, l, r, y0)** for 4th order.
   * Similar parameters to Euler's Method, except **y0** does not have a **t0** parameter since initial steps are computed using Euler's method for setup.
5. **Taylor's Method**
   * **Function:** **taylors\_method(f, dd, h, l, r, y0, t0=0)**
   * **Parameters:**
     + **f**: The ODE as a function of **t** and **y**.
     + **dd**: A list of the derivatives of **f** with respect to **t**, starting with the first derivative.
     + **h**: The step size.
     + **l**: The left (start) bound of the interval.
     + **r**: The right (end) bound of the interval.
     + **y0**: The initial value of **y**.
     + **t0**: The initial value of **t** (default is 0).
   * **Returns:** A tuple **(ts, ys)** of time values **ts** and corresponding solution values **ys**.

**Note**

* Ensure that the step size **h** is chosen appropriately for the problem at hand, considering the trade-off between computational cost and the accuracy of the solution.
* For more complex ODEs or systems requiring higher accuracy, consider using higher-order methods like Runge-Kutta 4th Order or Adams-Bashforth 4th Order.

**Numerical Methods Package: Differentiation**

Overview

This package provides various numerical methods for calculating derivatives of functions or datasets. It includes forward, backward, and centered finite difference methods, as well as Richardson extrapolation for enhanced accuracy. These methods can handle both analytical functions and discrete data sets.

Installation

No specific installation steps are provided, but ensure that **FiniteDiffCoefficients.py** is accessible in your project environment as it is required for coefficient lookup.

Functions

1. **Finite Difference Methods**
   * **Forward Finite Difference** (**forward\_finite\_difference**)
   * **Centered Finite Difference** (**centered\_finite\_difference**)
   * **Backward Finite Difference** (**backward\_finite\_difference**)

**Common Parameters:**

* + **mode**: Specifies the input type. Use **'f'** for functions and **'d'** for data arrays.
  + **f\_or\_data**: The function to differentiate or the dataset as an array/list.
  + **x**: The point of differentiation for functions, or the index in the dataset.
  + **h**: The step size for differentiation (default **1e-5**).
  + **o**: The order of differentiation (default first order if unspecified).
  + **a**: The accuracy of the finite difference approximation. This parameter selects the specific coefficients for differentiation.

**Returns:** The derivative of the function or data at point **x** or index **x**, respectively.

1. **Richardson Extrapolation** (**richardson\_extrapolation**)

Enhances the accuracy of the derivative calculation using Richardson extrapolation.

**Additional Parameters:**

* + **method**: Specifies which finite difference method to use (**'forward'**, **'centered'**, or **'backward'**).
  + Rest parameters are the same as the finite difference methods.

**Returns:** The extrapolated derivative of the function or data at the specified point or index.

Notes

* For data (**'d'**) mode, ensure indices are within bounds of the dataset.
* Choose **h** carefully to balance between accuracy and numerical stability.
* Higher-order and accuracy settings require corresponding coefficients in **FiniteDiffCoefficients**.

Authors and Acknowledgments

* **Author:** Graeme Wiltrout
* **Advisor:** T. Fogarty

This documentation aims to provide a quick start for users needing to apply differentiation techniques in their projects, whether for analytical functions or discrete datasets.

**Numerical Methods Package: Integration**

Overview

This package offers a suite of algorithms for numerically approximating integrals. It includes both basic geometric approaches and advanced techniques. The geometric methods use simple shapes like rectangles and trapezoids to approximate areas under curves, while advanced techniques utilize mathematical strategies for more precise approximations.

Installation

Ensure all dependencies are properly installed and accessible within your project. This package requires a callable function as input and numerical bounds for the interval of integration.

Geometric Approximation Methods

1. **Left Endpoint Method** (**left\_endpoint**)
   * Approximates the integral by summing rectangles using the left endpoint for the height.
2. **Midpoint Method** (**midpoint**)
   * Uses the midpoint of each interval for the height of rectangles, offering better accuracy than endpoint methods for certain functions.
3. **Right Endpoint Method** (**right\_endpoint**)
   * Similar to the Left Endpoint Method but uses the right endpoint for determining the height of rectangles.
4. **Trapezoid Method** (**trapezoid**)
   * Approximates the integral by summing trapezoids, utilizing both the left and right endpoints for a better approximation.

Advanced Integration Techniques

1. **Simpson's Rule** (**simpson**)
   * Utilizes parabolic arcs to approximate the area under the curve, requiring an even number of intervals. Adjusts the number of intervals to be even if an odd number is provided.
2. **Gaussian Quadrature** (**gaussian\_quadrature**)
   * Employs a weighted sum of function values at specific points (roots) within the interval for a highly accurate approximation. This method uses precomputed roots and weights.
3. **Romberg Integration** (**romberg**)
   * Combines the trapezoidal rule with Richardson extrapolation, progressively increasing accuracy with a given depth. Trades computational effort for accuracy.

Notes

* Choose the step size **h** (for geometric methods) or the number of intervals **n** (for advanced methods) thoughtfully to balance accuracy and computational efficiency.
* For the Gaussian Quadrature method, available degrees (n values) are limited to precomputed roots and weights. Extending this method requires additional computations or resources for more degrees.
* Romberg Integration's **depth** parameter controls the accuracy; higher depths mean more precise results at the cost of increased computation.

Authors and Acknowledgments

* **Author:** Graeme Wiltrout
* **Advisor:** T. Fogarty

This package is designed to provide users with flexible and easy-to-use tools for numerical integration across a variety of applications, from simple to complex integrands.