pyfod

Release 0.1.1

Paul R. Miles, Graham T. Pash

May 23, 2020

CONTENTS

The pyfod package is a Python repository for performing fractional-order derivative operations. Several different definitions of fractional derivative are available within the package:

- Riemann-Liouville
- Caputo (development)
- Grünwald-Letnikov

For now, the package is designed specifically for problems where the fractional order is between 0 and 1. Accommodating a broader range of fractional order values will be a feature added as time permits.

CHAPTER

ONE

INSTALLATION

This code can be found on the Github project page. To install the master branch directly from Github,

```
pip install git+https://github.com/prmiles/pyfod.git
```

You can also clone the repository and run `python setup.py install`.

CHAPTER

TWO

GETTING STARTED

CHAPTER

THREE

FEEDBACK

- Feature request
- Bug report

CHAPTER

FOUR

CONTENTS

4.1 pyfod package

4.1.1 Submodules

4.1.2 pyfod.fod module

This module provides support for three common definitions of fractional derivative in the limiting case of $\alpha \in [0, 1)$. The definitions available include:

- Riemann-Liouville - *riemannliouville()*
- Caputo - *caputo()*
- Grünwald-Letnikov - *grunwaldletnikov()*

For more details regarding this definitions of fractional derivatives please see [Pod98].

Note: In each method you are required to provide a function handle. Depending on the method being used, you may need to define your function using sympy to allow for extended numerical precision.

---

pyfod.fod.**caputo**(*f*, *lower*, *upper*, *dt=0.0001*, *alpha=0.0*, *df=None*, *quadrature='GLegRS'*, ***kwargs*)

Caputo fractional derivative calculator for $\alpha \in [0, 1)$.

The general definition for Caputo fractional derivative is

$$D_C^{\alpha}[f(t)] = \frac{1}{\Gamma(n-\alpha)} \int_0^t \frac{f(s)^{(n)}}{(t-s)^{\alpha+1-n}} ds,$$

where $n = \lceil \alpha \rceil$. In the limiting case where $\alpha \in [0, 1)$ this further simplifies to

$$D_C^{\alpha}[f(t)] = \frac{1}{\Gamma(1-\alpha)} \int_0^t \frac{f(s)^{(1)}}{(t-s)^{\alpha}} ds.$$

To evaluate this we simply need to define a finite-difference scheme for approximating $f(s)^{(1)}$.

Args:

- **f** (def): Function handle.
- **lower** (float): Lower limit - should be zero.
- **upper** (float): Upper limit, i.e., point at which fractional derivative is being evaluated.

Kwargs: name (type) - default

- **dt** (float) - *1e-4*: Time step, $t_{j+1} - t_j$.
- **alpha** (float) - *0*: Order of fractional derivative.
- **df** (def) - *None*: Finite difference function. See tutorials for examples of how to utilize this feature.
- **quadrature** (str) - *'glegrs'*: Quadrature method
- **kwargs**: Quadrature specific settings.

Returns: dict

- *fd*: Fractional derivative.
- *i1*: Value of integral.
- *q1*: Quadrature object.

pyfod.fod.**grunwaldletnikov**(*f*, *lower*, *upper*, *n=100*, *dt=None*, *alpha=0.0*)

Grünwald-Letnikov fractional derivative calculator.

We have implemented the reverse Grünwald-Letnikov definition.

$$D_G^\alpha[f(t)] = \lim_{h\to 0} \frac{1}{h^\alpha} \sum_{0\leq m<\infty} (-1)^m \binom{\alpha}{m} f(t-mh).$$

Note: The package as a whole was built for problems where $\alpha \in [0, 1)$; however, this definition for Grünwald-Letnikov does not necessarily have the same constraints. It has not been tested for values of $\alpha \geq 1$, but in principle it will work.

Args:

- **f** (def): Function handle.
- **lower** (float): Lower limit - should be zero.
- **upper** (float): Upper limit, i.e., point at which fractional derivative is being evaluated.

Kwargs: name (type) - default

- **n** (int) - *100*: Number of terms to be used in approximation.
- **dt** (float) - *1e-4*: Time step. If *dt is None*, then the value of *dt* will be *(upper - lower)/n*.
- **alpha** (float) - *0*: Order of fractional derivative.

Returns: dict

- *fd*: Fractional derivative.

pyfod.fod.**riemannliouville**(*f*, *lower*, *upper*, *dt=0.0001*, *alpha=0.0*, *quadrature='GLegRS'*, ***kwargs*)

Riemann-Liouville fractional derivative calculator for $\alpha \in [0, 1)$.

The general definition for Riemann-Liouville fractional derivative is

$$D_{RL}^\alpha[f(t)] = \frac{1}{\Gamma(n-\alpha)} \frac{d^n}{dt^n} \int_0^t \frac{f(s)}{(t-s)^{\alpha+1-n}} ds,$$

where $n = \lceil \alpha \rceil$. In the limiting case where $\alpha \in [0, 1)$ this further simplifies to

$$D_{RL}^\alpha[f(t)] = \frac{1}{\Gamma(1-\alpha)} \frac{d}{dt} \int_0^t \frac{f(s)}{(t-s)^{\alpha}} ds.$$

By defining

$$F[t] = \int_{t_0}^{t} (t-s)^{-\alpha} f(s) ds,$$

we can numerically approximate this definition of fractional derivative as

$$D^{\alpha}_{RL}[f(t)] = \chi \frac{d}{dt} F[t] \approx \chi \frac{F(t_{j+1}) - F(t_j)}{t_{j+1} - t_j},$$

where $\chi = \Gamma(1-\alpha)^{-1}$. For more details regarding this approach please see [AGomezA17] and [MPOS18].

Args:

- **f** (def): Function handle.
- **lower** (`float`): Lower limit - should be zero.
- **upper** (`float`): Upper limit, i.e., point at which fractional derivative is being evaluated.

Kwargs: name (type) - default

- **dt** (`float`) - *1e-4*: Time step, $t_{j+1} - t_j$.
- **alpha** (`float`) - *0*: Order of fractional derivative.
- **quadrature** (`str`) - *'glegrs'*: Quadrature method
- **kwargs**: Quadrature specific settings.

Returns: `dict`

- *fd*: Fractional derivative
- *i1*: Value of integral $F(t_{j+1})$.
- *i2*: Value of integral $F(t_j)$.
- *q1*: Quadrature object for $F(t_{j+1})$.
- *q2*: Quadrature object for $F(t_j)$.

4.1.3 pyfod.quadrature module

This module contains a variety of classes for performing quadrature. While this module was developed to support the fractional derivative calculators in *fod*, the quadrature methods themselves are still applicable to a wide variety of integration problems.

The quadrature methods are defined to approximate integrals of the form

$$I = \int_{t_0}^{t} \frac{f(t)}{(b-s)^{\alpha}} ds,$$

where b is typically equal to t. The user is only required to provide the function $f(t)$ and the limits of integration. We observe that by setting $\alpha = 0$, this integral is transformed to

$$I = \int_{t_0}^{t} f(t) ds,$$

which is applicable to many problems.

Classes:

- *GaussLegendre*

- *GaussLaguerre*
- *RiemannSum*
- *GaussLegendreRiemannSum*
- *GaussLegendreGaussLaguerre*

class pyfod.quadrature.**GaussLaguerre**(*deg=5*, *lower=0.0*, *upper=1.0*, *alpha=0.0*, *f=None*, *extend_precision=True*, *n_digits=30*, *singularity=None*)

Bases: object

Gauss-Laguerre quadrature.

Kwargs: name (type) - default

- **deg** (int) - *5*: Degree of laguerre polynomials.
- **lower** (float) - *0.0*: Lower limit of integration.
- **upper** (float) - *1.0*: Upper limit of integration.
- **alpha** (float) - *0.0*: Exponent of singular kernel.
- **f** (def) - *None*: Function handle.
- **extend_precision** (bool) - *True*: Flag to use sympy extended precision.
- **n_digits** (int) - *30*: Number of digits in extended precision.
- **singularity** (float) - *None*: Location of singularity.

integrate(*f=None*)

Evaluate the integral.

The user can assign a function during initial creation of the quadrature object, or they can send it here.

Kwargs: name (type) - default

- **f** (def) - *None*: Function handle.

Note: The function, **f**, should output an array, with shape (n,) or (n, 1).

---

update_weights(*alpha=None*)

Update quadrature weights.

The quadrature weights are a function of α. To facilitate usage of the quadrature object, you can update the weights with a new α without creating a whole new object.

Args:

- **alpha** (float): Exponent of singular kernel.

class pyfod.quadrature.**GaussLegendre**(*ndom=5*, *deg=5*, *lower=0.0*, *upper=1.0*, *alpha=0.0*, *f=None*, *singularity=None*)

Bases: object

Gauss-Legendre quadrature.

Kwargs: name (type) - default

- **ndom** (int) - *5*: Number of quadrature intervals.
- **deg** (int) - *5*: Degree of legendre polynomials.
- **lower** (float) - *0.0*: Lower limit of integration.

- **upper** (float) - *1.0*: Upper limit of integration.
- **alpha** (float) - *0.0*: Exponent of singular kernel.
- **f** (def) - *None*: Function handle.
- **singularity** (float) - *None*: Location of singularity.

integrate(*f=None*)
Evaluate the integral.

The user can assign a function during initial creation of the quadrature object, or they can send it here.

Kwargs: name (type) - default

- **f** (def) - *None*: Function handle.

Note: The function, **f**, should output an array, with shape (n,) or (n, 1).

---

update_weights(*alpha=None*)
Update quadrature weights.

The quadrature weights are a function of α. To facilitate usage of the quadrature object, you can update the weights with a new α without creating a whole new object.

Args:

- **alpha** (float): Exponent of singular kernel.

class pyfod.quadrature.**GaussLegendreGaussLaguerre**(*ndom=5*, *gleg_deg=4*, *glag_deg=20*, *percent=0.9*, *ts=None*, *lower=0.0*, *upper=1.0*, *alpha=0.0*, *f=None*, *extend_precision=True*, *n_digits=30*)

Bases: object

Gauss-Legendre, Gauss-Laguerre quadrature.

Kwargs: name (type) - default

- **ndom** (int) - *5*: Number of Gauss-Legendre quadrature intervals.
- **gleg_deg** (int) - *4*: Degree of legendre polynomials.
- **glag_deg** (int) - *20*: Degree of laguerre polynomials.
- **percent** (float) - *0.9*: Percentage of interval in which to use Gauss-Legendre method.
- **ts** (float) - *None*: User-defined time to switch from Gauss-Legendre to Riemann-Sum quadrature.
- **lower** (float) - *0.0*: Lower limit of integration.
- **upper** (float) - *1.0*: Upper limit of integration.
- **alpha** (float) - *0.0*: Exponent of singular kernel.
- **f** (def) - *None*: Function handle.
- **extend_precision** (bool) - *True*: Flag to use sympy extended precision.
- **n_digits** (int) - *30*: Number of digits in extended precision.

integrate(*f=None*)
Evaluate the integral.

The user can assign a function during initial creation of the quadrature object, or they can send it here.

Kwargs: name (type) - default

- **f** (def) - *None*: Function handle.

Note: The function, **f**, should output an array, with shape (n,) or (n, 1).

---

update_weights(*alpha=None*)
Update quadrature weights.

The quadrature weights are a function of α. To facilitate usage of the quadrature object, you can update the weights with a new α without creating a whole new object.

Args:

- **alpha** (float): Exponent of singular kernel.

class pyfod.quadrature.**GaussLegendreRiemannSum**(*ndom=5*, *deg=4*, *nrs=20*, *percent=0.9*, *ts=None*, *lower=0.0*, *upper=1.0*, *alpha=0.0*, *f=None*)

Bases: object

Gauss-Legendre, Riemann-Sum quadrature.

Kwargs: name (type) - default

- **ndom** (int) - *5*: Number of Gauss-Legendre quadrature intervals.
- **deg** (int) - *4*: Degree of legendre polynomials.
- **nrs** (int) - *20*: Number of Riemann-Sum quadrature intervals.
- **percent** (float) - *0.9*: Percentage of interval in which to use Gauss-Legendre method.
- **ts** (float) - *None*: User-defined time to switch from Gauss-Legendre to Riemann-Sum quadrature.
- **lower** (float) - *0.0*: Lower limit of integration.
- **upper** (float) - *1.0*: Upper limit of integration.
- **alpha** (float) - *0.0*: Exponent of singular kernel.
- **f** (def) - *None*: Function handle.

integrate(*f=None*)
Evaluate the integral.

The user can assign a function during initial creation of the quadrature object, or they can send it here.

Kwargs: name (type) - default

- **f** (def) - *None*: Function handle.

Note: The function, **f**, should output an array, with shape (n,) or (n, 1).

---

update_weights(*alpha=None*)
Update quadrature weights.

The quadrature weights are a function of α. To facilitate usage of the quadrature object, you can update the weights with a new α without creating a whole new object.

Args:

- **alpha** (float): Exponent of singular kernel.

class pyfod.quadrature.**RiemannSum**(*n=5*, *lower=0.0*, *upper=1.0*, *alpha=0.0*, *f=None*, *singularity=None*)

Bases: object

Riemann-Sum quadrature.

Kwargs: name (type) - default

- **n** (int) - *5*: Number of quadrature intervals.
- **lower** (float) - *0.0*: Lower limit of integration.
- **upper** (float) - *1.0*: Upper limit of integration.
- **alpha** (float) - *0.0*: Exponent of singular kernel.
- **f** (def) - *None*: Function handle.
- **singularity** (float) - *None*: Location of singularity.

integrate(*f=None*)

Evaluate the integral.

The user can assign a function during initial creation of the quadrature object, or they can send it here.

Kwargs: name (type) - default

- **f** (def) - *None*: Function handle.

Note: The function, **f**, should output an array, with shape (n,) or (n, 1).

---

update_weights(*alpha=None*)

Update quadrature weights.

The quadrature weights are a function of α. To facilitate usage of the quadrature object, you can update the weights with a new α without creating a whole new object.

Args:

- **alpha** (float): Exponent of singular kernel.

4.1.4 pyfod.utilities module

pyfod.utilities.**check_alpha**(*alpha*)

Check value of fractional order.

The package is designed to work on problems where the fractional order has a value in in the range [0, 1).

Args:

- **alpha** (float): Order of fractional derivative.

Raises:

- System exit for *ZeroDivisionError*.

pyfod.utilities.**check_input**(*value*, *varname=None*)

Check value is defined.

This routine checks that value is defined.

Args:

- **value** (user defined): Value being tested

Kwargs: name (type) - default

- **varname** (str) - *None*: Name of variable/value being tested. This provides a more descriptive output for debugging.

If not properly defined

Raises:

- System exist for no value defined.

pyfod.utilities.**check_node_type**(*n*)
Check that number of nodes is an integer.

Args:

- **n** (user defined): Number of nodes.

Returns:

- *int(n)*

pyfod.utilities.**check_range**(*lower*, *upper*, *value*)
Check that value falls within [lower, upper] range.

Args:

- **lower** (float): Lower limit
- **upper** (float): Upper limit
- **value** (float): Value to check with respect to range

If **value** in range,

Returns:

- **value** (float)

else

Raises:

- System exit for value out of domain.

pyfod.utilities.**check_singularity**(*singularity*, *upper*)
Check singularity was defined.

This routine checks if user provided a singularity location. Default behavior is to use the upper limit of integration as the singularity location.

Args:

- **singularity** (float): User defined location
- **upper** (float): Upper limit

Returns:

- **upper** - if *singularity is None*
- **singularity** - if *singularity not None*

pyfod.utilities.**check_value**(*value*, *default_value*, *varname=None*)
Check value with respect to default.

This routine checks that value is defined by either a user-defined value or default value. If still *None*, then something went wrong and this raises an error message.

Args:

- **value** (user defined): Value being tested
- **default_value** (method defined): Default value

Kwargs: name (type) - default

- **varname** (`str`) - *None*: Name of variable/value being tested. This provides a more descriptive output for debugging.

If properly defined

Returns:

- **value**

else

Raises:

- System exist for no value defined.

4.1.5 Module contents

CHAPTER

FIVE

REFERENCES

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[AGomezA17] Abdon Atangana and JF Gómez-Aguilar. Numerical approximation of riemann-liouville definition of fractional derivative: from riemann-liouville to atangana-baleanu. *Numerical Methods for Partial Differential Equations*, 2017.

[MPOS18] Paul R. Miles, Graham T. Pash, William S. Oates, and Ralph C. Smith. Numerical techniques to model fractional-order nonlinear viscoelasticity in soft elastomers. In *ASME 2018 Conference on Smart Materials, Adaptive Structures and Intelligent Systems*, V001T03A021. American Society of Mechanical Engineers, 2018.

[Pod98] Igor Podlubny. *Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications*. Volume 198. Elsevier, 1998.

PYTHON MODULE INDEX

p

INDEX