

# Package ‘Convolutioner’

January 20, 2025

**Type** Package

**Title** Convolution of Data

**Version** 0.1.0

**Maintainer** Federico Maria Vivaldi <federico-vivaldi@virgilio.it>

**Description** General functions for convolutions of data. Moving average, running median, and other filters are available.

Bibliography regarding the functions can be found in the following text.  
Richard G. Brereton (2003) <ISBN:9780471489771>.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Federico Maria Vivaldi [aut, cre]

**Repository** CRAN

**Date/Publication** 2021-03-11 10:40:02 UTC

## Contents

Hamming . . . . .	2
Hann . . . . .	2
MA . . . . .	3
RMS . . . . .	4
sine . . . . .	4
test_data . . . . .	5

<b>Index</b>	<b>6</b>
--------------	----------

---

Hamming

*Hamming window filter.*

---

### Description

This function return the data smoothed using the an Hamming window filter. Data are smoothed using a cosine window with particular coefficients.

### Usage

```
Hamming(raw_data, buffer_size = 5)
```

### Arguments

<code>raw_data</code>	Data upon which the algorithm is applied
<code>buffer_size</code>	number of points the algorithm use to compute the coefficients of the Hann window

### Value

Smoothed data using Hann Window filter

### Examples

```
raw_data = c(1:100)
smoothed_data = Hamming(raw_data)
```

---

Hann

*Hann window filter.*

---

### Description

This function return the data smoothed using the an Hann window filter. Data are smoothed using a cosine window.

### Usage

```
Hann(raw_data, buffer_size = 5)
```

### Arguments

<code>raw_data</code>	Data upon which the algorithm is applied
<code>buffer_size</code>	number of points the algorithm use to compute the coefficients of the Hann window

**Value**

Smoothed data using Hann Window filter

**Examples**

```
raw_data = c(1:100)
smoothed_data = Hann(raw_data)
```

---

MA

*Moving average filter.*

---

**Description**

This function return the data smoothed using the basic moving average algorithm. For each chunk of data of size equal to the `buffer_size` parameter is calculated the average and this value is used as the `i` term of the newly smoothed data. zero padding is applied for initial and final values

**Usage**

```
MA(raw_data, buffer_size = 5)
```

**Arguments**

<code>raw_data</code>	Data upon which the algorithm is applied
<code>buffer_size</code>	number of points the algorithm use to compute the average

**Value**

Smoothed data using moving average algorithm

**Examples**

```
raw_data = c(1:100)
smoothed_data = MA(raw_data)
```

RMS

*Running median smoothing.*

---

**Description**

This function return the data smoothed using the running median algorithm. For each chunk of data of size equal to the `buffer_size` parameter is calculated the median and this value is used as the `i` term of the newly smoothed data. For initial and final values zero padding is applied.

**Usage**

```
RMS(raw_data, buffer_size = 5)
```

**Arguments**

<code>raw_data</code>	Data upon which the algorithm is applied
<code>buffer_size</code>	number of points the algorithm use to compute the median

**Value**

Smoothed data using running median algorithm

**Examples**

```
raw_data = c(1:100)
smoothed_data = RMS(raw_data)
```

---

sine*Sine window filter.*

---

**Description**

This function return the data smoothed using the a sine window filter.

**Usage**

```
sine(raw_data, buffer_size = 5)
```

**Arguments**

<code>raw_data</code>	Data upon which the algorithm is applied
<code>buffer_size</code>	number of points the algorithm use to compute the coefficients of the Hann window

**Value**

Smoothed data using Hann Window filter

**Examples**

```
raw_data = c(1:100)
smoothed_data = sine(raw_data)
```

---

test_data	<i>Test data generator</i>
-----------	----------------------------

---

**Description**

Generate test data in order to test the filtering functions. To a signal function is added random noise contribution. V0.1 = noise is assumed gaussian

**Usage**

```
test_data(
  amplitude = 1,
  f = 100,
  npoints = 1000,
  type = "sinusoidal",
  x0 = 0,
  noise_contribution = 100
)
```

**Arguments**

amplitude	amplitude of the signal, default = 1
f	frequency of the sinusoidal signal, default = 100
npoints	number of points of the time serie
type	type of signal, default = sinusoidal. Available types: sinusoidal, gaussian
x0	signal position for gaussian type. Default = 0
noise_contribution	percentage pointing the maximum wanted signal/noise ratio. Default = 10

**Value**

A time serie with added random noise.

**Examples**

```
test_data()
```

# Index

Hamming, [2](#)

Hann, [2](#)

MA, [3](#)

RMS, [4](#)

sine, [4](#)

test\_data, [5](#)