Algorithms for Unevenly Spaced Time Series: Moving Averages and Other Rolling Operators

Andreas Eckner*

First version: January 2010
Major revision: April 2017
Current version: September 13, 2018

Abstract

This paper describes algorithms for efficiently calculating certain rolling time series operators for unevenly spaced data. In particular, we show how to calculate simple moving averages (SMAs), exponential moving averages (EMAs), and related operators in linear time with respect to the number of observations in a time series. An implementation of the algorithms is provided in the programming language `C` and as an R package.[1]

1 Introduction

There exists an extensive body of literature on analyzing *equally*-spaced time series data, see Tong (1990), Brockwell and Davis (1991), Hamilton (1994), Fan and Yao (2003), Lütkepohl (2010), Box et al. (2015), and Brockwell and Davis (2016). As a consequence, efficient algorithms have been developed for many computational questions, and implementations are available for software environments such as Fortran, Matlab, and R. Many of the underlying algorithms were developed at a time when limitations in computing resources favored an analysis of equally spaced data (and the use of linear Gaussian models), because in this case efficient linear algebra routines can be used.

As a result, fewer methods exists specifically for analyzing and processing *unevenly*-spaced (also called unequally- or irregularly-spaced) time series data, even though such data naturally occurs in many industrial and scientific domains, such as astronomy, biology, (paleo)climatology, economics, finance, geology, and network traffic analysis.

For such data, Müller (1991) and Dacorogna et al. (2001) recommend, due to the computational simplicity, to use exponential moving averages (EMAs) as the main building block of time series operators. In particular, Müller (1991) argues that the sequential computation of EMAs "is more efficient than the computation of any differently weighted MA". This paper shows that many other time series operators can in fact be calculated just as efficiently.

The main focus of this paper are rolling time series operators, such as moving averages, that allow to extract a certain piece of local information about a time series (typically within

*Comments are welcome at andreas@eckner.com

[1] See github.com/andreas50/utsAlgorithms and github.com/andreas50/utsOperators, respectively.

a rolling time window of a fixed length $\tau > 0$). The generic structure for most such operators is as follows:

```
for each time window of length tau with end point equal to an observation time {
  do some calculation on
    the observation values and times in the current rolling window
}
c
```

Generic Rolling Time Series Operator

If the calculation within each window runs in linear time with respect to the number of observations in the said window, then the total time for applying such a rolling time series operator is proportional to (i) the length of the time series, times (ii) the average number of observations in each time window (or equivalently, the average observation density times the window length τ). However, in many cases it is possible to devise a much more efficient algorithm with execution time proportional only to the length of the time series and independent of the window length τ. This paper presents such algorithms for SMAs, EMAs, and various other rolling time series operators.

1.1 Framework

This section provides a brief introduction to unevenly spaced time series and the basic structure of rolling time series operators for such objects. For a more detailed exposition see Eckner (2017).

We use the notation $((t_n, X_n) : 1 \le n \le N(X))$ or $(X_{t_n} : 1 \le n \le N(X))$ to denote an unevenly spaced time series X with strictly-increasing observation times $T(X) = (t_1, \ldots, t_{N(X)})$ and observation values $V(X) = (X_1, \ldots, X_{N(X)})$, where $N(X)$ denotes the length of the time series.

For a time point $t \in \mathbb{R}$, $X[t]_{\text{last}}$ denotes the most recent (i.e. last) observation value of X at or before time t, $X[t]_{\text{next}}$ denotes the next available observation value of X at or after time t, while $X[t]_{\text{linear}}$ denotes the linearly interpolated value of X at time t. We call these quantities the sampled value of X at time t with last-point, next-point, and linear interpolation, respectively. Sampled values before the first observation time, t_1, are taken to be equal to the first observation value, X_{t_1}. While potentially not appropriate for some applications, this convention avoids the treatment of a multitude of special cases in the exposition below.[2]

The algorithms in this paper have as input (i) an array `values` containing the observation values, (ii) an array `times` containing the observation times, and (iii) a parameter τ describing the temporal horizon of a time series operator, such as the length of a moving average window. The output is an array `out` of same length as the input arrays. Indices of arrays start at one. For integers $n \le m$, `n:m` denotes the array of numbers $[n, n+1, \ldots, m-1, m]$. For brevity of the presentation, although not in the accompanying implementation, we ignore variable declarations, memory allocation, and special cases for time series of length zero or one.

Many of the algorithms below use a half-open rolling time window of the form $(t - \tau, t]$ to keep track of observations relevant to the calculation of a time series operator at a time $t \in T(X)$. Specifically, the variable `right` denotes the index corresponding to the right edge of

[2] A software implementation might instead use a special symbol to denote a value that is not available. For example, `R` uses the constant `NA`, which propagates through all steps of an analysis, because the result of a calculation involving `NAs` is also `NA`.

the rolling time window, while the variable `left` denotes the index of the left-most observation inside the rolling time window. In particular,

$$t - \tau < \texttt{times[left]} \leq \texttt{times[right]} = t.$$

With this notation, the generic algorithm for rolling time series operators becomes:

```
left = 1;
for (right in 1:N(X)) {
  // Shrink window on the left end
  while (times[left] <= times[right] - tau))
    left = left + 1;

  // Calculate output for current rolling window
  out[right] = do_some_calculation(values[left:right], times[left:right]);
}
c
```

Generic Algorithm for Rolling Time Series Operators

Here, `values[left:right]` and `times[left:right]` denote the array of observation values and times, respectively, in the current time window.

In many cases of interest, the function `do_some_calculation` will run in linear time with respect to the number of observations inside the current time window. The algorithms in this paper improve upon the efficiency of this generic algorithm by reusing results from the previous time window to determine the output value of the current time window.

Remark 1. *In some cases, it is desirable to store every single observation value within the rolling time window in a self-balancing binary search tree, such as an AVL tree, see Knuth (1998). Such a data structure allows searches, insertions, and deletions to be carried out in logarithmic time. However, the additional complexity involved in the implementation pays off only for very large datasets. For time series of moderate length, a simpler algorithm is often just as efficient.*

2 Rolling Summary Statistics

The simplest possible rolling operators calculate a univariate summary statistic of the observation values inside a rolling time window, without regard to the spacing of observations.

2.1 Rolling Sum, Average, Product, and Number of Observations

The rolling average can be used for summarizing the average value of a time series over a certain time horizon, for example, for the purpose of smoothing noisy observations. It can be calculated efficiently by keeping track of (i) the number and (ii) the sum of observation values in a time window of length τ that moves forward in time. Both values are updated whenever a new observation enters or leaves the time window. The pseudocode of the algorithm is as follows:

```
left = 1; roll_sum = 0;
for (right in 1:N(X)) {
  // Expand window on right end
  roll_sum = roll_sum + values[right];

  // Shrink window on left end
  while (times[left] <= times[right] - tau) {
    roll_sum = roll_sum - values[left];
    left = left + 1;
  }

  // Save average value for current time window
  out[right] = roll_sum / (right - left + 1);
}
c
```

Rolling Average

This algorithm can be trivially modified to calculate the total number and sum of observation values in a rolling time window of length $\tau > 0$.

```
left = 1;
for (right in 1:N(X)) {
  while (times[left] <= times[right] - tau)
    left = left + 1;
  out[right] = right - left + 1;
}
c
```

Rolling Number of Observations

```
left = 1; roll_sum = 0;
for (right in 1:N(X)) {
  roll_sum = roll_sum + values[right];
  while (times[left] <= times[right] - tau) {
    roll_sum = roll_sum - values[left];
    left = left + 1;
  }
  out[right] = roll_sum;
}
c
```

Rolling Sum

Note that for a fixed time window width $\tau > 0$, by construction, the rolling average of a time series equals the rolling sum divided by the rolling number of observations of the same time series.

Finally, the rolling product can be calculated in a similar manner, but care must be taken to avoid divisions by zero when a zero observation value leaves the rolling time window.

2.2 Rolling Maximum and Minimum

Assume given a function `max_index(array, start, end)` that returns the index of the largest element in an array between and including the indices `start` and `end`. The rolling maximum in a time window of length $\tau > 0$ can be calculated as follows:

```
left = 1; max_pos = 1;
for (right in 1:N(X)) {
  // Expand interval on right end
  if (values[right] >= values[max_pos])
    max_pos = right;

  // Shrink interval on left end
  while (times[left] <= times[right] - tau)
    left++;

  // Recalculate position of maximum if old maximum dropped out
  if (max_pos < left)
    max_pos = max_index(values, left, right);

  // Save maximum for current time window
  out[right] = values[max_pos];
}
c
```

Rolling Maximum

The rolling minimum in a time window of length $\tau > 0$ can be calculated in a similar manner.

Remark 2. *An alternative algorithm could use a self-balancing binary search tree to keep track of the observation values inside the rolling time window, see Remark 1. The minimum and maximum element of such a data structure can be extracted in logarithmic time, which gives a better worst-case performance than the algorithm above.*[3]

3 Simple Moving Averages

Like the rolling average, simple moving averages (SMAs) can be used for summarizing the average value of a time series over a certain time horizon.

Definition 3.1. *For an unevenly spaced time series X, we define three versions of the simple moving average (SMA) of length $\tau > 0$. For $t \in T(X)$,*

(i) $\mathrm{SMA}_{\mathrm{last}}(X, \tau)_t = \frac{1}{\tau} \int_0^\tau X[t-s]_{\mathrm{last}} \, \mathrm{d}s$,

(ii) $\mathrm{SMA}_{\mathrm{next}}(X, \tau)_t = \frac{1}{\tau} \int_0^\tau X[t-s]_{\mathrm{next}} \, \mathrm{d}s$,

(iii) $\mathrm{SMA}_{\mathrm{linear}}(X, \tau)_t = \frac{1}{\tau} \int_0^\tau X[t-s]_{\mathrm{linear}} \, \mathrm{d}s$,

where in all cases the observation times of the input and output time series are identical.

Notice that the rolling average from Section 2.1 gives the same weight to each observation inside the rolling time window. The SMAs, on the other hand, implicitly assign different weights to different observations, depending on the spacing of observation times. However, for equally spaced time series data with rolling window length τ an integer multiple of the observation time spacing, the simple moving average $\mathrm{SMA}_{\mathrm{next}}$ is identical to the rolling average, see Eckner (2017).

[3] The worst case is attained for a strictly decreasing time series, because in this case the position of maximum value, `max_pos`, has to be calculated from scratch each time an observation drops out of the time window.

When to use which operator?

A rolling average is ideal for analyzing discrete *events*; for example, to calculate the average number of casualties per deadly car accident over the past twelve months, or to determine the average number of IBM common shares traded on the NYSE per executed order during the past 30 minutes.

On the other hand, the SMA_{last} can be used to analyze discrete observation *values*; for example, to calculate the average FED funds target rate[4] over the past three years. In such a case, it is desirable to weight each observation value by the amount of time it remained unchanged.

The SMA_{next} has a similar purpose for time-reversed processes and, as mentioned above, under certain circumstances coincides with the rolling average for equally spaced time series.

Finally, the $\text{SMA}_{\text{linear}}$ can be used to estimate the rolling average value of a discretely-observed continuous-time stochastic processes with observation times that are independent of the observation values, see Eckner (2017).

3.1 SMA_{last} and SMA_{next}

The simple moving average SMA_{last} can be efficiently calculated in a manner similar to the rolling average. However, now the spacing of observations leaving and entering the rolling time window needs to be taken into account. Figure 1 visualizes two key steps in the algorithm, which is as follows:

```
left = 1; roll_area = left_area = values[1] * tau; out[1] = values[1];
for (right in 2:N(X)) {
  // Expand interval on right end
  roll_area = roll_area + values[right-1] * (times[right] - times[right-1]);

  // Remove truncated area on left end
  roll_area = roll_area - left_area;

  // Shrink interval on left end
  t_left_new = times[right] - tau;
  while (times[left] <= t_left_new) {
    roll_area = roll_area - values[left] * (times[left+1] - times[left]);
    left = left + 1;
  }

  // Add truncated area on left end
  left_area = values[max(1, left-1)] * (times[left] - t_left_new)
  roll_area = roll_area + left_area;

  // Save SMA value for current time window
  out[right] = roll_area / tau;
}
c
```

SMA_{last}

Note that the value of `left_area`, calculated towards the end of the loop, is reused in the next iteration close to the top of the loop.

[4]The FED funds target rate is the desired interest rate (by the Federal Reserve) at which depository institutions (such as a savings bank) lend balances held at the Federal Reserve to other depository institutions overnight. See www.federalreserve.gov/fomc/fundsrate.htm for details.

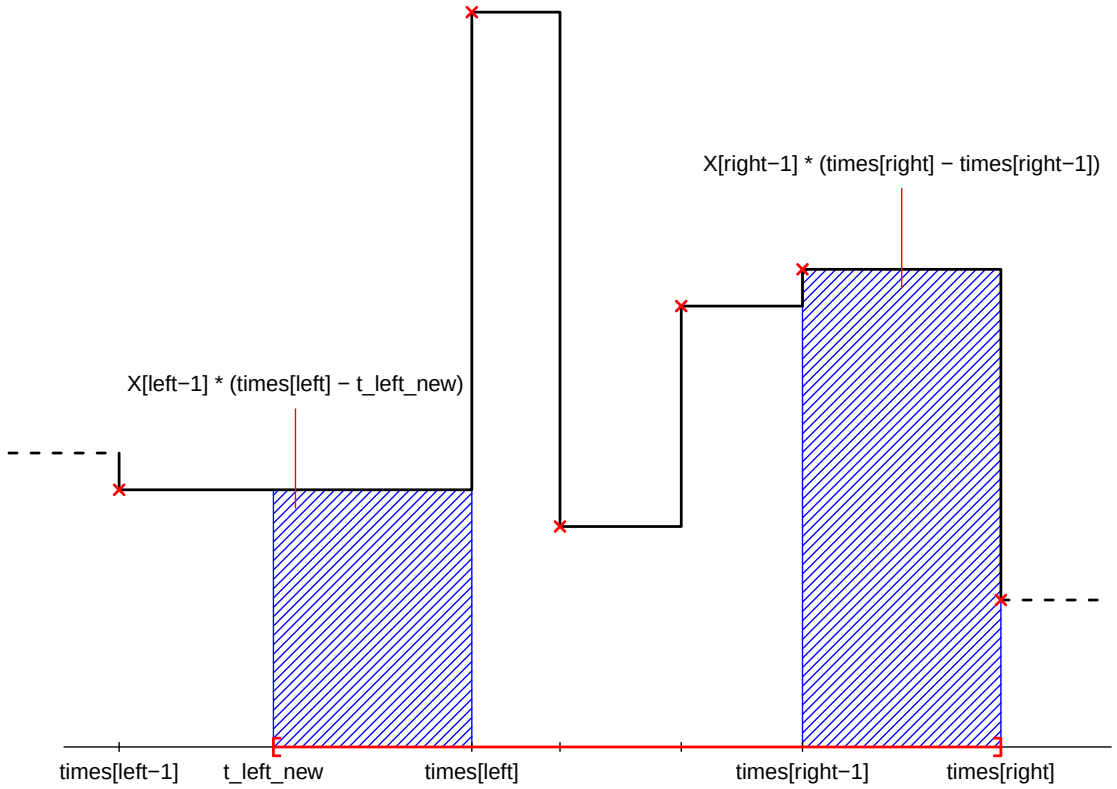

Figure 1: **Areas involved in the calculation of the simple moving average** $\mathrm{SMA}_{\mathrm{last}}(X, \tau)$**.** The x's mark the observations of the time series, and the step function is the corresponding sample path with last-point sampling scheme. The shaded area on the right-hand side has to be added to the variable `roll_area` in order to expand the rolling time window to the right. Depending on the position in the loop of the algorithm, the shaded area on the left-hand side has to be either added or subtracted from the variable `roll_area`.

The algorithm for calculating the simple moving average $\mathrm{SMA}_{\mathrm{next}}$ is almost identical and therefore not shown here, but provided in the implementation.

3.2 $\mathrm{SMA}_{\mathrm{linear}}$

The simple moving average $\mathrm{SMA}_{\mathrm{linear}}$ can be calculated like the $\mathrm{SMA}_{\mathrm{last}}$, except that areas entering and leaving the rolling time window are now trapezoids instead of rectangles, see Figure 2. To this end, we define a helper function that calculates the area of the trapezoid with coordinates of the corners $(x_2, 0), (x_2, y_2), (x_3, 0)$, and (x_3, y_3), where y_2 is obtained by linear interpolation of (x_1, y_1) and (x_3, y_3) evaluated at x_2.

```
trapezoid = function(x1, x2, x3, y1, y3) {
  // Degenerate cases
  if (x2 == x3 or x2 < x1)
    return (x3 - x2) * y1;

  w = (x3 - x2) / (x3 - x1);
  y2 = y1 * w + y3 * (1 - w);
  return (x3 - x2) * (y2 + y3) / 2;
}
c
```

$\text{SMA}_{\text{linear}}$ helper function - Trapezoid Area

The `if` statement treats the two special cases of (i) a trapezoid with zero area, and (ii) the first observation has not yet left the rolling time window. The pseudo-code for the main part of the algorithm is as follows:

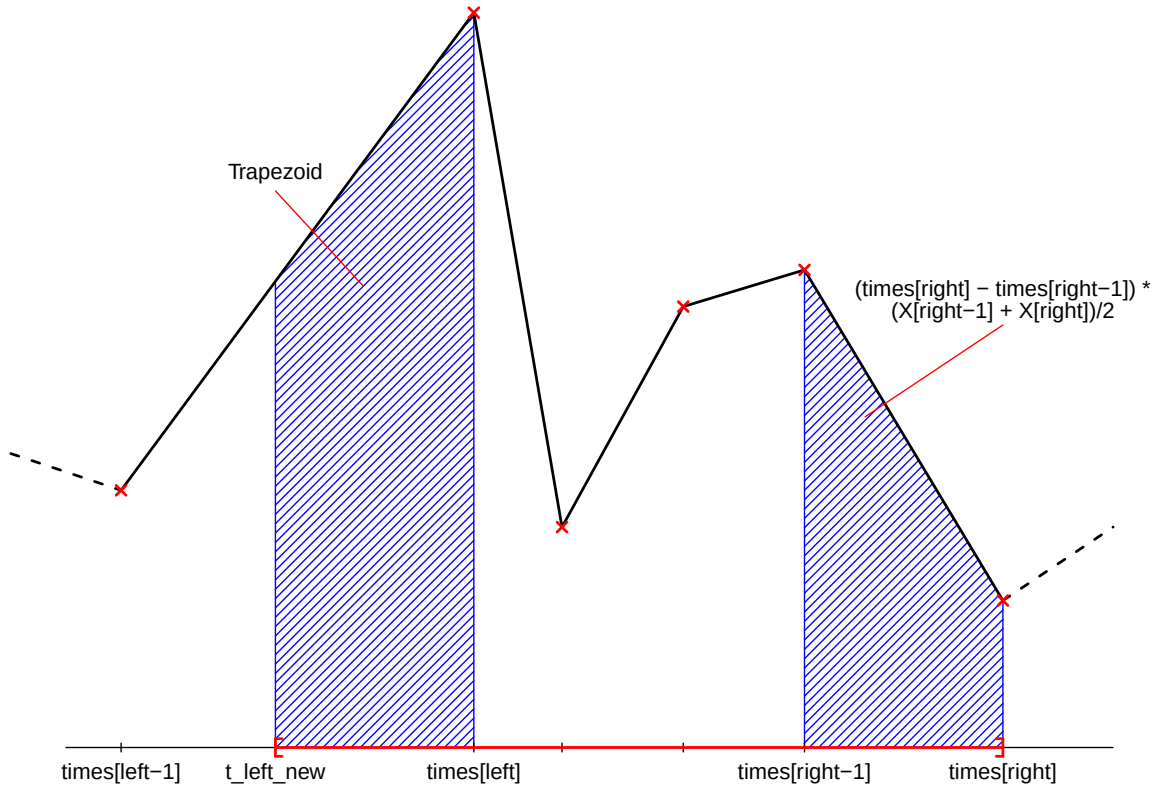

Figure 2: **Areas involved in the calculation of the simple moving average** $\text{SMA}_{\text{linear}}(X, \tau)$**.** The x's mark the observations of the time series, and the piecewise-linear function is the corresponding sample path with linear interpolation. The shaded area on the right-hand side has to be added to the variable `roll_area` in order to expand the rolling time window to the right. Depending on the position in the loop of the algorithm, the shaded area on the left-hand side has to be either added or subtracted from the variable `roll_area`.

```
left = 1; roll_area = left_area = values[1] * tau; out[1] = values[1];
for (right in 2:N(X)) {
  // Expand interval on right end
  roll_area = roll_area + (values[right-1] + values[right])/2 *
                          (times[right] - times[right-1]);

  // Remove truncated area on left end
  roll_area = roll_area - left_area;

  // Shrink interval on left end
  t_left_new = times[right] - tau;
  while (times[left] <= t_left_new) {
    roll_area = roll_area - (values[left] + values[left+1]) / 2 *
                            (times[left+1] - times[left]);
    left = left + 1;
  }

  // Add truncated area on left end
  left_area = trapezoid(times[max(1, left-1)], t_left_new, times[left],
                        values[max(1, left-1)], values[left]);
  roll_area = roll_area + left_area;

  // Save SMA value for current time window
  out[right] = roll_area / tau;
}
c
```

$\text{SMA}_{\text{linear}}$

As before, the value of `left_area`, calculated towards the end of the loop, is reused in the next iteration close to the top of the loop.

4 Exponential Moving Averages

Like SMAs, exponential moving averages (EMAs) can be used for summarizing the average value of a time series over a certain time horizon, except that more weight is given to more recent observations.

Definition 4.1. *For an unevenly spaced time series X, we define three versions of the exponential moving average (EMA) of length $\tau > 0$. For $t \in T(X)$,*

(i) $\text{EMA}_{\text{last}}(X, \tau)_t = \frac{1}{\tau} \int_0^\infty X[t-s]_{\text{last}} e^{-s/\tau} \, \mathrm{d}s,$

(ii) $\text{EMA}_{\text{next}}(X, \tau)_t = \frac{1}{\tau} \int_0^\infty X[t-s]_{\text{next}} e^{-s/\tau} \, \mathrm{d}s,$

(iii) $\text{EMA}_{\text{linear}}(X, \tau)_t = \frac{1}{\tau} \int_0^\infty X[t-s]_{\text{linear}} e^{-s/\tau} \, \mathrm{d}s,$

where in all cases the observation times of the input and output time series are identical.

For reasons that will become clear shortly, we discuss the EMA_{next} first.

4.1 EMA_{next}

It is easy to show that Definition 4.1 (ii) is equivalent to

```
out[1] = values[1];
for (j in 2:N(X)) {
  w = exp(-(times[j] - times[j-1]) / tau);
  out[j] = out[j-1] * w + values[j] * (1-w);
}
c
```

EMA_{next}

This algorithm is identical to the algorithm commonly used for equally spaced time series. Therefore, applying the usual EMA for equally spaced data to unevenly spaced data implicitly applies next-point interpolation to the time series, something that, depending on the application, may not be desirable.

4.2 EMA_{last}

It is easy to show that Definition 4.1 (i) is equivalent to

$$\text{EMA}_{\text{last}}(X,\tau)_t = \begin{cases} X_{t_1} & \text{if } t = t_1, \\ (1 - e^{-\Delta t_n/\tau})X_{t_{n-1}} + e^{-\Delta t_n/\tau}\,\text{EMA}_{\text{last}}(X,\tau)_{t_{n-1}} & \text{if } t = t_n > t_1 \end{cases}$$

for $t \in T(X)$, which is the EMA_{next} of the original time series with lagged observation values. Hence, the $\text{EMA}_{\text{last}}(X,\tau)$ can be calculated recursively as well:

```
out[1] = values[1];
for (j in 2:N(X)) {
  w = exp(-(times[j] - times[j-1]) / tau);
  out[j] = out[j-1] * w + values[j-1] * (1-w);
}
c
```

EMA_{last}

4.3 $\text{EMA}_{\text{linear}}$

The $\text{EMA}_{\text{linear}}$ can also be calculated recursively, see Müller (1991).

```
out[1] = values[1];
for (j in 2:N(X)) {
  tmp = (times[j] - times[j-1]) / tau;
  w = exp(-tmp);
  w2 = (1 - w) / tmp;
  out[j] = out[j-1] * w + values[j] * (1 - w2) + values[j-1] * (w2 - w);
}
c
```

$\text{EMA}_{\text{linear}}$

5 Two-Sided, Non-Causal Rolling Operators

The rolling time series operators discussed so far are all causal, that is, the output value at each time point depends only on past observations. However, for some applications, such as the smoothing of noisy data, it is desirable to use a two-sided rolling time window. The resulting time series operator is non-causal, because the output can also depend on future observations.

The half-open rolling time window is now of the form $(t-\tau, t+\eta]$, where $\tau > 0$ ($\eta \geq 0$) is the width of the rolling time window before (after) the current output time. Again, the variable `left` denotes the index of the left-most observation inside the rolling time window, while the variable `right` denotes the index of to the right-most observation inside of the rolling time window. In particular,

$$t - \tau < \texttt{times[left]} \leq \texttt{times[right]} \leq t + \eta.$$

With this notation, the generic algorithm for two-sided rolling time series operators is:

```
left = 1; right = 1;
for (pos in 1:N(X)) {
  // Expand window on the right end
  while (right < N(X)) && (times[right + 1] <= times[pos] + eta)
    right = right + 1;

  // Shrink window on the left end
  while (times[left] <= times[pos] - tau))
    left = left + 1;

  // Calculate output for current rolling window
  out[pos] = do_some_calculation(values[left:right], times[left:right]);
}
c
```

Generic Algorithm for Two-Sided Rolling Time Series Operators

Note that a separate loop index (different from `right`) is now needed.

It is relatively straightforward to adapt the algorithms in Section 2, 3, and 2.2 to allow a two-sided rolling time window, which has been done in the accompanying implementation.

6 More General Operators

This section briefly discusses how the above operators can be combined and generalized.

6.1 Convolution Operators

Many of the operators discussed so far are examples of convolution operators, see Gilles Zumbach (2001), Dacorogna et al. (2001), and Eckner (2017). For the purpose of our discussion, the following simplified definition suffices:

Definition 6.1. *For a time series X and function f on $\mathbb{R} \times \mathbb{R}_+$ satisfying suitable integrability conditions, the convolution of X with f with last-point sampling, $X *_{\text{last}} f$, is a time series with*

$$T(X *_{\text{last}} f) = T(X),$$
$$(X *_{\text{last}} f)_t = \int_0^\infty f(X[t-s]_{\text{last}}, s)\, \mathrm{d}s, \qquad t \in T(X *_{\text{last}} f),$$

while the convolution of X with f with linear interpolation, $X *_{\text{linear}} f$, is a time series with

$$T(X *_{\text{linear}} f) = T(X),$$
$$(X *_{\text{linear}} f)_t = \int_0^\infty f(X[t-s]_{\text{linear}}, s)\, \mathrm{d}s, \qquad t \in T(X *_{\text{linear}} f).$$

It is easy to see that the SMAs in Section 3 are convolution operators with $f(x,s) = \frac{1}{\tau}\mathbf{1}_{\{0\le s\le\tau\}}$, while the EMAs in Section 4 are convolution operators with $f(x,s) = \frac{1}{\tau}e^{-s/\tau}$. The SMA and EMA algorithms can be generalized to calculate convolutions with functions of the form

$$f(x,s) = g(x)\frac{1}{\tau}\mathbf{1}_{\{0\le s\le\tau\}},$$

where g is a real-valued function. For example, $g(x) = x^k$ for $k \in \mathbb{N}$ allows to calculate the rolling k-th time series moment, denoted by $\mathrm{m}(X,\tau,k)$, where $k = 1$ is a simple moving average. In the case of linear interpolation, the algorithm typically requires numerical integration, because the shaded areas in Figure 2 are irregular. Alternatively, one can first apply the function g to the observation values of X, and then apply the simple moving average $\mathrm{SMA}_{\mathrm{last}}$ or $\mathrm{SMA}_{\mathrm{linear}}$ to the transformed time series.

6.2 Combining Operators

Time series operators that are built using the operators above can also be efficiently calculated. For example, the rolling variance of a time series over a time horizon τ can be calculated as

$$\begin{aligned}\sigma^2(X,\tau) &= \mathrm{SMA}_{\mathrm{last}}(X^2,\tau) - (\mathrm{SMA}_{\mathrm{last}}(X,\tau))^2\\ &= \mathrm{m}(X,\tau,2) - \mathrm{m}(X,\tau,1)^2,\end{aligned}$$

where X^k for a time series X is short-hand notation for taking the k-th moment of the individual time series observations.

Similarly, the average true range (ATR), a popular robust measure of price volatility, can be calculated as

$$\begin{aligned}\mathrm{ATR}(X,\rho,\tau) &= \mathrm{SMA}_{\mathrm{last}}(\mathrm{range}(X,\rho),\tau)\\ &= \mathrm{SMA}_{\mathrm{last}}(\mathrm{roll\_max}(X,\rho),\tau) - \mathrm{SMA}_{\mathrm{last}}(\mathrm{roll\_min}(X,\rho),\tau),\end{aligned}$$

where $\rho > 0$ is the range horizon, and $\tau > \rho$ is the smoothing horizon.

Finally, convolutions $X *_{\mathrm{last}} f$ and $X *_{\mathrm{linear}} f$ with densities of the form $f(x,s) = g(x)h(s)$, where $g : \mathbb{R} \to \mathbb{R}$ and $h : \mathbb{R}_+ \to \mathbb{R}$ are reasonably smooth functions, can be approximated by a linear combination of SMAs of the transformed time series $g(X)$, where

$$\begin{aligned}T(g(X)) &= T(X),\\ (g(X))_t &= g(X_t), \qquad t \in T(g(X)).\end{aligned}$$

7 Numeric Stability

The efficiency of many algorithms above rests on reusing a result or intermediate quantity from the previous time window to determine the output value for the current time window. For example, `roll_sum` for calculating the rolling average and sum, and `roll_area` for various SMAs. For long time series, the large number of additions and subtractions in updating these quantities can lead to the accumulation of round-off errors due to the finite precision of floating point numbers. See Higham (1993) for a comparison of summation algorithms for floating point numbers.

Moreover, if the rolling time window contains a very large number of observations or if the observation values cover several orders of magnitude, then the intermediate quantity can be several magnitudes larger than typical observation values. Under these conditions, round-off errors that are the relatively small at the scale of the intermediate quantity translate to much larger relative errors at the scale of typical observation values, a phenomenon commonly referred to as *catastrophic cancellation*.

The Kahan (1965) summation algorithm (also known as *compensated summation*) can be used to bound the size of the accumulated round-off error in the summation of a sequence of floating point numbers. To incorporate this method into the algorithms above, we define the following helper function to add together the two variables `sum` and `addend`:

```
compensated_addition = function(sum, addend, comp) {
  addend = addend - comp;
  sum_new = sum + addend;
  comp = (sum_new - sum) - addend;
  return (sum_new, comp)
}
c
```

Compensated Addition

Here, `comp` is a compensation for the accumulated numeric error from previous calls of this function. Note that the function returns a 2-tuple of numbers; the updated sum, and the updated accumulated numeric error.

Using this helper function, the rolling average algorithm from Section 2.1 becomes:[5]

```
left = 1; roll_sum = 0; comp = 0;
for (right in 1:N(X)) {
  // Expand window on the right
  (roll_sum, comp) = compensated_addition(roll_sum, values[right], comp)

  // Shrink window on the left
  while (times[left] <= times[right] - tau) {
    (roll_sum, comp) = compensated_addition(roll_sum, -values[left], comp)
    left = left + 1;
  }

  //  Save average value for current time window
  out[right] = roll_sum / (right - left + 1);
}
c
```

Numerically Stable Rolling Average

The other algorithms can be modified in a similar manner to make them numerically stable.

8 Conclusion

We have shown how to efficiently calculate several rolling operators for unevenly spaced time series. These operators in turn may serve as building blocks for more complicated data transformations.

[5] For performance reasons, it is recommended to inline the code of the helper function to avoid the function call overhead.

References

Box, G. E. P., G. M. Jenkins, G. C. Reinsel, and G. M. Ljung (2015). *Time Series Analysis: Forecasting and Control* (5th ed.). Wiley Series in Probability and Statistics.

Brockwell, P. J. and R. A. Davis (1991). *Time Series: Theory and Methods* (Second ed.). Springer.

Brockwell, P. J. and R. A. Davis (2016). *Introduction to Time Series and Forecasting* (3rd ed.). Springer.

Dacorogna, M. M., R. Gençay, U. A. Müller, R. B. Olsen, and O. V. Pictet (2001). *An Introduction to High-Frequency Finance*. Academic Press.

Eckner, A. (2017). Some properties of operators for unevenly spaced time series. Working Paper.

Fan, J. and Q. Yao (2003). *Nonlinear Time Series: Nonparametric and Parametric Methods*. Springer Series in Statistics.

Gilles Zumbach, U. A. M. (2001). Operators on inhomogeneous time series. *International Journal of Theoretical and Applied Finance 4*, 147–178.

Hamilton, J. (1994). *Time Series Analysis*. Princeton University Press.

Higham, N. J. (1993). The accuracy of floating point summation. *SIAM Journal on Scientific Computing 14*(4), 783–799.

Kahan, W. (1965). Further remarks on reducing truncation errors. *Communications of the ACM 8*(1), 40.

Knuth, D. E. (1998). *The Art of Computer Programming* (Second ed.), Volume 3: Sorting and Searching. Addison-Wesley Professional.

Lütkepohl, H. (2010). *New Introduction to Multiple Time Series Analysis*. Springer.

Müller, U. A. (1991). Specially weighted moving averages with repeated application of the ema operator. Working Paper, Olsen and Associates, Zurich, Switzerland.

Tong, H. (1990). *Non-linear Time Series: A Dynamical System Approach*. Oxford University Press.