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

Andreas Eckner*

First version: January 2010
Current version: October 13, 2013

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. A web appendix[1] provides an implementation of these algorithms in the programming language `C` and a package (forthcoming) for the statistical software `R`.

Keywords: unevenly spaced time series, unequally spaced time series, irregularly spaced time series, moving average, simple moving average, exponential moving average, R

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), Brockwell and Davis (2002), Fan and Yao (2003), Box et al. (2004), and Lütkepohl (2010). As a consequence, efficient algorithms have been developed for many computational questions, and implementations are available for software environments such as Fortran, Matlab®, R, and SAS®. 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, 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.

*Comments are welcome at andreas@eckner.com.
[1]See www.eckner.com/research.html.

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 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
}
```

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 Basic 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 much more detailed exposition see Eckner (2012).

We use the notation $((t_n, X_n) : 1 \leq n \leq N(X))$ or $(X_{t_n} : 1 \leq n \leq N(X))$ to denote an unevenly spaced time series X with 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 $t \in \mathbb{R}$, $X[t]$ denotes the most recent observation value of X at or before time t, while $X[t]_{\text{lin}}$ denotes the linearly interpolated value of X at time t. We call these quantities the sampled value X at time t with last-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 $n \leq m$, $n : m$ denotes the array of numbers $[n, n+1, \ldots m-1, m]$. For brevity of the presentation (but not in the accompanying implementation) we ignore memory allocation, numerical noise, 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 the rolling time window, while the variable `left` denotes the index of the left-most observation

[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 any calculation involving `NAs` is also `NA`.

inside the rolling time window. In other words,

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

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]);
}
```

Generic Algorithm for Rolling Time Series Operators

Here, `values[left:right]` and `times[left:right]` denotes the array of observation values and times, respectively, in the current rolling 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 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 extra complexity involved in the implementation pays off only for extremely large datasets. For time series of moderate length, a simpler algorithm is often just as efficient.*

2 Simple Moving Averages

Moving averages are used for summarizing the average value of a time series over a certain time horizon, for example, for the purpose of smoothing noisy observations. For equally spaced time series data there is only one way of calculating simple moving averages (SMAs) and exponential moving averages (EMAs). For unevenly spaced data, however, there exist multiple alternatives due to the choice of the sampling scheme. See Eckner (2012), Section 8 for a systematic analysis.

Definition 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) $\mathrm{SMA}(X,\tau)_t = \frac{1}{\tau}\int_0^\tau X[t-s]\,ds$,

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

(iii) $\mathrm{SMA}_{\mathrm{eq}}(X,\tau)_t = \mathrm{avg}\{X_s : s \in T(X) \cap (t-\tau, t]\}$

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

The first SMA can be used to analyze discrete observation *values*; for example, to calculate the average FED funds target rate[3] over the past three years. In such a case, it is desirable to weigh each observation value by the amount of time it remained unchanged. The SMA_{eq} is ideal for analyzing discrete *events*; for example, calculating the average number of casualties per deadly car accident over the past twelve months, or determining the average number of IBM common shares traded on the NYSE per executed order during the past 30 minutes. The SMA_{lin} 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 (2012), Theorem 6.18.

2.1 SMA_{eq}

The simple moving average SMA_{eq} can be calculated efficiently by keeping track of (i) the number and (ii) sum of observation values in a 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 SMA value for current time window
  out[right] = roll_sum / (right - left + 1);
}
```

SMA_{eq} Algorithm

For very large datasets, to avoid numerical noise due to a potentially large number of additions and subtractions, it is recommended to occasionally calculate the sum of observation values in the current window from scratch.

2.2 SMA

The simple moving average SMA can be efficiently calculated in a manner similar to the SMA_{eq}. 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:

[3]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.

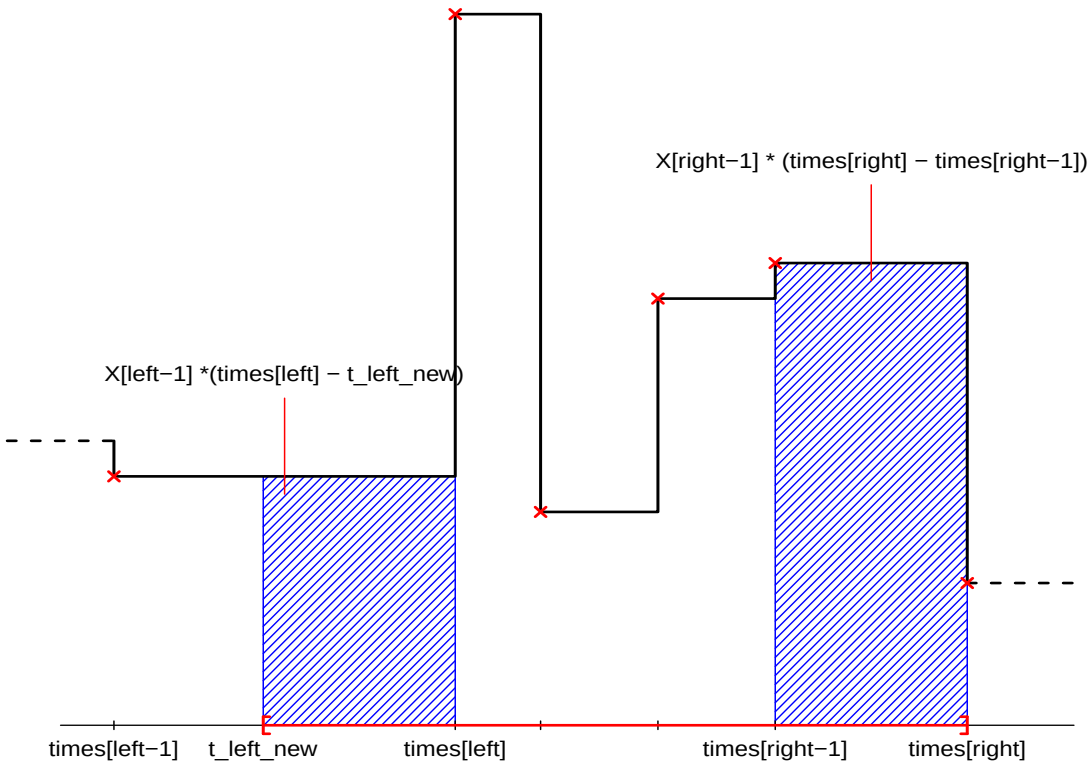

Figure 1: **Areas involved in the calculation of the simple moving average SMA(X, τ).** The step function is the sample path of a time series X with last-point sampling scheme. The shaded area on the right-hand side has to be added to the variable `roll_area` in order to expand the rolling time window to the right. Depending on the position in the 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] * (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;
}
```

SMA Algorithm

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.

2.3 SMA_{lin}

The simple moving average SMA_{lin} can be calculated like the SMA, 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) {
  if (x2 == x3 or x2 < x1)
    return (x3 - x2) * y1;
  else {
    weight = (x3 - x2) / (x3 - x1);
    y2 = y1 * weight + y3 * (1 - weight);
    return (x3 - x2) * (y2 + y3) / 2;
  }
}
```

SMA_{lin} helper function - Trapezoid Area

The second and third line the function treat 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:

```
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;
}
```

SMA_{lin} Algorithm

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.

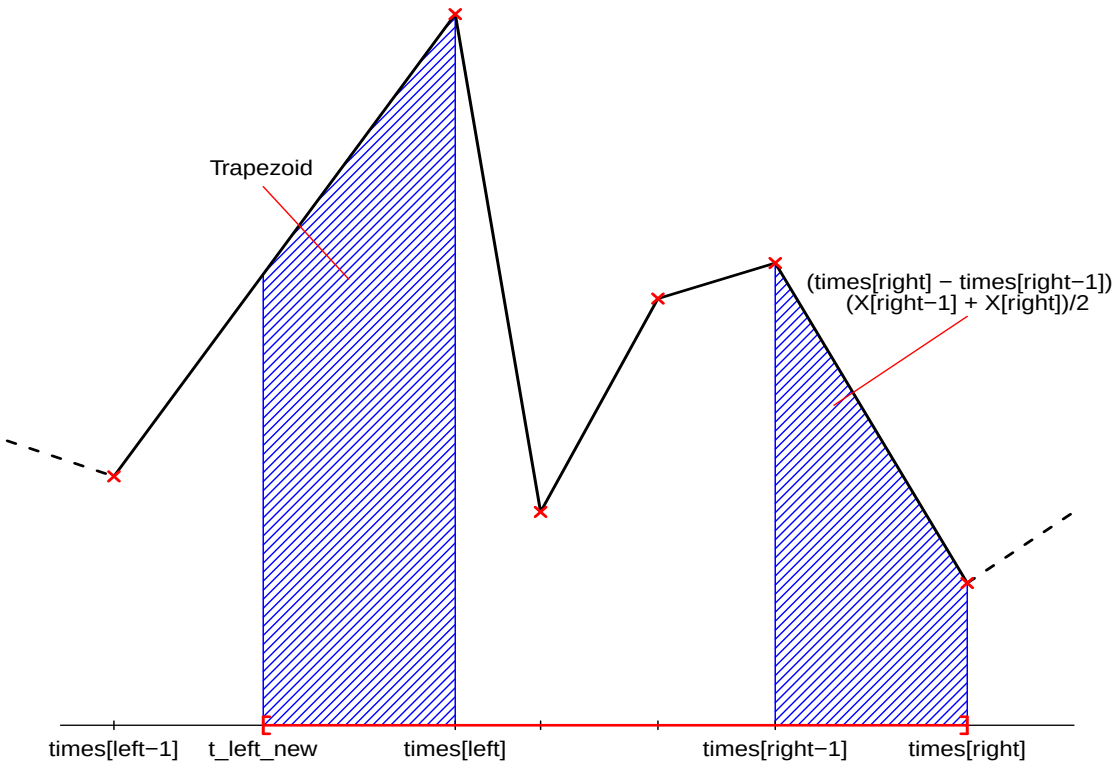

Figure 2: **Areas involved in the calculation of the simple moving average $\mathbf{SMA}_{\mathrm{lin}}(X, \tau)$.** The piecewise-linear function is the sample path of a time series X with linear interpolation. The shaded area on the right-hand side has to be added to the variable `roll_area` in order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

3 Exponential Moving Averages

Definition 2. *For an unevenly spaced time series X, we define three versions of the exponential moving average (EMA) of length $\tau > 0$. For $t \in \{t_1, \ldots, t_{N(X)}\}$,*

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

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

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

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

The first EMA uses last-point sampling, the second EMA uses linear interpolation, and the last EMA is motivated by the corresponding definition for equally spaced time series.

3.1 EMA_{eq}

By construction, the EMA_{eq} can be calculated recursively:

```
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);
}
```

EMA_{eq} Algorithm

3.2 EMA

It is easy to show that Definition 2 (i) is equivalent to

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

for $t \in T\left(X\right)$, which is the EMA_{eq} of the original time series with shifted observation values. Hence, the $\text{EMA}(X,\tau)$ 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);
}
```

EMA Algorithm

3.3 EMA_{lin}

The EMA_{lin} 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);
}
```

EMA_{lin} Algorithm

4 Other Rolling Operators

This section presents algorithms for rolling time series operators that are not moving averages, but nevertheless have a similar structure.

4.1 Rolling Sum and Number of Observations

The algorithm for the SMA_{eq} can be 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;
}
```

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;
}
```

Rolling Sum of Observations

For a fixed window width τ, by construction, the $\mathrm{SMA}_{\mathrm{eq}}$ of a time series equals the rolling sum divided by the rolling number of observations of the same time series.

4.2 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 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];
}
```

Rolling Maximum

The rolling minimum in a 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.*[4]

[4]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 for each time window.

5 More General Operators

This section briefly discusses how the operators discussed so far can be combined and generalized.

5.1 Convolution Operators

Many of the operators discussed so far are examples of convolution operators, see Gilles Zumbach (2001), Dacorogna et al. (2001), and Eckner (2012). For the purpose of our discussion, the following simplified definition suffices:

Definition 3. *For a time series X and function f on $\mathbb{R} \times \mathbb{R}_+$ satisfying suitable integrability conditions, the convolution $X * f$ of X with f (with last-point sampling) is a time series with*

$$\begin{aligned} T(X * f) &= T(X), \\ (X * f)_t &= \int_0^\infty f(X[t-s], s)\, ds, \qquad t \in T(X * f), \end{aligned}$$

and the convolution with linear-interpolation is given by

$$\begin{aligned} T(X *_{\text{lin}} f) &= T(X), \\ (X *_{\text{lin}} f)_t &= \int_0^\infty f(X[t-s]_{\text{lin}}, s)\, ds, \qquad t \in T(X *_{\text{lin}} f). \end{aligned}$$

The algorithms for the simple moving averages in Section 2 can be modified to calculate convolutions with functions of the form

$$f(x,t) = g(x) \frac{1}{\tau} \mathbf{1}_{\{0 \le t \le \tau\}}, \tag{5.1}$$

where $\tau > 0$ and 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 $k = 1$ amounts to calculating 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, we can first apply the function g in (5.1) to the observation values of X, and then apply the simple moving average SMA_{eq}, SMA, or $\mathrm{SMA}_{\text{lin}}$ to the transformed time series.

5.2 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

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

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

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

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

Finally, convolutions $X * f$ with densities of the form $f(x,t) = g(x)\,h(t)$, 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 simple moving averages of the transformed time series $g(X)$, where

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

6 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.

References

Box, G. E. P., G. M. Jenkins, and G. C. Reinsel (2004). *Time Series Analysis: Forecasting and Control* (4th 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 (2002). *Introduction to Time Series and Forecasting* (Second 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. (2012). A framework for the analysis of unevenly-spaced time series data. 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.

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.