accrued: Tools for visualizing data quality of partially accruing data

Julie Eaton
University of Washington Tacoma
`jreaton@uw.edu`

Ian Painter
University of Washington
`ipainter@u.washington.edu`

June 22, 2015

Abstract

A time series is *partially accruing* if data for each time point are accrued piecemeal and become more complete over time. The R package **accrued** provides tools for summarizing and visualizing data quality for partially accruing data. This vignette describes the data classes defined in **accrued** and the summary and visualization methods associated with each, with examples.

Keywords: Partially accruing data, data quality, data visualization

1 Introduction

The **accrued** package contains methods for visualizing data quality for partially accruing data. A time series is *partially accruing* if data for each time point are accrued piecemeal and become more complete over time. Data including the number of emergency visits per day (total counts) and the number of daily visits due to ILI were sent from various emergency departments to a single source. Data were received at different times, giving the time series a partially accrued structure.

The primary feature of partially accruing data we note is the effect of *accrual lag*—the difference between the *event time* (emergency department visit date) versus *record time* (also referred to as upload date), which is the time when the data become available to the system. Our summary and visualization methods primarily revolve around visualizing aspects of the data quality affected by accrual lag (referred to here as *lag*, though this is fundamentally different from the traditional definition of lag in time series analysis).

Due to the piecemeal accrual of data, the value of an indicator for a a particular event date changes over time. The following definitions help clarify different ways of characterizing the indicator value. The *current value* of an indicator refers to the value of an indicator for a particular event date as of the current date. The *complete data value* of the indicator refers to the value of an indicator for a particular event date once all data for the event date has been received. Finally, the *lagged value* of the indicator refers to the value of an indicator for a particular event date a fixed number of days after that date. The number of days lagged is specified so that, for example, the five-day lagged value for a particular event date is the current value five days after the event date.

The methods provided in **accrued** were developed for understanding the data quality issues surrounding the Distribute project for real time influenza-like-illness (ILI) surveillance (Distribute: [2]and [1]; methods: [3] and [4]).

2 Classes defined in accrued

The main class used in **accrued** is the "`accrued`" class. This class is a wrapper for a matrix containing partially accrued data. The function `data.accrued` creates an object of the "`accrued`" class. The only required argument, `x`, is a matrix of time series with rows as consecutive event dates (encounter dates) and columns as accrual lag. Accrual lag must be nonnegative integers and are are assumed to increase one unit with each column. Optional arguments are `start` (start date) and `final` (the vector of complete data). If `start` is omitted, the start date defaults to "1" and if `final` is omitted, then the right-most column of data is used in its stead. The matrix value in the ith row and jth column of `data` represent the value for the event date i, known $j-1$ time units after the event date.

To create the "`accrued`" object:

```
> library(accrued)
> data(accruedDataExample)
> accrued_data = data.accrued(accruedDataExample)
> names(accrued_data)
```

```
[1] "final" "data"  "start"
```

The accrued data in the "`accrued`" class can be extracted using "`data`"

```
> accrued_data[["data"]][101:115,1:10]
```

```
     0   1   2   3   4   5   6   7   8   9
101  0 177 286 373 457 478 478 478 478 478
102 NA 178 266 365 393 466 466 466 466 466
103  0 143 290 405 405 416 463 463 473 473
104  0 105 275 331 466 476 476 489 489 489
105  0 140 312 396 459 459 491 491 491 491
106  0  99 290 379 379 450 450 450 450 450
107 NA  NA  NA  NA 420 434 434 434 434 434
108  0 204 204 351 409 452 488 489 497 497
109 13  13 326 366 469 471 471 471 471 471
110 NA 148 320 371 413 413 413 413 413 413
111 NA 156 293 324 455 482 482 486 486 486
112 NA  NA  NA  NA 408 408 441 441 441 441
113 NA  NA  NA  NA  NA 479 479 479 479 479
114  0 102 264 264 427 428 433 433 433 433
115  0 168 168 327 423 461 461 461 461 461
```

The `summary` function for the "`accrued`" class provides a succinct summary of the data. Below is an example summary. The first column represents the percentage of dates on which a positive number of counts lagged by $0, 1, 2, \ldots, 20$ days were received. The second column represents the mean percentage of counts relative to the final counts received, for each lag. The third column represents the mean counts received for each lag.

```
> summary(accrued_data)
```

```
Summary of accrued data object with  time points.

        Upload Percent  Proportion  Mean Count  Quartile 1  Median  Quartile 3
 Lag
 0      0.61            0.00          1.5       0.00        0.00    0.00
 1      0.84            0.26        122.9       0.19        0.25    0.33
 2      0.86            0.56        268.8       0.50        0.57    0.65
 3      0.86            0.75        358.8       0.71        0.77    0.81
 4      0.93            0.92        437.4       0.89        0.93    0.97
 5      1.00            0.97        462.3       0.95        1.00    1.00
 6      1.00            0.99        471.1       0.99        1.00    1.00
 7      1.00            0.99        470.5       1.00        1.00    1.00
 8      1.00            0.99        473.0       1.00        1.00    1.00
 9      1.00            1.00        477.3       1.00        1.00    1.00
```

```
10     1.00           1.00       477.4     1.00       1.00     1.00
11     1.00           1.00       477.5     1.00       1.00     1.00
12     1.00           1.00       477.6     1.00       1.00     1.00
final  1.00           1.00       477.6     1.00       1.00     1.00
```

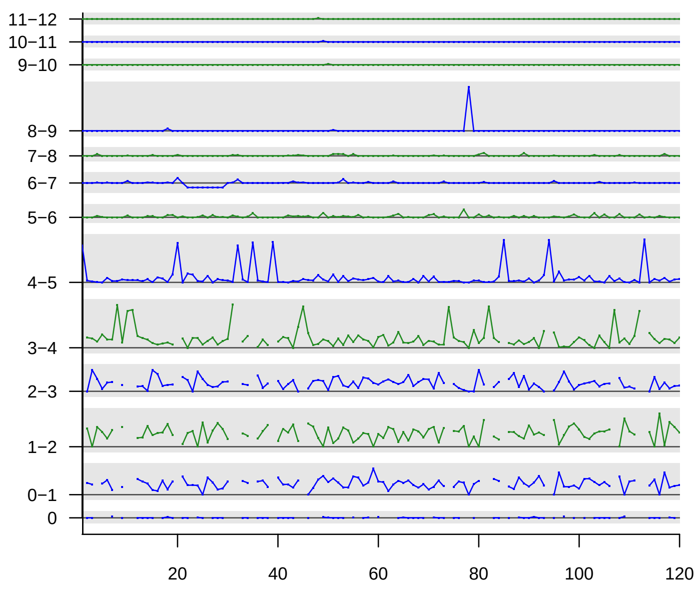
3 Stacklag difference plot

The default plot for an object of the “`accrued`” class is the stacklag difference plot. An example of this plot is shown in Figure 1 and is produced using the following function call.

```
> plot(accrued_data)
```

This plot shows changes in count values from one lag to the next by stacking scaled time series of differences between values. The x-axis shows the event date, and the $(Z + 1)$th layer shows the difference in count indicator values between lag Z and lag $(Z-1)$, for $Z \geq 1$. For layer 1, the counts received at lag 0 are shown, and at any date if no counts have been received for the current or any previous lags no value is shown.

Figure 1: Stacklag difference plot for the example data, the default plot for an “`accrued`” object.

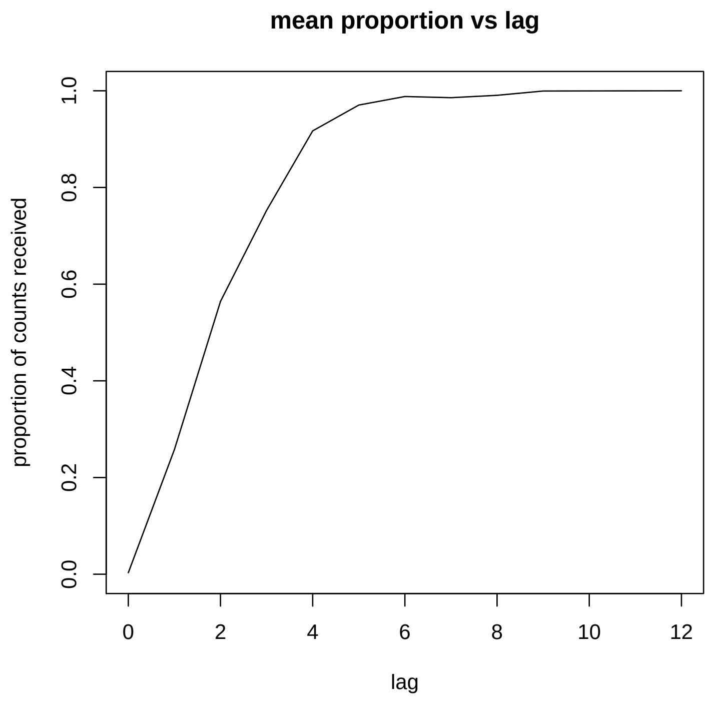


4 Summary completion curve

The "`summary.accrued`" function returns an object of type "`summary.accrued`", which has a default plot function, called the *summary completion curve*, which graphs the second column of the `print.summary.accrued` output against lag, that is, mean percentage versus lag. The summary completion curve for the example data is shown in Figure 2. It is produced using the function call

```
> plot(summary(accrued_data))
```

Figure 2: Summary completion curve for the example data, the default plot for a `summary.accrued` object.



5 Bar code plot

The bar code plot is a sparkline graph showing a vertical line for each date on which an record was received. This function requires the data be in the form of counts.

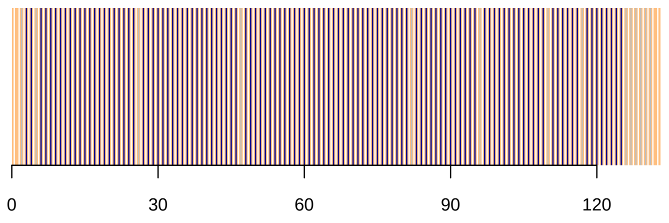
```
> accrued_data_reduced = data.accrued(accruedDataExample)
```

The bar code plot call is made here

```
> barcode(accrued_data_reduced)
```

The bar code plot for the example reduced data is shown in Figure 3.

Figure 3: Bar code plot for the example data.



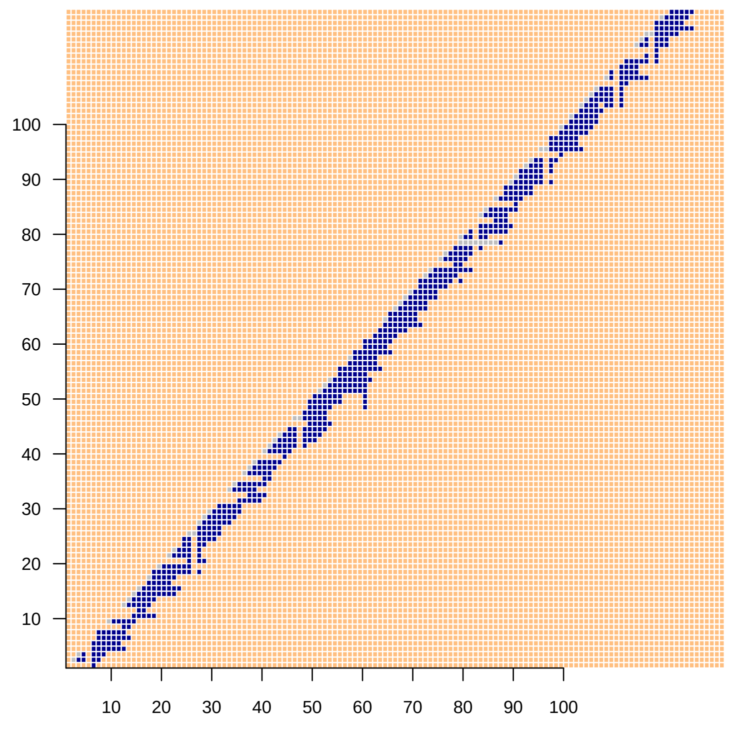
6 Upload pattern plot

The upload pattern plot displays indicators of whether non-zero counts were received on a particular day. This function has two display options. In both cases count data are required. In the first way, record dates (upload dates) are represented on the horizontal axis and event dates (encounter dates) are represented on the vertical axis. This is the default plot (the parameter `horizontal` is set to `FALSE`). The point with coordinates (i, j) is plotted if a positive count was received on that record date for that particular event date. Since event dates must always occur on or before record dates, this produces a set of points with integer coordinates that lie on or below the line $y = x$. The upload pattern plots can be difficult to view if too many upload dates are included.

```
> uploadPattern(accrued_data_reduced)
```

and the plot is shown in Figure 4.

Figure 4: Upload plot for the example data.



A more compact way of representing upload information is by replacing the vertical axis with lag instead of event date. Recall that for a particular record-event combination, the associated accrual lag is

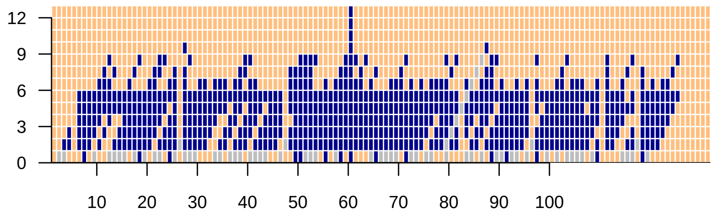
$$\text{(accrual lag)} = \text{(record date)} - \text{(event date)}.$$

To obtain this plot, the optional argument `horiztonal` must be set to `TRUE` (its default value is `FALSE`). The plot is produced using the code

```
> uploadPattern(accrued_data_reduced, horizontal=TRUE)
```

and resulting plot appears in Figure 5.

Figure 5: Horiztonal upload plot for the example data.



7 Arrayed lag time series

The function `laggedTSarray` generates time series of indicator values, arrayed by lag. Its only required argument is an object of the "`accrued`" class. A running median line using the previous `daysOfHistory` days of data is drawn to show the average pattern of each time series. Lines at plus and minus two median absolute deviations (MAD) from the median to show the variability of each time series, both using a `daysOfHistory` window. If `daysOfHistory` is not specified, it defaults to 30.

The following code produces arrayed lag time series for all lags. This produces as many pages of plots as there are lags.

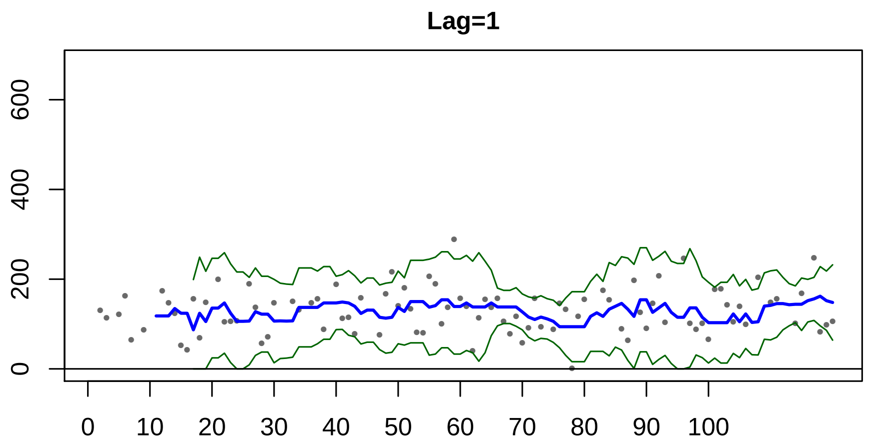
```
> laggedTSarray(accrued_data, daysOfHistory=20)
```

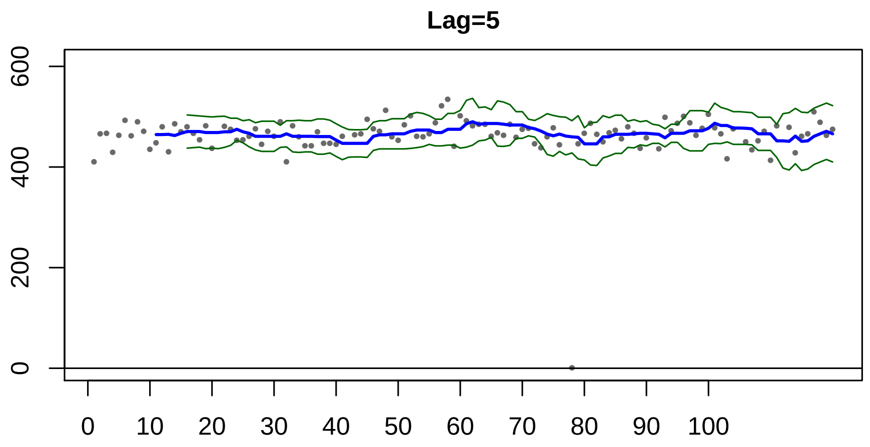
The following code produces a time series for a lag of 1 only.

```
> laggedTSarray(accrued_data, daysOfHistory=20, lags=c(1))
```

The output of three separate calls is shown in Figure 6.

Figure 6: Arrayed lag time series using 20 days of previous data for lag=1 and 5.





8 Lag histograms

The completion distribution refers to the percentage of total counts received for each lag. The lag histograms show the completion distribution for lags ranging from 0 to 14 days. For a fixed lag Z and for each event date, the proportion of counts available relative to the final counts present for that date by that lag is computed. That is,

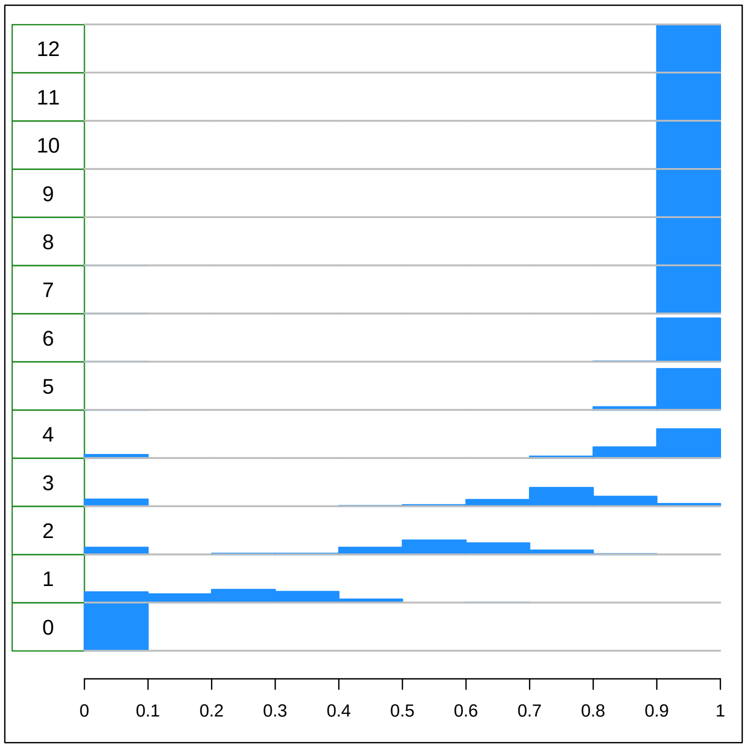
$$\text{(percent complete, } Z \text{ days lagged)} = \frac{\text{(counts, } Z \text{ days lagged)}}{\text{(final counts)}}.$$

Percentages above 100 are rounded down. Repeating this calculation over all event dates yields a completion distribution. The function `lagHistogram` graphs the completion distribution. Its only required argument is an object of the "`accrued`" class. The function call for producing the lag histograms is shown here:

```
> lagHistogram(accrued_data)
```

The lag histograms for the example data appear in Figure 7. Note that only lags 0 to 14 are shown, even though the data contain more lags.

Figure 7: Lag histograms for the example data.



9 Errors

For partially accruing data, we refer to count error as the difference between the final count for a day and the lagged count for a day. One expects that the error in the total counts decreases as lag increases. The function `accruedErrors` inputs an "`accrued`" object and outputs one of the "`accruedErrors`" class. `accruedErrors` computes the count errors using the error function `func`. The default error function is the difference between final and lagged observation.

```
> dat = data.accrued(accruedDataExample)
> countErrors = accruedErrors(dat)

> summary(countErrors)  # the summary function prints the 0.1, 0.5, and 0.9 quantiles.

        0   1     2     3    4    5  6    7 8 9 10 11 12
0.1 437.2 288 136.2  53.2  5.0  0.0  0  0.0 0 0  0  0  0
0.5 472.0 351 206.0 116.0 32.0  1.5  0  0.0 0 0  0  0  0
0.9 513.8 439 291.6 178.8 78.6 45.0 24 11.1 0 0  0  0  0

> errorQuantileSummary(countErrors, quantiles=c(0.1,0.2,0.5,0.8,0.9))

        0   1     2     3    4    5    6    7 8 9 10 11 12
0.1 437.2 288 136.2  53.2  5.0  0.0  0.0  0.0 0 0  0  0  0
0.2 449.4 305 153.8  71.8 11.0  0.0  0.0  0.0 0 0  0  0  0
0.5 472.0 351 206.0 116.0 32.0  1.5  0.0  0.0 0 0  0  0  0
0.8 498.6 408 257.6 152.2 61.8 32.2 10.2  4.2 0 0  0  0  0
0.9 513.8 439 291.6 178.8 78.6 45.0 24.0 11.1 0 0  0  0  0

> # errorQuantileSummary allows the user to specify a desired quantile vector

> plot(countErrors)
> # generic plot call does not allow user to specify quantiles plotted.
> # Quantiles plotted are 0.1, 0.5, and 0.9.
>
> plot.accruedErrors(countErrors, quantiles=c(0.2,0.5,0.8))
> # this version of the plot call allows the user to specify the quantiles.
```

In Figure 8 the count errors are shown.

The ILI ratio is defined by

$$\text{(ILI ratio)} = \frac{\text{(ILI counts)}}{\text{(total counts)}}.$$

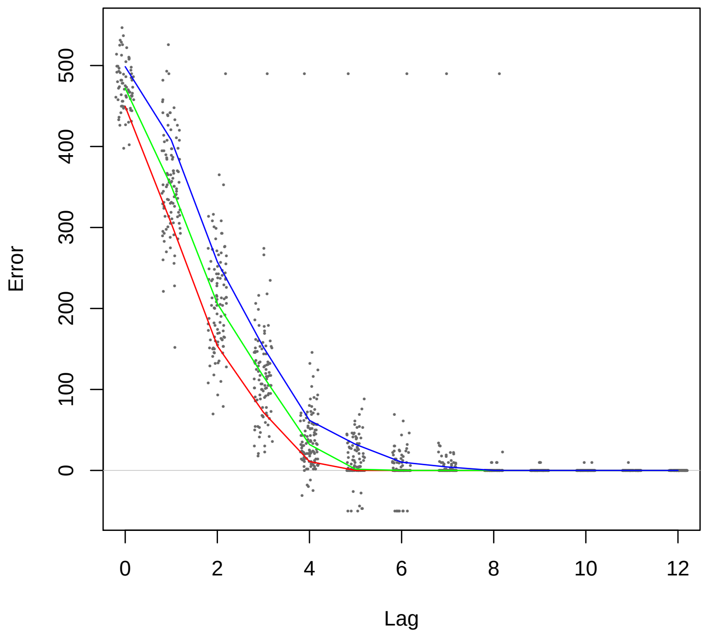
The ILI ratio error is given by

$$\text{(ILI ratio error, lagged by } Z \text{ days)} = \frac{\dfrac{\text{(ILI counts, lagged by } Z \text{ days)}}{\text{(total counts, lagged by } Z \text{ days)}}}{\dfrac{\text{(final ILI counts)}}{\text{(final total counts)}}}. \tag{1}$$

If two arguments are passed to the `accruedErrors` function instead of one (both must be counts), then a ratio is computed (the ILI ratio), and the log error ratio is computed as the default error function, that is, the log of (1) is returned. In the ratio computation, the first argument (`x`) is the denominator, and the second argument (`y`) is the numerator. The function call appears below, and the associated plot appears in Figure 9

```
> data(accruedDataILIExample)
> dat2 = data.accrued(accruedDataILIExample)
> combinedErrors = accruedErrors(x=dat, y=dat2)
> plot.accruedErrors(x=combinedErrors, quantiles=c(0.1, 0.2, 0.5, 0.8, 0.9))
```

Figure 8: Count errors for the example data.

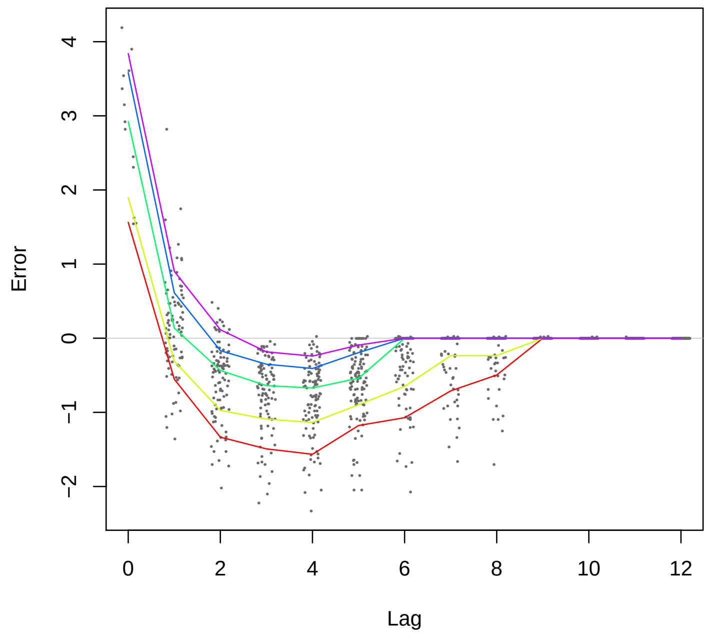


The user can also specify the error function. In the following example, the square root function is used to compute the error of the counts contained in `accrued_data`. Note that the "x" in the argument of the function `func` is *different* from the first argument of `accruedErrors`, `x`. The plot of the errors appears in Figure 10.

```
> errors = accruedErrors(x=dat, func = function(a,b) {sqrt(a) - sqrt(b)} )
> round(summary(errors), 4)

          0       1      2      3      4      5      6      7 8 9 10 11 12
0.1 19.4071  8.1285 3.4802 1.2612 0.1161 0.0000 0.0000 0.0000 0 0  0  0  0
0.5 21.5174 10.5648 5.4165 2.7745 0.7422 0.0351 0.0000 0.0000 0 0  0  0  0
0.9 22.5566 14.4148 7.9905 4.3689 1.7687 1.0298 0.5254 0.2501 0 0  0  0  0
```

Figure 9: Log of the error in ILI ratio for the example data.



References

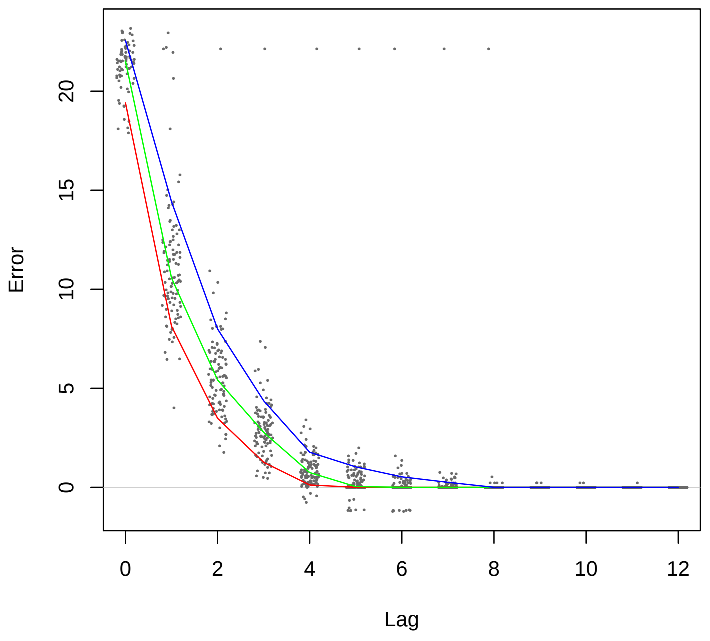
[1] W. Lober, B. Reeder, I. Painter, D. Revere, P. Bugni, K. Goldov J. McReynolds, E. Webster, and D. Olson. Technical description of the distribute project: A community-based syndromic surveillance system implementation. *Online Journal of Public Health Informatics*, 5, 2014.

[2] D. Olson, M. Paladini, W. Lober, L. Buckeridge, and I. D. W. Group. Applying a new model for sharing population health data to national syndromic influenza surveillance: Distribute project proof of concept, 2006 to 2009. *PLOS Currents Influenza*, 3, 2011.

[3] I. Painter, J. Eaton, D. Olson, D. Revere, and W. Lober. How good is your data. 2011 ISDS Conference abstract. *Emerging Health Threats Journal*, 4, 2011.

[4] I. Painter, J. Eaton, D. Olson, D. Revere, and W. Lober. Visualizing data quality: tools and views. 2011 ISDS Conference abstract. *Emerging Health Threats Journal*, 4, 2011.

Figure 10: Graph of count errors with the user specified error function.


Error
Lag