Estimating spot volatility in R: The `spotvol` function in the `highfrequency` package

Maarten Schermer

August 28, 2014

1 General usage

The `spotvol` function offers several methods to estimate spot volatility and its intraday seasonality, using high-frequency data. The spot volatility $\sigma_{t,n}$ can be described as the the volatility of returns on a certain moment n during day t. These returns $r_{t,n}$ are commonly modeled as

$$r_{t,n} = \sigma_{t,n}\varepsilon_{t,n}, \tag{1}$$

where $\varepsilon_{t,n} \sim \text{IID}(0, \frac{1}{N})$ is a white noise process. N denotes the number of intraday periods, and we will denote the number of days with T.

The spot volatility estimates $\hat{\sigma}_{t,n}$ can be estimated from these returns using the `spotvol` function. The estimates are stored as the variable `spot` in a `spotvol` object, which is a list containing certain outputs of the function, depending on the method used. These estimates can be called in the following way

```
library(highfrequency)
data(sample_real5minprices)
out <- spotvol(sample_real5minprices)
class(out)

## [1] "spotvol"

sigma_hat <- out$spot
```

1.1 Input data: prices or returns

The `spotvol` function accepts two sorts of input:

1. An `xts` object containing price data.

2. A `matrix` containing return data.

The first can be used for raw price data, i.e. from the NYSE TAQ database. The `highfrequency` package offers several methods to process data files into `xts` format. The `spotvol` function will then aggregate this price data to the frequency specified by the user and calculate the return series.

```
str(sample_real5minprices)

## An 'xts' object on 2005-03-04 09:30:00/2005-06-01 16:00:00 containing:
##   Data: num [1:4819, 1] 105 105 105 105 104 ...
##   Indexed by objects of class: [timeDate] TZ:
##   xts Attributes:
##  NULL

out <- spotvol(sample_real5minprices)
```

The second way of inputting data is through a `matrix`, which can be convenient if your data has already been processed. No further permutations will be made to the data, as it is assumed to be in the right format. Therefore, it should satisfy the following restrictions:

- The returns should be equispaced, i.e. the time between each pair of observations should be equal.
- Each row of the matrix should correspond to a day.
- Each column of the matrix should correspond to an intraday time interval.

For example, a matrix containing 5-minute returns of 60 days of 24h exchange rate data should have 60 rows and 288 columns:

```
str(sample_returns_5min)

## Error:  object 'sample_returns_5min' not found

out <- spotvol(sample_returns_5min)

## Error:  object 'sample_returns_5min' not found
```

1.2 Market opening times and time zones

Working with `xts` objects means that every observation in your data should have a timestamp. This is convenient, but can also pose some problems (e.g. with timezones). The `spotvol` function does not require the data to be timestamped: you can also supply raw return data in matrix form (see 1.1).

Market opening and closing times can be specified by the arguments `marketopen`

and `marketclose`. They should be entered in 24h format and in the timezone specified by `tz`, which should also be the timezone of your `xts` data. If your `xts` object does not have a timezone attribute specified, it will be assigned value `tz`. Note that all of your data must be in the same timezone. Problems can arise when using a daylight saving dependent timezone, such as CET, which changes to CEST during summer. It is advised to use a timezone which is invariant to daylight saving, such as GMT/UTC.

The time entered as `marketopen` should always be before `marketclose`. This can be a problem in case of 24h data, which lacks open and close times, or the opening time would be the same as the closing time. As the `spotvol` function processes the data on a daily basis, some problems arise when a 24h-period of data spans multiple days. There are two ways around this: either have your data always start at 0:00:00, or avoid time issues by entering your data as a `matrix`.

1.3 Method specific parameters

The `spotvol` function offers several methods to estimate the spot volatility. These are all listed in the help pages of the function and package. The user can specify the method to be used with the `method` argument, which accepts the following values:

- `"detper"` (default)
- `"stochper"`
- `"kernel"`
- `"piecewise"`

Each of these has different parameters to be specified by the user. These can all be entered as arguments to the `spotvol` function, although they will only be accepted if they are actually used by the specified method. The 'Details' section of the `spotvol` help page lists which parameters are used by each method. It is not required to specify all parameters; the function will even work if none are entered, by using default values.

1.4 Output

The `spotvol` function returns an object of the `spotvol` class. This is a list of which the contents depend on the used method. In any case, it includes the variable `spot`, which contains all spot volatility estimates as an `xts` or `matrix` object (depending on the input). The following example shows the different outputs for methods `"detper"` and `"stochper"`:

```
out1 <- spotvol(sample_real5minprices, method = "detper")
out2 <- spotvol(sample_real5minprices, method = "stochper")
```

```
names(out1)

## [1] "spot"     "daily"    "periodic"

names(out2)

## [1] "spot" "par"
```

1.5 Plotting

To get a quick overview of the output, the `plot` function can be called on `spotvol` objects. The S3 method `plot.spotvol` has been implemented to show the spot volatility estimates, as well as any other components of the `spotvol` object that can be visually represented, depending on the used method. An additional argument that can be passed to `plot.spotvol` is `length`, which can limit the amount of data points to be plotted. Examples are shown in section 2.

2 Methods

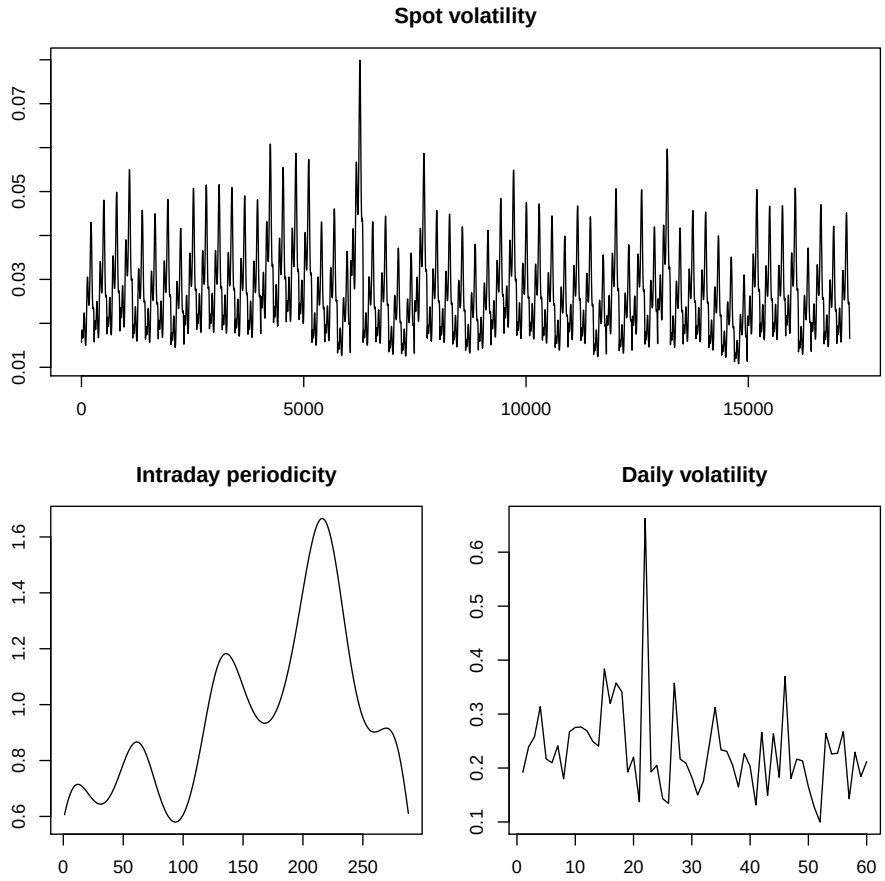
Each method implemented in the `spotvolatility` package has been proposed in an academic paper. For detailed descriptions, refer to these papers written by the original authors of the methods. References can be found at the end of this document or in the help page of this package, where you can also find a short overview of each method. In this section, we show some examples for each method.

2.1 Deterministic periodicity - `"detper"`

This is the default method. The spot volatility is decomposed into a deterministic periodic factor f_i (identical for every day in the sample) and a daily factor s_t (identical for all observations within a day). Both components are then estimated separately. For more details, see Taylor and Xu (1997) and Andersen and Bollerslev (1997). The jump robust versions by Boudt et al. (2011) have also been implemented.

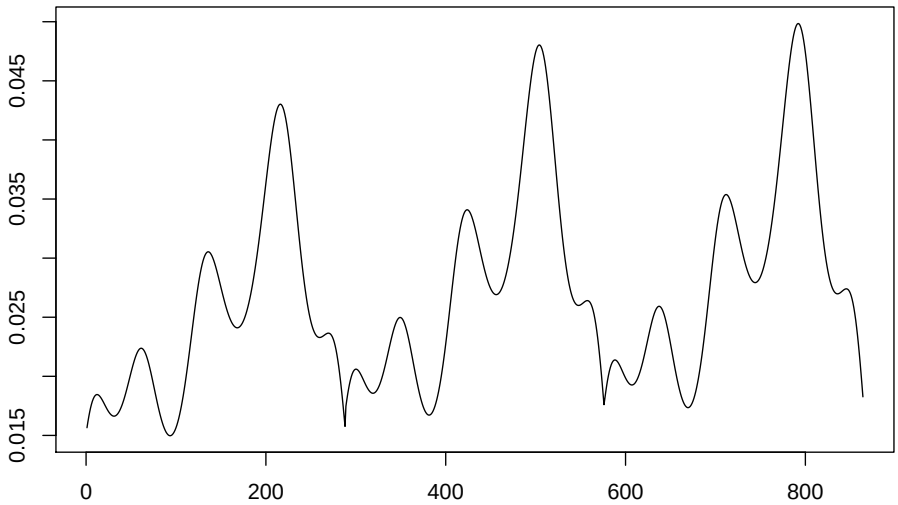
First, we can use the `plot` function to get a quick view of the results. This will call the `plot.spotvolatility` function, which automatically shows all relevant results contained in the `spotvol` object. When estimated by the deterministic method, it plots the general spot volatility estimates in the top panel, and the intraday periodicity pattern and daily volatilies in the bottom panels:

```
data(sample_returns_5min)
vol_detper <- spotvol(sample_returns_5min, method = "detper")
plot(vol_detper)
```



Now, to get a better look at the spot volatility estimates, we need to zoom in. This time, we only plot the first 3 days of the spot volatility. Recall that since we entered the input as a matrix, the output `vol_detper$spot` is also a matrix, with each row representing 1 day of estimates:

```
plot(as.numeric(t(vol_detper$spot[1:3,])), type = "l")
```



Now, it is easier to see how this method works. The estimated periodicity pattern (the second panel in the previous plot) is repeated every day, and it is multiplied by the estimate of the daily volatility (the third panel in the previous plot). To get the plot to zoom in, we also could have run

```
plot(vol_detper, length = 3*288)
```

2.2 Stochastic periodicity - `"stochper"`

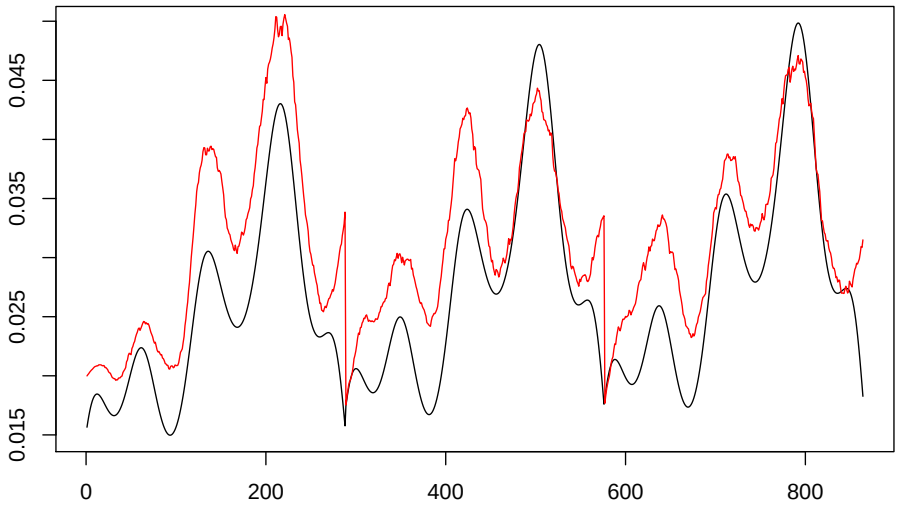
This method by Beltratti and Morana (2001) assumes the periodicity factor to be stochastic. The spot volatility estimation is split into four components: a random walk, an autoregressive process, a stochastic cyclical process and a deterministic cyclical process. The model is estimated using a quasi-maximum likelihood method based on the Kalman Filter. The package FKF is used to apply the Kalman filter. In addition to the spot volatility estimates, all parameter estimates are returned.

Like most stochastic volatility models, this model can be difficult to estimate. Optimization can take a long time, and results can depend on the initial values used in the estimation of the parameters. Therefore, these values can be specified by the user, as well as optimization restrictions:

```
init <- list(sigma = 0.03, sigma_mu = 0.005, sigma_h = 0.007,
    sigma_k = 0.06, phi = 0.194, rho = 0.986, mu = c(1.87, -0.42),
    delta_c = c(0.25, -0.05, -0.2, 0.13, 0.02), delta_s = c(-1.2,
        0.11, 0.26, -0.03, 0.08))
vol_stochper <- spotvol(sample_returns_5min, method = "stochper",
    init = init, control = list(maxit = 20))
```

The `control` argument will be passed down to `optim`, so it will accept all options listed in `?optim`. We can now compare the spot volatility estimates of the stochastic method to those calculated previously by the deterministic model:

```
plot(as.numeric(t(vol_detper$spot[1:3,])), type = "l")
lines(as.numeric(t(vol_stochper$spot[1:3,])), col = "red")
```



2.3 Nonparametric filtering - `"kernel"`

This method by Kristensen (2010) filters the spot volatility in a nonparametric way by applying kernel weights to the standard realized volatility estimator. Different kernels and bandwidths can be used to focus on specific characteristics of the volatility process.

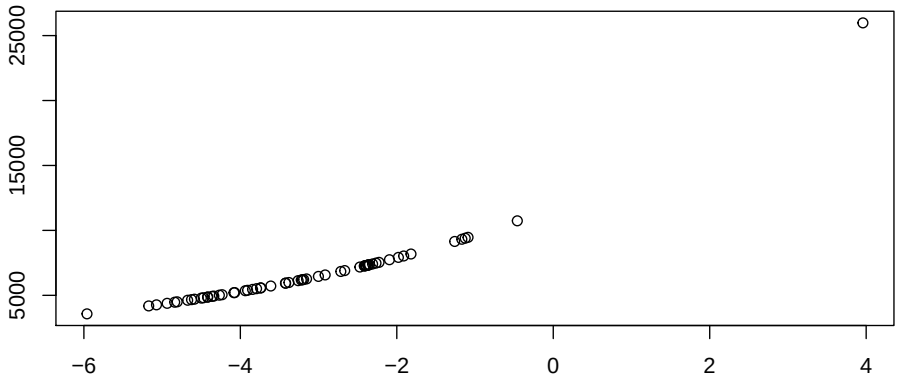
Estimation results heavily depend on the bandwidth parameter h, so it is important that this parameter is well chosen. However, it is difficult to come up with a method that determines the optimal bandwidth for any kind of data or kernel that can be used. Although some estimation methods are provided, it is advised that you specify h yourself, or make sure that the estimation results are appropiate.

An easy way to get a quick estimate of h is to use the R function `bw.nrd0`. The kernels will be calculated on a daily basis using the time index of each data point. Therefore, the bandwidth should be calculated from the same data, i.e. a vector of intraday times, for example in seconds. For data consisting of 5-minute returns, this would be

```
h1 = bw.nrd0((1:nrow(sample_returns_5min))*(5*60))
vol3 <- spotvol(sample_returns_5min, method = "kernel", h = h1)
```

Another quick estimator has been implemented, one that multiplies the above simple estimate by the quarticity of the returns on each day. It can be used by the option `est = "quarticity"`. The quarticity is a measure of the variance of the spot volatility. Days with higher quarticity will be assigned a larger bandwidth, to reduce the influence of single large returns. When using the method "kernel", in addition to the spot volatility estimates, all used values of the bandwidth h are returned, so we can compare them to the quarticity values:

```
quarticity = (nrow(sample_returns_5min)/3) * rowSums(sample_returns_5min^4)
vol4 <- spotvol(sample_returns_5min, method = "kernel", est = "quarticity")
plot(log(quarticity), vol4$par$h)
```

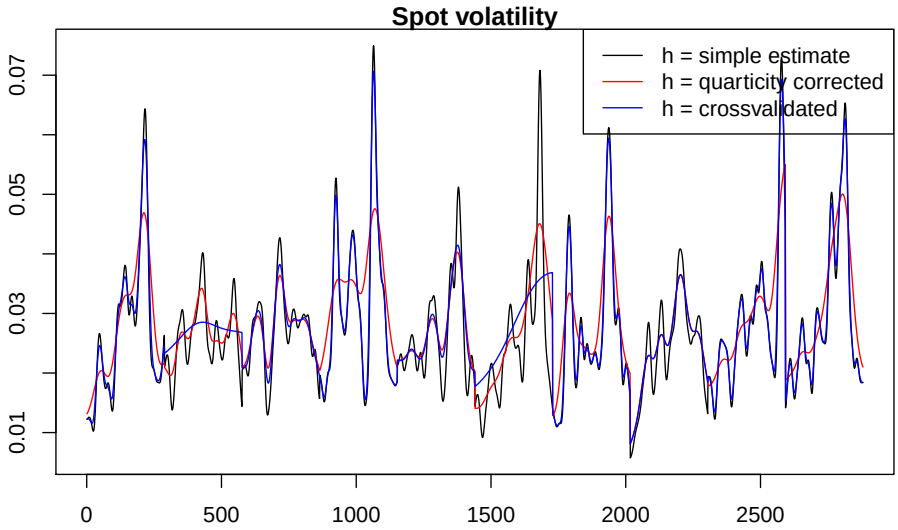


One way to estimate h, is by using cross-validation. For each day in the sample, h is chosen as to minimize the Integrated Square Error, which is a function of h. However, this function often has multiple local minima, or no minima at all ($h \to \infty$). To ensure a reasonable optimum is reached, strict boundaries have to be imposed on h. These can be specified by lower and upper, which by default are $0.1n^{-0.2}$ and $n^{-0.2}$ respectively, where n is the number of observations in a day. Crossvalidation can be used by specifying the option `est = "cv"`:

```
vol5 <- spotvol(sample_returns_5min, method = "kernel", est = "cv")
```

Now, we can easily compare the estimates of each bandwidth selection method by adding them to the same plot (`plot.spotvol` does not show any additional plots besides the spot volatility estimates for this method):

```
plot(vol3, length = 2880)
lines(as.numeric(t(vol4$spot))[1:2880], col = "red")
lines(as.numeric(t(vol5$spot))[1:2880], col = "blue")
legend("topright", c("h = simple estimate", "h = quarticity corrected",
    "h = crossvalidated"), col = c("black", "red", "blue"), lty = 1)
```



This plot compares the spot volatility estimates of the first 10 days. The simple estimator usually yields the smallest bandwidth, which makes it more sensitive to shocks. On the 2nd and 6th day (observations 289-576 and 1441-1728 in the plot, respectively), the crossvalidation estimate has hit its upper boundary, meaning the minimization problem had no feasible solution on those days.

2.4 Piecewise constant volatility - `"piecewise"`

This nonparametric method by Fried (2012) assumes the volatility to be piecewise constant over local windows. Robust two-sample tests are applied to detect changes in variability between subsequent windows. The spot volatility can then be estimated by evaluating regular realized volatility estimators within each local window.

An example from Fried (2012) consists of simulated data from the student's t-distribution. The volatility is equal to $\sqrt{(5/3)}$ for the first 1000 observations, $1.5\sqrt{(5/3)}$ from 1001 to 2000, $2\sqrt{(5/3)}$ from 2001 to 2500, and $\sqrt{(5/3)}$ again for the last 500 observations. We organize these simulated values in a matrix containing 500 observations per row:

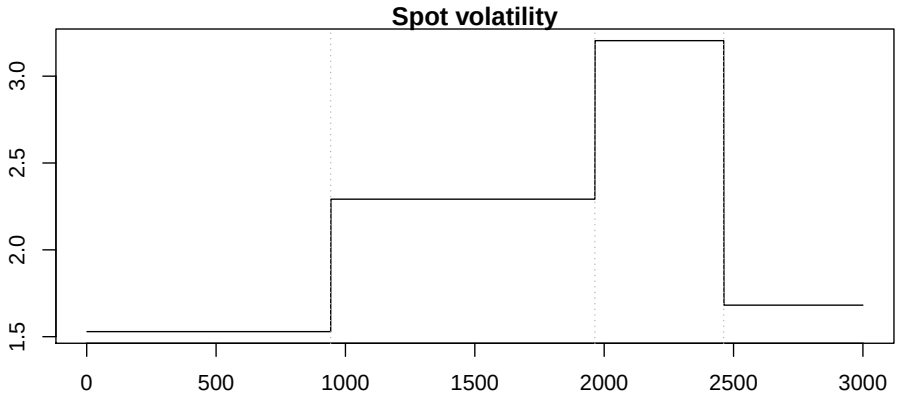
```
simdata <- matrix(sqrt(5/3) * rt(3000, df = 5), ncol = 500, byrow = TRUE)
simdata <- c(1, 1, 1.5, 1.5, 2, 1) * simdata
```

When we apply the piecewise constant volatility method the resulting `spotvol` object will contain both the spot volatility estimates and the detected change points in the volatility level. When plotting it, these change points will be visualized:

```
vol6 <- spotvol(simdata, method = "piecewise", m = 200, n = 100,
    online = FALSE)

## Detecting change points...
## Change detected at observation 943 ...
## Change detected at observation 1964 ...
## Change detected at observation 2462 ...

plot(vol6)
```

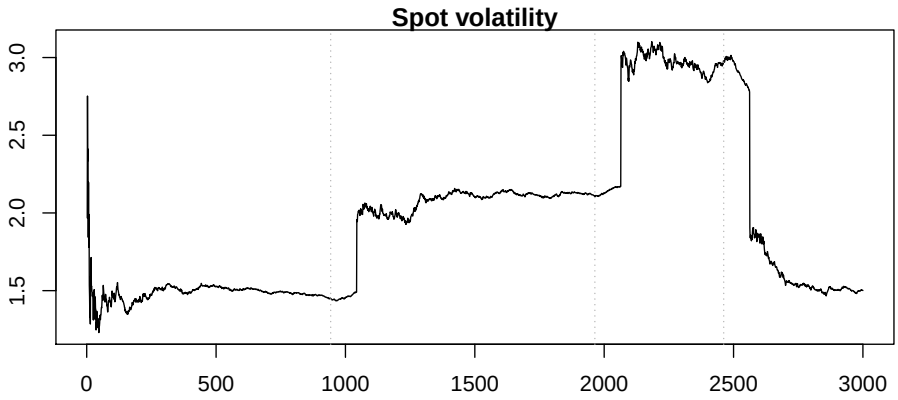


Note that these results depend on randomly generated values, which can vary when you repeat this example. Here, we applied ex post estimation, using all observations between two change points to estimate the volatility. Another option is to apply online estimation, where only observations up to point t are used in the estimation of the volatility at point t. This results in slowly changing estimates, depending on the lengths m and n of the reference and test windows:

```
vol7 <- spotvol(simdata, method = "piecewise", m = 200, n = 100,
    online = TRUE, volest = "tau")

## Detecting change points...
## Change detected at observation 943 ...
## Change detected at observation 1964 ...
## Change detected at observation 2462 ...

plot(vol7)
```



This can lead to a few unreliable estimates after each change point, when only a couple of observations are available. We also used a different estimator of the volatility for this plot. It can be specified by the option `volest`.

2.5 GARCH with intraday seasonality - "garch"

The package also includes an option to apply GARCH models, implemented by the `rugarch` package, to estimate spot volatility from intraday data. This is done by including external regressors in the GARCH model. These regressors are based on a flexible Fourier form, which was also used in the stochastic and deterministic periodicity estimation methods.

The `rugarch` package offers several GARCH specifications, which can be chosen from using the `model` argument in the `spotvol` function. Depending on the data, some models will not reach convergence in the minimization algorithm though, so it is advised to try multiple specifications. As an example, we will compare a standard GARCH(1,1) model to an eGARCH(1,1) specification:

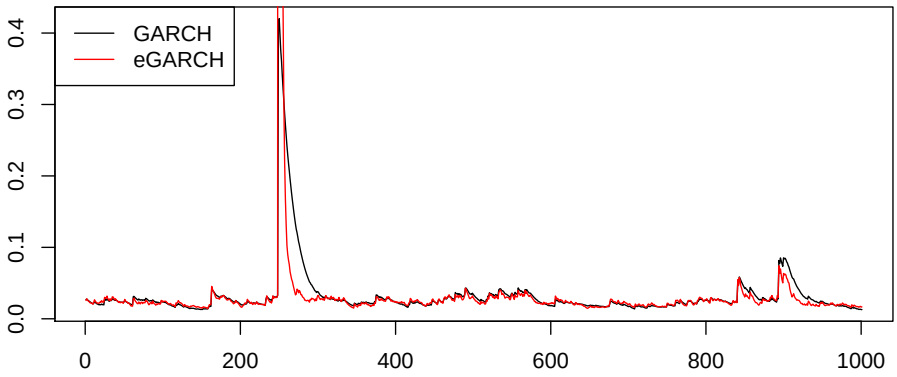
```
vol8 <- spotvol(sample_returns_5min, method = "garch", model = "sGARCH",
    solver.control = list(maxeval = 1000))

## Fitting sGARCH model...

vol9 <- spotvol(sample_returns_5min, method = "garch", model = "eGARCH",
    solver.control = list(maxeval = 1000))

## Fitting eGARCH model...

plot(as.numeric(t(vol8$spot))[6000:7000], type = "l")
lines(as.numeric(t(vol9$spot))[6000:7000], col = "red")
legend("topleft", c("GARCH", "eGARCH"), col = c("black", "red"),
    lty = 1)
```



This plot highlights the huge effect an outlier can have on the estimation. The GARCH models seem particulary susceptible to these, the eGARCH in this case even more so than the standard GARCH model.

References

Andersen, T. G. and T. Bollerslev (1997). Intraday periodicity and volatility persistence in financial markets. Journal of Empirical Finance 4, 115-158.

Beltratti, A. and C. Morana (2001). Deterministic and stochastic methods for estimation of intraday seasonal components with high frequency data. Economic Notes 30, 205-234.

Boudt K., Croux C. and Laurent S. (2011). Robust estimation of intraweek periodicity in volatility and jump detection. Journal of Empirical Finance 18, 353-367.

Fried, Roland (2012). On the online estimation of local constant volatilities. Computational Statistics and Data Analysis 56, 3080-3090.

Kristensen, Dennis (2010). Nonparametric filtering of the realized spot volatility: A kernel-based approach. Econometric Theory 26, 60-93.

Taylor, S. J. and X. Xu (1997). The incremental volatility information in one million foreign exchange quotations. Journal of Empirical Finance 4, 317-340.