# **quantmod** Charts Overview

Joshua M. Ulrich & ChuuHsiang Hung

August 21, 2016

**Abstract**

This vignette introduces the major update of **quantmod** for the GSoC "plot.xts for Performance and Risk" project. It gives a brief overview of the graphics and display wrapper functionality contained in quantmod including most of the charts. We develop the examples using randomly generated OHLCV data from **TTR**. Since S4 method for chartSeries has been deprecated, we will introduce the structure of new plot object and the relevant functions.

# Contents

**1 Introduction** **1**

**2 Set up quantmod** **1**
2.1 Install quantmod . . . . . 2
2.2 Load and review data . . . . . 2

**3 Create Charts** **2**
3.1 Create financial charts . . . . . 2
3.2 Add built-in indicators . . . . . 10
3.3 Manage chart colors . . . . . 10
3.4 Arrange technical indicators . . . . . 12
3.5 Add lines, points and shading regime . . . . . 17
3.6 Zoom chart . . . . . 18

**4 Conclusion** **18**

# 1 Introduction

`quantmod` is a library of functions designed for charting OHLC data and adding technical indicator functions in **TTR** to the chart. As of the developed version 0.4-7, we rewrite

chartSeries and its relevant functions based on plot.xts in Joshua's **xts**, developed version 0.10-0, to improve graphing primitives and convenience for in data exploration. New code is backwards compatible with interfaces of other functions in the corresponding packages such as **blotter** and **quantstrat**.

This vignette provides a demonstration of some of the capabilities of `quantmod`. We focus on the charting functionality but comment on the new structure along the way. These examples are not intended to be complete, but they should provide an indication of how to use the new chartSeries and create custom addTA functions.

# 2 Set up quantmod

These examples assume the reader has basic knowledge of R and understands how to install R, read and manipulate data, and create basic calculations. For further assistance, please see `http://cran.r-project.org/doc/manuals/R-intro.pdf` and other available materials at `http://cran.r-project.org`. This section will begin with installation, discuss the example data set, and provide an overview of charts attributes that will be used in the examples that follow.

## 2.1 Install quantmod

As of developed version 0.4-7 for GSoC project, quantmod is only available via GitHub, and the dependent xts with version 0.10-0 is needed to replicate the examples that follow. R users with connectivity need to install the latest `devtools` to install package from GitHub. Users can visit `https://github.com/hadley/devtools` to get more information. After it is installed, you can simply type:

```
> install_github('naturalsmen/quantmod', ref = 'design')
> install_github('joshuaulrich/xts') # install developed version of xts
```

A number of packages are required, including `xts`, `zoo`, `TTR` and the suggested packages `DBI`, `RMySQL`, `RSQLite`,`timeSeries`, `its`, `XML` and `downloader`. After installing `quantmod`, load it into your active R session using `library("quantmod")`.

```
> library('quantmod')
```

## 2.2 Load and review data

First we load the data used in all of the examples that follow. As you can see in Figure 1, `ttrc` is a xts object that contains columns of Open, High, Low, Close and Volume starting January 2, 1985 to December 31, 2006. Since there are 5550 observations in `ttrc`, we only select period from 1997 to 2001 with about 1000 observations.

Figure 1: First Lines of the ttrc

```
> data(ttrc, package="TTR")
> head(ttrc)

        Date Open High  Low Close  Volume
1 1985-01-02 3.18 3.18 3.08  3.08 1870906
2 1985-01-03 3.09 3.15 3.09  3.11 3099506
3 1985-01-04 3.11 3.12 3.08  3.09 2274157
4 1985-01-07 3.09 3.12 3.07  3.10 2086758
5 1985-01-08 3.10 3.12 3.08  3.11 2166348
6 1985-01-09 3.12 3.17 3.10  3.16 3441798

> ttrc <- xts(ttrc[,-1], ttrc[,1])
> class(ttrc)

[1] "xts" "zoo"

> ttrc <- ttrc["1997::2001"]
```

# 3 Create Charts

Since the appearance of charts are the same as the previous version, We will focus on introducing the new structure of chartSeries and the relevant functions with charts.

## 3.1 Create financial charts

candlesticks chart has been a main feature in quantmod. `chartSeries` is a charting tool designed for creating standard financial charts for xts-based object and can integrate with addTA functions to implement technical analysis. Possible chart styles include candles, matches (1 pixel candles), bars, and lines. In this chapter we introduce the charting functionality of `chartSeries` by different chart styles and subset period. We will use candlesticks chart for the examples in the later chapter.

### candlesticks

In Figure 2, we create a plot object `cs`. As you can see, `cs` is an environment which is different from the original S4 objects, `chob` and `chobTA` that have been deprecated. Users can get the names of `cs` to see functions used to manage the plot object. Then we can

Figure 2: Example of Candlesticks

```
> cs <- chartSeries(x=ttrc, type='candlesticks', theme=chartTheme('black'))
> class(cs)

[1] "replot_xts"  "environment"

> names(cs)

 [1] "get_frame"     "set_window"    "get_ylim"      "next_frame"
 [5] "set_ylim"      "replot"        "update_frames" "reset_ylim"
 [9] "set_asp"       "add_frame"     "get_asp"       "set_frame"
[13] "subset"        "get_actions"   "remove_frame"  "get_xlim"
[17] "Env"           "set_pad"       "set_xlim"      "add"

> cs
```

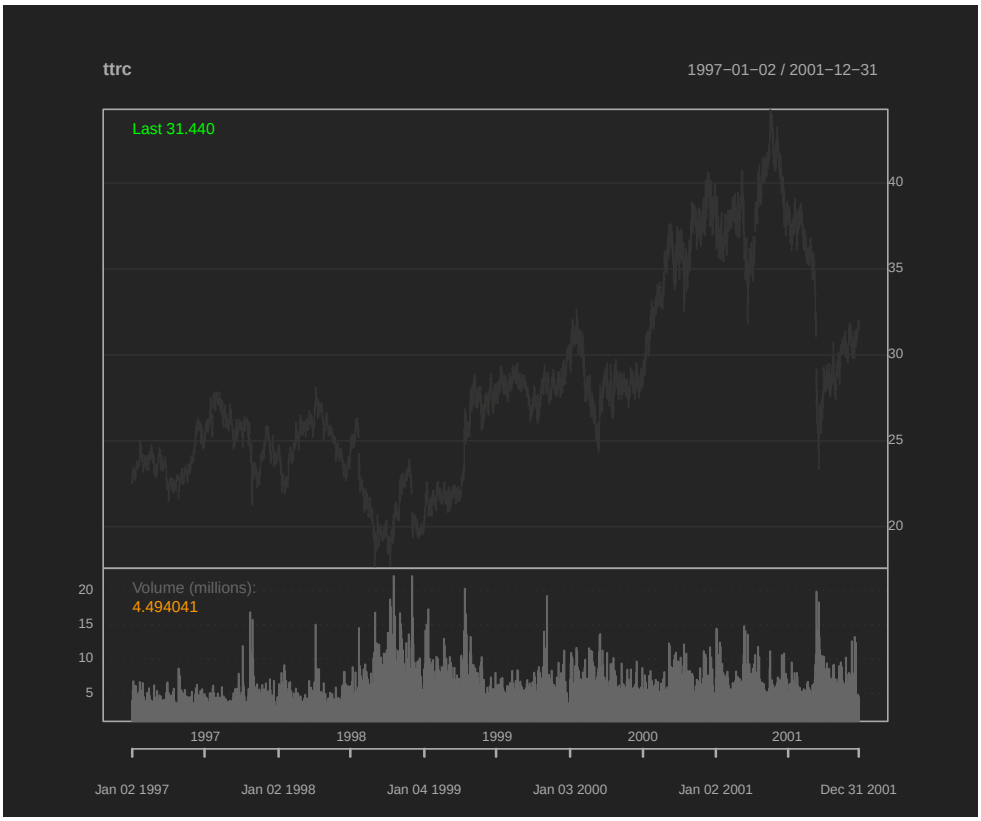


Figure 3: Example of Matchsticks

```
> cs <- chartSeries(x=ttrc, type='matchsticks', theme=chartTheme('black'))
> cs
```


see the candlesticks chart with theme 'black'. For 'candlesticks' and 'matchsticks', data need to be OHLC or OHLCV based xts object to create chart. The type of 'matchsticks creates 1 pixel candles. In default, 'green' and 'red' colors of the candlesticks represent price rising and price decreasing respectively. Users can manage the chart colors by calling `chartTheme`. We will introduce it in the later chapter.

### matchsticks

When series is greater than 300 days, the border color will block the color of price movement thus making chart unreadable. Users can set type to 'matchsticks' to draw 1 pixel candlesticks to avoid this issue or use the default setting. As you can see in Figure 3, the price movement is more readable with 1 pixel candlesticks for large series.

Figure 4: Example of Subset Series

```
> cs <- chartSeries(x=ttrc, subset="1998-06::1999-01")
> cs
```


### subset

Users are allowed to enlarge a certain period by specifying `subset` with insterested date. The date needs to be given in specific format. For an entire year or month, you can simply specify "yyyy" and "yyyy-mm". 1-day subset is not allowed here. As of the period that across the years or months, you can use "::" or "/" to separate two periods: "yyyy::yyyy" or "yyyy/yyyy". In Figure 4, we subset the period from June 1998 to January 1999 where two sudden drops in price occur. Users can also call `zoomChart` or `zooom` to view the subset series. We will introduce them in the later chapter

### multi.col

As showed in Figure 5. `chartSeries` also supports 4 color candle pattern to provide

Figure 5: Example of 4 Color Candle Pattern

```
> cs <- chartSeries(x=ttrc, subset="1998-06::1999-01", multi.col=TRUE)
> cs
```

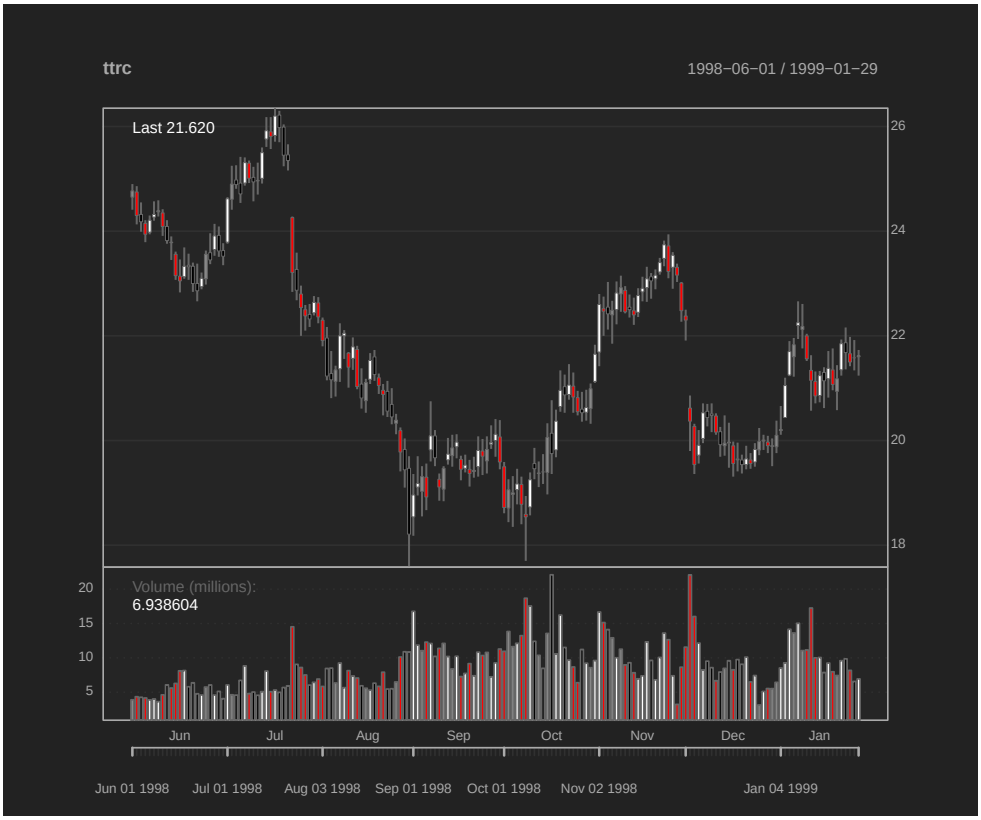
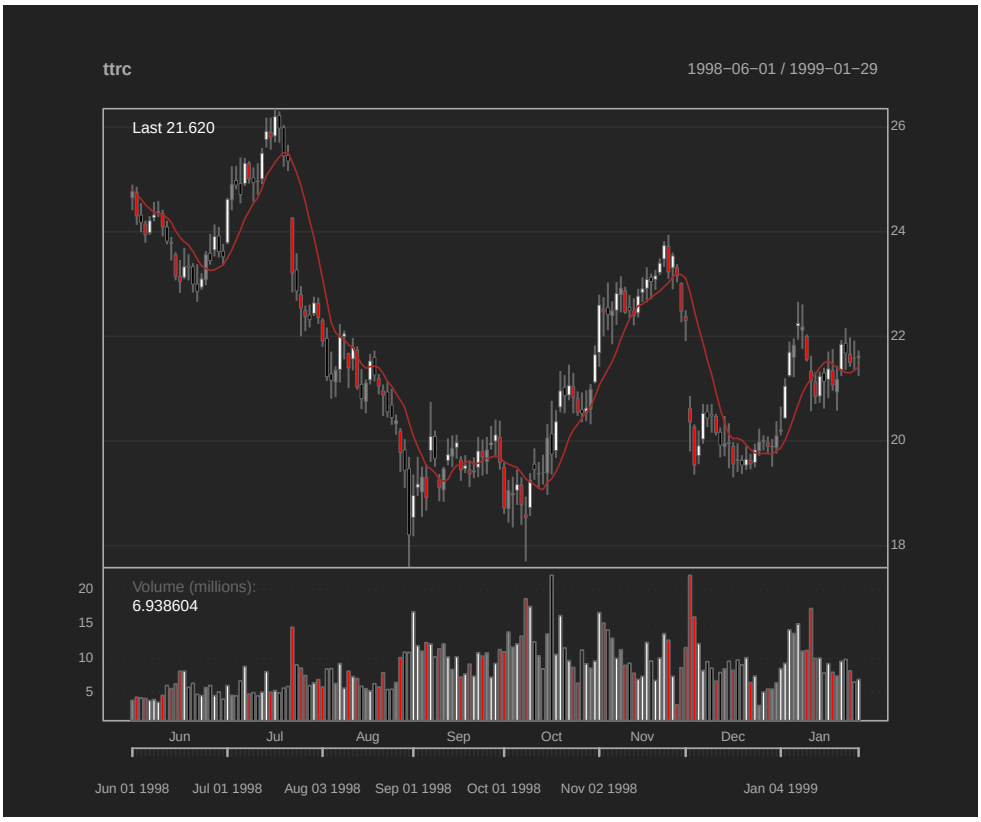


Figure 6: Example of Moving Averages

```
> cs <- addSMA(n=10)
> cs
```




further time-series data exploration. It implements a color coding scheme with grey, white, red and black colors. users can type `help(chartSeries)` to get more information about the rules.

To analyze the movement of the series, users can call addTA functions such as `addSMA`, `addRSI`, `addMACD`, etc., to add technical indicators to the chart. Those functions will first calculate the technical indicators from **TTR** and add the result series to the chart. In Figure 6, we add simple moving average to `cs`, the plot object we created ealier. As you can see, MA is added to the price panel. Users can choose if they want to draw MAs on a new panel by setting `overlay=FALSE` for various moving averages. Multiple numbers of periods to average over are also supported. In Figure 7, we add 5-day SMA to 10-day SMA to `cs`.

Figure 7: Example of Multiple Periods

```
> cs <- addSMA(n=10:15)
> cs
```

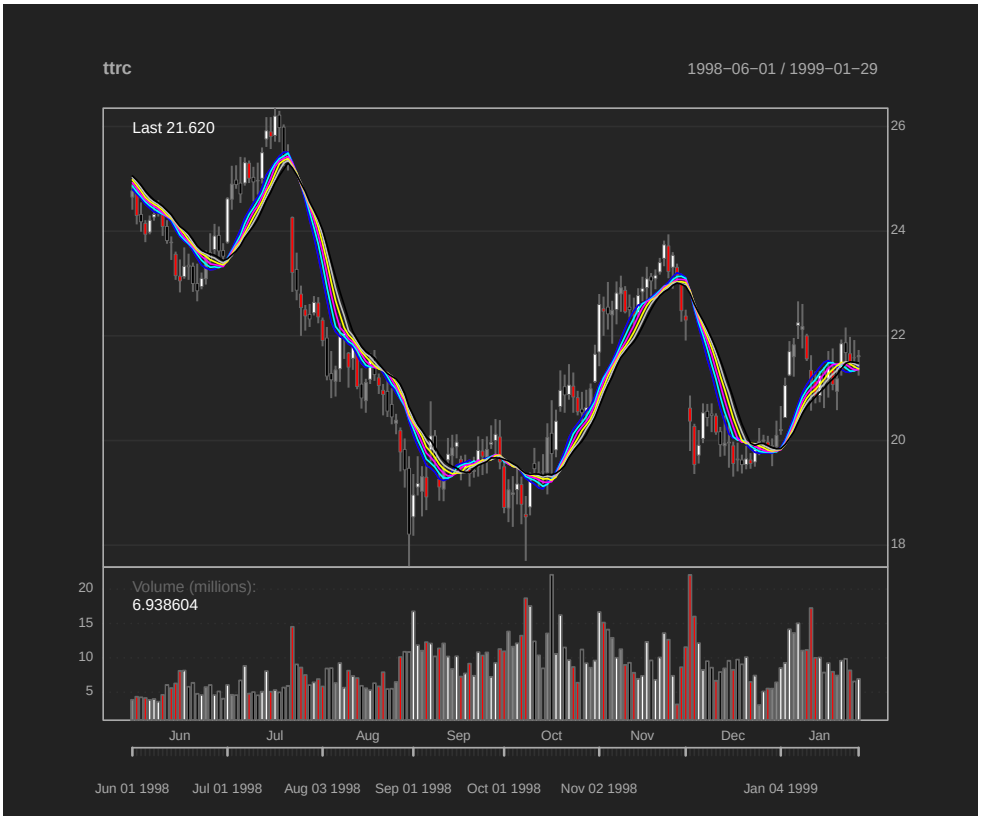


Figure 8: Call Indicators from chartSeries

```
> cs <- chartSeries(ttrc, TA=c("addRSI();addBBands()"), TAsep=";")
> cs
```


## 3.2 Add built-in indicators

Users can also add technical indicators by specifying addTA functions to `chartSeries`. In Figure 8, we add bollinger bands and relative strength index to the chart. As you can see, because the functions will be called inside of `chartSeries` by `eval()`, the specified TAs need to be in character. You can use different symbols to separate the functions. In this example, we use "semicolon" to divide `addRSI` and `addBBands` functions so we must specify `TAsep` with the corresponding symbol. Users can also specify a character vector or list to TA: `c("addRSI()", "addBBands()")` or `list("addRSI()", "addBBands()")`. By doing so, `sepTA` can be ignored.

## 3.3 Manage chart colors

Figure 9: Name of Themes and Color Palates

```
> theme.name <- names(.chart.theme)
> theme.name

[1] "white"      "white.mono" "black"      "black.mono" "beige"
[6] "wsj"

> col.palates <- chartTheme(theme.name[2])
> names(col.palates)

 [1] "fg.col"       "bg.col"       "grid.col"     "border"       "minor.tick"
 [6] "major.tick"   "up.col"       "dn.col"       "dn.up.col"    "up.up.col"
[11] "dn.dn.col"    "up.dn.col"    "up.border"    "dn.border"    "dn.up.border"
[16] "up.up.border" "dn.dn.border" "up.dn.border" "main.col"     "sub.col"
[21] "fill"         "Expiry"       "BBands.col"   "BBands.fill"  "BBands"
[26] "theme.name"

> head(col.palates, n=6L)

$fg.col
[1] "#666666"

$bg.col
[1] "#FFFFFF"

$grid.col
[1] "#CCCCCC"

$border
[1] "#666666"

$minor.tick
[1] "#CCCCCC"

$major.tick
[1] "#888888"
```

Except for the various moving average indicators which can control line colors by specifying `col`, users need to call `chartTheme` to do that. We have set up some specific color palattes designed to create readable line and bar graphs with sepcific objectives in `.chart.theme`. Figure 9 shows an example of the existing theme names and the correspondent color palates. As you can see, `col.palates` is a list with several elements to set up chart colors and has `chart.theme` class. Users can see parameters by entering `plotObject`$Env$`theme`$< TA >$`col`.

**theme**

Then we create candlesticks with specific theme name. There are color settings for Bollinger Bands in some themes already. Figure 10 shows candlesticks chart with Bollinger Bands, Relative Strength Index and Moving Average Convergence Divergence Indicator under theme 'beige' created by `chartTheme`. As you can see, the background color turns into light yellow color. Both bands and the middle moving average are in orange and green color, respectively.

Next we show how to set up custom colors for the technical indicators added to the chart in Figure 11. To compare to the Figure 10, we keep using theme 'beige' but change the color of RSI from blue to red. `chartTheme` creates custom colors for TAs and follows the following rules:

- chartTheme(. . . , add<TA.name>=list(col=list(. . . )))

Whether `col` should be added or not depends on the columns of TAs used. For example, Exponential Moving Average calculates the ease of movement values (emv) and the smoothed emv. So you should type:

- chartTheme(addEMV=list(col=list(emv, emvMA)))

Please type `demo(chartTheme)` to see the corresponding color names of addTA functions and the demonstration of `chartTheme` or type `help(chartTheme)` to get further information.

## 3.4 Arrange technical indicators

Since there are many technical indicators from **TTR**, it may be troublesome for users to replace an indicator with a new one to apply to the current chart or swap the locations of existing TAs. We provide users some functions to manipulate TAs directly without calling `chartSeries`. In this section, we will introduce the functions that allow users to see, drop and move TAs on the current chart.

Figure 10: Candlesticks Chart with 'beige' Theme

```
> cs.beige <- chartSeries(ttrc, theme=chartTheme('beige'))
> cs.beige <- addBBands()
> cs.beige <- addRSI()
> cs.beige
```

Figure 11: Example of Managing TA colors

```
> rsi.beige <- chartTheme('beige',
+                         addRSI=list(col=list(rsi='red', dot='white')))
> cs.rsi.beige <- chartSeries(ttrc, theme=rsi.beige)
> cs.rsi.beige <- addBBands()
> cs.rsi.beige <- addRSI()
> cs.rsi.beige
```

Figure 12: Example of list TA

```
> cs <- chartSeries(ttrc,
+                   TA=c("addVo();addMACD();addRSI();addBBands();
+                         addDEMA();addSMA()"),
+                   TAsep=";")
> head(listTA(chob=cs), n=3L)

[[1]]
addVo()

[[2]]
addMACD()

[[3]]
addRSI()

> cs
```

Figure 13: Example of TA Arrangement

```
> # drop Bollinger Bands
> cs <- dropTA(ta="BBands")
> # swap frames of volume and macd
> cs <- swapTA(ta1="Vo", ta2="MACD", occ1=1, occ2=1)
> # move SMA to be behind of DEMA
> cs <- moveTA(ta="SMA", pos=1)
> cs
```

Figure 14: Example of Series

```
> cs <- chartSeries(ttrc)
> cs <- addLines(x=Lo(ttrc["1998-06::1999-01"]), col="red")
> cs
```


First we create a chart with volume and several indicators, MACD, RSI, BBands, DEMA and SMA and show the existing TAs by `listTA` in Figure 12. It returns a list of function calls. To save spcae, we only show the first three elements.

Then we perform the arrangements to the current chart in Figure 13 and print the result.

## 3.5 Add lines, points and shading regime

To add more than just technical indicators, users can add series, points and shading area with `addLines`, `addPoints` and `addShading`, respectively. We show example of adding Low Price of `ttrc` to the chart. As you can see in Figure 14, specified series will be added

to the corresponding period. Horizontal and vertical lines are also supported.

As showed in Figure 15, users can add points with three ways depending on the value of `x` and `y`. First, if a xts object is passed to `addPoints`, it is similar to `addLines` but with points. The points will be added to the corresponding period automatically without specifying locations. Second, with location specified to `x` and value to `y`, the form is similar to `points` but `y` is required to be a xts object. Third, when y is `NULL`, points will be added to the desired period. Relying on the value of `offset`, when it is 1, Low Price of the series is drawn; when it is greater than 1, High Price is drawn.

To emphasize an event or a sudden drop in prce, we can add shading regime to the desired period. In Figure 16, shading area is added from August 2015 to September 2015 with the default fill color of BBands in "black" theme.

### 3.6 Zoom chart

As we mentioned in section Create charts, Figure 4, to view a subset period of a long term series you have two choices:

1. Specify a desired period to `subset` when `chartSeries` is called.

2. Call `zoomChart` or `zooom` functions.

Since we have introduced the first functionality earlier, here we make a demonstration for `zoomChart`. `zooom` is an interactive chart version of zoomChart which utilizes the standard `R` device interaction tool `locator` to estimate the desired subset. This estimate is then passed to zoomChart for actual redrawing. Because of its interactive functionality, we will not give an example here but users can type `demo(zoom)` to get further information.

`zoomChart` allows not only a desired range of periods but a statement which consists of three elements first/last, a number and periodicity. You can also zoom back to full data by `zoomChart()` as showed in Figure 17.

## 4 Conclusion

With that short overview of a few of the capabilities provided by new **quantmod**, we hope that the accompanying package and documentation will make it easier for users to understand the new structure of quantmod. If you think there's an important gap or possible improvement to be made, please don't hesitate to contact us.

Figure 15: Example of Points

```
> cs <- chartSeries(ttrc)
> # specify x with xts object
> cs <- addPoints(Lo(ttrc["2000"]))
> # specify both x and y, location and value, respectively
> cs <- addPoints(1:200, Cl(ttrc[1:200]), col="blue")
> # specify x with location, y=NULL
> cs <- addPoints(201:300, offset=1, col="pink")
> cs
```

Figure 16: Example of Shading Regime

```
> cs <- chartSeries(ttrc)
> cs <- addShading("1998-06::1999-01", col=cs$Env$theme$BBands$col$fill)
> cs
```

Figure 17: Example of zoomChart

```
> cs <- chartSeries(ttrc, TA=c("addVo()", "addMACD()"))
> zoomChart("last 3 months")
```