FRE7241 Algorithmic Portfolio Management

Lecture#3, Spring 2024

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

April 2, 2024

NYU | TANDON SCHOOL OF ENGINEERING

The Stop-loss Strategy

Stop-loss rules are used to reduce losses in case of a significant drawdown in prices.

For example, a simple stop-loss rule is to sell the stock if its price drops below the stop-loss level, equal to the stop-loss percentage times the previous maximum price.

The stock is bought back after the price recovers, for example after the price reaches its previous maximum price.

The stop-loss strategy trades a single $1 of stock, and is either long $1 of stock or it's flat ($0 of stock).

The stop-loss strategy is trend-following because it expects prices to continue dropping.

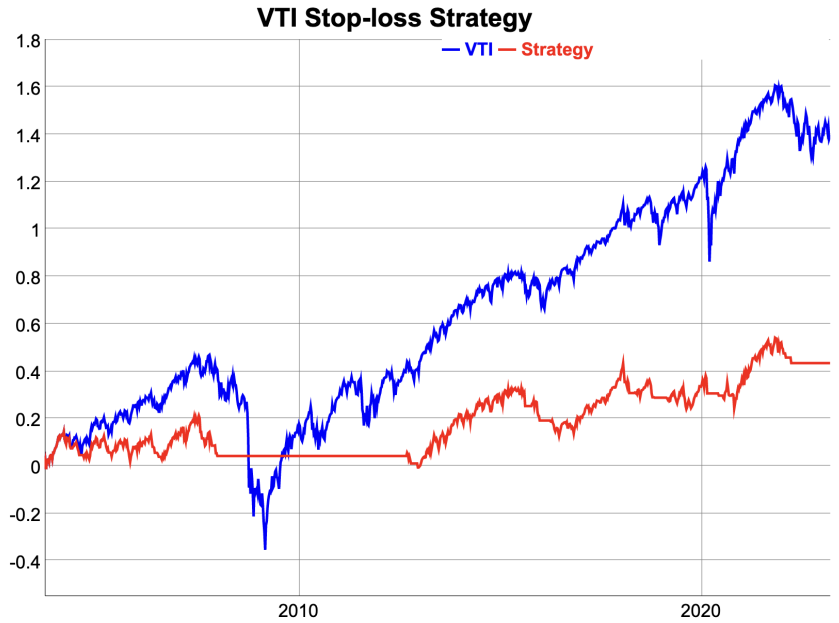
```
> # Calculate the VTI prices and returns
> pricev <- na.omit(rutils::etfenv$prices$VTI)
> nrows <- NROW(pricev)
> datev <- zoo::index(pricev)
> retp <- rutils::diffit(log(pricev))
> # Simulate stop-loss strategy
> stopl <- 0.05 # Stop-loss percentage
> pricem <- cummax(pricev) # Trailing maximum prices
> # Calculate the drawdown
> dd <- (pricev - pricem)
> pnls <- retp # Initialize PnLs
> for (i in 1:(nrows-1)) {
+ # Check for stop-loss
+   if (dd[i] < -stopl*pricem[i])
+     pnls[i+1] <- 0 # Set PnLs = 0 if in stop-loss
+ }  # end for
> # Same but without using loops in R
> pnls2 <- retp
> insl <- rutils::lagit(dd < -stopl*pricem)
> pnls2 <- ifelse(insl, 0, pnls2)
> all.equal(pnls, pnls2, check.attributes=FALSE)
```

Stop-loss Strategy Performance

The stop-loss strategy underperforms because it waits too long for prices to recover.

And the stop-loss strategy also underperforms because it gets *"whipsawed"* when prices are range-bound without a trend.

When prices are range-bound without a trend, the stop-loss strategy often stops because of a drawdown (goes flat risk), but if the prices soon rebound, then it's forced to buy back the stock. (This is called a *"whipsaw"*.)




```
> # Combine the data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot the stop-loss strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Stop-loss Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

```
> # Plot dygraph with shading
> # Create colors for background shading
> indic <- (rutils::diffit(insl) != 0) # Indices of stop-loss
> crossd <- c(datev[indic], datev[nrows]) # Dates of stop-loss
> shadev <- ifelse(insl[indic] == 1, "antiquewhite", "lightgreen")
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main="VTI Stop-loss Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], c
+ }  # end for
> # Plot the dygraph object
> dyplot
```

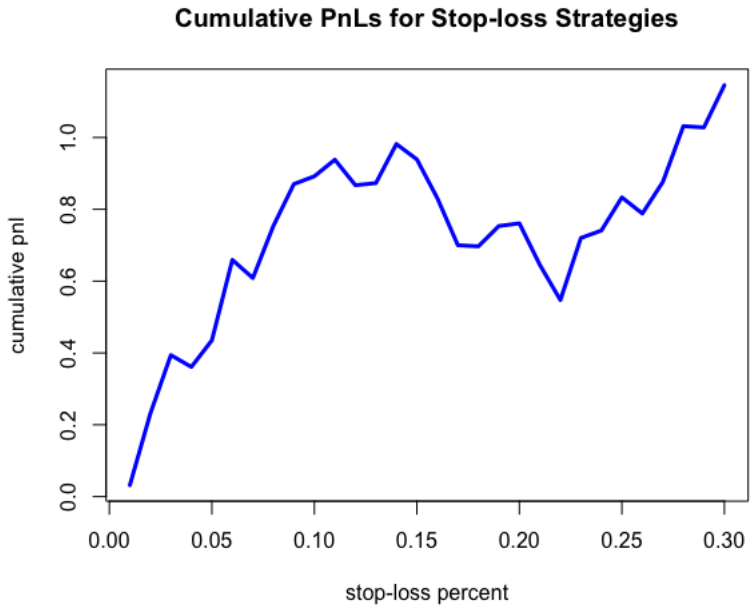
Optimal Stop-loss Strategy

Stop-loss rules can reduce the largest drawdowns but they also tend to reduce cumulative returns for stocks with good returns.

The best performing stop-loss strategy for *VTI* has the largest stop-loss percentage - i.e. it almost never enters into a stop-loss.

That's because *VTI* has had positive returns in the last 20 years, so a stop-loss rule had little benefit.

That's why many quantitative investment funds do not use stop-loss rules if they have strong convictions that the stocks will have positive returns.




```
> # Simulate multiple stop-loss strategies
> dd <- (pricev - pricem)
> stopv <- 0.01*(1:30)
> pnlc <- sapply(stopv, function(stopl) {
+   pnls <- retp
+   insl <- rutils::lagit(dd < -stopl*pricem)
+   pnls <- ifelse(insl, 0, pnls)
+   sum(pnls)
+ })  # end sapply
```

```
> # Plot cumulative pnls for stop-loss strategies
> plot(x=stopv, y=pnlc,
+    main="Cumulative PnLs for Stop-loss Strategies",
+    xlab="stop-loss percent", ylab="cumulative pnl",
+    t="l", lwd=3, col="blue")
```

Stop-Start Strategy

The stop-loss strategy can be improved by introducing a start-gain rule: buy back the stock if its price rebounds from the previous minimum and exceeds the start-gain level.

The stop-start strategy implements both stop-loss events due to price drawdowns and start-gain events due to price draw-ups.

In order to determine the stop-loss and start-gain events, the stop-start strategy follows the trailing maximum and trailing minimum prices.

A start-gain event is when the stock price rebounds from the previous minimum and exceeds the start-gain level, equal to the start-gain percentage times the previous minimum price.

After the start-gain level is crossed, the strategy buys back the stock which was sold under the stop-loss.

The stop-start strategy is trend-following because it profits if price trends are persistent. But it loses if prices are range-bound.

```
> # Define function for simulating a stop-start strategy
> sim_stopstart <- function(stopl) {
+   maxp <- pricev[1] # Trailing maximum price
+   minp <- pricev[1] # Trailing minimum price
+   insl <- FALSE # Is in stop-loss?
+   insg <- FALSE # Is in start-gain?
+   pnls <- retp # Initialize PnLs
+   for (i in 1:nrows) {
+     if (insl) { # In stop-loss
+ pnls[i] <- 0 # Set PnLs = 0 if in stop-loss
+ minp <- min(minp, pricev[i]) # Update minimum price to current pri
+ if (pricev[i] > ((1 + stopl)*minp)) { # Check for start-gain
+   insg <- TRUE # Is in start-gain?
+   insl <- FALSE # Is in stop-loss?
+   maxp <- pricev[i] # Reset trailing maximum price
+ }  # end if
+     } else if (insg) { # In start-gain
+ maxp <- max(maxp, pricev[i]) # Update maximum price to current pri
+ if (pricev[i] < ((1 - stopl)*maxp)) { # Check for stop-loss
+   insl <- TRUE # Is in stop-loss?
+   insg <- FALSE # Is in start-gain?
+   minp <- pricev[i] # Reset trailing minimum price
+ }  # end if
+     } else { # Warmup period
+ # Update the maximum and minimum prices
+ maxp <- max(maxp, pricev[i])
+ minp <- min(minp, pricev[i])
+ # Update the stop-loss and start-gain indicators
+ insl <- (pricev[i] < ((1 - stopl)*maxp)) # Is in stop-loss?
+ insg <- (pricev[i] > ((1 + stopl)*minp)) # Is in start-gain?
+     }  # end if
+   }  # end for
+   return(pnls)
+ } # end sim_stopstart
```

Stop-Start Strategy Performance

The stop-start strategy spends less time in a stop-loss because prices tend to rebound sharply after a steep loss.

The start-gain rule tends to quickly override the stop-loss rule, so that the strategy is long the stock after it recovers after a stop-loss.

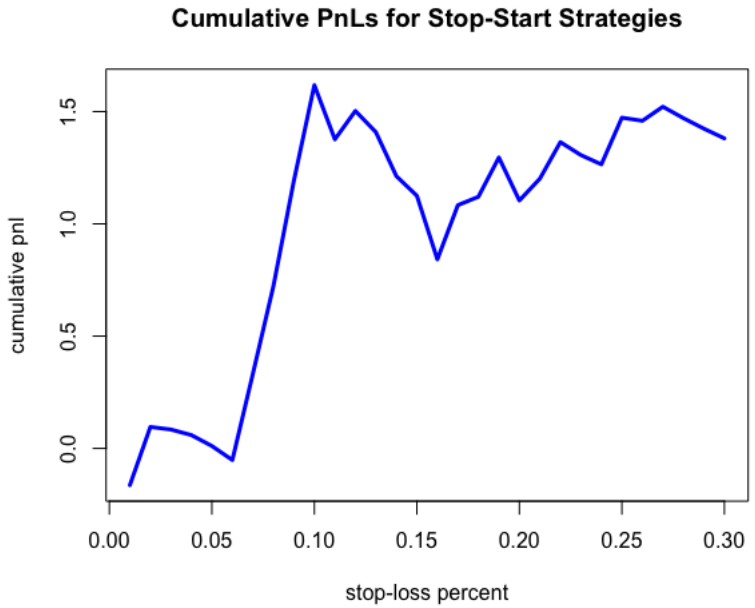
```
> # Simulate stop-start strategy
> pnls <- sim_stopstart(0.1)
> # Combine the data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("VTI", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot the stop-loss strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```


```
> # Plot dygraph with shading
> # Create colors for background shading
> insl <- (pnls == 0) # Is in stop-loss?
> indic <- (rutils::diffit(insl) != 0) # Indices of crosses
> crossd <- c(datev[indic], datev[nrows]) # Dates of crosses
> shadev <- ifelse(insl[indic] == 1, "antiquewhite", "lightgreen")
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main="VTI Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], c
+ }  # end for
> # Plot the dygraph object
> dyplot
```

Optimal Stop-Start Strategy

The stop-start strategy performs much better than the stop-loss strategy because it captures stock gains.

```
> # Simulate multiple stop-loss strategies
> stopv <- 0.01*(1:30)
> pnlc <- sapply(stopv, function(stopl) {
+   sum(sim_stopstart(stopl))
+ })  # end sapply
> stopl <- stopv[which.max(pnlc)]
```

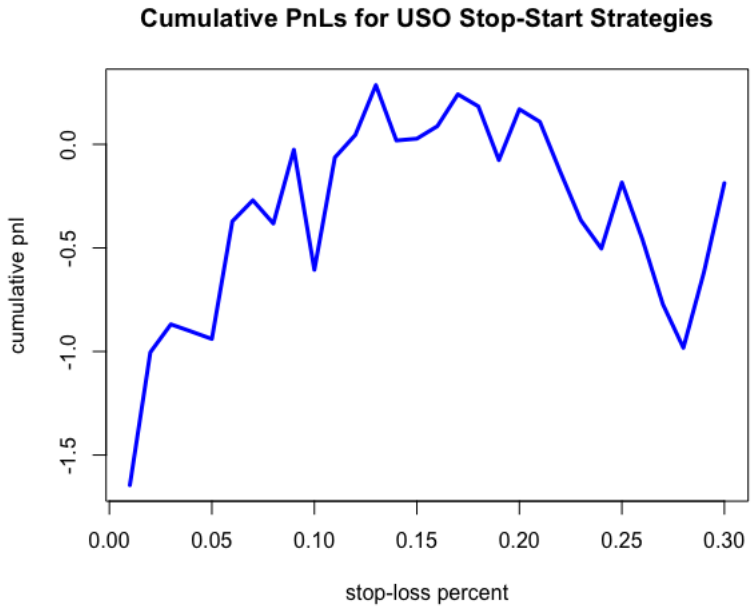



```
> # Plot cumulative pnls for stop-loss strategies
> plot(x=stopv, y=pnlc,
+    main="Cumulative PnLs for Stop-Start Strategies",
+    xlab="stop-loss percent", ylab="cumulative pnl",
+    t="l", lwd=3, col="blue")
```

Stop-Start Strategy for Other ETFs

The stop-start strategy can prevent losses for stocks with significant negative returns, like the *USO* ETF (oil fund).

```
> # Calculate the USO prices and returns
> pricev <- na.omit(rutils::etfenv$prices$USO)
> nrows <- NROW(pricev)
> datev <- zoo::index(pricev)
> retp <- rutils::diffit(log(pricev))
> # Simulate multiple stop-start strategies
> stopv <- 0.01*(1:30)
> pnlc <- sapply(stopv, function(stopl) {
+   sum(sim_stopstart(stopl))
+ })  # end sapply
```




```
> # Plot cumulative pnls for stop-start strategies
> plot(x=stopv, y=pnlc,
+    main="Cumulative PnLs for USO Stop-Start Strategies",
+    xlab="stop-loss percent", ylab="cumulative pnl",
+    t="l", lwd=3, col="blue")
```

Optimal Stop-Start Strategy For *USO*

The stop-start strategy for the *USO* ETF has performed well because *USO* has had very negative returns in the last 20 years.

Stop-start strategies are able to preserve profits for the best performing stocks, and avoid losses for the worst performing stocks.

```
> # Simulate optimal stop-start strategy for USO
> stopl <- stopv[which.max(pnlc)]
> pnls <- sim_stopstart(stopl)
> # Combine the data
> wealthv <- cbind(retp, pnls)
> colnamev <- c("USO", "Strategy")
> colnames(wealthv) <- colnamev
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # dygraph plot the stop-start strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="USO Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```


```
> # Plot dygraph with shading
> # Create colors for background shading
> insl <- (pnls == 0) # Is in stop-loss?
> indic <- (rutils::diffit(insl) != 0) # Indices of crosses
> crossd <- c(datev[indic], datev[nrows]) # Dates of crosses
> shadev <- ifelse(insl[indic] == 1, "antiquewhite", "lightgreen")
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main="USO Stop-Start Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], c
+ }  # end for
> # Plot the dygraph object
> dyplot
```

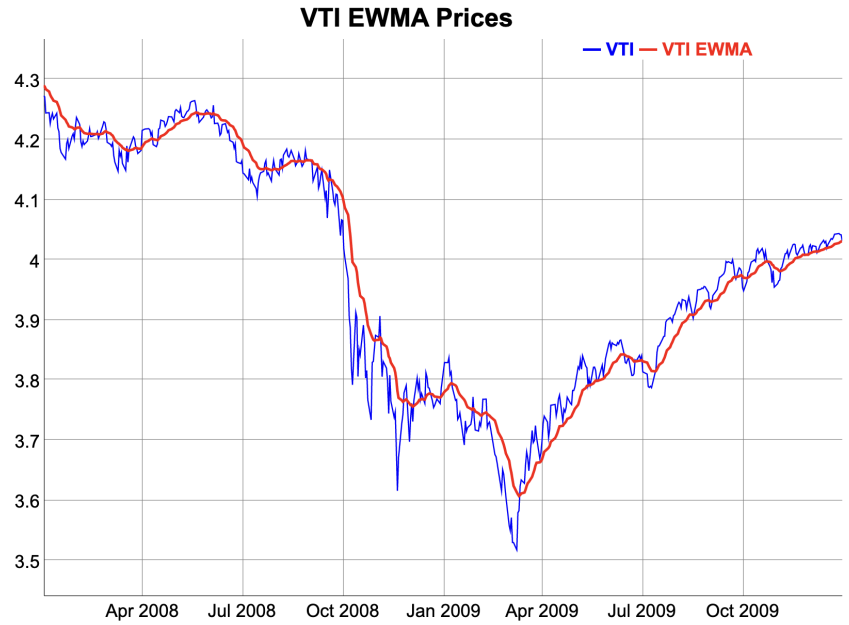
EMA Price Technical Indicator

The *Exponentially Weighted Moving Average Price* (*EMA*) is defined as the weighted average of prices over a rolling interval:

$$p_i^{EMA} = (1 - \lambda) \sum_{j=0}^{\infty} \lambda^j p_{i-j}$$

Where the decay factor λ determines the rate of decay of the *EMA* weights, with smaller values of λ producing faster decay, giving more weight to recent prices, and vice versa.

The function `HighFreq::roll_wsum()` calculates the convolution of a time series with a vector of weights.




```
> # Extract the log VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> closep <- quantmod::Cl(ohlc)
> colnames(closep) <- "VTI"
> nrows <- NROW(closep)
> # Calculate the EMA weights
> lookb <- 111
> lambda <- 0.9
> weightv <- lambda^(0:lookb)
> weightv <- weightv/sum(weightv)
> # Calculate the EMA prices as a convolution
> emacpp <- HighFreq::roll_sumw(closep, weightv=weightv)
> pricev <- cbind(closep, emacpp)
> colnames(pricev) <- c("VTI", "VTI EMA")
```

```
> # Dygraphs plot with custom line colors
> colnamev <- colnames(pricev)
> dygraphs::dygraph(pricev["2008/2009"], main="VTI EMA Prices") %>%
+   dySeries(name=colnamev[1], strokeWidth=1, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, col="red") %>%
+   dyLegend(show="always", width=200)
> # Standard plot of  EMA prices with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> colorv <- c("blue", "red")
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pricev["2009"], theme=plot_theme,
+        lwd=2, name="VTI EMA Prices")
> legend("topleft", legend=colnames(pricev), y.intersp=0.5,
+  inset=0.1, bg="white", lty=1, lwd=6, cex=0.8,
+  col=plot_theme$col$line.col, bty="n")
```

Recursive EMA Price Indicator

The *EMA* prices can be calculated recursively as follows:

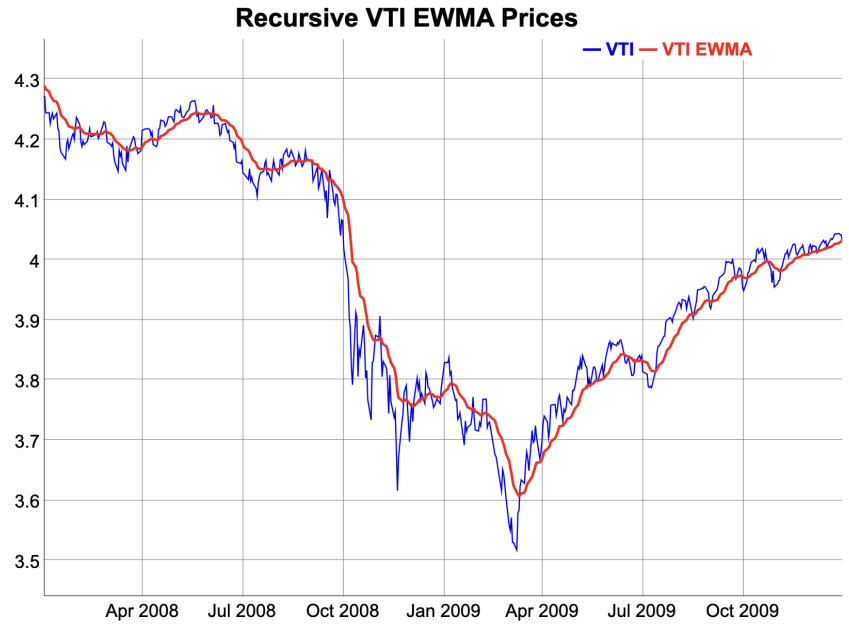
$$p_i^{EMA} = (1 - \lambda)p_i + \lambda p_{i-1}^{EMA}$$

Where the decay factor λ determines the rate of decay of the *EMA* weights, with smaller values of λ producing faster decay, giving more weight to recent prices, and vice versa.

The recursive *EMA* prices are slightly different from those calculated as a convolution, because the convolution uses a fixed look-back interval.

The compiled C++ function `stats:::C_rfilter()` calculates the exponentially weighted moving average prices recursively.

The function `HighFreq::run_mean()` calculates the exponentially weighted moving average prices recursively.




```
> # Calculate the EMA prices recursively using C++ code
> emar <- .Call(stats:::C_rfilter, closep, lambda, c(as.numeric(cl
> # Or R code
> # emar <- filter(closep, filter=lambda, init=as.numeric(closep[1
> emar <- (1-lambda)*emar
> # Calculate the EMA prices recursively using RcppArmadillo
> emacpp <- HighFreq::run_mean(closep, lambda=lambda)
> all.equal(drop(emacpp), emar)
> # Compare the speed of C++ code with RcppArmadillo
> library(microbenchmark)
> summary(microbenchmark(
+   Rcpp=HighFreq::run_mean(closep, lambda=lambda),
+   rfilter=.Call(stats:::C_rfilter, closep, lambda, c(as.numeric(
```

```
> # Dygraphs plot with custom line colors
> pricev <- cbind(closep, emacpp)
> colnames(pricev) <- c("VTI", "VTI EMA")
> colnamev <- colnames(pricev)
> dygraphs::dygraph(pricev["2008/2009"], main="Recursive VTI EMA Pri
+   dySeries(name=colnamev[1], strokeWidth=1, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, col="red") %>%
+   dyLegend(show="always", width=200)
> # Standard plot of  EMA prices with custom line colors
> plot_theme <- chart_theme()
> colorv <- c("blue", "red")
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pricev["2009"], theme=plot_theme,
+        lwd=2, name="VTI EMA Prices")
> legend("topleft", legend=colnames(pricev), y.intersp=0.5,
+  inset=0.1, bg="white", lty=1, lwd=6, cex=0.8,
+  col=plot_theme$col$line.col, bty="n")
```

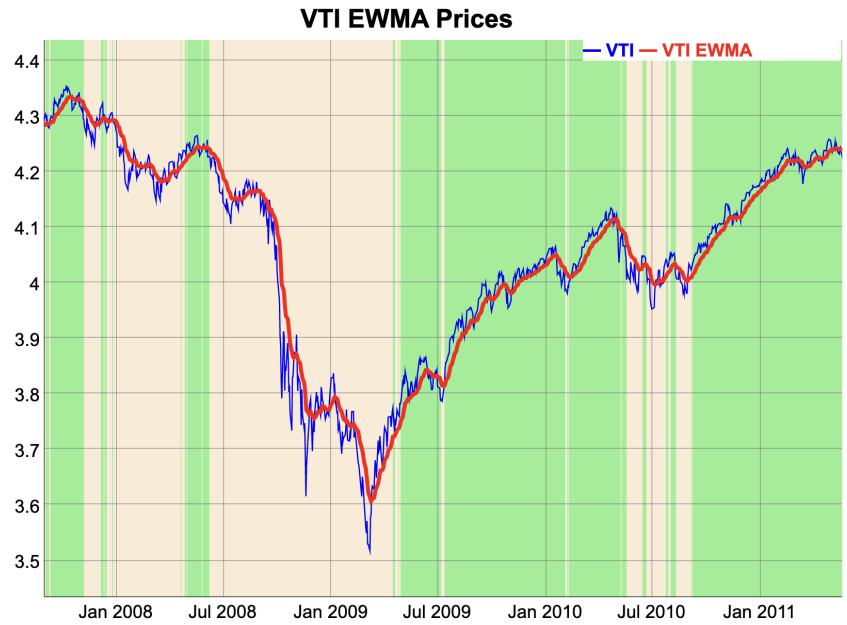
The EMA Crossover Strategy

The trend following *EMA Crossover* strategy switches its stock position depending if the current price is above or below the *EMA*.

If the stock price is above the *EMA* price, then the strategy switches to long $1 dollar of stock, and if it is below, to short $1 dollar of stock.

The strategy holds the same position until the *EMA* crosses over the current price (either from above or below), and then it switches its position.

The strategy is therefore always either long $1 dollar of stock or short $1 dollar of stock.




```
> # Calculate the EMA prices recursively using C++ code
> lambda <- 0.984
> emacpp <- HighFreq::run_mean(closep, lambda=lambda)
> # Calculate the positions, either: -1, 0, or 1
> indic <- sign(closep - emacpp)
> posv <- rutils::lagit(indic, lagg=1)
> # Create colors for background shading
> crossd <- (rutils::diffit(posv) != 0)
> shadev <- posv[crossd]
> crossd <- c(zoo::index(shadev), end(posv))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen",
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(pricev, main="VTI EMA Prices") %>%
+   dySeries(name=colnamev[1], strokeWidth=1, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=3, col="red") %>%
+   dyLegend(show="always", width=200)
```

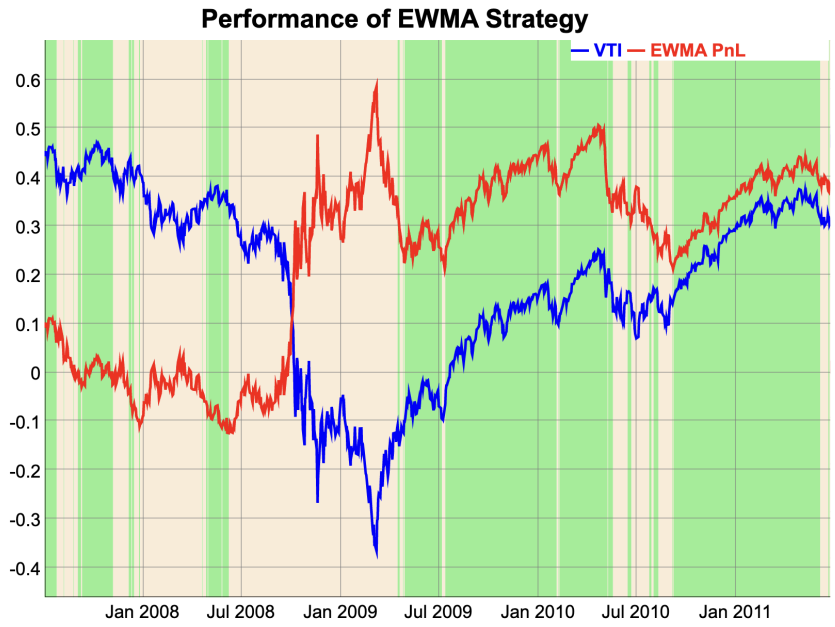
```
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], c
+ }  # end for
> # Plot the dygraph object
> dyplot
> # Standard plot of EMA prices with position shading
> quantmod::chart_Series(pricev, theme=plot_theme,
+        lwd=2, name="VTI EMA Prices")
> add_TA(posv > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posv < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("topleft", legend=colnames(pricev),
+  inset=0.1, bg="white", lty=1, lwd=6, y.intersp=0.5,
+  col=plot_theme$col$line.col, bty="n")
```

EMA Crossover Strategy Performance

The crossover strategy trades at the *Close* price on the same day that prices cross the *EMA*, which may be difficult in practice.

The crossover strategy performance is worse than the underlying asset (*VTI*), but it has a negative correlation to it, which is very valuable when building a portfolio.

```
> # Calculate the daily profits and losses of EMA strategy
> retp <- rutils::diffit(closep)  # VTI returns
> pnls <- retp*posv
> colnames(pnls) <- "EMA"
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "EMA PnL")
> # Annualized Sharpe ratio of EMA strategy
> sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
> # Plot dygraph of EMA strategy wealth
> # Create dygraph object without plotting it
> colorv <- c("blue", "red")
> dyplot <- dygraphs::dygraph(cumsum(wealthv), main="Performance o
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>%
+     dyShading(from=crossd[i], to=crossd[i+1], color=shadev[i])
+ }  # end for
> # Plot the dygraph object
> dyplot
```



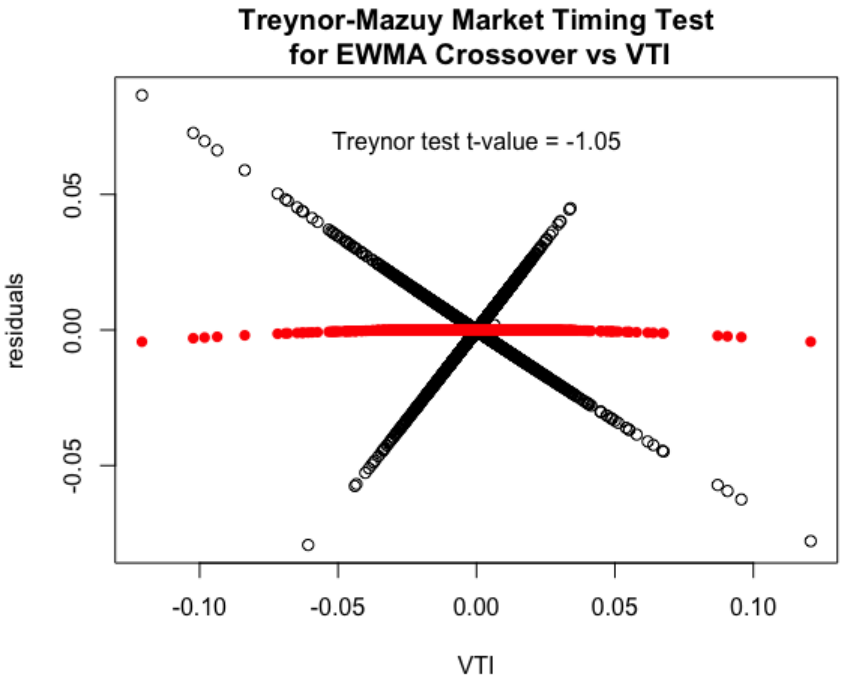

```
> # Standard plot of EMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+         name="Performance of EMA Strategy")
> add_TA(posv > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posv < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("top", legend=colnames(wealthv), y.intersp=0.5,
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

EMA Crossover Strategy Market Timing Skill

The EMA crossover strategy shorts the market during significant selloffs, but otherwise doesn't display market timing skill.

The t-value of the *Treynor-Mazuy* test is negative, but not statistically significant.

```
> # Test EMA crossover market timing of VTI using Treynor-Mazuy test
> desm <- cbind(pnls, retp, retp^2)
> desm <- na.omit(desm)
> colnames(desm) <- c("EMA", "VTI", "treynor")
> regmod <- lm(EMA ~ VTI + treynor, data=desm)
> summary(regmod)
> # Plot residual scatterplot
> resids <- (desm$EMA - regmod$coeff["VTI"]*retp)
> resids <- regmod$residuals
> plot.default(x=retp, y=resids, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for EMA Crossover
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retp
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retp, y=fitv, pch=16, col="red")
> text(x=0.0, y=0.8*max(resids), paste("Treynor test t-value =", tvalue))
```

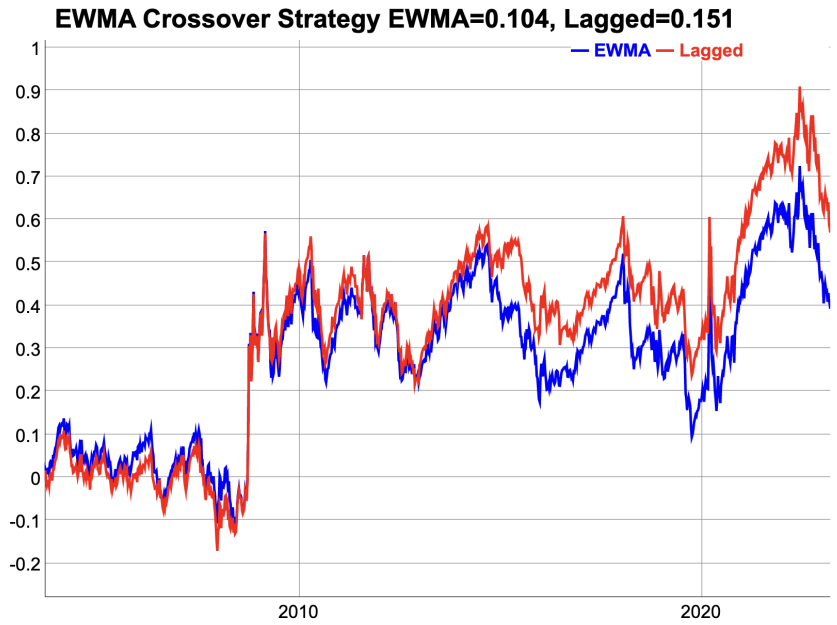


EMA Crossover Strategy With Lag

The crossover strategy suffers losses when prices are range-bound without a trend, because whenever it switches position the prices soon change direction. (This is called a *"whipsaw"*.)

To prevent whipsaws and over-trading, the crossover strategy may choose to delay switching positions until the indicator repeats the same value for several periods.

There's a tradeoff between switching positions too early and risking a whipsaw, and waiting too long and missing an emerging trend.




```
> # Determine trade dates right after EMA has crossed prices
> indic <- sign(closep - emacpp)
> # Calculate the positions from lagged indicator
> lagg <- 2
> indic <- HighFreq::roll_sum(indic, lagg)
> # Calculate the positions, either: -1, 0, or 1
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(indic == lagg, 1, posv)
> posv <- ifelse(indic == (-lagg), -1, posv)
> posv <- zoo::na.locf(posv, na.rm=FALSE)
> posv <- xts::xts(posv, order.by=zoo::index(closep))
> # Lag the positions to trade in next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the PnLs of lagged strategy
> pnlslag <- retp*posv
> colnames(pnlslag) <- "Lagged Strategy"
```

```
> wealthv <- cbind(pnls, pnlslag)
> colnames(wealthv) <- c("EMA", "Lagged")
> # Annualized Sharpe ratios of EMA strategies
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # Plot both strategies
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main=paste("EMA Crossover
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

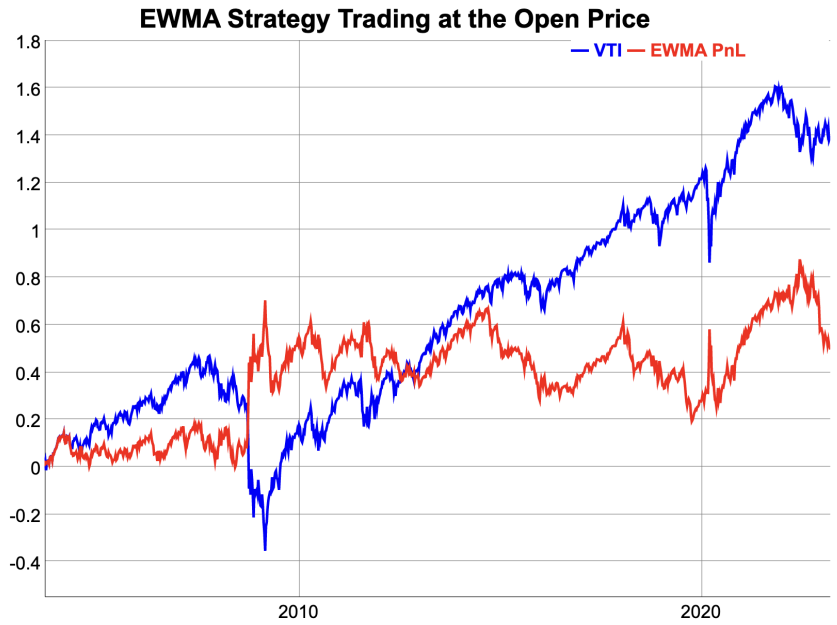
EMA Strategy Trading at the Open Price

In practice it may not be possible to trade immediately at the *Close* price on the same day that prices cross the *EMA*.

Then the strategy may trade at the *Open* price on the next day.

The Profit and Loss (*PnL*) on a trade date is the sum of the realized *PnL* from closing the old position, plus the unrealized *PnL* after opening the new position.

```
> # Calculate the positions, either: -1, 0, or 1
> indic <- sign(closep - emacpp)
> posv <- rutils::lagit(indic, lagg=1)
> # Calculate the daily pnl for days without trades
> pnls <- retp*posv
> # Determine trade dates right after EMA has crossed prices
> crossd <- which(rutils::diffit(posv) != 0)
> # Calculate the realized pnl for days with trades
> openp <- quantmod::Op(ohlc)
> closelag <- rutils::lagit(closep)
> poslag <- rutils::lagit(posv)
> pnls[crossd] <- poslag[crossd]*(openp[crossd] - closelag[crossd])
> # Calculate the unrealized pnl for days with trades
> pnls[crossd] <- pnls[crossd] +
+   posv[crossd]*(closep[crossd] - openp[crossd])
> # Calculate the wealth
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "EMA PnL")
> # Annualized Sharpe ratio of EMA strategy
> sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
```




```
> # Plot dygraph of EMA strategy wealth
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="EMA Strategy Tradi
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # Standard plot of EMA strategy wealth
> quantmod::chart_Series(cumsum(wealthv)[endd], theme=plot_theme,
+        name="EMA Strategy Trading at the Open Price")
> legend("top", legend=colnames(wealthv),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

EMA Crossover Strategy With Transaction Costs

The *bid-ask spread* is the percentage difference between the *ask* (offer) minus the *bid* prices, divided by the *mid* price.

The bid-ask spread for many liquid ETFs is about 1 basis point. For example the *XLK ETF*

Let n_t be the number of shares of the stock owned at time t, and let p_t be their price.

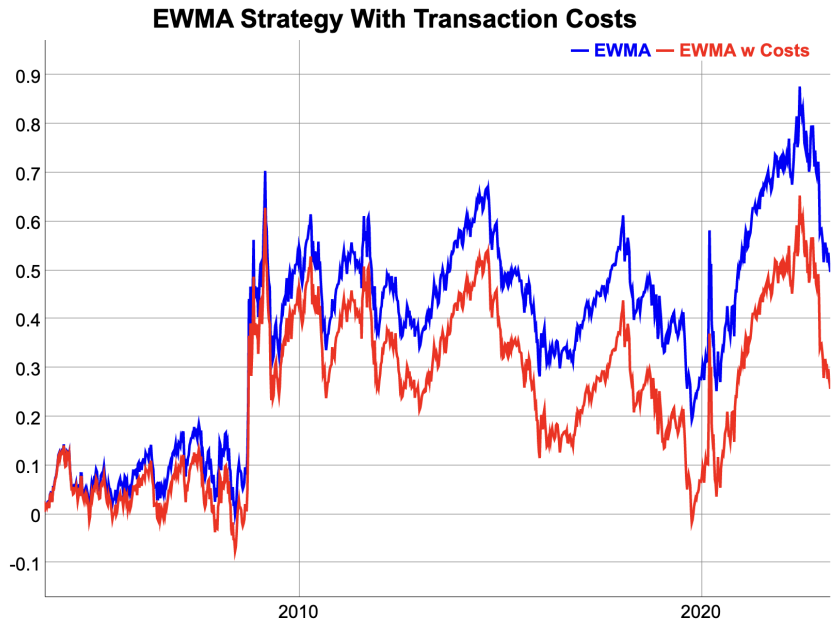
Then the traded dollar amount of the stock is equal to the change in the number of shares times the stock price: $\Delta n_t p_t$.

The the *transaction costs* c^r due to the *bid-ask spread* are equal to half the *bid-ask spread* δ times the absolute value of the traded dollar amount of the stock:

$$c^r = \frac{\delta}{2} |\Delta n_t| \, p_t$$

If d_t is the dollar amount of the stock owned at time t then the *transaction costs* c^r are equal to:

$$c^r = \frac{\delta}{2} |\Delta d_t|$$




```
> # bidask equal to 1 bp for liquid ETFs
> bidask <- 0.001
> # Calculate the transaction costs
> costs <- 0.5*bidask*abs(poslag - posv)
> # Plot strategy with transaction costs
> wealthv <- cbind(pnls, pnls - costs)
> colnames(wealthv) <- c("EMA", "EMA w Costs")
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="EMA Strategy With T
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Simulation Function for EMA Crossover Strategy

The *EMA* strategy can be simulated by a single function, which allows the analysis of its performance depending on its parameters.

The function `sim_ema()` performs a simulation of the *EMA* strategy, given an *OHLC* time series of prices, and a decay factor λ.

The function `sim_ema()` returns the *EMA* strategy positions and returns, in a two-column *xts* time series.

```
> sim_ema <- function(ohlc, lambda=0.9, lookb=333, bidask=0.001,
+                     trend=1, lagg=1) {
+   closep <- quantmod::Cl(ohlc)
+   retp <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate the EMA prices
+   emacpp <- HighFreq::run_mean(closep, lambda=lambda)
+   # Calculate the indicator
+   indic <- trend*sign(closep - emacpp)
+   if (lagg > 1) {
+     indic <- HighFreq::roll_sum(indic, lagg)
+     indic[1:lagg] <- 0
+   }  # end if
+   # Calculate the positions, either: -1, 0, or 1
+   posv <- rep(NA_integer_, nrows)
+   posv[1] <- 0
+   posv <- ifelse(indic == lagg, 1, posv)
+   posv <- ifelse(indic == (-lagg), -1, posv)
+   posv <- zoo::na.locf(posv, na.rm=FALSE)
+   posv <- xts::xts(posv, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posv <- rutils::lagit(posv, lagg=1)
+   # Calculate the PnLs of strategy
+   pnls <- retp*posv
+   costs <- 0.5*bidask*abs(rutils::diffit(posv))
+   pnls <- (pnls - costs)
+   # Calculate the strategy returns
+   pnls <- cbind(posv, pnls)
+   colnames(pnls) <- c("positions", "pnls")
+   pnls
+ }  # end sim_ema
```

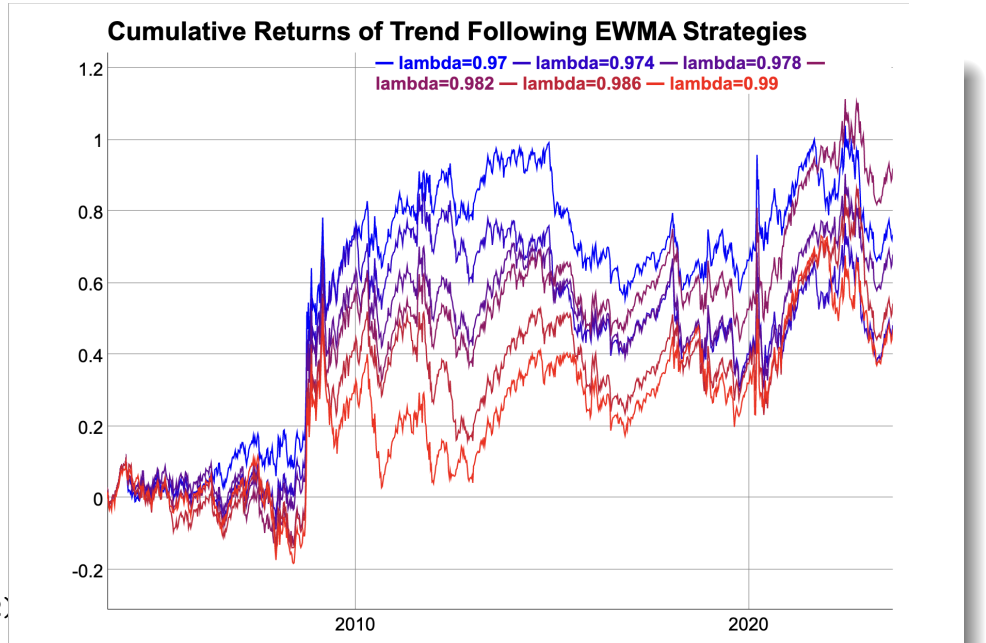
Simulating Multiple Trend Following EMA Strategies

Multiple *EMA* strategies can be simulated by calling the function `sim_ema()` in a loop over a vector of λ parameters.

But `sim_ema()` returns an *xts* time series, and `sapply()` cannot merge *xts* time series together.

So instead the loop is performed using `lapply()` which returns a list of *xts*, and the list is merged into a single *xts* using the functions `do.call()` and `cbind()`.

```
> lambdav <- seq(from=0.97, to=0.99, by=0.004)
> # Perform lapply() loop over lambdav
> pnltrend <- lapply(lambdav, function(lambda) {
+   # Simulate EMA strategy and Calculate the returns
+   sim_ema(ohlc=ohlc, lambda=lambda, lookb=lookb, bidask=0, lagg=2)
+ })  # end lapply
> pnltrend <- do.call(cbind, pnltrend)
> colnames(pnltrend) <- paste0("lambda=", lambdav)
```




```
> # Plot dygraph of multiple EMA strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnltrend))
> endd <- rutils::calc_endpoints(pnltrend, interval="weeks")
> dygraphs::dygraph(cumsum(pnltrend)[endd], main="Cumulative Returns
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=400)
> # Plot EMA strategies with custom line colors
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(pnltrend), theme=plot_theme,
+   name="Cumulative Returns of EMA Strategies")
> legend("topleft", legend=colnames(pnltrend), inset=0.1,
+   bg="white", cex=0.8, lwd=rep(6, NCOL(pnltrend)),
+   col=plot_theme$col$line.col, bty="n")
```

Simulating EMA Strategies Using Parallel Computing

Simulating *EMA* strategies naturally lends itself to parallel computing, since the simulations are independent from each other.

The function `parLapply()` is similar to `lapply()`, and performs loops under *Windows* using parallel computing on several CPU cores.

The resulting list of time series can then be collapsed into a single *xts* series using the functions `rutils::do_call()` and `cbind()`.

```
> # Initialize compute cluster under Windows
> library(parallel)
> ncores <- detectCores() - 1  # Number of cores
> compclust <- makeCluster(detectCores()-1)
> clusterExport(compclust,
+   varlist=c("ohlc", "lookb", "sim_ema"))
> # Perform parallel loop over lambdav under Windows
> pnltrend <- parLapply(compclust, lambdav, function(lambda) {
+   library(quantmod)
+   # Simulate EMA strategy and Calculate the returns
+   sim_ema(ohlc=ohlc, lambda=lambda, lookb=lookb)[, "pnls"]
+ })  # end parLapply
> stretduster(compclust)  # Stop R processes over cluster under Wind
> # Perform parallel loop over lambdav under Mac-OSX or Linux
> pnltrend <- mclapply(lambdav, function(lambda) {
+   library(quantmod)
+   # Simulate EMA strategy and Calculate the returns
+   sim_ema(ohlc=ohlc, lambda=lambda, lookb=lookb)[, "pnls"]
+ }, mc.cores=ncores)  # end mclapply
> pnltrend <- do.call(cbind, pnltrend)
> colnames(pnltrend) <- paste0("lambda=", lambdav)
```

Optimal Decay Factor of Trend Following EMA Strategies

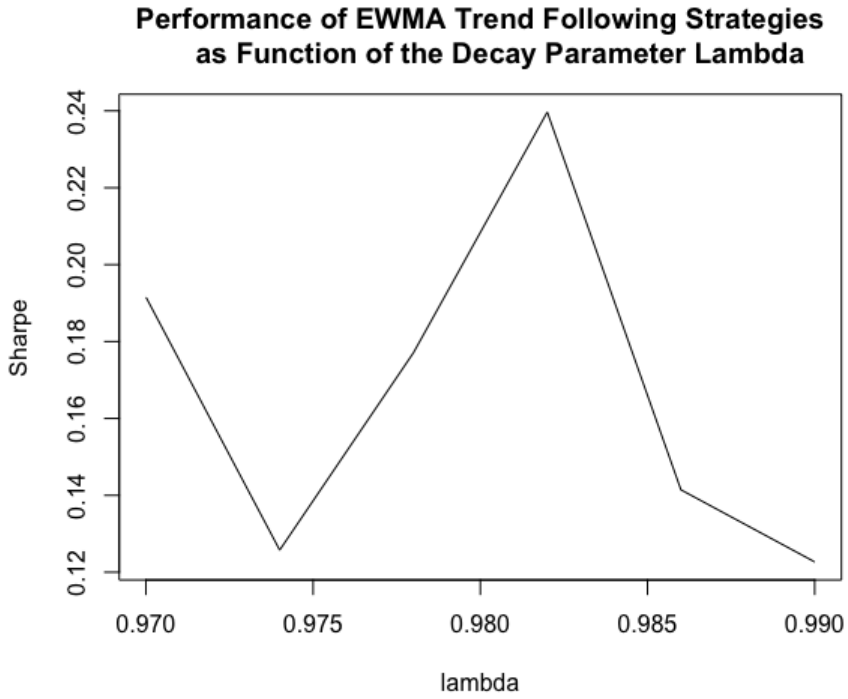
The performance of trend following *EMA* strategies depends on the λ decay factor, with smaller λ parameters performing better than larger ones.

The optimal λ parameter applies significant weight to returns $8 - 12$ months in the past, which is consistent with research on trend following strategies.

The *Sharpe ratios* of *EMA* strategies with different λ parameters can be calculated by performing an `sapply()` loop over the *columns* of returns.

`sapply()` treats the columns of *xts* time series as list elements, and loops over the columns.

Performing loops in `R` over the *columns* of returns is acceptable, but `R` loops over the *rows* of returns should be avoided.

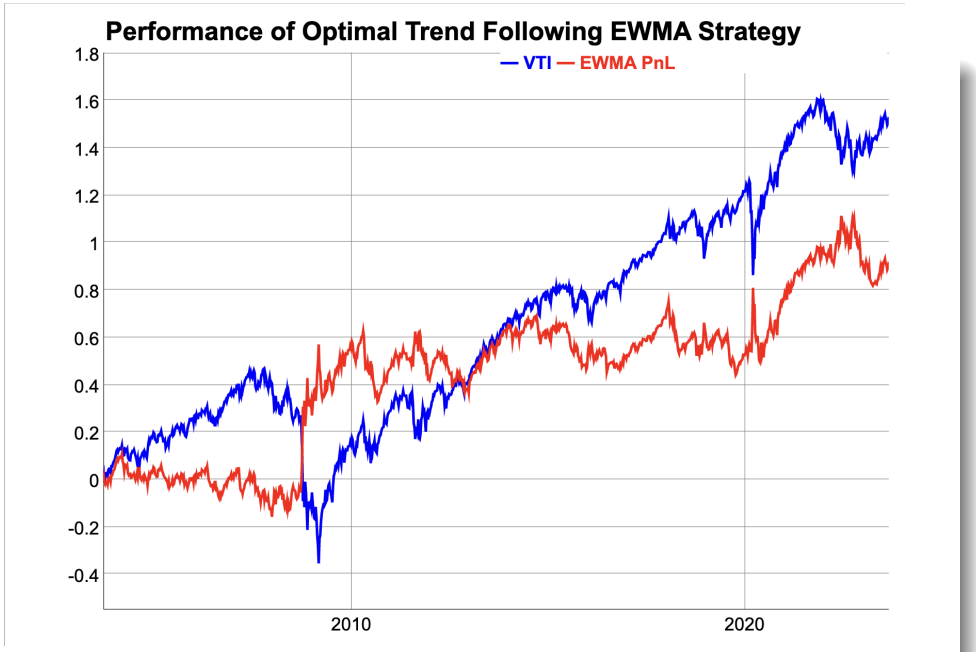



```
> # Calculate the annualized Sharpe ratios of strategy returns
> sharpetrend <- sqrt(252)*sapply(pnltrend, function(xtsv) {
+   mean(xtsv)/sd(xtsv)
+ })  # end sapply
> # Plot Sharpe ratios
> dev.new(width=6, height=5, noRStudioGD=TRUE)
> plot(x=lambdav, y=sharpetrend, t="l",
+      xlab="lambda", ylab="Sharpe",
+      main="Performance of EMA Trend Following Strategies
+      as Function of the Decay Factor Lambda")
```

Optimal Trend Following EMA Strategy

The best performing trend following *EMA* strategy has a relatively small λ parameter, corresponding to slower weight decay (giving more weight to past pricev), and producing less frequent trading.

```
> # Calculate the optimal lambda
> lambda <- lambdav[which.max(sharpetrend)]
> # Simulate best performing strategy
> ematrend <- sim_ema(ohlc=ohlc, lambda=lambda, bidask=0, lagg=2)
> posv <- ematrend[, "positions"]
> trendopt <- ematrend[, "pnls"]
> wealthv <- cbind(retp, trendopt)
> colnames(wealthv) <- c("VTI", "EMA PnL")
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> cor(wealthv)[1, 2]
> # Plot dygraph of EMA strategy wealth
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Performance of Opt
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```



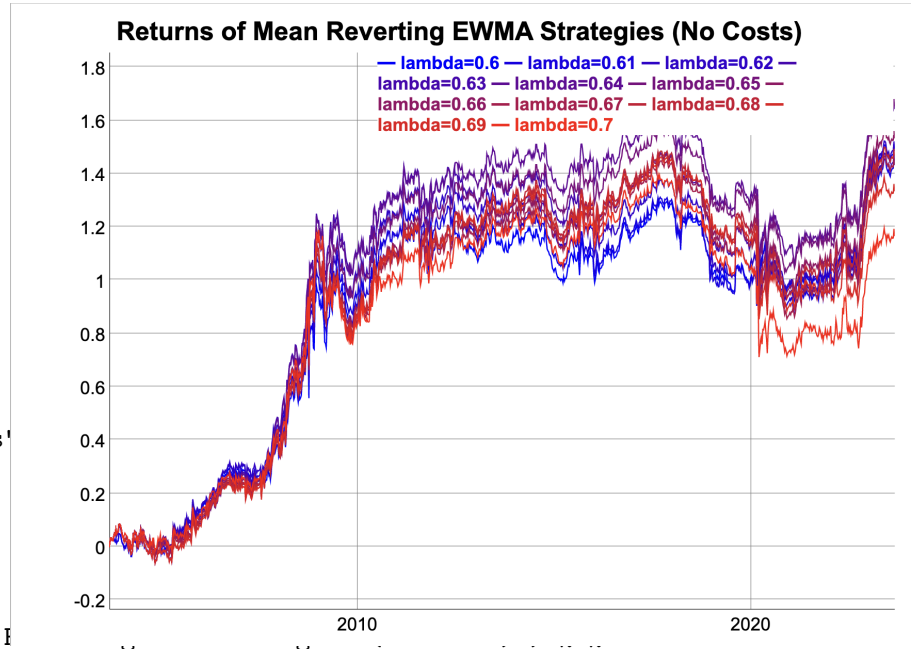

```
> # Plot EMA PnL with position shading
> # Standard plot of EMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+        name="Performance of EMA Strategy")
> add_TA(posv > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posv < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("top", legend=colnames(wealthv),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

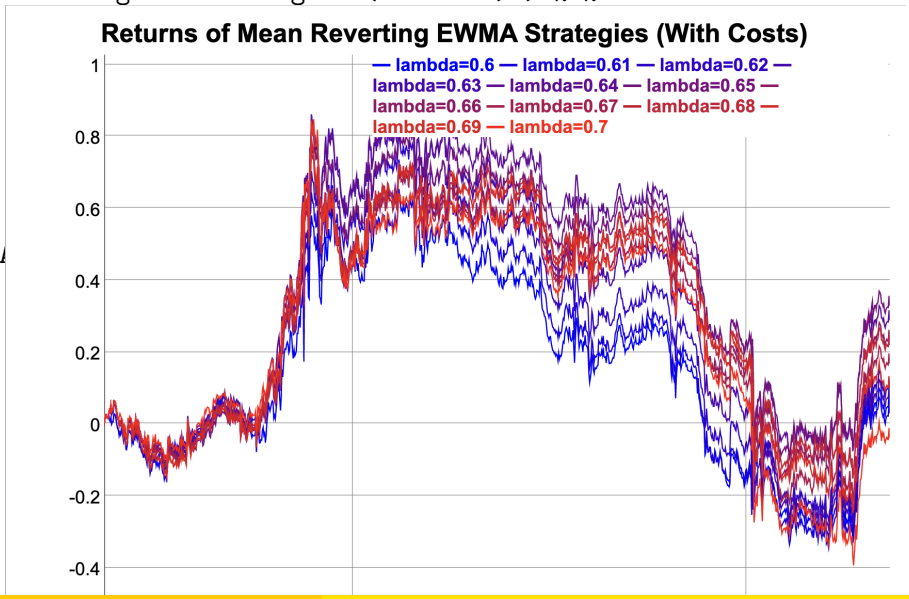
Mean Reverting EMA Crossover Strategies

Mean reverting EMA crossover strategies can be simulated using function `sim_ema()` with argument `trend=(-1)`.

The profitability of mean reverting strategies can be significantly improved by using limit orders, to reduce transaction costs.

```
> lambdav <- seq(0.6, 0.7, 0.01)
> # Perform lapply() loop over lambdav
> pnlrevert <- lapply(lambdav, function(lambda) {
+   # Simulate EMA strategy and Calculate the returns
+   sim_ema(ohlc=ohlc, lambda=lambda, bidask=0, trend=(-1))[, "pnls'
+ })  # end lapply
> pnlrevert <- do.call(cbind, pnlrevert)
> colnames(pnlrevert) <- paste0("lambda=", lambdav)
> # Plot dygraph of mean reverting EMA strategies
> colorv <- colorRampPalette(c("blue", "red"))(NROW(lambdav))
> dygraphs::dygraph(cumsum(pnlrevert)[endd], main="Returns of Mean R
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=400)
> # Plot EMA strategies with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pnlrevert,
+   theme=plot_theme, name="Cumulative Returns of Mean Reverting EMA
> legend("topleft", legend=colnames(pnlrevert),
+   inset=0.1, bg="white", cex=0.8, lwd=6,
+   col=plot_theme$col$line.col, bty="n")
```





Performance of Mean Reverting EMA Strategies

The *Sharpe ratios* of *EMA* strategies with different λ parameters can be calculated by performing an `sapply()` loop over the *columns* of returns.

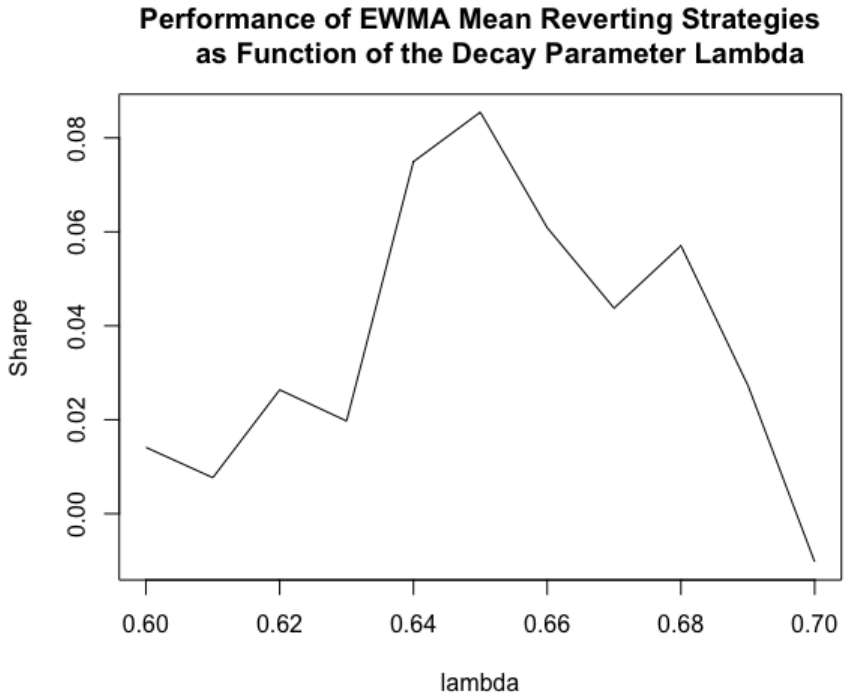
`sapply()` treats the columns of *xts* time series as list elements, and loops over the columns.

Performing loops in `R` over the *columns* of returns is acceptable, but `R` loops over the *rows* of returns should be avoided.

The performance of mean reverting *EMA* strategies depends on the λ parameter, with performance decreasing for very small or very large λ parameters.

For too large λ parameters, the trading frequency is too high, causing high transaction costs.

For too small λ parameters, the trading frequency is too low, causing the strategy to miss profitable trades.



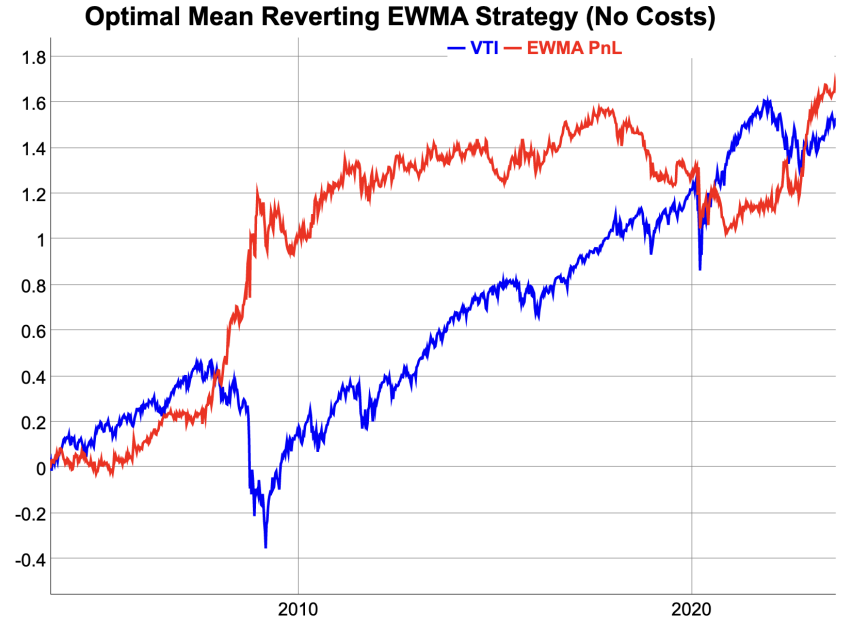

```
> # Calculate the Sharpe ratios of strategy returns
> sharperevert <- sqrt(252)*sapply(pnlrevert, function(xtsv) {
+   mean(xtsv)/sd(xtsv)
+ })  # end sapply
> plot(x=lambdav, y=sharperevert, t="l",
+      xlab="lambda", ylab="Sharpe",
+      main="Performance of EMA Mean Reverting Strategies
+      as Function of the Decay Factor Lambda")
```

Optimal Mean Reverting EMA Strategy

Reverting the direction of the trend following *EMA* strategy creates a mean reverting strategy.

The best performing mean reverting *EMA* strategy has a relatively large λ parameter, corresponding to faster weight decay (giving more weight to recent prices), and producing more frequent trading.

But a too large λ parameter also causes very high trading frequency, and high transaction costs.




```
> # Calculate the optimal lambda
> lambda <- lambdav[which.max(sharperevert)]
> # Simulate best performing strategy
> emarevert <- sim_ema(ohlc=ohlc, bidask=0.0,
+   lambda=lambda, trend=(-1))
> posv <- emarevert[, "positions"]
> revertopt <- emarevert[, "pnls"]
> wealthv <- cbind(retp, revertopt)
> colnames(wealthv) <- c("VTI", "EMA PnL")
> # Plot dygraph of EMA strategy wealth
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Optimal Mean Reve
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

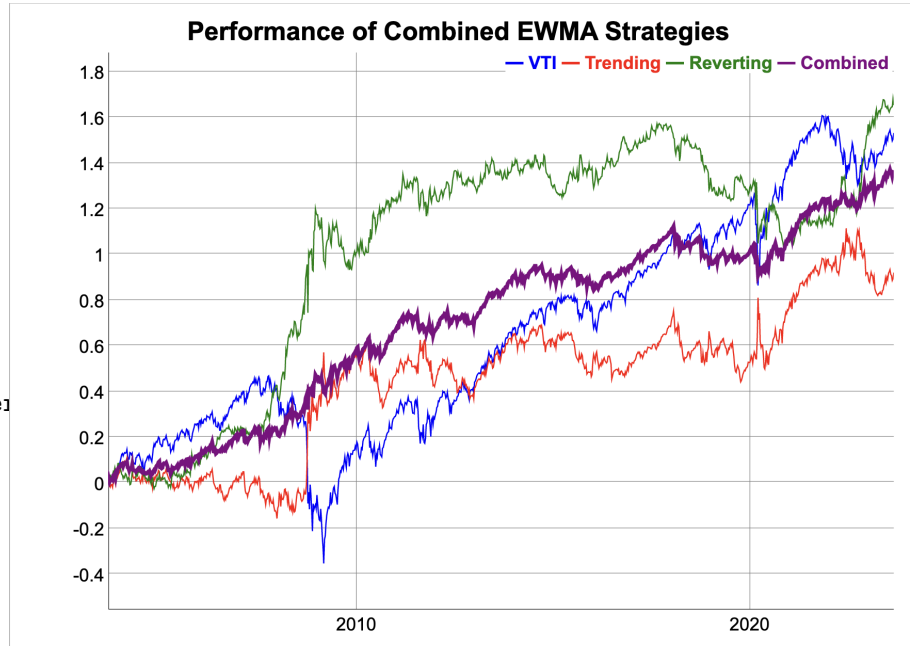
```
> # Standard plot of EMA strategy wealth
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(wealthv), theme=plot_theme,
+        name="Optimal Mean Reverting EMA Strategy")
> add_TA(posv > 0, on=-1, col="lightgreen", border="lightgreen")
> add_TA(posv < 0, on=-1, col="lightgrey", border="lightgrey")
> legend("top", legend=colnames(wealthv),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

Combining Trend Following and Mean Reverting Strategies

The returns of trend following and mean reverting strategies are usually negatively correlated to each other, so combining them can achieve significant diversification of risk.

The main advantage of EMA crossover strategies is that they provide positive returns and a diversification of risk with respect to static stock portfolios.

```
> # Calculate the correlation between trend following and mean rever
> trendopt <- ematrend[, "pnls"]
> colnames(trendopt) <- "trend"
> revertopt <- emarevert[, "pnls"]
> colnames(revertopt) <- "revert"
> cor(cbind(retp, trendopt, revertopt))
> # Calculate the combined strategy
> combstrat <- (retp + trendopt + revertopt)/3
> colnames(combstrat) <- "combined"
> # Calculate the annualized Sharpe ratio of strategy returns
> retc <- cbind(retp, trendopt, revertopt, combstrat)
> colnames(retc) <- c("VTI", "Trending", "Reverting", "Combined")
> sqrt(252)*sapply(retc, function(xtsv) mean(xtsv)/sd(xtsv))
```

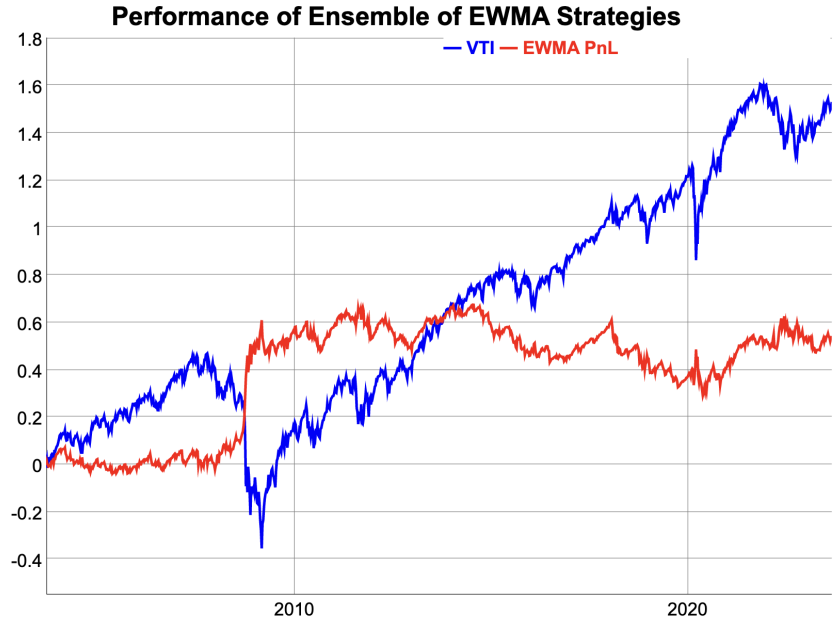



```
> # Plot dygraph of EMA strategy wealth
> colorv <- c("blue", "red", "green", "purple")
> dygraphs::dygraph(cumsum(retc)[endd], main="Performance of Combine
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=200)
> # Standard plot of EMA strategy wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(pnls, theme=plot_theme,
+        name="Performance of Combined EMA Strategies")
> legend("topleft", legend=colnames(pnls),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

Ensemble of EMA Strategies

Instead of selecting the best performing *EMA* strategy, one can choose a weighted average of strategies (ensemble), which corresponds to allocating positions according to the weights.

The weights can be chosen to be proportional to the Sharpe ratios of the *EMA* strategies.



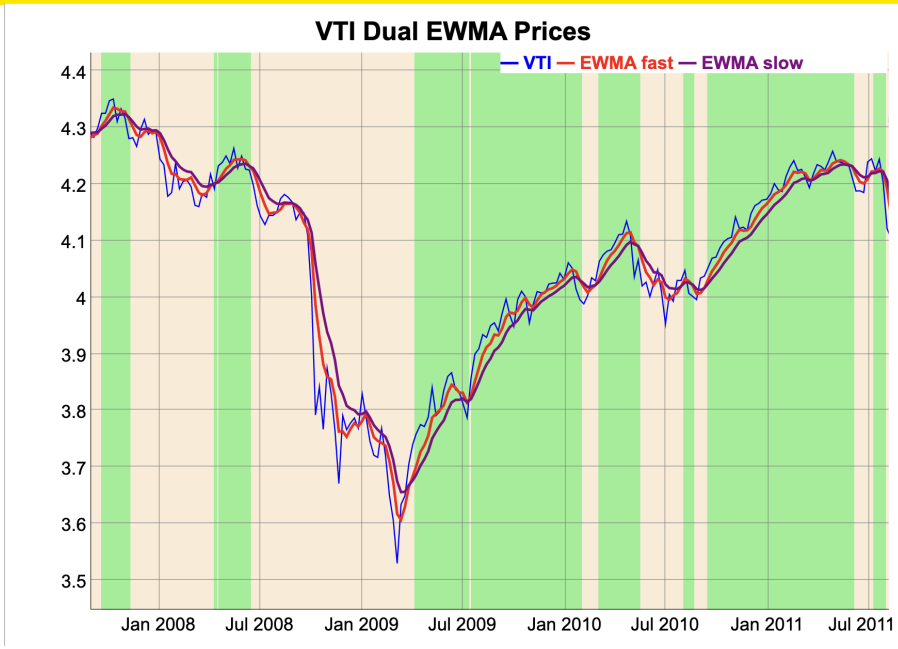

```
> # Calculate the weights proportional to Sharpe ratios
> weightv <- c(sharpetrend, sharperevert)
> weightv[weightv < 0] <- 0
> weightv <- weightv/sum(weightv)
> retc <- cbind(pnltrend, pnlrevert)
> retc <- retc %*% weightv
> retc <- cbind(retp, retc)
> colnames(retc) <- c("VTI", "EMA PnL")
> # Plot dygraph of EMA strategy wealth
> colorv <- c("blue", "red")
> dygraphs::dygraph(cumsum(retc)[endd], main="Performance of Ensemble of EMA Strategies") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # Standard plot of EMA strategy wealth
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(retc), theme=plot_theme,
+        name="Performance of Ensemble of EMA Strategies")
> legend("topleft", legend=colnames(pnls),
+  inset=0.05, bg="white", lty=1, lwd=6,
+  col=plot_theme$col$line.col, bty="n")
```

Simulating the Dual EMA Crossover Strategy

In the *Dual EMA Crossover* strategy, the stock position depends on the difference between two moving averages.

The stock position flips when the fast moving *EMA* crosses the slow moving *EMA*.

```
> # Calculate the fast and slow EMAs
> lambdaf <- 0.89
> lambdas <- 0.95
> # Calculate the EMA prices
> emaf <- HighFreq::run_mean(closep, lambda=lambdaf)
> emas <- HighFreq::run_mean(closep, lambda=lambdas)
> # Calculate the EMA prices
> pricev <- cbind(closep, emaf, emas)
> colnames(pricev) <- c("VTI", "EMA fast", "EMA slow")
> # Calculate the positions, either: -1, 0, or 1
> indic <- sign(emaf - emas)
> lagg <- 2
> indic <- HighFreq::roll_sum(indic, lagg)
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(indic == lagg, 1, posv)
> posv <- ifelse(indic == (-lagg), -1, posv)
> posv <- zoo::na.locf(posv, na.rm=FALSE)
> posv <- xts::xts(posv, order.by=zoo::index(closep))
> posv <- rutils::lagit(posv, lagg=1)
```



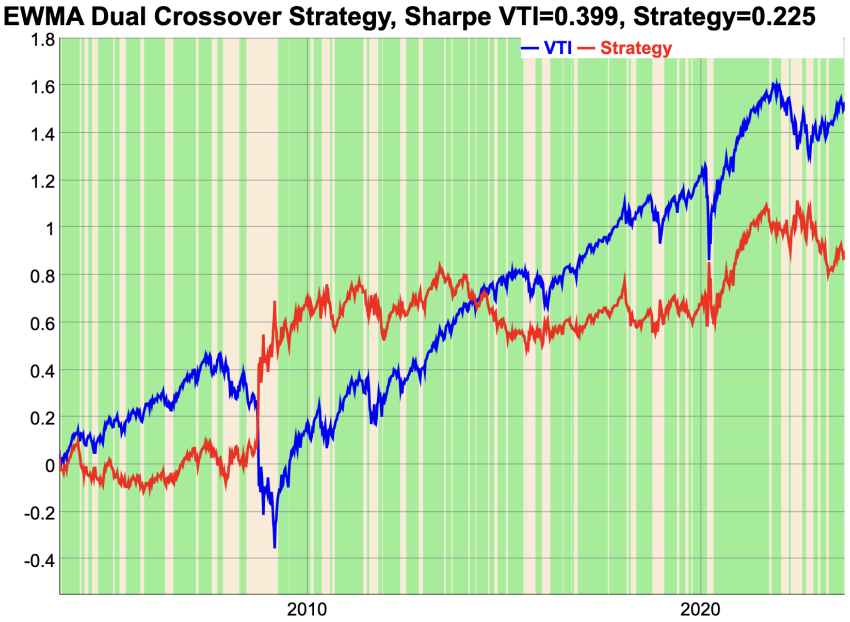

```
> # Create colors for background shading
> crossd <- (rutils::diffit(posv) != 0)
> shadev <- posv[crossd]
> crossd <- c(zoo::index(shadev), end(posv))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen", "
> # Plot dygraph
> colnamev <- colnames(pricev)
> dyplot <- dygraphs::dygraph(pricev, main="VTI Dual EMA Prices") %>
+   dySeries(name=colnamev[1], strokeWidth=1, col="blue") %>%
+   dySeries(name=colnamev[2], strokeWidth=2, col="red") %>%
+   dySeries(name=colnamev[3], strokeWidth=2, col="purple") %>%
+   dyLegend(show="always", width=200)
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], c
+ }  # end for
> dyplot
```

Dual EMA Crossover Strategy Performance

The crossover strategy suffers losses when prices are range-bound without a trend, because whenever it switches position the prices soon change direction. (This is called a *"whipsaw"*.)

The crossover strategy performance is worse than the underlying asset (*VTI*), but it has a negative correlation to it, which is very valuable when building a portfolio.

```
> # Calculate the daily profits and losses of strategy
> pnls <- retp*posv
> colnames(pnls) <- "Strategy"
> wealthv <- cbind(retp, pnls)
> # Annualized Sharpe ratio of Dual EMA strategy
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
```




```
> # Plot Dual EMA strategy
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main=paste("EMA Dual Crossover Strategy, Sharpe", paste(paste(na
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], c
+ }  # end for
> # Plot the dygraph object
> dyplot
```

Simulation Function for the Dual EMA Crossover Strategy

The *Dual EMA* strategy can be simulated by a single function, which allows the analysis of its performance depending on its parameters.

The function `sim_ema2()` performs a simulation of the *Dual EMA* strategy, given an *OHLC* time series of prices, and two decay factors λ_1 and λ_2.

The function `sim_ema2()` returns the *EMA* strategy positions and returns, in a two-column *xts* time series.

```
> sim_ema2 <- function(ohlc, lambdaf=0.1, lambdas=0.01,
+                bidask=0.001, trend=1, lagg=1) {
+   if (lambdaf >= lambdas) return(NA)
+   closep <- quantmod::Cl(ohlc)
+   retp <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate the EMA prices
+   emaf <- HighFreq::run_mean(closep, lambda=lambdaf)
+   emas <- HighFreq::run_mean(closep, lambda=lambdas)
+   # Calculate the positions, either: -1, 0, or 1
+   indic <- sign(emaf - emas)
+   if (lagg > 1) {
+     indic <- HighFreq::roll_sum(indic, lagg)
+     indic[1:lagg] <- 0
+   }  # end if
+   posv <- rep(NA_integer_, nrows)
+   posv[1] <- 0
+   posv <- ifelse(indic == lagg, 1, posv)
+   posv <- ifelse(indic == (-lagg), -1, posv)
+   posv <- zoo::na.locf(posv, na.rm=FALSE)
+   posv <- xts::xts(posv, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posv <- rutils::lagit(posv, lagg=1)
+   # Calculate the PnLs of strategy
+   pnls <- retp*posv
+   costs <- 0.5*bidask*abs(rutils::diffit(posv))
+   pnls <- (pnls - costs)
+   # Calculate the strategy returns
+   pnls <- cbind(posv, pnls)
+   colnames(pnls) <- c("positions", "pnls")
+   pnls
+ }  # end sim_ema2
```

Dual EMA Strategy Performance Matrix

Multiple *Dual EMA* strategies can be simulated by calling the function `sim_ema2()` in two loops over the vectors of λ parameters.

The function `outer()` calculates the values of a function over a grid spanned by two variables, and returns a matrix of function values.

The function `Vectorize()` performs an `apply()` loop over the arguments of a function, and returns a vectorized version of the function.

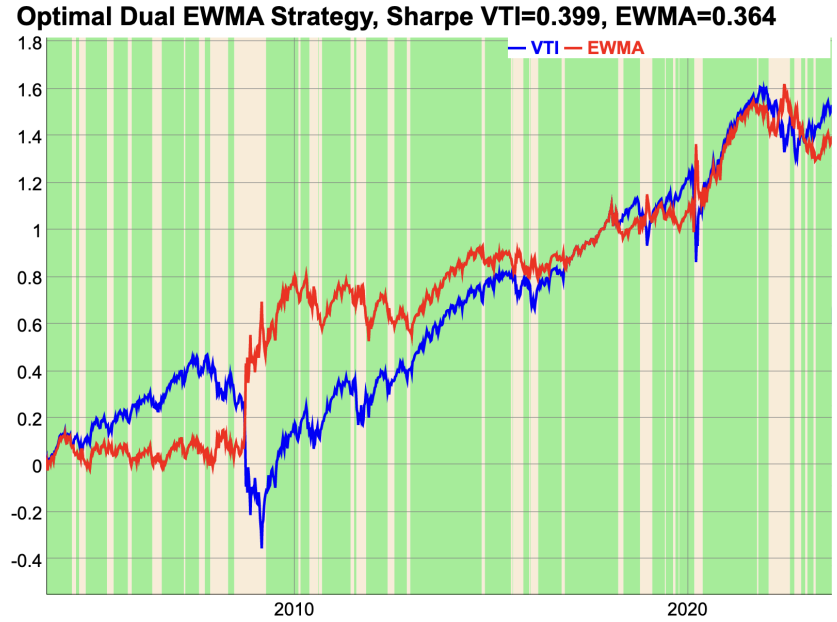
```
> lambdafv <- seq(from=0.85, to=0.99, by=0.01)
> lambdasv <- seq(from=0.85, to=0.99, by=0.01)
> # Calculate the Sharpe ratio of dual EMA strategy
> calc_sharpe <- function(ohlc, lambdaf, lambdas, bidask, trend, lag
+   if (lambdaf >= lambdas) return(NA)
+   pnls <- sim_ema2(ohlc=ohlc, lambdaf=lambdaf, lambdas=lambdas,
+     bidask=bidask, trend=trend, lagg=lagg)[, "pnls"]
+   sqrt(252)*mean(pnls)/sd(pnls)
+ }  # end calc_sharpe
> # Vectorize calc_sharpe with respect to lambdaf and lambdas
> calc_sharpe <- Vectorize(FUN=calc_sharpe,
+   vectorize.args=c("lambdaf", "lambdas"))
> # Calculate the matrix of PnLs
> sharpem <- outer(lambdafv, lambdasv, FUN=calc_sharpe, ohlc=ohlc,
+            bidask=0.0, trend=1, lagg=2)
> # Or perform two sapply() loops over lambda vectors
> sharpem <- sapply(lambdasv, function(lambdas) {
+   sapply(lambdafv, function(lambdaf) {
+     if (lambdaf >= lambdas) return(NA)
+     calc_sharpe(ohlc=ohlc, lambdaf=lambdaf, lambdas=lambdas,
+           bidask=0.0, trend=1, lagg=2)
+   })  # end sapply
+ })  # end sapply
> colnames(sharpem) <- lambdasv
> rownames(sharpem) <- lambdafv
```

Optimal Dual EMA Strategy

The best *Dual EMA* strategy performs better than the best *single EMA* strategy, because it has an extra parameter that can be adjusted to improve in-sample performance.

But this doesn't guarantee better out-of-sample performance.

```
> # Calculate the PnLs for the optimal strategy
> whichv <- which(sharpem == max(sharpem, na.rm=TRUE), arr.ind=TRUE)
> lambdaf <- lambdafv[whichv[1]]
> lambdas <- lambdasv[whichv[2]]
> emopt <- sim_ema2(ohlc=ohlc, lambdaf=lambdaf, lambdas=lambdas,
+   bidask=0.0, trend=1, lagg=2)
> pnls <- emopt[, "pnls"]
> wealthv <- cbind(retp, pnls)
> colnames(wealthv)[2] <- "EMA"
> # Calculate the Sharpe and Sortino ratios
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Annualized Sharpe ratio of Dual EMA strategy
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # The crossover strategy has a negative correlation to VTI
> cor(wealthv)[1, 2]
```

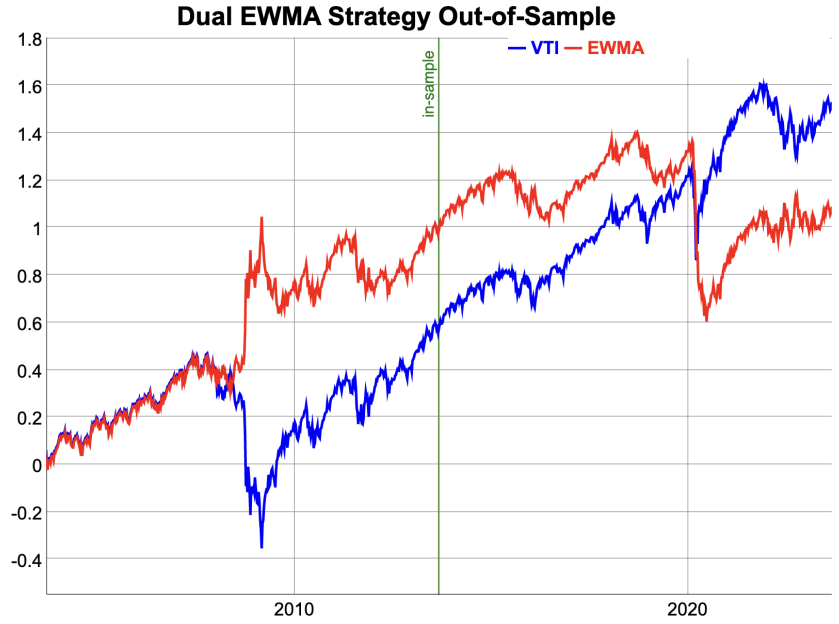



```
> # Create colors for background shading
> posv <- emopt[, "positions"]
> crossd <- (rutils::diffit(posv) != 0)
> shadev <- posv[crossd]
> crossd <- c(zoo::index(shadev), end(posv))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen", "
> # Plot Optimal Dual EMA strategy
> dyplot <- dygraphs::dygraph(cumsum(wealthv), main=paste("Optimal D
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1], c
+ }  # end for
> # Plot the dygraph object
> dyplot
```

Performance of Dual Crossover Strategy *Out-of-Sample*

In-sample, the best *Dual EMA* strategy performs better than *VTI*, because it has two parameters that can be adjusted to improve performance.

But out-of-sample, the best *Dual EMA* strategy performs worse than *VTI*, because it's been *overfitted* in-sample.




```
> # Define in-sample and out-of-sample intervals
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Calculate the matrix of PnLs
> sharpem <- outer(lambdafv, lambdasv,
+            FUN=calc_sharpe, ohlc=ohlc[insample, ],
+            bidask=0.0, trend=1, lagg=2)
> colnames(sharpem) <- lambdasv
> rownames(sharpem) <- lambdafv
> # Calculate the PnLs for the optimal strategy
> whichv <- which(sharpem == max(sharpem, na.rm=TRUE), arr.ind=TRU
> lambdaf <- lambdafv[whichv[1]]
> lambdas <- lambdasv[whichv[2]]
> pnls <- sim_ema2(ohlc=ohlc, lambdaf=lambdaf, lambdas=lambdas,
+             bidask=0.0, trend=1, lagg=2)[, "pnls"]
> wealthv <- cbind(retp, pnls)
> colnames(wealthv)[2] <- "EMA"
> # Calculate the Sharpe and Sortino ratios in-sample and out-of-sample
> sqrt(252)*sapply(wealthv[insample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv[outsample, ], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```

```
> # Dygraphs plot with custom line colors
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Dual EMA Strategy
+   dyEvent(zoo::index(wealthv[last(insample)]), label="in-sample",
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
```

Volume-Weighted Average Price Indicator

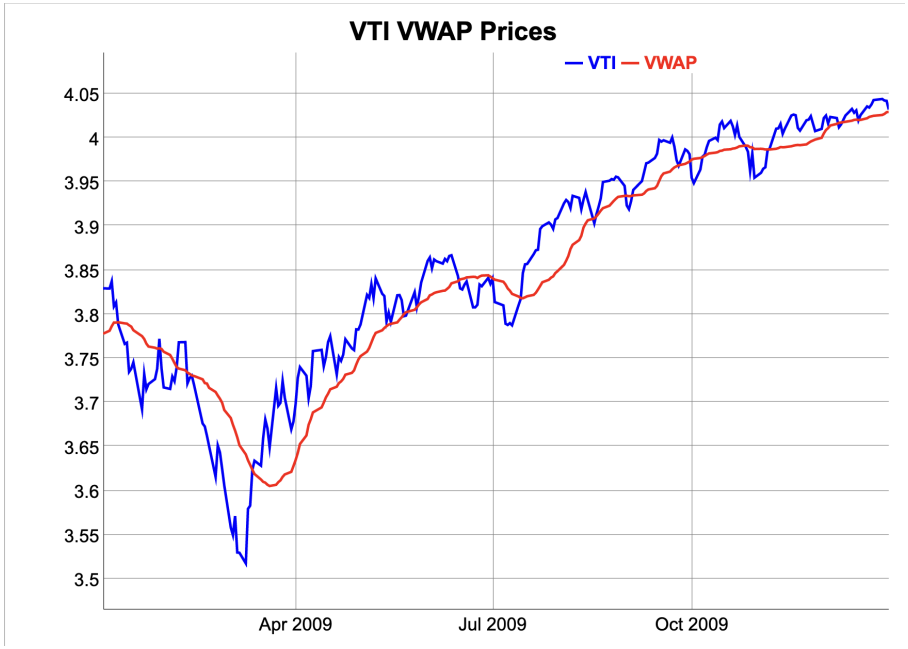
The Volume-Weighted Average Price (*VWAP*) is defined as the sum of prices multiplied by trading volumes, divided by the sum of volumes:

$$p_t^{VWAP} = \frac{\sum_{j=0}^{n} v_{t-j} p_{t-j}}{\sum_{j=0}^{n} v_{t-j}}$$

The *VWAP* applies more weight to prices with higher trading volumes, which allows it to react more quickly to recent market volatility.

The drawback of the *VWAP* indicator is that it applies large weights to prices far in the past.

The *VWAP* is often used as a technical indicator in trend following strategies.




```
> # Calculate the log OHLC prices and volumes
> ohlc <- rutils::etfenv$VTI
> closep <- log(quantmod::Cl(ohlc))
> colnames(closep) <- "VTI"
> volumv <- quantmod::Vo(ohlc)
> colnames(volumv) <- "Volume"
> nrows <- NROW(closep)
> # Calculate the VWAP prices
> lookb <- 21
> vwap <- HighFreq::roll_sum(closep, lookb=lookb, weightv=volumv)
> colnames(vwap) <- "VWAP"
> pricev <- cbind(closep, vwap)
```

```
> # Dygraphs plot with custom line colors
> colorv <- c("blue", "red")
> dygraphs::dygraph(pricev["2009"], main="VTI VWAP Prices") %>%
+   dyOptions(colors=colorv, strokeWidth=2)
> # Plot VWAP prices with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colors
> quantmod::chart_Series(pricev["2009"], theme=plot_theme,
+        lwd=2, name="VTI VWAP Prices")
> legend("bottomright", legend=colnames(pricev),
+  inset=0.1, bg="white", lty=1, lwd=6, cex=0.8,
+  col=plot_theme$col$line.col, bty="n")
```

Recursive VWAP Price Indicator

The *VWAP* prices p^{VWAP} can also be calculated as the ratio of the volume weighted prices μ^{pv} divided by the mean trading volumes μ^{v}:

$$p^{VWAP} = \frac{\mu^{pv}}{\mu^{v}}$$

The volume weighted prices μ^{pv} and the mean trading volumes μ^{v} are both calculated recursively:

$$\mu^{v}_{t} = \lambda\mu^{v}_{t-1} + (1-\lambda)v_t$$

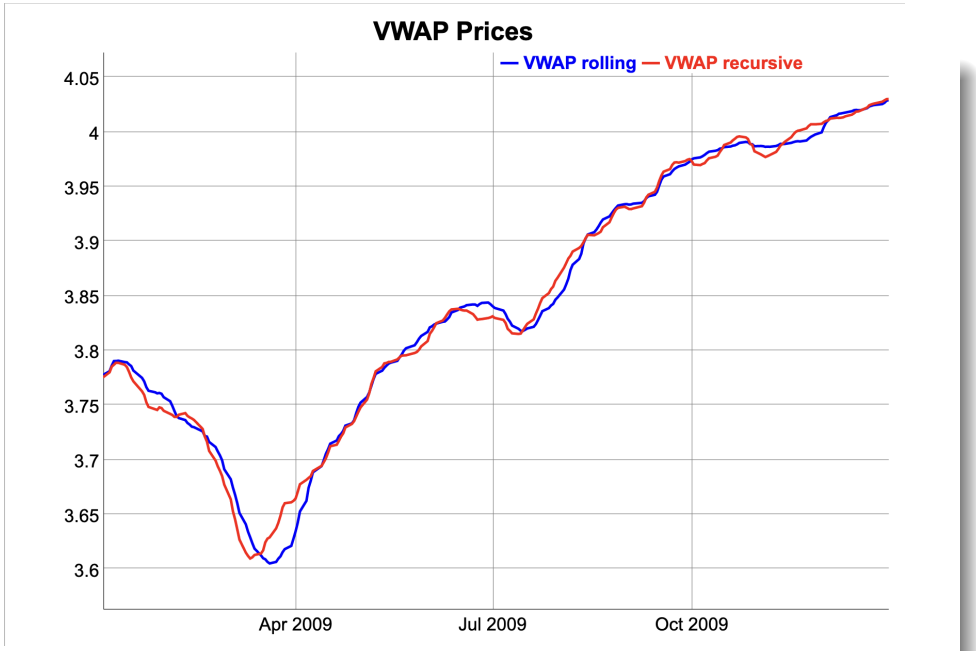
$$\mu^{pv}_{t} = \lambda\mu^{pv}_{t-1} + (1-\lambda)v_t p_t$$

The recursive *VWAP* prices are slightly different from those calculated as a convolution, because the convolution uses a fixed look-back interval.

The advantage of the recursive *VWAP* indicator is that it gradually "forgets" about large trading volumes far in the past.

The compiled C++ function `stats:::C_rfilter()` calculates the trailing weighted values recursively.

The function `HighFreq::run_mean()` also calculates the trailing weighted values recursively.




```
> # Calculate the VWAP prices recursively using C++ code
> lambda <- 0.9
> volumer <- .Call(stats:::C_rfilter, volumv, lambda, c(as.numeric(v
> pricer <- .Call(stats:::C_rfilter, volumv*closep, lambda, c(as.nu
> vwapr <- pricer/volumer
> # Calculate the VWAP prices recursively using RcppArmadillo
> vwapcpp <- HighFreq::run_mean(closep, lambda=lambda, weightv=volu
> all.equal(vwapr, drop(vwapcpp))
> # Dygraphs plot the VWAP prices
> pricev <- xts(cbind(vwap, vwapr), zoo::index(ohlc))
> colnames(pricev) <- c("VWAP rolling", "VWAP recursive")
> dygraphs::dygraph(pricev["2009"], main="VWAP Prices") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
```

Simulating the VWAP Crossover Strategy

In the trend following *VWAP Crossover* strategy, the stock position switches depending if the current price is above or below the *VWAP*.

If the current price crosses above the *VWAP*, then the strategy switches its stock position to a fixed unit of long risk, and if it crosses below, to a fixed unit of short risk.

To prevent whipsaws and over-trading, the crossover strategy delays switching positions until the indicator repeats the same value for several periods.


```
> # Calculate the VWAP prices recursively using RcppArmadillo
> lambda <- 0.99
> vwapcpp <- HighFreq::run_mean(closep, lambda=lambda, weightv=volu
> # Calculate the positions from lagged indicator
> indic <- sign(closep - vwapcpp)
> lagg <- 2
> indic <- HighFreq::roll_sum(indic, lagg)
> # Calculate the positions, either: -1, 0, or 1
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(indic == lagg, 1, posv)
> posv <- ifelse(indic == (-lagg), -1, posv)
> posv <- zoo::na.locf(posv, na.rm=FALSE)
> posv <- xts::xts(posv, order.by=zoo::index(closep))
> # Lag the positions to trade in next period
> posv <- rutils::lagit(posv, lagg=1)
> # Calculate the PnLs of VWAP strategy
> retp <- rutils::diffit(closep)  # VTI returns
> pnls <- retp*posv
> colnames(pnls) <- "VWAP"
> wealthv <- cbind(retp, pnls)
> colnamev <- colnames(wealthv)
```

```
> # Annualized Sharpe ratios of VTI and VWAP strategy
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
> # Create colors for background shading
> crossd <- (rutils::diffit(posv) != 0)
> shadev <- posv[crossd]
> crossd <- c(zoo::index(shadev), end(posv))
> shadev <- ifelse(drop(zoo::coredata(shadev)) == 1, "lightgreen",
> # Plot dygraph of VWAP strategy
> # Create dygraph object without plotting it
> dyplot <- dygraphs::dygraph(cumsum(wealthv),
+   main=paste("VWAP Crossover Strategy, Sharpe", paste(paste(names
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=200)
> # Add shading to dygraph object
> for (i in 1:NROW(shadev)) {
+   dyplot <- dyplot %>% dyShading(from=crossd[i], to=crossd[i+1],
+ }  # end for
> # Plot the dygraph object
```

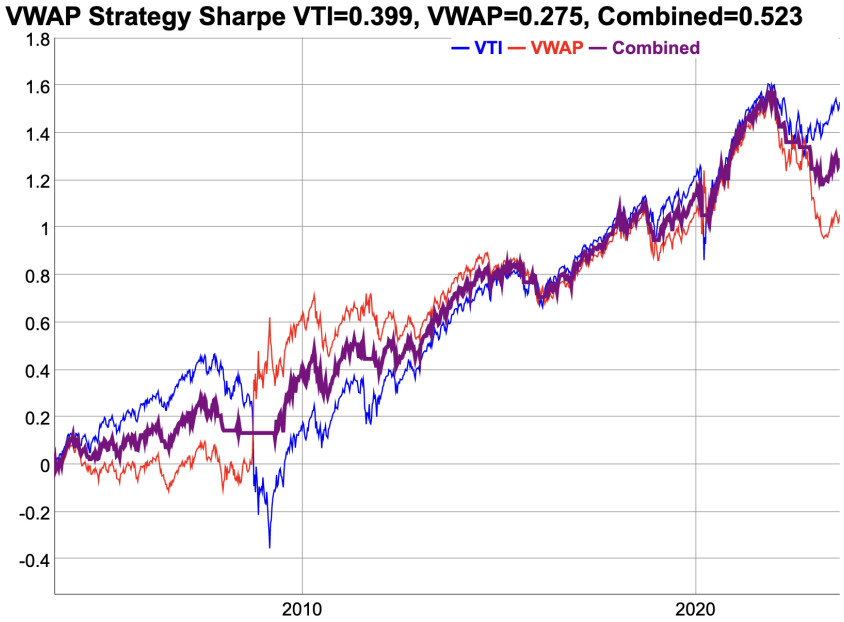
Combining VWAP Crossover Strategy with Stocks

Even though the *VWAP* strategy doesn't perform as well as a static buy-and-hold strategy, it can provide risk reduction when combined with it.

This is because the *VWAP* strategy has a negative correlation with respect to the underlying asset.

In addition, the *VWAP* strategy performs well in periods of extreme market selloffs, so it can provide a hedge for a static buy-and-hold strategy.

The *VWAP* strategy serves as a dynamic put option in periods of extreme market selloffs.




```
> # Calculate the correlation of VWAP strategy with VTI
> cor(retp, pnls)
> # Combine VWAP strategy with VTI
> wealthv <- cbind(retp, pnls, 0.5*(retp+pnls))
> colnames(wealthv) <- c("VTI", "VWAP", "Combined")
> sharper <- sqrt(252)*sapply(wealthv, function (x) mean(x)/sd(x))
```

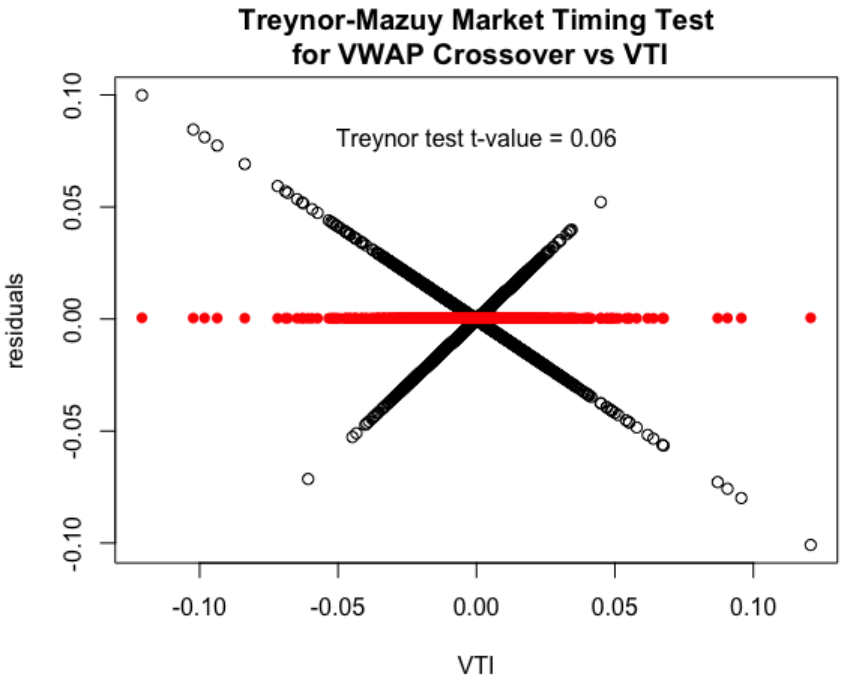
```
> # Plot dygraph of VWAP strategy combined with VTI
> colorv <- c("blue", "red", "purple")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   paste("VWAP Strategy Sharpe", paste(paste(names(sharper), round(
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dySeries(name="Combined", label="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=200)
```

VWAP Crossover Strategy Market Timing Skill

The VWAP crossover strategy shorts the market during significant selloffs, but otherwise doesn't display market timing skill.

The t-value of the *Treynor-Mazuy* test is negative, but not statistically significant.

```
> # Test VWAP crossover market timing of VTI using Treynor-Mazuy tes
> desm <- cbind(pnls, retp, retp^2)
> desm <- na.omit(desm)
> colnames(desm) <- c("VWAP", "VTI", "treynor")
> regmod <- lm(VWAP ~ VTI + treynor, data=desm)
> summary(regmod)
> # Plot residual scatterplot
> resids <- (desm$VWAP - regmod$coeff["VTI"]*retp)
> resids <- regmod$residuals
> # x11(width=6, height=6)
> plot.default(x=retp, y=resids, xlab="VTI", ylab="residuals")
> title(main="Treynor-Mazuy Market Timing Test\n for VWAP Crossover
> # Plot fitted (predicted) response values
> coefreg <- summary(regmod)$coeff
> fitv <- regmod$fitted.values - coefreg["VTI", "Estimate"]*retp
> tvalue <- round(coefreg["treynor", "t value"], 2)
> points.default(x=retp, y=fitv, pch=16, col="red")
> text(x=0.0, y=0.8*max(resids), paste("Treynor test t-value =", tvalue))
```



Simulation Function for VWAP Crossover Strategy

The *VWAP* strategy can be simulated by a single function, which allows the analysis of its performance depending on its parameters.

The function `sim_vwap()` performs a simulation of the *VWAP* strategy, given an *OHLC* time series of prices, and the length of the look-back interval (`lookb`).

The function `sim_vwap()` returns the *VWAP* strategy positions and returns, in a two-column *xts* time series.

```
> sim_vwap <- function(ohlc, lambda=0.9, bidask=0.001, trend=1, lagg
+   closep <- log(quantmod::Cl(ohlc))
+   volumv <- quantmod::Vo(ohlc)
+   retp <- rutils::diffit(closep)
+   nrows <- NROW(ohlc)
+   # Calculate the VWAP prices
+   vwap <- HighFreq::run_mean(closep, lambda=lambda, weightv=volumv
+   # Calculate the indicator
+   indic <- trend*sign(closep - vwap)
+   if (lagg > 1) {
+     indic <- HighFreq::roll_sum(indic, lagg)
+     indic[1:lagg] <- 0
+   }  # end if
+   # Calculate the positions, either: -1, 0, or 1
+   posv <- rep(NA_integer_, nrows)
+   posv[1] <- 0
+   posv <- ifelse(indic == lagg, 1, posv)
+   posv <- ifelse(indic == (-lagg), -1, posv)
+   posv <- zoo::na.locf(posv, na.rm=FALSE)
+   posv <- xts::xts(posv, order.by=zoo::index(closep))
+   # Lag the positions to trade on next day
+   posv <- rutils::lagit(posv, lagg=1)
+   # Calculate the PnLs of strategy
+   pnls <- retp*posv
+   costs <- 0.5*bidask*abs(rutils::diffit(posv))
+   pnls <- (pnls - costs)
+   # Calculate the strategy returns
+   pnls <- cbind(posv, pnls)
+   colnames(pnls) <- c("positions", "pnls")
+   pnls
+ }  # end sim_vwap
```

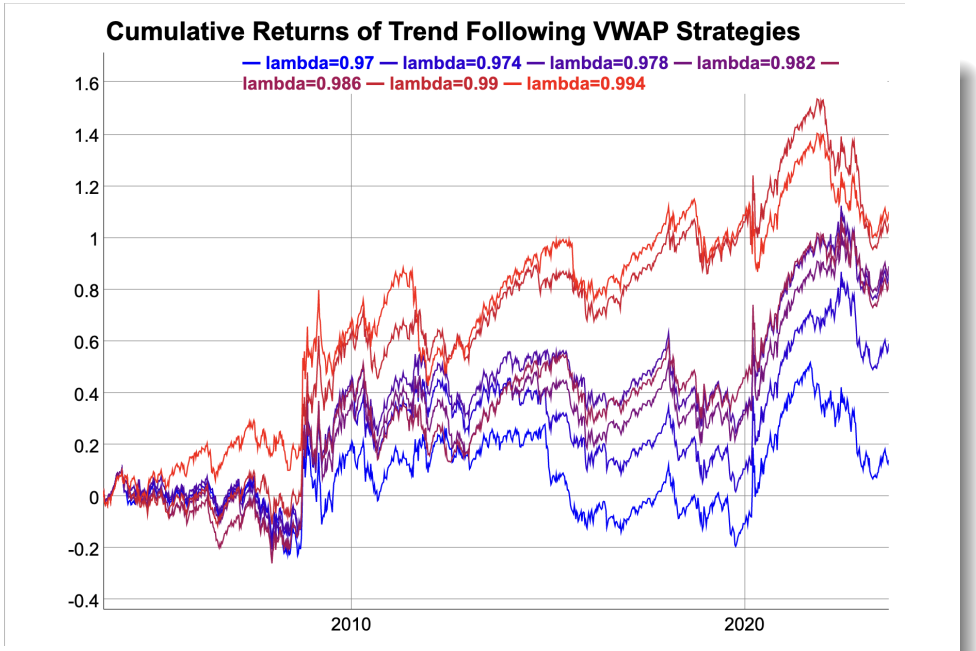
Simulating Multiple Trend Following VWAP Strategies

Multiple *VWAP* strategies can be simulated by calling the function `sim_vwap()` in a loop over a vector of λ parameters.

But `sim_vwap()` returns an *xts* time series, and `sapply()` cannot merge *xts* time series together.

So instead the loop is performed using `lapply()` which returns a list of *xts*, and the list is merged into a single *xts* using the functions `do.call()` and `cbind()`.

```
> lambdav <- seq(from=0.97, to=0.995, by=0.004)
> # Perform lapply() loop over lambdav
> pnls <- lapply(lambdav, function(lambda) {
+   # Simulate VWAP strategy and Calculate the returns
+   sim_vwap(ohlc=ohlc, lambda=lambda, bidask=0, lagg=2)[, "pnls"]
+ })  # end lapply
> pnls <- do.call(cbind, pnls)
> colnames(pnls) <- paste0("lambda=", lambdav)
```




```
> # Plot dygraph of multiple VWAP strategies
> colorv <- colorRampPalette(c("blue", "red"))(NCOL(pnls))
> dygraphs::dygraph(cumsum(pnls)[endd], main="Cumulative Returns of
+   dyOptions(colors=colorv, strokeWidth=1) %>%
+   dyLegend(show="always", width=500)
> # Plot VWAP strategies with custom line colors
> x11(width=6, height=5)
> plot_theme <- chart_theme()
> plot_theme$col$line.col <- colorv
> quantmod::chart_Series(cumsum(pnls), theme=plot_theme,
+   name="Cumulative Returns of VWAP Strategies")
> legend("topleft", legend=colnames(pnls), inset=0.1,
+   bg="white", cex=0.8, lwd=rep(6, NCOL(pnls)),
+   col=plot_theme$col$line.col, bty="n")
```

Backtesting of Rolling Strategies

Backtesting is the simulation of a trading strategy on historical data.

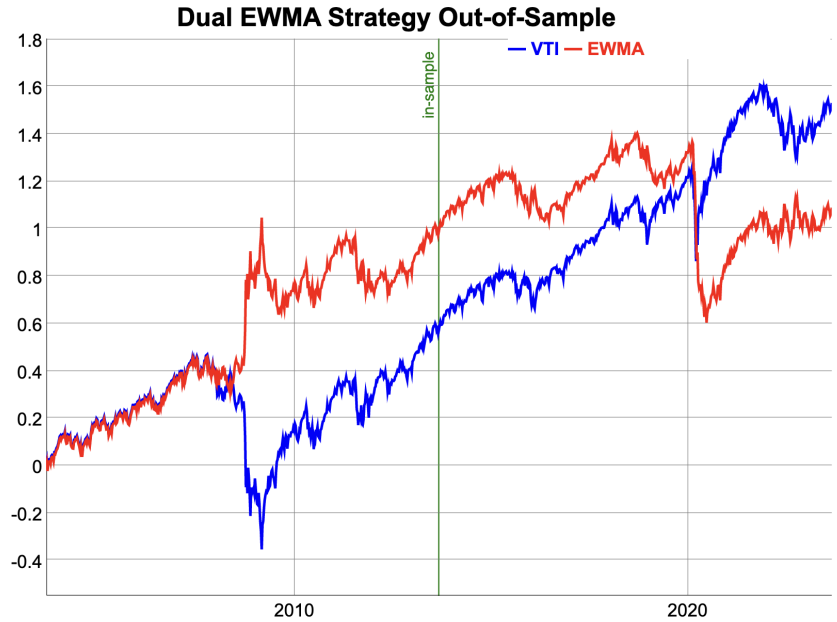
A *rolling strategy* can be *backtested* by specifying the parameter updating frequency, the formation interval, and the holding period:

- Calculate the *end points* for parameter updating,
- Define an objective function for parameter optimization,
- Calculate the optimal parameters in the in-sample formation interval,
- Calculate the out-of-sample strategy returns,
- Calculate the transaction costs and subtract them from the strategy returns.

The *backtesting* redefines the problem of finding (tuning) the optimal trading strategy parameters, into the problem of finding the optimal *backtest* (meta-model) parameters: the updating frequency, the formation interval, and the holding period.

The advantage of using the *backtest* meta-model is that it can reduce the number of parameters that need to be optimized.

Using a different updating frequency in the *backtest* can produce different values for the optimal trading strategy parameters.




```
> # Dygraphs plot with custom line colors
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd], main="Dual Crossover Stra
+   dyEvent(zoo::index(wealthv[last(insample)]), label="in-sample",
+   dyOptions(colors=c("blue", "red"), strokeWidth=2)
```

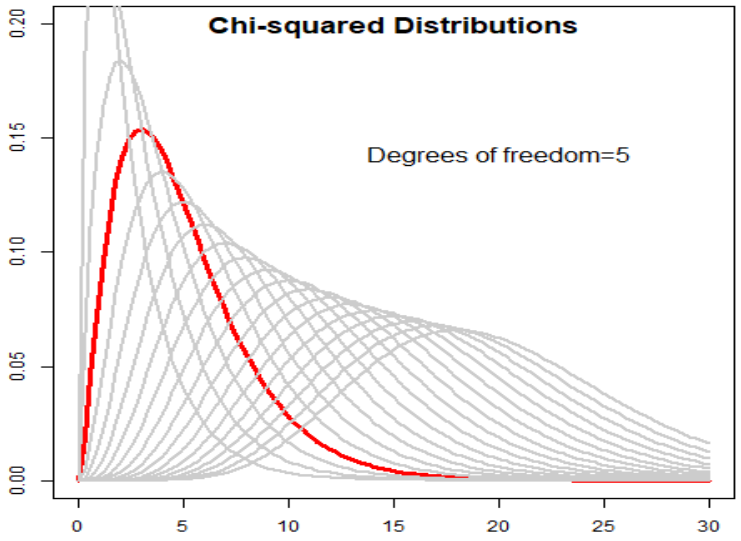
Plotting Using Expression Objects

It's sometimes convenient to create an *expression* object containing plotting commands, to be able to later create plots using it.

The function `quote()` produces an *expression* object without evaluating it.

The function `eval()` evaluates an *expression* in a specified *environment*.

```
> # Create a plotting expression
> expv <- quote({
+   degf <- 2:20
+   rangev <- (1:NROW(degf))
+   indeks <- 4
+   # Plot a curve
+   curve(expr=dchisq(x, df=degf[indeks]),
+ xlim=c(0, 30), ylim=c(0, 0.2),
+ xlab="", ylab="", lwd=3, col="red")
+   # Add grey lines to plot
+   for (it in rangev[-indeks]) {
+     curve(expr=dchisq(x, df=degf[it]),
+   xlim=c(0, 30), ylim=c(0, 0.2),
+   xlab="", ylab="", lwd=2, col="grey80", add=TRUE)
+   }  # end for
+   # Add title
+   title(main="Chi-squared Distributions", line=-1.5, cex.main=1.5)
+   # Add legend
+   text(x=20, y=0.15, labels=paste0("Degrees of freedom=",
+       degf[indeks]), pos=1, cex=1.3)
+ })  # end quote
```




```
> # View the plotting expression
> expv
> # Create plot by evaluating the plotting expression
> x11(width=6, height=4)
> eval(expv)
```

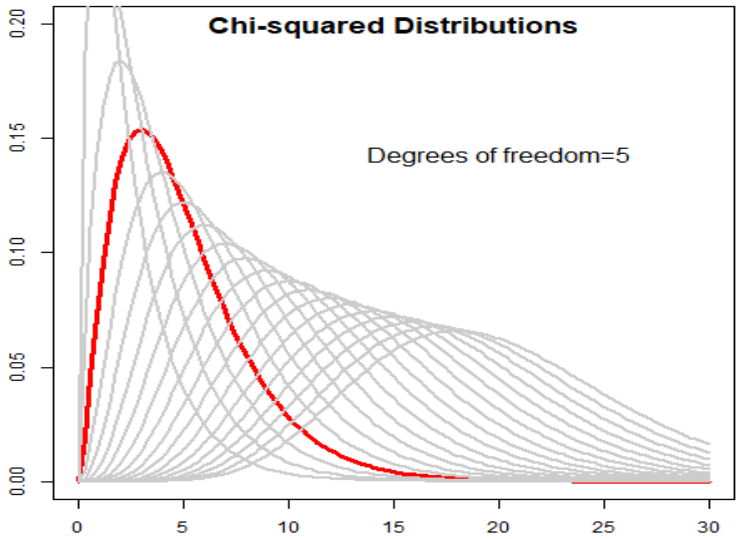
Animated Plots Using Package *animation*

The package *animation* allows creating animated plots in the form of *gif* and *html* documents.

The function `saveGIF()` produces a *gif* image with an animated plot.

The function `saveHTML()` produces an *html* document with an animated plot.

```
> library(animation)
> # Create an expression for creating multiple plots
> expv <- quote({
+   degf <- 2:20
+   rangev <- (1:NROW(degf))
+   # Set image refesh interval
+   animation::ani.options(interval=0.5)
+   # Create multiple plots with curves
+   for (indeks in rangev) {
+     curve(expr=dchisq(x, df=degf[indeks]),
+   xlim=c(0, 30), ylim=c(0, 0.2),
+   xlab="", ylab="", lwd=3, col="red")
+     # Add grey lines to plot
+     for (it in rangev[-indeks]) {
+       curve(expr=dchisq(x, df=degf[it]),
+     xlim=c(0, 30), ylim=c(0, 0.2),
+     xlab="", ylab="", lwd=2, col="grey80", add=TRUE)
+     }  # end for
+     # Add title
+     title(main="Chi-squared Distributions", line=-1.5, cex.main=
+     # Add legend
+     text(x=20, y=0.15, labels=paste0("Degrees of freedom=",
+       degf[indeks]), pos=1, cex=1.3)
+   }  # end for
+ })  # end quote
```




```
> # Create plot by evaluating the plotting expression
> x11(width=6, height=4)
> eval(expv)
> # Create gif with animated plot
> animation::saveGIF(expr=eval(expv),
+   movie.name="chi_squared.gif",
+   img.name="chi_squared")
> # Create html with animated plot
> animation::saveHTML(expr=eval(expv),
+   img.name="chi_squared",
+   htmlfile="chi_squared.html",
+   description="Chi-squared Distributions")  # end saveHTML
```

Dynamic Documents Using *R markdown*

markdown is a simple markup language designed for creating documents in different formats, including *pdf* and *html*.

R Markdown is a modified version of *markdown*, which allows creating documents containing *math formulas* and `R` code embedded in them.

An *R Markdown* document (with extension `.Rmd`) contains:

- A *YAML* header,
- Text in *R Markdown* code format,
- Math formulas (equations), delimited using either single "$" symbols (for inline formulas), or double "$$" symbols (for display formulas),
- `R` code chunks, delimited using either single "`" backtick symbols (for inline code), or triple "```" backtick symbols (for display code).

The packages *rmarkdown* and *knitr* compile `R` documents into either *pdf*, *html*, or *MS Word* documents.

````
---
title: "My First R Markdown Document"
author: Jerzy Pawlowski
date: ''`r format(Sys.time(), "%m/%d/%Y")`''
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

# install package quantmod if it can't be loaded success
if (!require("quantmod"))
  install.packages("quantmod")
```

### R Markdown
This is an *R Markdown* document. Markdown is a simple f

One of the advantages of writing documents *R Markdown*

You can read more about publishing documents using *R* h
https://algoquant.github.io/r,/markdown/2016/07/02/Publi

You can read more about using *R* to create *HTML* docum
https://algoquant.github.io/2016/07/05/Interactive-Plots

Clicking the **Knit** button in *RStudio*, compiles the

Example of an *R* code chunk:
```{r cars}
summary(cars)
```

### Plots in *R Markdown* documents

Plots can also be embedded, for example:
```{r pressure, echo=FALSE}
plot(pressure)
````
````

Package *shiny* for Creating Interactive Applications

The package *shiny* creates interactive applications running in R, with their outputs presented as live visualizations.

Shiny allows changing the model parameters, recalculating the model, and displaying the resulting outputs as plots and charts.

A *shiny app* is a file with *shiny* commands and R code.

The *shiny* code consists of a *shiny interface* and a *shiny server*.

The *shiny interface* contains widgets for data input and an area for plotting.

The *shiny server* contains the R model code and the plotting code.

The function `shiny::fluidPage()` creates a GUI layout for the user inputs of model parameters and an area for plots and charts.

The function `shiny::renderPlot()` renders a plot from the outputs of a live model.

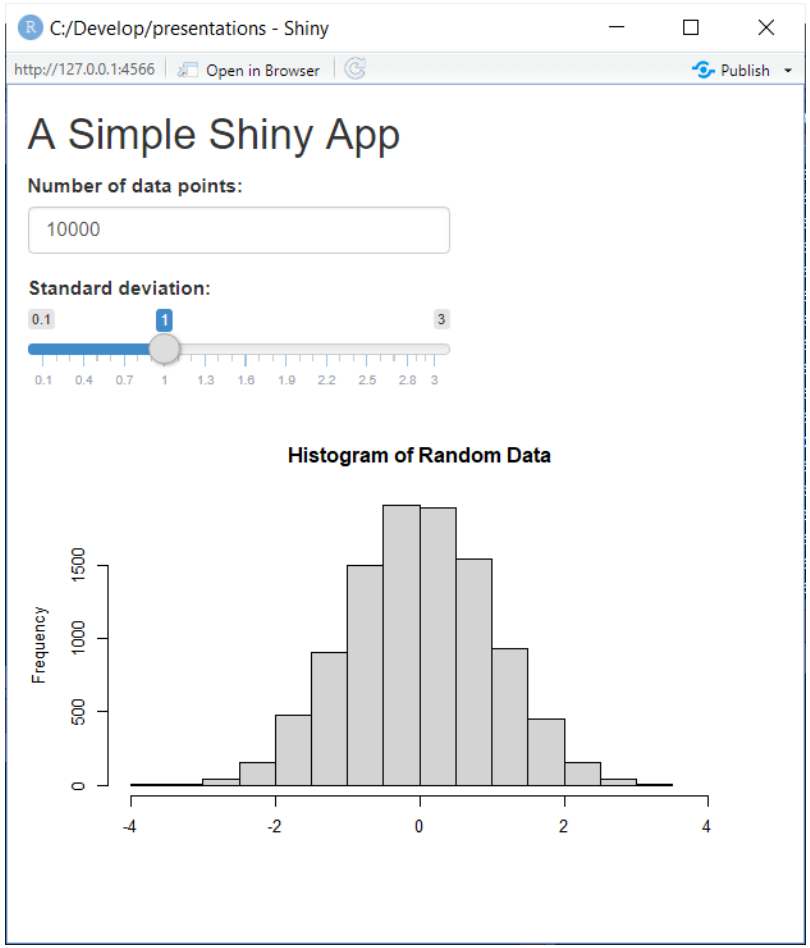
The function `shiny::shinyApp()` creates a `shiny` app from a *shiny interface* and a *shiny server*.

```
> ## App setup code that runs only once at startup.
> ndata <- 1e4
> stdev <- 1.0
>
> ## Define the user interface
> uiface <- shiny::fluidPage(
+   # Create numeric input for the number of data points.
+   numericInput("ndata", "Number of data points:", value=ndata),
+   # Create slider input for the standard deviation parameter.
+   sliderInput("stdev", label="Standard deviation:",
+         min=0.1, max=3.0, value=stdev, step=0.1),
+   # Render plot in a panel.
+   plotOutput("plotobj", height=300, width=500)
+ )  # end user interface
>
> ## Define the server function
> servfun <- function(input, output) {
+   output$plotobj <- shiny::renderPlot({
+     # Simulate the data
+     datav <- rnorm(input$ndata, sd=input$stdev)
+     # Plot the data
+     par(mar=c(2, 4, 4, 0), oma=c(0, 0, 0, 0))
+     hist(datav, xlim=c(-4, 4), main="Histogram of Random Data")
+   })  # end renderPlot
+ }  # end servfun
>
> # Return a Shiny app object
> shiny::shinyApp(ui=uiface, server=servfun)
```

Running Shiny Apps in *RStudio*

A *shiny app* can be run by pressing the "Run App" button in *RStudio*.

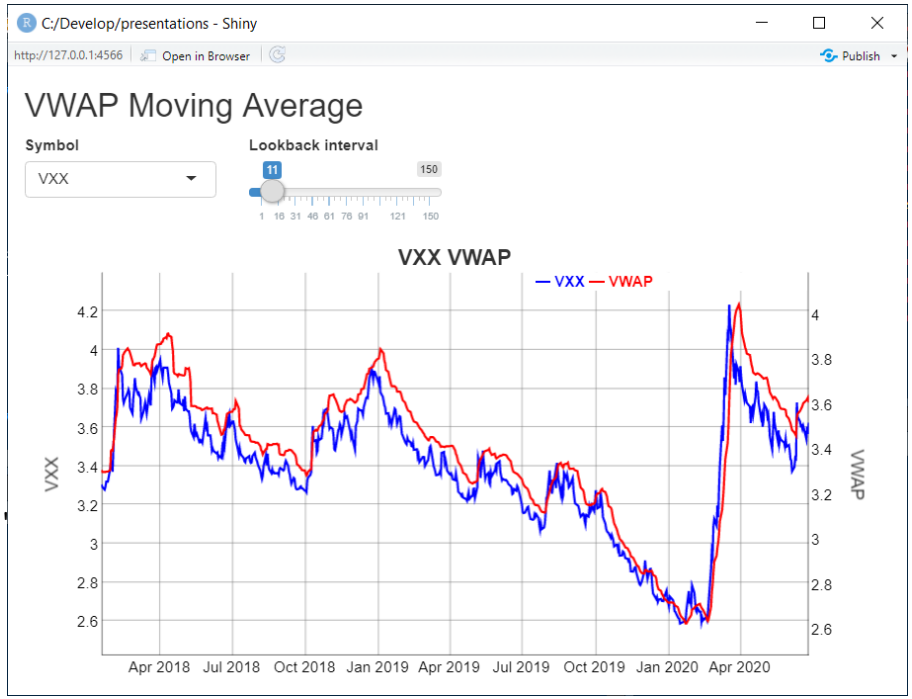
When the *shiny app* is run, the *shiny* commands are translated into *JavaScript* code, which creates a graphical user interface (GUI) with buttons, sliders, and boxes for data input, and also with the output plots and charts.



Positioning and Sizing Widgets Within the Shiny GUI

The functions `shiny::fluidRow()` and `shiny::column()` allow positioning and sizing widgets within the *shiny* GUI.

```
> ## Create elements of the user interface
> uiface <- shiny::fluidPage(
+   titlePanel("VWAP Moving Average"),
+   # Create single row of widgets with two slider inputs
+   fluidRow(
+     # Input stock symbol
+     column(width=3, selectInput("symbol", label="Symbol",
+                           choices=symbolv, selected=symbol)),
+     # Input look-back interval
+     column(width=3, sliderInput("lookb", label="Lookback interval"
+                           min=1, max=150, value=11, step=1))
+   ),  # end fluidRow
+   # Create output plot panel
+   mainPanel(dygraphs::dygraphOutput("dyplot"), width=12)
+ )  # end fluidPage interface
```



Shiny Apps With Reactive Expressions

The package *shiny* allows specifying reactive expressions which are evaluated only when their input data is updated.

Reactive expressions avoid performing unnecessary calculations.

If the reactive expression is invalidated (recalculated), then other expressions that depend on its output are also recalculated.

This way calculations cascade through the expressions that depend on each other.

The function `shiny::reactive()` transforms an expression into a reactive expression.

```
> ## Define the server function
> servfun <- shiny::shinyServer(function(input, output) {
+   # Get the close and volume data in a reactive environment
+   closep <- shiny::reactive({
+     # Get the data
+     ohlc <- get(input$symbol, data_env)
+     closep <- log(quantmod::Cl(ohlc))
+     volum <- quantmod::Vo(ohlc)
+     # Return the data
+     cbind(closep, volum)
+   })  # end reactive code
+
+   # Calculate the VWAP indicator in a reactive environment
+   vwapv <- shiny::reactive({
+     # Get model parameters from input argument
+     lookb <- input$lookb
+     # Calculate the VWAP indicator
+     closep <- closep()[, 1]
+     volum <- closep()[, 2]
+     vwapv <- HighFreq::roll_sum(tseries=closep*volum, lookb=lookb)
+     volumroll <- HighFreq::roll_sum(tseries=volum, lookb=lookb)
+     vwapv <- vwapv/volumroll
+     vwapv[is.na(vwapv)] <- 0
+     # Return the plot data
+     datav <- cbind(closep, vwapv)
+     colnames(datav) <- c(input$symbol, "VWAP")
+     datav
+   })  # end reactive code
+
+   # Return the dygraph plot to output argument
+   output$dyplot <- dygraphs::renderDygraph({
+     colnamev <- colnames(vwapv())
+     dygraphs::dygraph(vwapv(), main=paste(colnamev[1], "VWAP")) %>
+ dyAxis("y", label=colnamev[1], independentTicks=TRUE) %>%
+ dyAxis("y2", label=colnamev[2], independentTicks=TRUE) %>%
+ dySeries(name=colnamev[1], axis="y", label=colnamev[1], strokeWidt
+ dySeries(name=colnamev[2], axis="y2", label=colnamev[2], strokeWi
+   })  # end output plot
+ })  # end server code
```

Reactive Event Handlers

Event handlers are functions which evaluate expressions when an event occurs (like a button press).

The functions `shiny::observeEvent()` and `shiny::eventReactive()` are event handlers.

The function `shiny::eventReactive()` returns a value, while `shiny::observeEvent()` produces a side-effect, without returning a value.

The function `shiny::reactiveValues()` creates a list for storing reactive values, which can be updated by event handlers.

```
> ## Define the server function
> servfun <- shiny::shinyServer(function(input, output) {
+
+   # Create an empty list of reactive values.
+   value_s <- reactiveValues()
+
+   # Get input parameters from the user interface.
+   nrows <- reactive({
+     # Add nrows to list of reactive values.
+     value_s*nrows <- input$nrows
+     input$nrows
+   })  # end reactive code
+
+   # Broadcast a message to the console when the button is pressed.
+   observeEvent(eventExpr=input$button, handlerExpr={
+     cat("Input button pressed\n")
+   })  # end observeEvent
+
+   # Send the data when the button is pressed.
+   datav <- eventReactive(eventExpr=input$button, valueExpr={
+     # eventReactive() executes on input$button, but not on nrows()
+     cat("Sending", nrows(), "rows of data\n")
+     datav <- head(mtcars, input$nrows)
+     value_s$mpg <- mean(datav$mpg)
+     datav
+   })  # end eventReactive
+   #   datav
+
+   # Draw table of the data when the button is pressed.
+   observeEvent(eventExpr=input$button, handlerExpr={
+     datav <- datav()
+     cat("Received", value_s*nrows, "rows of data\n")
+     cat("Average mpg = ", value_s$mpg, "\n")
+     cat("Drawing table\n")
+     output$tablev <- renderTable(datav)
+   })  # end observeEvent
+
+ })  # end server code
>
```

Homework Assignment

Required

- Study all the lecture slides in *FRE7241_Lecture_3.pdf*, and run all the code in *FRE7241_Lecture_3.R*