FRE7241 Algorithmic Portfolio Management

Lecture#3, Spring 2025

Jerzy Pawlowski *jp3900@nyu.edu*

NYU Tandon School of Engineering

April 8, 2025

Autocorrelation Function of Time Series

The *autocorrelation* of lag k of a time series of returns r_t is equal to:

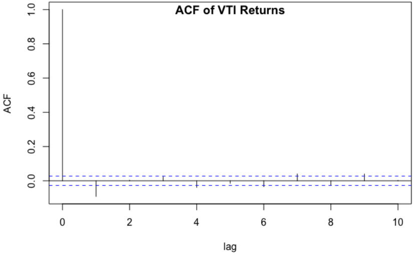
$$\rho_k = \frac{\sum_{t=k+1}^{n} (r_t - \bar{r})(r_{t-k} - \bar{r})}{(n-k)\,\sigma^2}$$

The *autocorrelation function* (*ACF*) is the vector of autocorrelation coefficients ρ_k.

The function `stats::acf()` calculates and plots the autocorrelation function of a time series.

The function `stats::acf()` has the drawback that it plots the lag zero autocorrelation (which is trivially equal to 1).

```
> # Open plot window under MS Windows
> x11(width=6, height=4)
> par(mar=c(3, 2, 1, 1), oma=c(1, 0, 0, 0))
> # Calculate VTI percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> retp <- drop(zoo::coredata(retp))
> # Plot autocorrelations of VTI returns using stats::acf()
> stats::acf(retp, lag=10, xlab="lag", main="")
> title(main="ACF of VTI Returns", line=-1)
> # Calculate two-tailed 95% confidence interval
> qnorm(0.975)/sqrt(NROW(retp))
```



The *VTI* time series of returns has small, but statistically significant negative autocorrelations.

The horizontal dashed lines are two-tailed confidence intervals of the autocorrelation estimator at 95% significance level: $\frac{\Phi^{-1}(0.975)}{\sqrt{n}}$.

But the visual inspection of the *ACF* plot alone is not enough to test whether autocorrelations are statistically significant or not.

Improved Autocorrelation Function

The function acf() has the drawback that it plots the lag zero autocorrelation (which is simply equal to 1).

Inspection of the data returned by acf() shows how to omit the lag zero autocorrelation.

The function acf() returns the *ACF* data invisibly, i.e. the return value can be assigned to a variable, but otherwise it isn't automatically printed to the console.

The function rutils::plot_acf() from package *rutils* is a wrapper for acf(), and it omits the lag zero autocorrelation.

```
> # Get the ACF data returned invisibly
> acfl <- acf(retp, plot=FALSE)
> summary(acfl)
> # Print the ACF data
> print(acfl)
> dim(acfl$acf)
> dim(acfl$lag)
> head(acfl$acf)
```

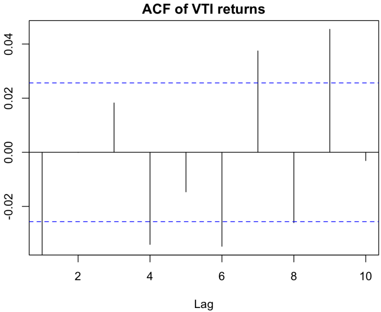
```
> plot_acf <- function(xtsv, lagg=10, plotobj=TRUE,
+                xlab="Lag", ylab="", main="", ...) {
+   # Calculate the ACF without a plot
+   acfl <- acf(x=xtsv, lag.max=lagg, plot=FALSE, ...)
+   # Remove first element of ACF data
+   acfl$acf <- array(data=acfl$acf[-1],
+     dim=c((dim(acfl$acf)[1]-1), 1, 1))
+   acfl$lag <- array(data=acfl$lag[-1],
+     dim=c((dim(acfl$lag)[1]-1), 1, 1))
+   # Plot ACF
+   if (plotobj) {
+     ci <- qnorm((1+0.95)/2)/sqrt(NROW(xtsv))
+     ylim <- c(min(-ci, range(acfl$acf[-1])),
+         max(ci, range(acfl$acf[-1])))
+     plot(acfl, xlab=xlab, ylab=ylab,
+    ylim=ylim, main="", ci=0)
+     title(main=main, line=0.5)
+     abline(h=c(-ci, ci), col="blue", lty=2)
+   }  # end if
+   # Return the ACF data invisibly
+   invisible(acfl)
+ }  # end plot_acf
```

Autocorrelations of Stock Returns

The *VTI* returns appear to have some small, yet significant negative autocorrelations at `lag=1`.

But the visual inspection of the *ACF* plot alone is not enough to test whether autocorrelations are statistically significant or not.

```
> # Autocorrelations of VTI returns
> rutils::plot_acf(retp, lag=10, main="ACF of VTI returns")
```



Ljung-Box Test for Autocorrelations of Time Series

The *Ljung-Box* test, tests if the autocorrelations of a time series are *statistically significant*.

The *null hypothesis* of the *Ljung-Box* test is that the autocorrelations are equal to zero.

The test statistic is:

$$Q = n(n+2) \sum_{k=1}^{maxlag} \frac{\hat{\rho}_k^2}{n-k}$$

Where `n` is the sample size, and the $\hat{\rho}_k$ are sample autocorrelations.

The *Ljung-Box* statistic follows the *chi-squared* distribution with *maxlag* degrees of freedom.

The *Ljung-Box* statistic is small for time series that have *statistically insignificant* autocorrelations.

The function `Box.test()` calculates the *Ljung-Box* test and returns the test statistic and its p-value.

```
> # Ljung-Box test for VTI returns
> # 'lag' is the number of autocorrelation coefficients
> Box.test(retp, lag=10, type="Ljung")
> # Ljung-Box test for random returns
> Box.test(rnorm(NROW(retp)), lag=10, type="Ljung")
> library(Ecdat)  # Load Ecdat
> macrodata <- as.zoo(Macrodat[, c("lhur", "fygm3")])
> colnames(macrodata) <- c("unemprate", "3mTbill")
> macrodiff <- na.omit(diff(macrodata))
> # Changes in 3 month T-bill rate are autocorrelated
> Box.test(macrodiff[, "3mTbill"], lag=10, type="Ljung")
> # Changes in unemployment rate are autocorrelated
> Box.test(macrodiff[, "unemprate"], lag=10, type="Ljung")
```

The *p*-value for *VTI* returns is small, and we conclude that the *null hypothesis* is `FALSE`, and that *VTI* returns do have some small autocorrelations.

The *p*-value for changes in econometric data is extremely small, and we conclude that the *null hypothesis* is `FALSE`, and that econometric data *are* autocorrelated.

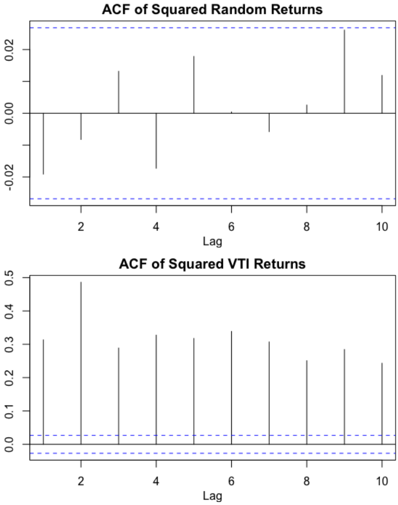
Autocorrelations of Squared VTI Returns

Squared random returns are not autocorrelated.

But squared *VTI* returns do have statistically significant autocorrelations.

The autocorrelations of squared asset returns are a very important feature.

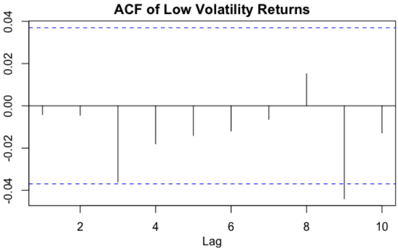
```
> # Open plot window under MS Windows
> x11(width=6, height=7)
> # Set two vertical plot panels
> par(mfrow=c(2,1))
> par(mar=c(3, 3, 2, 2), oma=c(0, 0, 0, 0), mgp=c(2, 1, 0))
> # Plot ACF of squared random returns
> rutils::plot_acf(rnorm(NROW(retp))^2, lag=10,
+  main="ACF of Squared Random Returns")
> # Plot ACF of squared VTI returns
> rutils::plot_acf(retp^2, lag=10,
+  main="ACF of Squared VTI Returns")
> # Ljung-Box test for squared VTI returns
> Box.test(retp^2, lag=10, type="Ljung")
```

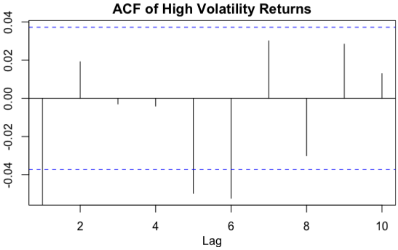


Autocorrelations in Intervals of Low and High Volatility

Stock returns have significant negative autocorrelations in time intervals with high volatility, but much less in time intervals with low volatility.

```
> # Calculate the weekly end points
> endd <- rutils::calc_endpoints(retp, interval="weeks")
> npts <- NROW(endd)
> # Calculate the monthly VTI volatilities and their median volatil
> stdev <- sapply(2:npts, function(endp) {
+   sd(retp[endd[endp-1]:endd[endp]])
+ })  # end sapply
> medianv <- median(stdev)
> # Calculate the stock returns of low volatility intervals
> retlow <- lapply(2:npts, function(endp) {
+   if (stdev[endp-1] <= medianv)
+     retp[endd[endp-1]:endd[endp]]
+ })  # end lapply
> retlow <- rutils::do_call(c, retlow)
> # Calculate the stock returns of high volatility intervals
> rethigh <- lapply(2:npts, function(endp) {
+   if (stdev[endp-1] > medianv)
+     retp[endd[endp-1]:endd[endp]]
+ })  # end lapply
> rethigh <- rutils::do_call(c, rethigh)
> # Plot ACF of low volatility returns
> rutils::plot_acf(retlow, lag=10,
+  main="ACF of Low Volatility Returns")
> Box.test(retlow, lag=10, type="Ljung")
> # Plot ACF of high volatility returns
> rutils::plot_acf(rethigh, lag=10,
+  main="ACF of High Volatility Returns")
> Box.test(rethigh, lag=10, type="Ljung")
```



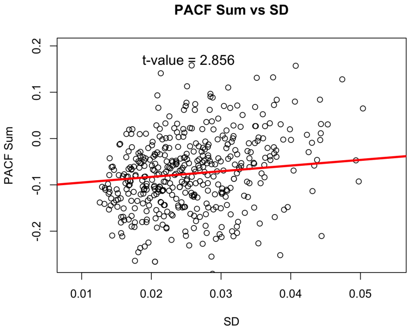


Autocorrelations of Low and High Volatility Stocks

Low volatility stocks have more significant negative autocorrelations than high volatility stocks.

The lowest volatility quantile of stocks has negative autocorrelations similar to *VTI*.

```
> # Load daily S&P500 stock returns
> load("/Users/jerzy/Develop/lecture_slides/data/sp500_returnstop.RI
> # Calculate the stock volatilities and the sum of the ACF
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> statm <- mclapply(retstock, function(retp) {
+   retp <- na.omit(retp)
+   # Calculate the sum of the ACF
+   acfsum <- sum(pacf(retp, lag=10, plot=FALSE)$acf)
+   # Calculate the Ljung-Box statistic
+   lbstat <- unname(Box.test(retp, lag=10, type="Ljung")$statistic)
+   c(stdev=sd(retp), acfsum=acfsum, lbstat=lbstat)
+ }, mc.cores=ncores)  # end mclapply
> statm <- do.call(rbind, statm)
> statm <- as.data.frame(statm)
> # Calculate the ACF sum for stock volatility quantiles
> confl <- seq(0.1, 0.9, 0.1)
> stdq <- quantile(statm[, "stdev"], confl)
> acfq <- quantile(statm[, "acfsum"], confl)
> plot(stdq, acfq, xlab="volatility", ylab="PACF Sum",
+      main="PACF Sum vs Volatility")
> # Compare the ACF sum for stock volatility quantile with VTI
> acfq[1]
> sum(pacf(na.omit(rutils::etfenv$returns$VTI), lag=10, plot=FALSE)
```



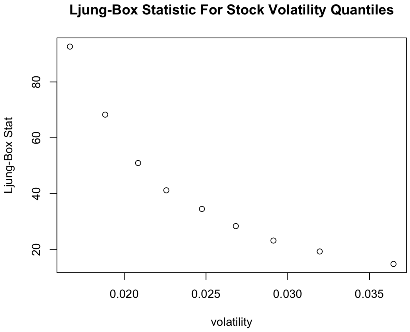

```
> # Scatter plot of the sum of the ACF vs the standard deviation
> regmod <- lm(acfsum ~ stdev, data=statm)
> plot(acfsum ~ stdev, data=statm, xlab="SD", ylab="PACF Sum",
+      xlim=c(0.5*min(stdq), 1.5*max(stdq)), ylim=c(1.5*min(acfq), 7
+      main="PACF Sum vs SD")
> abline(regmod, lwd=3, col="red")
> tvalue <- summary(regmod)$coefficients[2, "t value"]
> tvalue <- round(tvalue, 3)
> text(x=mean(stdq), y=6*max(acfq),
+      lab=paste("t-value =", tvalue), lwd=2, cex=1.2)
```

Ljung-Box Statistics of Low and High Volatility Stocks

A larger value of the *Ljung-Box* statistic means that the autocorrelations are more statistically significant.

The lowest volatility quantile of stocks has slightly more significant negative autocorrelations than *VTI* does.

```
> # Compare the Ljung-Box statistics for lowest volatility stocks wi
> lbstatq <- rev(quantile(statm[, "lbstat"], confl))
> lbstatq[1]
> Box.test(na.omit(rutils::etfenv$returns$VTI), lag=10, type="Ljung"
```



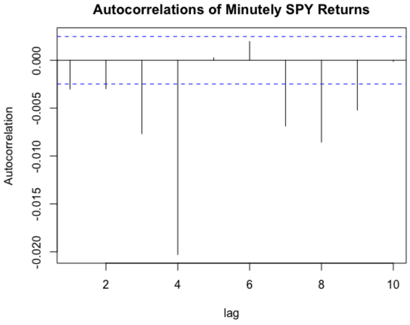

```
> # Plot Ljung-Box test statistic for volatility quantiles
> plot(stdq, lbstatq, xlab="volatility", ylab="Ljung-Box Stat",
+      main="Ljung-Box Statistic For Stock Volatility Quantiles")
```

Autocorrelations of High Frequency Returns

The package *HighFreq* contains three time series of intraday 1-minute *OHLC* price bars, called `SPY`, `TLT`, and `VXX`, for the *SPY*, *TLT*, and *VXX* ETFs.

Minutely *SPY* returns have statistically significant negative autocorrelations.

```
> # Calculate SPY log prices and percentage returns
> ohlc <- HighFreq::SPY
> ohlc[, 1:4] <- log(ohlc[, 1:4])
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> retp <- rutils::diffit(closep)
> colnames(retp) <- "SPY"
> # Open plot window under MS Windows
> x11(width=6, height=4)
> # Open plot window on Mac
> dev.new(width=6, height=4, noRStudioGD=TRUE)
> # Plot the autocorrelations of minutely SPY returns
> acfl <- rutils::plot_acf(as.numeric(retp), lag=10,
+      xlab="lag", ylab="Autocorrelation", main="")
> title("Autocorrelations of Minutely SPY Returns", line=1)
> # Calculate the sum of autocorrelations
> sum(acfl$acf)
```



Autocorrelations as Function of Aggregation Interval

For *minutely SPY* returns, the *Ljung-Box* statistic is large and its *p*-value is very small, so we can conclude that *minutely SPY* returns have statistically significant autocorrelations.

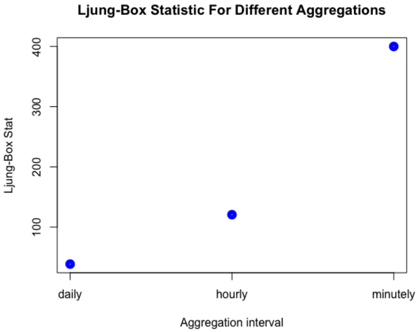
The level of the autocorrelations depends on the sampling frequency, with higher frequency returns having more significant negative autocorrelations.

SPY returns aggregated to longer time intervals are less autocorrelated.

As the returns are aggregated to a lower periodicity, they become less autocorrelated, with daily returns having almost insignificant autocorrelations.

The function `rutils::to_period()` aggregates an *OHLC* time series to a lower periodicity.

```
> # Ljung-Box test for minutely SPY returns
> Box.test(retp, lag=10, type="Ljung")
> # Calculate hourly SPY percentage returns
> closeh <- quantmod::Cl(xts::to.period(x=ohlc, period="hours"))
> retsh <- rutils::diffit(closeh)
> # Ljung-Box test for hourly SPY returns
> Box.test(retsh, lag=10, type="Ljung")
> # Calculate daily SPY percentage returns
> closed <- quantmod::Cl(xts::to.period(x=ohlc, period="days"))
> retd <- rutils::diffit(closed)
> # Ljung-Box test for daily SPY returns
> Box.test(retd, lag=10, type="Ljung")
```




```
> # Ljung-Box test statistics for aggregated SPY returns
> lbstat <- sapply(list(daily=retd, hourly=retsh, minutely=retp),
+   function(rets) {
+     Box.test(rets, lag=10, type="Ljung")$statistic
+ })  # end sapply
> # Plot Ljung-Box test statistic for different aggregation interval
> plot(lbstat, lwd=6, col="blue", xaxt="n",
+      xlab="Aggregation interval", ylab="Ljung-Box Stat",
+      main="Ljung-Box Statistic For Different Aggregations")
> # Add X-axis with labels
> axis(side=1, at=(1:3), labels=c("daily", "hourly", "minutely"))
```

Volatility as a Function of the Aggregation Interval

The estimated volatility σ scales as the *power* of the length of the aggregation time interval Δt:

$$\frac{\sigma_t}{\sigma} = \Delta t^H$$

Where H is the *Hurst* exponent, σ is the return volatility, and σ_t is the volatility of the aggregated returns.

If returns follow *Brownian motion* then the volatility scales as the *square root* of the length of the aggregation interval (`H = 0.5`).

If returns are *mean reverting* then the volatility scales slower than the *square root* (`H < 0.5`).

If returns are *trending* then the volatility scales faster than the *square root* (`H > 0.5`).

The length of the daily time interval is often approximated to be equal to `390 = 6.5*60` minutes, since the exchange trading session is equal to `6.5` hours, and daily volatility is dominated by the trading session.

The daily volatility is exaggerated by price jumps over the weekends and holidays, so it should be scaled.

The minutely volatility is exaggerated by overnight price jumps.

```
> # Daily SPY volatility from daily returns
> sd(retd)
> # Minutely SPY volatility scaled to daily interval
> sqrt(6.5*60)*sd(retp)
> # Minutely SPY returns without overnight price jumps (unit per sec
> retp <- retp/rutils::diffit(xts::.index(retp))
> retp[1] <- 0
> # Daily SPY volatility from minutely returns
> sqrt(6.5*60)*60*sd(retp)
> # Daily SPY returns without weekend and holiday price jumps (unit
> retd <- retd/rutils::diffit(xts::.index(retd))
> retd[1] <- 0
> # Daily SPY volatility without weekend and holiday price jumps
> 24*60*60*sd(retd)
```

The package *HighFreq* contains three time series of intraday 1-minute *OHLC* price bars, called `SPY`, `TLT`, and `VXX`, for the *SPY*, *TLT*, and *VXX* ETFs.

The function `rutils::to_period()` aggregates an *OHLC* time series to a lower periodicity.

The function `zoo::index()` extracts the time index of a time series.

The function `xts::.index()` extracts the time index expressed in the number of seconds.

Hurst Exponent From Volatility

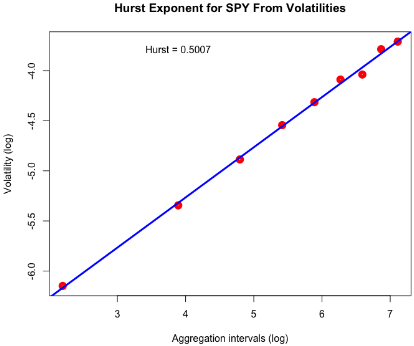
For a single aggregation interval, the *Hurst exponent* H is equal to:

$$H = \frac{\log \sigma_t - \log \sigma}{\log \Delta t}$$

For a vector of aggregation intervals Δt, the *Hurst exponent* H is equal to the regression slope between the logarithms of the aggregated volatilities σ_t versus the logarithms of the aggregation intervals Δt:

$$H = \frac{\mathrm{cov}(\log \sigma_t, \log \Delta t)}{\mathrm{var}(\log \Delta t)}$$

```
> # Calculate volatilities for vector of aggregation intervals
> aggv <- seq.int(from=3, to=35, length.out=9)^2
> volv <- sapply(aggv, function(agg) {
+   naggs <- nrows %/% agg
+   endd <- c(0, nrows - naggs*agg + (0:naggs)*agg)
+   # endd <- rutils::calc_endpoints(closep, interval=agg)
+   sd(rutils::diffit(closep[endd]))
+ })  # end sapply
> # Calculate the Hurst from single data point
> volog <- log(volv)
> agglog <- log(aggv)
> (last(volog) - first(volog))/(last(agglog) - first(agglog))
> # Calculate the Hurst from regression slope using formula
> hurstexp <- cov(volog, agglog)/var(agglog)
> # Or using function lm()
> regmod <- lm(volog ~ agglog)
> coef(regmod)[2]
```




```
> # Plot the volatilities
> x11(width=6, height=4)
> par(mar=c(4, 4, 2, 1), oma=c(1, 1, 1, 1))
> plot(volog ~ agglog, lwd=6, col="red",
+      xlab="Aggregation intervals (log)", ylab="Volatility (log)",
+      main="Hurst Exponent for SPY From Volatilities")
> abline(model, lwd=3, col="blue")
> text(agglog[2], volog[NROW(volog)-1],
+      paste0("Hurst = ", round(hurstexp, 4)))
```

Rescaled Range Analysis

The range $R_{\Delta t}$ of prices p_t over an interval Δt, is the difference between the highest attained price minus the lowest:

$$R_t = \max_{\Delta t} [p_\tau] - \min_{\Delta t} [p_\tau]$$

The *Rescaled Range* $RS_{\Delta t}$ is equal to the range $R_{\Delta t}$ divided by the standard deviation of the price differences σ_t: $RS_{\Delta t} = R_t / \sigma_t$.

The *Rescaled Range* $RS_{\Delta t}$ for a time series of prices is calculated by:

- Dividing the time series into non-overlapping intervals of length Δt,
- Calculating the *rescaled range* $RS_{\Delta t}$ for each interval,
- Calculating the average of the *rescaled ranges* $RS_{\Delta t}$ for all the intervals.

Rescaled Range Analysis (R/S) consists of calculating the average *rescaled range* $RS_{\Delta t}$ as a function of the length of the aggregation interval Δt.

```
> # Calculate cumulative SPY returns
> closep <- cumsum(retp)
> nrows <- NROW(closep)
> # Calculate the rescaled range
> agg <- 500
> naggs <- nrows %/% agg
> endd <- c(0, nrows - naggs*agg + (0:naggs)*agg)
> # Or
> # endd <- rutils::calc_endpoints(closep, interval=agg)
> rrange <- sapply(2:NROW(endd), function(np) {
+   indeks <- (endd[np-1]+1):endd[np]
+   diff(range(closep[indeks]))/sd(retp[indeks])
+ })  # end sapply
> mean(rrange)
> # Calculate the Hurst from single data point
> log(mean(rrange))/log(agg)
```

Hurst Exponent From Rescaled Range

The average *Rescaled Range* $RS_{\Delta t}$ is proportional to the length of the aggregation interval Δt raised to the power of the *Hurst exponent* H:

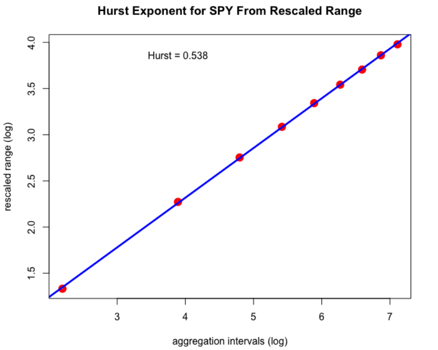
$$RS_{\Delta t} \propto \Delta t^H$$

So the *Hurst exponent* H is equal to:

$$H = \frac{\log RS_{\Delta t}}{\log \Delta t}$$

The Hurst exponents calculated from the *rescaled range* and the *volatility* are similar but not exactly equal because they use different methods to estimate price dispersion.

```
> # Calculate the rescaled range for vector of aggregation intervals
> rrange <- sapply(aggv, function(agg) {
+ # Calculate the end points
+   naggs <- nrows %/% agg
+   endd <- c(0, nrows - naggs*agg + (0:naggs)*agg)
+ # Calculate the rescaled ranges
+   rrange <- sapply(2:NROW(endd), function(np) {
+     indeks <- (endd[np-1]+1):endd[np]
+     diff(range(closep[indeks]))/sd(retp[indeks])
+   })  # end sapply
+   mean(na.omit(rrange))
+ })  # end sapply
> # Calculate the Hurst as regression slope using formula
> rangelog <- log(rrange)
> agglog <- log(aggv)
> hurstexp <- cov(rangelog, agglog)/var(agglog)
> # Or using function lm()
> regmod <- lm(rangelog ~ agglog)
> coef(regmod)[2]
```



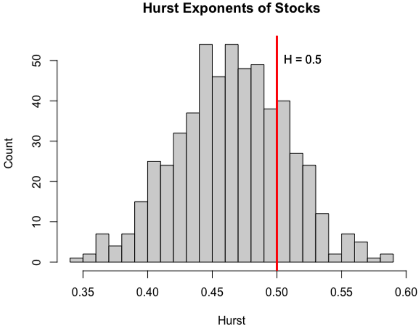

```
> plot(rangelog ~ agglog, lwd=6, col="red",
+      xlab="aggregation intervals (log)",
+      ylab="rescaled range (log)",
+      main="Hurst Exponent for SPY From Rescaled Range")
> abline(model, lwd=3, col="blue")
> text(agglog[2], rangelog[NROW(rangelog)-1],
+      paste0("Hurst = ", round(hurstexp, 4)))
```

Hurst Exponents of Stocks

The Hurst exponents of stocks are typically slightly less than 0.5, because their idiosyncratic risk components are mean-reverting.

The function `HighFreq::calc_hurst()` calculates the Hurst exponent in C++ using volatility ratios.

```
> # Load S&P500 constituent OHLC stock prices
> load("/Users/jerzy/Develop/lecture_slides/data/sp500.RData")
> class(sp500env$AAPL)
> head(sp500env$AAPL)
> # Calculate log stock prices after the year 2000
> pricev <- eapply(sp500env, function(ohlc) {
+   closep <- log(quantmod::Cl(ohlc)["2000/"])
+ # Ignore short lived and penny stocks (less than $1)
+   if ((NROW(closep) > 4000) & (last(closep) > 0))
+     return(closep)
+ })  # end eapply
> # Calculate the number of NULL prices
> sum(sapply(pricev, is.null))
> # Calculate the names of the stocks (remove NULL pricev)
> namev <- sapply(pricev, is.null)
> namev <- namev[!namev]
> namev <- names(namev)
> pricev <- pricev[namev]
> # Calculate the Hurst exponents of stocks
> aggv <- trunc(seq.int(from=3, to=10, length.out=5)^2)
> hurstv <- sapply(pricev, HighFreq::calc_hurst, aggv=aggv)
> # Dygraph of stock with largest Hurst exponent
> namev <- names(which.max(hurstv))
> dygraphs::dygraph(get(namev, pricev), main=namev)
> # Dygraph of stock with smallest Hurst exponent
> namev <- names(which.min(hurstv))
> dygraphs::dygraph(get(namev, pricev), main=namev)
```



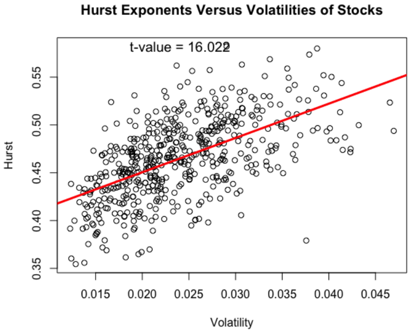

```
> # Plot a histogram of the Hurst exponents of stocks
> hist(hurstv, breaks=20, xlab="Hurst", ylab="Count",
+      main="Hurst Exponents of Stocks")
> # Add vertical line for H = 0.5
> abline(v=0.5, lwd=3, col='red')
> text(x=0.5, y=50, lab="H = 0.5", pos=4)
```

Stock Volatility and Hurst Exponents

There is a strong relationship between stock volatilities and Hurst exponents.

More volatile stocks tend to have larger Hurst exponents, closer to 0.5.

```
> # Calculate the volatility of stocks
> volv <- sapply(pricev, function(closep) {
+     sqrt(HighFreq::calc_var(HighFreq::diffit(closep)))
+ })  # end sapply
> # Dygraph of stock with highest volatility
> namev <- names(which.max(volv))
> dygraphs::dygraph(get(namev, pricev), main=namev)
> # Dygraph of stock with lowest volatility
> namev <- names(which.min(volv))
> dygraphs::dygraph(get(namev, pricev), main=namev)
> # Calculate the regression of the Hurst exponents versus volatilit
> regmod <- lm(hurstv ~ volv)
> summary(regmod)
```



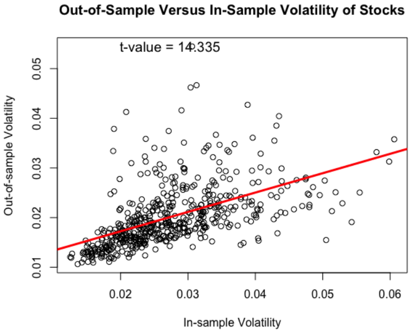

```
> # Plot scatterplot of the Hurst exponents versus volatilities
> plot(hurstv ~ volv, xlab="Volatility", ylab="Hurst",
+      main="Hurst Exponents Versus Volatilities of Stocks")
> # Add regression line
> abline(model, col='red', lwd=3)
> tvalue <- summary(regmod)$coefficients[2, "t value"]
> tvalue <- round(tvalue, 3)
> text(x=mean(volv), y=max(hurstv),
+      lab=paste("t-value =", tvalue), lwd=2, cex=1.2)
```

Out-of-Sample Volatility of Stocks

There is a strong relationship between *out-of-sample* and *in-sample* stock volatility.

Highly volatile stocks *in-sample* also tend to have high volatility *out-of-sample*.

```
> # Calculate the in-sample volatility of stocks
> volatis <- sapply(pricev, function(closep) {
+     sqrt(HighFreq::calc_var(HighFreq::diffit(closep["/2010"])))
+ })  # end sapply
> # Calculate the out-of-sample volatility of stocks
> volatos <- sapply(pricev, function(closep) {
+     sqrt(HighFreq::calc_var(HighFreq::diffit(closep["2010/"])))
+ })  # end sapply
> # Calculate the regression of the out-of-sample versus in-sample v
> regmod <- lm(volatos ~ volatis)
> summary(regmod)
```



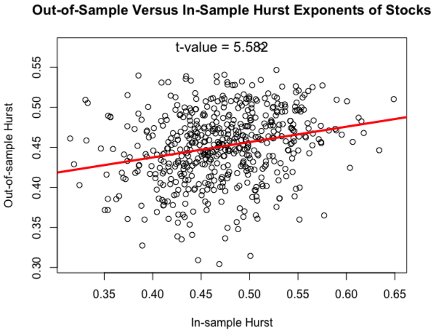

```
> # Plot scatterplot of the out-of-sample versus in-sample volatilit
> plot(volatos ~ volatis, xlab="In-sample Volatility", ylab="Out-of-
+      main="Out-of-Sample Versus In-Sample Volatility of Stocks")
> # Add regression line
> abline(model, col='red', lwd=3)
> tvalue <- summary(regmod)$coefficients[2, "t value"]
> tvalue <- round(tvalue, 3)
> text(x=mean(volatis), y=max(volatos),
+      lab=paste("t-value =", tvalue), lwd=2, cex=1.2)
```

Out-of-Sample Hurst Exponents of Stocks

The *out-of-sample* Hurst exponents of stocks have a significant positive correlation to the *in-sample* Hurst exponents.

That means that stocks with larger *in-sample* Hurst exponents tend to also have larger *out-of-sample* Hurst exponents (but not always).

This is because stock volatility persists *out-of-sample*, and Hurst exponents are larger for higher volatility stocks.




```
> # Calculate the in-sample Hurst exponents of stocks
> hurstis <- sapply(pricev, function(closep) {
+   HighFreq::calc_hurst(closep["/2010"], aggv=aggv)
+ })  # end sapply
> # Calculate the out-of-sample Hurst exponents of stocks
> hurstos <- sapply(pricev, function(closep) {
+   HighFreq::calc_hurst(closep["2010/"], aggv=aggv)
+ })  # end sapply
> # Calculate the regression of the out-of-sample versus in-sample Hurst exponents
> regmod <- lm(hurstos ~ hurstis)
> summary(regmod)
```

```
> # Plot scatterplot of the out-of-sample versus in-sample Hurst exp
> plot(hurstos ~ hurstis, xlab="In-sample Hurst", ylab="Out-of-sampl
+      main="Out-of-Sample Versus In-Sample Hurst Exponents of Stock
> # Add regression line
> abline(model, col='red', lwd=3)
> tvalue <- summary(regmod)$coefficients[2, "t value"]
> tvalue <- round(tvalue, 3)
> text(x=mean(hurstis), y=max(hurstos),
+      lab=paste("t-value =", tvalue), lwd=2, cex=1.2)
```

The Bid-Ask Spread

The *bid-ask spread* is the difference between the best ask (offer) price minus the best bid price in the market.

The *bid-ask spread* can be estimated from the differences between the execution prices of consecutive buy and sell market orders (roundtrip trades).

Market orders are orders to buy or sell a stock immediately at the best available price in the market. Market orders guarantee that the trade will be executed, but they do not guarantee the execution price. Market orders are subject to the *bid-ask spread*.

Limit orders are orders to buy or sell a stock at the limit price or better (the investor sets the limit price). Limit orders do not guarantee that the trade will be executed, but they guarantee the execution price. Limit orders are placed only for a certain time when they are "live".

Market orders are executed by matching them with live limit orders through a matching engine at an exchange.

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

The most liquid *SPY ETF* usually trades at a *bid-ask spread* of only one tick (cent=$0.01, or about 0.2 basis points).

In reality the *bid-ask spread* is not static and depends on many factors, such as market liquidity (trading volume), volatility, and the time of day.

```
> # Load the roundtrip trades
> dtable <- data.table::fread("/Users/jerzy/Develop/lecture_slides/d
> nrows <- NROW(dtable)
> class(dtable$timefill)
> # Sort the trades according to the execution time
> dtable <- dtable[order(dtable$timefill)]
> # Calculate the dollar bid-ask spread
> pricebuy <- dtable$price[dtable$side == "buy"]
> pricesell <- dtable$price[dtable$side == "sell"]
> bidask <- mean(pricebuy-pricesell)
> # Calculate the percentage bid-ask spread
> bidask/mean(pricesell)
```

Autocorrelations of Stock Returns

Daily stock returns often exhibit negative autocorrelations at one-day lags.

The *VTI* returns appear to have some small, yet significant negative autocorrelations at `lag=1`, and some positive autocorrelations at larger lags.

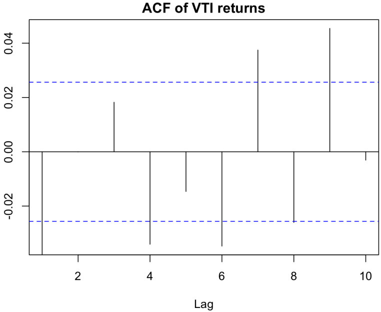
The *autocorrelation* of lag k of a time series of returns r_t is equal to:

$$\rho_k = \frac{\sum_{t=k+1}^{n} (r_t - \bar{r})(r_{t-k} - \bar{r})}{(n-k)\,\sigma^2}$$

The function `rutils::plot_acf()` calculates and plots the autocorrelations of a time series.

But the visual inspection of the *ACF* plot alone is not enough to test whether autocorrelations are statistically significant or not.

```
> # Calculate the daily VTI percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the autocorrelations of daily VTI returns
> rutils::plot_acf(retp, lag=10, main="ACF of VTI returns")
```



Daily Mean Reverting Strategy

The daily mean reverting strategy buys or sells short $1 of stock at the end of each day (depending on the sign of the previous daily return), and holds the position until the next day.

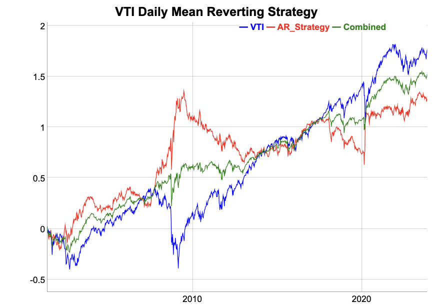
If the previous daily return was positive, it sells short $1 of stock. If the previous daily return was negative, it buys $1 of stock.

The mean reverting strategy has lower returns than *VTI*, but it has a very low correlation to *VTI*, so it has a positive *alpha*.

Thanks to its low correlation to *VTI*, the mean reverting strategy provides diversification of risk, and combined with *VTI*, a higher *Sharpe* ratio than *VTI* alone (it has a positive marginal *alpha*).

Strategies which have low or negative correlations to stocks, can contribute a significant marginal *alpha*, even if they have low returns.

```
> # Simulate the mean reverting strategy
> posv <- -rutils::lagit(sign(retp), lagg=1)
> pnls <- retp*posv
> # Subtract transaction costs from the pnls
> bidask <- 0.0001 # The bid-ask spread is equal to 1 basis point
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Calculate the strategy beta and alpha
> betac <- cov(pnls, retp)/var(retp)
> alphac <- mean(pnls) - betac*mean(retp)
```



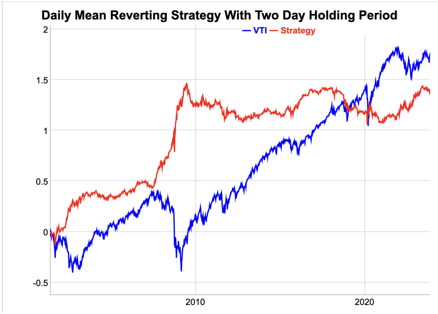

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI Daily Mean Reverting Strategy") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+
+   dyLegend(show="always", width=300)
```

Daily Mean Reverting Strategy With a Holding Period

The daily mean reverting strategy can be improved by combining the daily returns from the previous two days. This is equivalent to holding the position for two days, instead of rolling it daily.

The daily mean reverting strategy with a holding period performs better than the simple daily strategy because of risk diversification.

```
> # Simulate mean reverting strategy with two day holding period
> posv <- -rutils::roll_sum(sign(retp), lookb=2)/2
> pnls <- retp*rutils::lagit(posv)
```



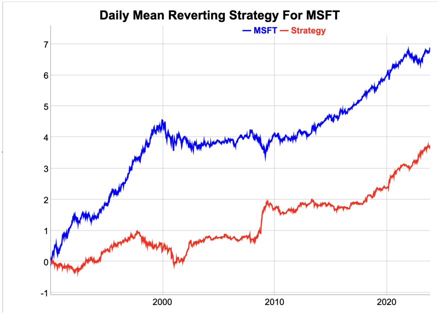

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Mean Reverting Strategy With Two Day Holding Period"
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Daily Mean Reverting Strategy For Stocks

Some daily stock returns exhibit stronger negative autocorrelations than ETFs.

But the daily mean reverting strategy doesn't perform well for many stocks.

```
> # Load daily S&P500 stock returns
> load(file="/Users/jerzy/Develop/lecture_slides/data/sp500_returns.
> retp <- na.omit(retstock$MSFT)
> rutils::plot_acf(retp)
> # Simulate mean reverting strategy with two day holding period
> posv <- -rutils::roll_sum(sign(retp), lookb=2)/2
> pnls <- retp*rutils::lagit(posv)
```



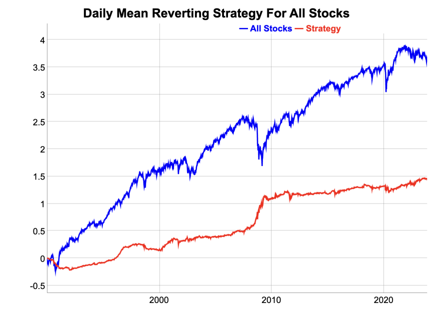

```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("MSFT", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Mean Reverting Strategy For MSFT") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Daily Mean Reverting Strategy For All Stocks

The combined daily mean reverting strategy for all *S&P500* stocks performed well prior to and during the 2008 financial crisis, but was flat afterwards.

Averaging the stock returns using the function `rowMeans()` with `na.rm=TRUE` is equivalent to rebalancing the portfolio so that stocks with `NA` returns have zero weight.

```
> # Simulate mean reverting strategy for all S&P500 stocks
> library(parallel)  # Load package parallel
> ncores <- detectCores() - 1
> pnll <- mclapply(retstock, function(retp) {
+   retp <- na.omit(retp)
+   posv <- -rutils::roll_sum(sign(retp), lookb=2)/2
+   retp*rutils::lagit(posv)
+ }, mc.cores=ncores)  # end mclapply
> pnls <- do.call(cbind, pnll)
> pnls <- rowMeans(pnls, na.rm=TRUE)
> # Calculate the average returns of all S&P500 stocks
> datev <- zoo::index(retstock)
> datev <- datev[-1]
> indeks <- rowMeans(retstock, na.rm=TRUE)
> indeks <- indeks[-1]
```

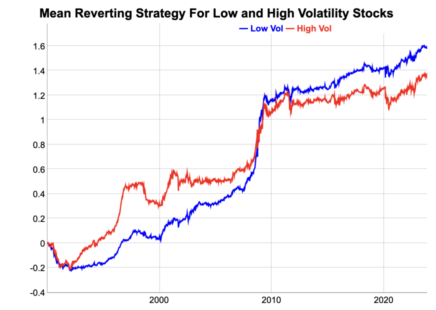



```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(indeks, pnls)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("All Stocks", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Daily Mean Reverting Strategy For All Stocks") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Mean Reverting Strategy For Low and High Volatility Stocks

The daily mean reverting strategy performs better for low volatility stocks than for high volatility stocks.

```
> # Calculate the stock volatilities
> volv <- mclapply(retstock, function(retp) {
+   sd(na.omit(retp))
+ }, mc.cores=ncores)  # end mclapply
> volv <- do.call(c, volv)
> # Calculate the median volatility
> medianv <- median(volv)
> # Calculate the pnls for low volatility stocks
> pnlovol <- do.call(cbind, pnll[volv < medianv])
> pnlovol <- rowMeans(pnlovol, na.rm=TRUE)
> # Calculate the pnls for high volatility stocks
> pnlhivol <- do.call(cbind, pnll[volv >= medianv])
> pnlhivol <- rowMeans(pnlhivol, na.rm=TRUE)
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(pnlovol, pnlhivol)
> wealthv <- xts::xts(wealthv, datev)
> colnames(wealthv) <- c("Low Vol", "High Vol")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Mean Reverting Strategy For Low and High Volatility Stocks
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

The EMA Mean-Reversion Strategy

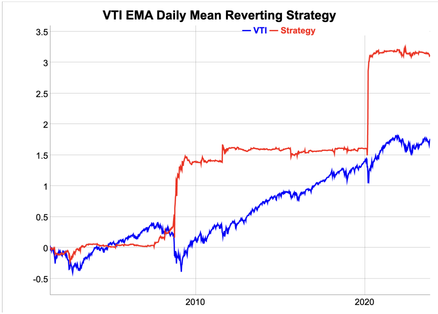
The *EMA* mean-reversion strategy holds either long stock positions or short positions proportional to minus the trailing *EMA* of past returns.

The strategy adjusts its stock position at the end of each day, just before the close of the market.

The strategy takes very large positions in periods of high volatility, when returns are large and highly anti-correlated.

The strategy makes profits mostly in periods of high volatility, but otherwise it's not very profitable.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> # Calculate the EMA returns recursively using C++ code
> retma <- HighFreq::run_mean(retp, lambda=0.1)
> # Calculate the positions and PnLs
> posv <- -rutils::lagit(retma, lagg=1)
> pnls <- retp*posv
> # Subtract transaction costs from the pnls
> bidask <- 0.0001 # The bid-ask spread is equal to 1 basis point
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI EMA Daily Mean Reverting Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

The EMA Mean-Reversion Strategy Scaled By Volatility

Dividing the returns by their trailing volatility reduces the effect of time-dependent volatility.

Scaling the returns by their trailing volatilities reduces the profits in periods of high volatility, but doesn't improve profits in periods of low volatility.

The function `HighFreq::run_var()` calculates the trailing mean and variance of the returns r_t, by recursively weighting the past variance estimates σ^2_{t-1}, with the squared differences of the returns minus their trailing means $(r_t - \bar{r}_t)^2$, using the decay factor λ:

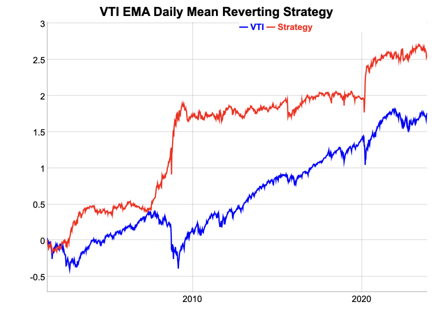
$$\bar{r}_t = \lambda \bar{r}_{t-1} + (1 - \lambda) r_t$$

$$\sigma^2_t = \lambda^2 \sigma^2_{t-1} + (1 - \lambda^2)(r_t - \bar{r}_t)^2$$

Where $\bar{r}_t$ and σ^2_t are the trailing mean and variance at time t.

The decay factor λ determines how quickly the mean and variance estimates are updated, with smaller values of λ producing faster updating, giving more weight to recent prices, and vice versa.

```
> # Calculate the EMA returns and volatilities
> volv <- HighFreq::run_var(retp, lambda=0.5)
> retma <- volv[, 1]
> volv <- sqrt(volv[, 2])
> # Scale the returns by their trailing volatility
> retsc <- ifelse(volv > 0, retp/volv, 0)
> # Calculate the positions and PnLs
> posv <- -rutils::lagit(retma, lagg=1)
> pnls <- retp*posv
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of mean reverting strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="VTI EMA Daily Mean Reverting Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Autoregressive Model of Stock Returns

The stock returns r_t can be fitted into an *autoregressive* model *AR(n)* with a constant intercept term φ_0:

$$r_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n} + \varepsilon_t$$

The *residuals* ε_t are assumed to be normally distributed, independent, and stationary.

The autoregressive model can be written in matrix form as:

$$\mathbf{r} = \varphi \, \mathbb{P} + \varepsilon$$

Where $\varphi = \{\varphi_0, \varphi_1, \varphi_2, \ldots \varphi_n\}$ is the vector of autoregressive coefficients.

The *autoregressive* model is equivalent to *multivariate* linear regression, with the *response* equal to the returns $\mathbf{r}$, and the columns of the *predictor matrix* $\mathbb{P}$ equal to the lags of the returns.

```
> # Calculate the VTI daily percentage returns
> retp <- na.omit(rutils::etfenv$returns$VTI)
> nrows <- NROW(retp)
> # Define the response and predictor matrices
> respv <- retp
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> # Add constant column for intercept coefficient phi0
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
```

Forecasting Stock Returns Using Autoregressive Models

The fitted autoregressive coefficients φ are equal to the *response* $\mathbf{r}$ multiplied by the inverse of the *predictor matrix* $\mathbb{P}$:

$$\varphi = \mathbb{P}^{-1}\,\mathbf{r}$$

The *in-sample* autoregressive forecasts of the returns are calculated by multiplying the predictor matrix by the fitted AR coefficients:

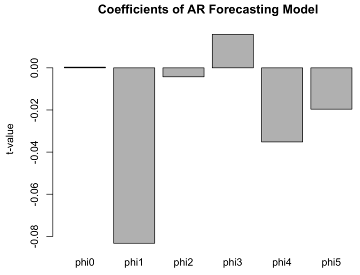
$$f_t = \varphi_0 + \varphi_1 r_{t-1} + \varphi_2 r_{t-2} + \ldots + \varphi_n r_{t-n}$$

For *VTI* returns, the intercept coefficient φ_0 has a small positive value, while the first autoregressive coefficient φ_1 has a small negative value.

This means that the autoregressive forecasting model is a combination of a static long stock position, plus a mean-reverting model which switches its stock position to the reverse of the previous day's return.

The function `MASS::ginv()` calculates the generalized inverse of a matrix.

```
> # Calculate the fitted autoregressive coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
```




```
> # Plot the AR coefficients
> coeffn <- paste0("phi", 0:(NROW(coeff)-1))
> barplot(coeff ~ coeffn, xlab="", ylab="t-value", col="grey",
+   main="Coefficients of AR Forecasting Model")
```

The t-values of the Autoregressive Coefficients

The forecast residuals are equal to the differences between the return forecasts minus the actual returns: $\varepsilon = f_t - r_t$

The variance of the autoregressive coefficients σ_φ^2 is equal to the variance of the forecast residuals σ_ε^2 divided by the squared *predictor matrix* $\mathbb{P}$:

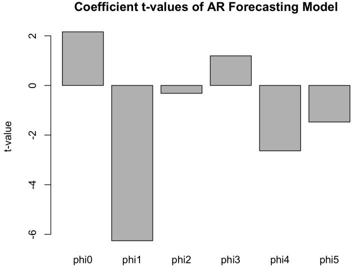
$$\sigma_\varphi^2 = \sigma_\varepsilon^2 (\mathbb{P}^T \mathbb{P})^{-1}$$

The t-values of the autoregressive coefficients are equal to the coefficient values divided by their volatilities:

$$\varphi_{tval} = \frac{\varphi}{\sigma_\varphi}$$

The intercept coefficient φ_0 and the first autoregressive coefficient φ_1 have statistically significant t-values.

```
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - respv)
> # The residuals are orthogonal to the predictors and the forecas
> round(cor(resids, fcasts), 6)
> round(sapply(predm[, -1], function(x) cor(resids, x)), 6)
> # Calculate the variance of the residuals
> varv <- sum(resids^2)/(nrows-NROW(coeff))
```




```
> # Calculate the predictor matrix squared
> pred2 <- crossprod(predm)
> # Calculate the covariance matrix of the AR coefficients
> covmat <- varv*MASS::ginv(pred2)
> coefsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coefsd)
> coeffn <- paste0("phi", 0:(NROW(coefft)-1))
> # Plot the t-values of the AR coefficients
> barplot(coefft ~ coeffn, xlab="", ylab="t-value", col="grey",
+   main="Coefficient t-values of AR Forecasting Model")
```

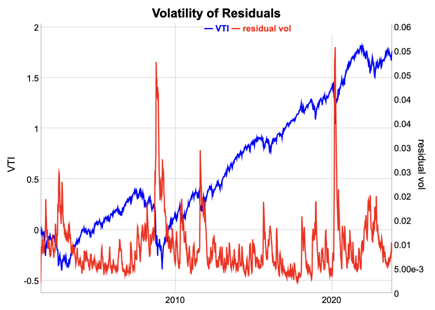
Residuals of Autoregressive Forecasting Model

The autoregressive model assumes stationary returns and residuals, with similar volatility over time.

In reality stock volatility is highly time dependent, so the volatility of the residuals is also time dependent.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Calculate the trailing volatility of the residuals
> residv <- sqrt(HighFreq::run_var(resids, lambda=0.9)[, 2])
```




```
> # Plot dygraph of volatility of residuals
> datav <- cbind(cumsum(retp), residv)
> colnames(datav) <- c("VTI", "residual vol")
> endd <- rutils::calc_endpoints(datav, interval="weeks")
> dygraphs::dygraph(datav[endd], main="Volatility of Residuals") %>%
+   dyAxis("y", label="VTI", independentTicks=TRUE) %>%
+   dyAxis("y2", label="residual vol", independentTicks=TRUE) %>%
+   dySeries(name="VTI", axis="y", strokeWidth=2, col="blue") %>%
+   dySeries(name="residual vol", axis="y2", strokeWidth=2, col="rec
+   dyLegend(show="always", width=300)
```

Autoregressive Strategy In-Sample

The first step in strategy development is optimizing it in-sample, even though in practice it can't be implemented. Because a strategy can't perform well out-of-sample if it doesn't perform well in-sample.

The autoregressive strategy invests dollar amounts of *VTI* stock proportional to the in-sample forecasts.

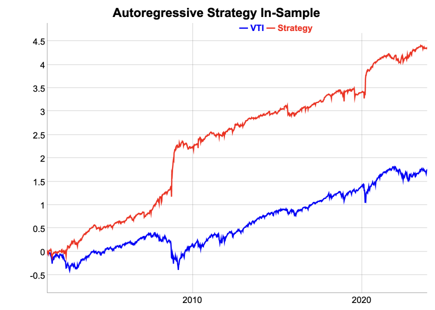
The in-sample autoregressive strategy performs well during periods of high volatility, but not as well in low volatility periods.

The dollar allocations of *VTI* stock are too large in periods of high volatility, which causes over-leverage and very high risk.

The leverage can be reduced by scaling (dividing) the forecasts by their trailing volatility.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Scale the forecasts by their volatility
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.2)[, 2])
> posv <- ifelse(fcastv > 0, fcasts/fcastv, 0)
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*posv
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

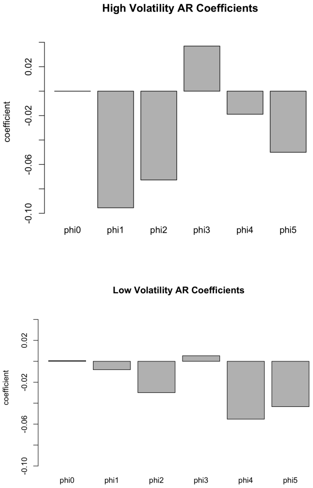
Autoregressive Coefficients in Periods of Low and High Volatility

The autoregressive model assumes stationary returns and residuals, with similar volatility over time. In reality stock volatility is highly time dependent.

The autoregressive coefficients in periods of high volatility are very different from those under low volatility.

In periods of high volatility, there are larger negative autocorrelations than in low volatility.

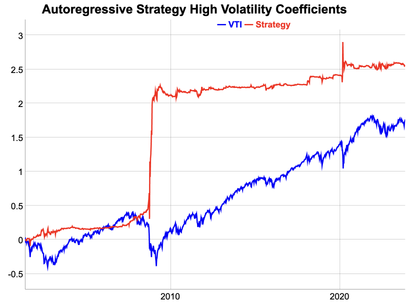
```
> # Calculate the high volatility AR coefficients
> respv <- retp["2008/2011"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffh <- drop(predinv %*% respv)
> coeffn <- paste0("phi", 0:(NROW(coeffh)-1))
> barplot(coeffh ~ coeffn, main="High Volatility AR Coefficients",
+   col="grey", xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
> # Calculate the low volatility AR coefficients
> respv <- retp["2012/2019"]
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(predm)), predm)
> predinv <- MASS::ginv(predm)
> coeffl <- drop(predinv %*% respv)
> barplot(coeffl ~ coeffn, main="Low Volatility AR Coefficients",
+   xlab="", ylab="coefficient", ylim=c(-0.1, 0.05))
```

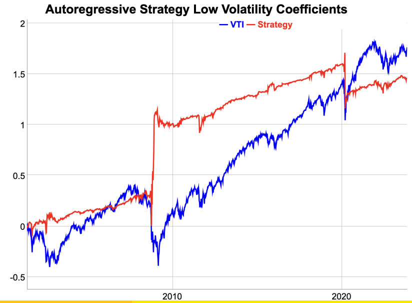


Performance of Low and High Volatility Autoregressive Coefficients

The autoregressive coefficients obtained from periods of high volatility are overfitted and only perform well in periods of high volatility. Similarly the low volatility coefficients.

```
> # Calculate the pnls for the high volatility AR coefficients
> predm <- lapply(1:orderp, rutils::lagit, input=retp)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> fcasts <- predm %*% coeffh
> pnlh <- retp*fcasts
> pnlh <- pnlh*sd(retp[retp<0])/sd(pnlh[pnlh<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnlh)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy High Volatility Coefficients") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
> # Calculate the pnls for the low volatility AR coefficients
> fcasts <- predm %*% coeffl
> pnll <- retp*fcasts
> pnll <- pnll*sd(retp[retp<0])/sd(pnll[pnll<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnll)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Low Volatility Coefficients") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```





The *Winsor* Function

Some models produce very large dollar allocations, leading to large portfolio leverage (dollars invested divided by the capital).

The *winsor function* maps the *model weight* w into the dollar amount for investment. The hyperbolic tangent function can serve as a winsor function:

$$W(x) = \frac{\exp(\lambda w) - \exp(-\lambda w)}{\exp(\lambda w) + \exp(-\lambda w)}$$

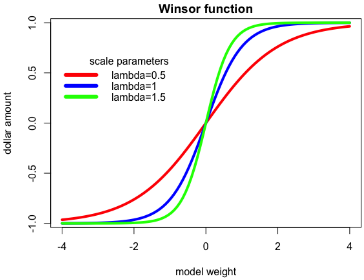
Where λ is the scale parameter.

The hyperbolic tangent is close to linear for small values of the *model weight* w, and saturates to $+1\$/-1\$$ for very large positive and negative values of the *model weight*.

The saturation effect limits (caps) the leverage in the strategy to $+1\$/-1\$$.

For very small values of the scale parameter λ, the invested dollar amount is linear for a wide range of *model weights*. So the strategy is mostly invested in dollar amounts proportional to the *model weights*.

For very large values of the scale parameter λ, the invested dollar amount jumps from $-1\$$ for negative *model weights* to $+1\$$ for positive *model weight* values. So the strategy is invested in either $-1\$$ or $+1\$$ dollar amounts.



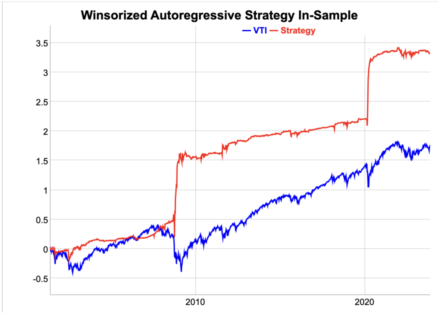

```
> lambdav <- c(0.5, 1, 1.5)
> colorv <- c("red", "blue", "green")
> # Define the winsor function
> winsorfun <- function(retp, lambdaf) tanh(lambdaf*retp)
> # Plot three curves in loop
> for (indeks in 1:3) {
+   curve(expr=winsorfun(x, lambda=lambdav[indeks]),
+ xlim=c(-4, 4), type="l", lwd=4,
+ xlab="model weight", ylab="dollar amount",
+ col=colorv[indeks], add=(indeks>1))
+ }  # end for
> # Add title and legend
> title(main="Winsor function", line=0.5)
> legend("topleft", title="scale parameters\n",
+    paste("lambdaf", lambdav, sep="="), inset=0.0, cex=1.0,
+    lwd=6, bty="n", y.intersp=0.3, lty=1, col=colorv)
```

Winsorized Autoregressive Strategy

The performance of the autoregressive strategy can be improved by fitting its coefficients using the *winsorized returns*, to reduce the effect of time-dependent volatility.

The performance can also be improved by *winsorizing* the forecasts, by reducing the leverage due to very large forecasts.

```
> # Winsorize the VTI returns
> retw <- winsorfun(retp/0.01, lambda=0.1)
> # Define the response and predictor matrices
> predm <- lapply(1:orderp, rutils::lagit, input=retw)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retw
> # Calculate the scaled in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.8)[, 2])
> fcastv[1:100] <- 1
> fcasts <- fcasts/fcastv
> # Winsorize the forecasts
> # fcasts <- winsorfun(fcasts/mad(fcasts), lambda=1.5)
```




```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Winsorized Autoregressive Strategy In-Sample") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

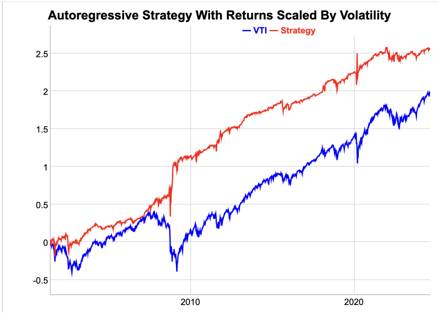
Autoregressive Strategy With Returns Scaled By Volatility

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by their trailing volatility.

Dividing the returns by their trailing volatility reduces the effect of time-dependent volatility.

The function `HighFreq::run_var()` calculates the trailing variance of a time series using exponential weights.

```
> # Scale the returns by their trailing volatility
> varv <- HighFreq::run_var(retp, lambda=0.99)[, 2]
> retsc <- ifelse(varv > 0, retp/sqrt(varv), 0)
> # Calculate the AR coefficients
> predm <- lapply(1:orderp, rutils::lagit, input=retsc)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% retsc
> # Calculate the scaled in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.8)[, 2])
> fcastv[1:100] <- 1
> fcasts <- fcasts/fcastv
```



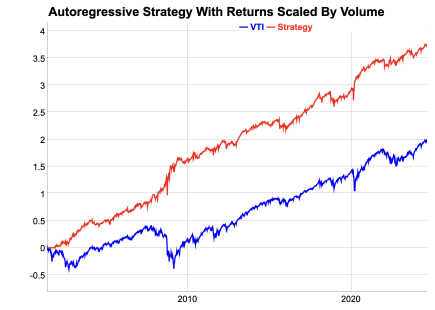

```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy With Returns Scaled By Volatility'
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Autoregressive Strategy in Trading Time

The performance of the autoregressive strategy can be improved by fitting its coefficients using returns divided by the trading volumes (returns in *trading time*).

Dividing the returns by the trading volumes reduces the effect of time-dependent volatility.

```
> # Calculate VTI returns and trading volumes
> ohlc <- rutils::etfenv$VTI
> datev <- zoo::index(ohlc)
> nrows <- NROW(ohlc)
> closep <- quantmod::Cl(ohlc)
> retp <- rutils::diffit(log(closep))
> volumv <- quantmod::Vo(ohlc)
> # Scale the returns using volume clock to trading time
> volumr <- HighFreq::run_mean(volumv, lambda=0.8)
> respv <- retp*volumr/volumv
> # Calculate the AR coefficients
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> # Calculate the scaled in-sample forecasts of VTI
> fcasts <- predm %*% coeff
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.8)[, 2])
> fcastv[1:100] <- 1
> fcasts <- fcasts/fcastv
```




```
> # Calculate the autoregressive strategy PnLs
> pnls <- retp*fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv["2010/"], function(x)
+     c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy With Returns Scaled By Volume") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

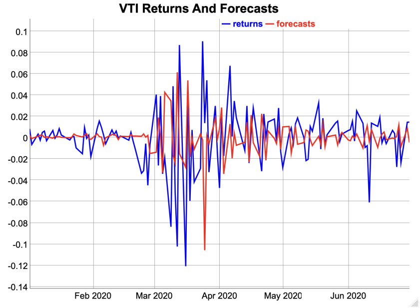
Mean Squared Error of the Autoregressive Forecasting Model

The accuracy of a forecasting model can be measured using the *mean squared error* and the *correlation*.

The mean squared error (*MSE*) of a forecasting model is the average of the squared forecasting errors ε_i, equal to the differences between the *forecasts* f_t minus the actual values r_t: $\varepsilon_i = f_t - r_t$:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (r_t - f_t)^2$$

```
> # Define the response and predictor matrices
> respv <- retp
> orderp <- 5
> predm <- lapply(1:orderp, rutils::lagit, input=respv)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, nrows), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> # Calculate the in-sample forecasts of VTI (fitted values)
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% respv
> fcasts <- predm %*% coeff
> # Calculate the correlation between forecasts and returns
> cor(fcasts, retp)
> # Calculate the forecasting errors
> errorf <- (fcasts - retp)
> # Mean squared error
> mean(errorf^2)
```




```
> # Plot the forecasts
> datav <- cbind(retp, fcasts)["2020-01/2020-06"]
> colnames(datav) <- c("returns", "forecasts")
> dygraphs::dygraph(datav,
+   main="VTI Returns And Forecasts") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

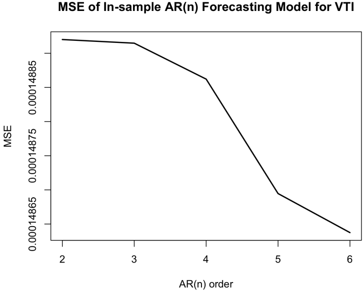
In-sample Order Selection of Autoregressive Forecasting

The mean squared errors (*MSE*) of the *in-sample* forecasts decrease steadily with the increasing order parameter *n* of the *AR(n)* forecasting model.

In-sample forecasting consists of first fitting an *AR(n)* model to the data, and calculating its coefficients.

The *in-sample* forecasts are calculated by multiplying the predictor matrix by the fitted AR coefficients.

```
> # Calculate the forecasts as a function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the fitted AR coefficients
+   predinv <- MASS::ginv(predm[, 1:ordern])
+   coeff <- predinv %*% respv
+   # Calculate the in-sample forecasts of VTI
+   drop(predm[, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcasts) <- paste0("n=", 2:NCOL(predm))
```




```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((respv - x)^2), cor=cor(respv, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcasts)
> # Plot forecasting MSE
> plot(x=2:NCOL(predm), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of In-sample AR(n) Forecasting Model for VTI")
```

Out-of-Sample Forecasting Using Autoregressive Models

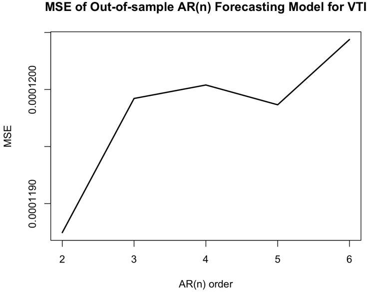
The mean squared errors (*MSE*) of the *out-of-sample* forecasts increase with the increasing order parameter *n* of the *AR(n)* model.

The reason for the increasing out-of-sample MSE is the *overfitting* of the coefficients to the training data for larger order parameters.

Out-of-sample forecasting consists of first fitting an *AR(n)* model to the training data, and calculating its coefficients.

The *out-of-sample* forecasts are calculated by multiplying the *out-of-sample* predictor matrix by the fitted AR coefficients.

```
> # Define in-sample and out-of-sample intervals
> nrows <- NROW(retp)
> insample <- 1:(nrows %/% 2)
> outsample <- (nrows %/% 2 + 1):nrows
> # Calculate the forecasts as a function of the AR order
> fcasts <- lapply(2:NCOL(predm), function(ordern) {
+   # Calculate the fitted AR coefficients
+   predinv <- MASS::ginv(predm[insample, 1:ordern])
+   coeff <- predinv %*% respv[insample]
+   # Calculate the out-of-sample forecasts of VTI
+   drop(predm[outsample, 1:ordern] %*% coeff)
+ })  # end lapply
> names(fcasts) <- paste0("n=", 2:NCOL(predm))
```




```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((respv[outsample] - x)^2), cor=cor(respv[outsample],
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- names(fcasts)
> # Plot forecasting MSE
> plot(x=2:NCOL(predm), y=mse[, 1],
+   xlab="AR(n) order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Out-of-sample AR(n) Forecasting Model for VTI")
```

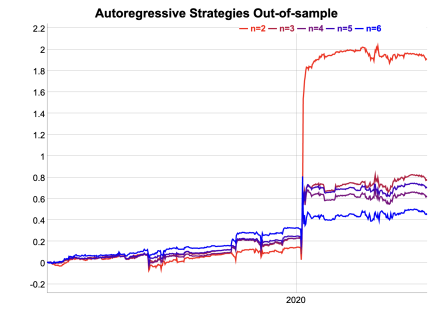
Out-of-Sample Autoregressive Strategy

The autoregressive strategy invests a dollar amount of *VTI* proportional to the AR forecasts.

The out-of-sample, risk-adjusted performance of the autoregressive strategy is better for a smaller order parameter *n* of the *AR(n)* model.

The optimal order parameter is `n = 2`, with a positive intercept coefficient φ_0 (since the average *VTI* returns were positive), and a negative coefficient φ_1 (because of strong negative autocorrelations in periods of high volatility).

Decreasing the order parameter of the autoregressive model is a form of *shrinkage* because it reduces the number of predictive variables.




```
> # Calculate the optimal AR coefficients
> predinv <- MASS::ginv(predm[insample, 1:2])
> coeff <- drop(predinv %*% respv[insample])
> # Calculate the out-of-sample PnLs
> pnls <- lapply(fcasts, function(fcast) {
+   pnls <- fcast*retp[outsample]
+   pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
+ })  # end lapply
> pnls <- rutils::do_call(cbind, pnls)
> colnames(pnls) <- names(fcasts)
```

```
> # Plot dygraph of out-of-sample PnLs
> colorv <- colorRampPalette(c("red", "blue"))(NCOL(pnls))
> colv <- colnames(pnls)
> endd <- rutils::calc_endpoints(pnls, interval="weeks")
> dygraphs::dygraph(cumsum(pnls)[endd],
+   main="Autoregressive Strategies Out-of-sample") %>%
+   dyOptions(colors=colorv, strokeWidth=2) %>%
+   dyLegend(width=300)
```

Out-of-Sample Autoregressive Forecasts

The autoregressive coefficients φ are equal to the in-sample *response* $\mathbf{r}$ times the inverse of the in-sample *predictor matrix* $\mathbb{P}$:

$$\varphi = \mathbb{P}^{-1} \mathbf{r}$$

The variance of the autoregressive coefficients σ^2_φ is equal to the variance of the in-sample forecast residuals σ^2_ε divided by the squared *predictor matrix* $\mathbb{P}$:

$$\sigma^2_\varphi = \sigma^2_\varepsilon (\mathbb{P}^T \mathbb{P})^{-1}$$

The t-values of the autoregressive coefficients are equal to the coefficient values divided by their volatilities:

$$\varphi_{tval} = \frac{\varphi}{\sigma_\varphi}$$

The *out-of-sample* autoregressive forecast f_t is equal to the single row of the predictor $\mathbb{P}_t$ times the fitted AR coefficients φ:

$$f_t = \varphi \mathbb{P}_t$$

The variance σ^2_f of the *forecast value* is equal to the inner product of the predictor $\mathbb{P}_t$ times the coefficient covariance matrix σ^2_φ:

$$\sigma^2_f = \mathbb{P}_t \, \sigma^2_\varphi \, \mathbb{P}^T_t$$

```
> # Define the look-back range
> lookb <- 100
> tday <- nrows
> startp <- max(1, tday-lookb)
> rangev <- startp:(tday-1)
> # Subset the response and predictors
> resps <- respv[rangev]
> preds <- predm[rangev]
> # Invert the predictor matrix
> predinv <- MASS::ginv(preds)
> # Calculate the fitted AR coefficients
> coeff <- predinv %*% resps
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- preds %*% coeff
> # Calculate the residuals (forecast errors)
> resids <- (fcasts - resps)
> # Calculate the variance of the residuals
> varv <- sum(resids^2)/(NROW(preds)-NROW(coeff))
> # Calculate the predictor matrix squared
> pred2 <- crossprod(preds)
> # Calculate the covariance matrix of the AR coefficients
> covmat <- varv*MASS::ginv(pred2)
> coefsd <- sqrt(diag(covmat))
> # Calculate the t-values of the AR coefficients
> coefft <- drop(coeff/coefsd)
> # Calculate the out-of-sample forecast
> predn <- predm[tday, ]
> fcast <- drop(predn %*% coeff)
> # Calculate the variance of the forecast
> varf <- drop(predn %*% covmat %*% t(predn))
> # Calculate the t-value of the out-of-sample forecast
> fcast/sqrt(varf)
```

Rolling Autoregressive Forecasting Model

The autoregressive coefficients can be calibrated dynamically over a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`lookb`).

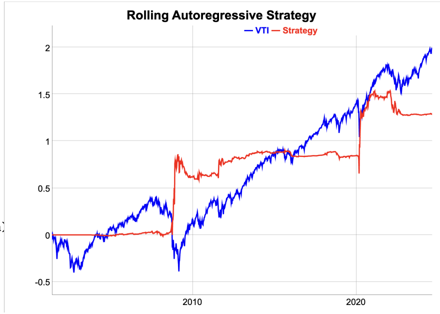
```
> # Perform rolling forecasting
> lookb <- 500
> fcasts <- parallel::mclapply(1:nrows, function(tday) {
+   if (tday > lookb) {
+     # Define the rolling look-back range
+     startp <- max(1, tday-lookb)
+     # startp <- 1 # Expanding look-back range
+     rangev <- startp:(tday-1) # In-sample range
+     # Subset the response and predictors
+     resps <- respv[rangev]
+     preds <- predm[rangev]
+     # Calculate the fitted AR coefficients
+     predinv <- MASS::ginv(preds)
+     coeff <- predinv %*% resps
+     # Calculate the in-sample forecasts of VTI (fitted values)
+     fcasts <- preds %*% coeff
+     # Calculate the residuals (forecast errors)
+     resids <- (fcasts - resps)
+     # Calculate the variance of the residuals
+     varv <- sum(resids^2)/(NROW(preds)-NROW(coeff))
+     # Calculate the covariance matrix of the AR coefficients
+     pred2 <- crossprod(preds)
+     covmat <- varv*MASS::ginv(pred2)
+     coefsd <- sqrt(diag(covmat))
+     coefft <- drop(coeff/coefsd) # t-values of the AR coefficients
+     # Calculate the out-of-sample forecast
+     predn <- predm[tday, ]
+     fcast <- drop(predn %*% coeff)
+     # Calculate the variance of the forecast
+     varf <- drop(predn %*% covmat %*% t(predn))
+     return(c(sd(resps), fcast=fcast, fstderr=sqrt(varf), coefft=co
+   } else {
+     return(c(volv=0, fcast=0, fstderr=0, coefft=rep(0, NCOL(predm)
+   } # end if
+ })  # end sapply
```

Rolling Autoregressive Strategy Performance

In the rolling autoregressive strategy, the autoregressive coefficients are calibrated on past data from a *rolling* look-back interval, and applied to calculating the *out-of-sample* forecasts.

The rolling autoregressive strategy performance depends on the length of the look-back interval.

```
> # Coerce fcasts to a time series
> fcasts <- t(fcasts)
> ncols <- NCOL(fcasts)
> colnames(fcasts) <- c("volv", "fcasts", "fstderr", colnames(predm)
> fcasts <- xts::xts(fcasts, zoo::index(retp))
> # Calculate the strategy PnLs
> pnls <- retp*fcasts$fcasts
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls)
> colnames(wealthv) <- c("VTI", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+ c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
```




```
> # Plot dygraph of the autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Rolling Autoregressive Strategy") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+ dyLegend(show="always", width=300)
```

Backtesting the Autoregressive Model

The *meta-parameters* of the *backtesting* function are the order *n* of the *AR(n)* model and the length of the look-back interval (`lookb`).

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

Backtesting is the simulation of a model on historical data to test its forecasting accuracy.

The autoregressive forecasting model can be *backtested* by calculating forecasts over either a *rolling* or an *expanding* look-back interval.

If the start date is fixed at the first row then the look-back interval is *expanding*.

The coefficients of the *AR(n)* process are fitted to past data, and then applied to calculating out-of-sample forecasts.

The *backtesting* procedure allows determining the optimal *meta-parameters* of the forecasting model: the order *n* of the *AR(n)* model and the length of look-back interval (`lookb`).

```
> # Define backtesting function
> sim_fcasts <- function(lookb=100, ordern=5, fixedlb=TRUE) {
+   # Perform rolling forecasting
+   fcasts <- sapply((lookb+1):nrows, function(tday) {
+     # Rolling look-back range
+     startp <- max(1, tday-lookb)
+     # Expanding look-back range
+     if (!fixedlb) {startp <- 1}
+     startp <- max(1, tday-lookb)
+     rangev <- startp:(tday-1) # In-sample range
+     # Subset the response and predictors
+     resps <- respv[rangev]
+     preds <- predm[rangev, 1:ordern]
+     # Invert the predictor matrix
+     predinv <- MASS::ginv(preds)
+     # Calculate the fitted AR coefficients
+     coeff <- predinv %*% resps
+     # Calculate the out-of-sample forecast
+     drop(predm[tday, 1:ordern] %*% coeff)
+   })  # end sapply
+   # Add warmup period
+   fcasts <- c(rep(0, lookb), fcasts)
+ }  # end sim_fcasts
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(lookb=100, ordern=5)
> c(mse=mean((fcasts - retp)^2), cor=cor(retp, fcasts))
```

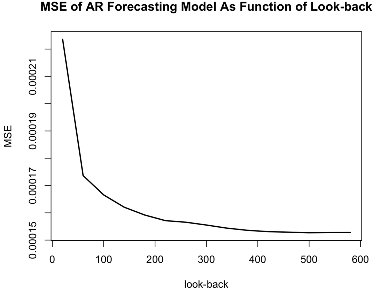
Forecasting Dependence On the Look-back Interval

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`lookb`).

The accuracy of the forecasting model increases with longer look-back intervals (`lookb`), because more data improves the estimates of the autoregressive coefficients.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> lookbv <- seq(20, 600, 40)
> fcasts <- parLapply(compclust, lookbv, sim_fcasts, ordern=6)
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(lookbv, sim_fcasts, ordern=6, mc.cores=ncores)
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- lookbv
> # Select optimal lookb interval
> lookb <- lookbv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=lookbv, y=mse[, 1],
+   xlab="look-back", ylab="MSE", type="l", lwd=2,
+   main="MSE of AR Forecasting Model As Function of Look-back")
```

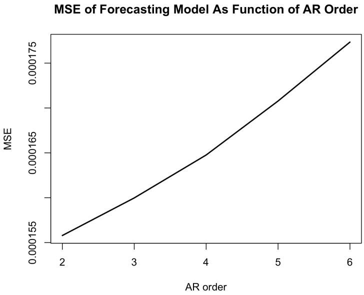
Order Dependence With Fixed Look-back

The *backtesting* function can be used to find the optimal *meta-parameters* of the autoregressive forecasting model.

The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model and on the length of the look-back interval (`lookb`).

The accuracy of the forecasting model decreases for larger AR order parameters, because of overfitting in-sample.




```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(compclust, orderv, sim_fcasts, lookb=lookb)
> stopCluster(compclust)  # Stop R processes over cluster under Wi
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   lookb=lookb, mc.cores=ncores)
```

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE of Forecasting Model As Function of AR Order")
```

Autoregressive Strategy With Fixed Look-back

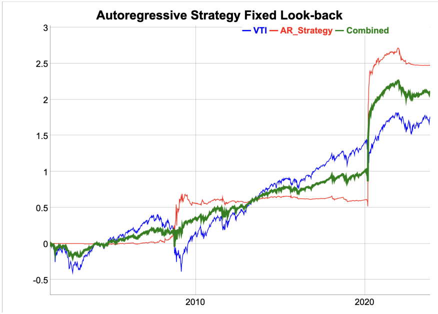
The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

Using the return forecasts as portfolio weights produces very large weights in periods of high volatility, and creates excessive risk.

To reduce excessive risk, a binary strategy can be used, with portfolio weights equal to the sign of the forecasts.

```
> # Simulate the rolling autoregressive forecasts
> fcasts <- sim_fcasts(lookb=lookb, ordern=ordern)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```




```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Fixed Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Order Dependence With Expanding Look-back

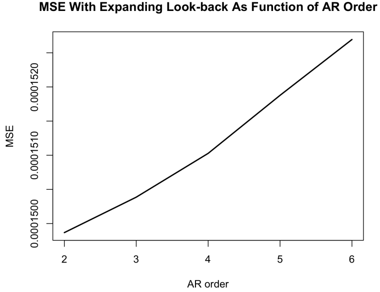
The two *meta-parameters* can be chosen by minimizing the *MSE* of the model forecasts in a *backtest* simulation.

The accuracy of the forecasting model depends on the order *n* of the *AR(n)* model.

Longer look-back intervals (`lookb`) are usually better for the autoregressive forecasting model.

The return forecasts are calculated just before the close of the markets, so that trades can be executed before the close.

```
> library(parallel)  # Load package parallel
> # Calculate the number of available cores
> ncores <- detectCores() - 1
> # Initialize compute cluster under Windows
> compclust <- makeCluster(ncores)
> # Perform parallel loop under Windows
> orderv <- 2:6
> fcasts <- parLapply(compclust, orderv, sim_fcasts,
+   lookb=lookb, fixedlb=FALSE)
> stopCluster(compclust)  # Stop R processes over cluster under Wi
> # Perform parallel bootstrap under Mac-OSX or Linux
> fcasts <- mclapply(orderv, sim_fcasts,
+   lookb=lookb, fixedlb=FALSE, mc.cores=ncores)
```



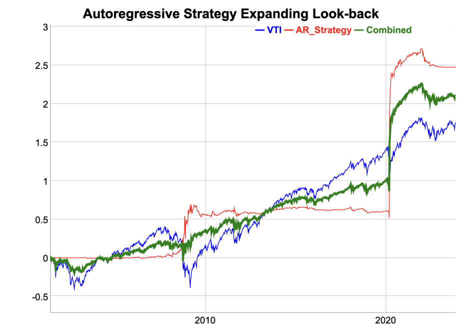

```
> # Calculate the mean squared errors
> mse <- sapply(fcasts, function(x) {
+   c(mse=mean((retp - x)^2), cor=cor(retp, x))
+ })  # end sapply
> mse <- t(mse)
> rownames(mse) <- orderv
> # Select optimal order parameter
> ordern <- orderv[which.min(mse[, 1])]
> # Plot forecasting MSE
> plot(x=orderv, y=mse[, 1],
+   xlab="AR order", ylab="MSE", type="l", lwd=2,
+   main="MSE With Expanding Look-back As Function of AR Order")
```

Autoregressive Strategy With Expanding Look-back

The model with an *expanding* look-back interval has better performance compared to the *fixed* look-back interval.

The autoregressive strategy returns are large in periods of high volatility, but much smaller in periods of low volatility. This because the forecasts are bigger in periods of high volatility, and also because the forecasts are more accurate, because the autocorrelations of stock returns are much higher in periods of high volatility.

```
> # Simulate the autoregressive forecasts with expanding look-back
> fcasts <- sim_fcasts(lookb=lookb, ordern=ordern, fixedlb=FALSE)
> # Calculate the strategy PnLs
> pnls <- fcasts*retp
> # Scale the PnL volatility to that of VTI
> pnls <- pnls*sd(retp[retp<0])/sd(pnls[pnls<0])
> wealthv <- cbind(retp, pnls, (retp+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
```



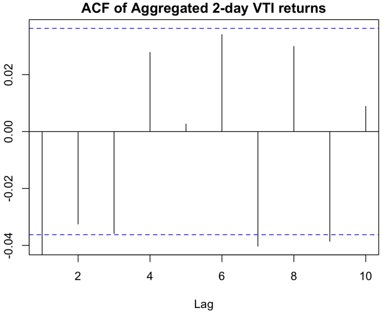

```
> # Annualized Sharpe ratios of VTI and AR strategy
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of AR strategy combined with VTI
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy Expanding Look-back") %>%
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dySeries(name="Combined", strokeWidth=3) %>%
+   dyLegend(show="always", width=300)
```

Autocorrelations of Aggregated Stock Returns

Stock returns aggregated over several days have much smaller autocorrelations.

But aggregating stock returns can reduce their noise and improve the performance of autoregressive strategies.

```
> # Calculate VTI returns over non-overlapping 2-day intervals
> pricev <- na.omit(rutils::etfenv$prices$VTI)
> reta <- rutils::diffit(log(pricev), lag=2)
> reta <- reta[2*(1:(NROW(pricev) %/% 2))]
> # Calculate the autocorrelations of daily VTI returns
> rutils::plot_acf(reta, lag=10, main="ACF of Aggregated 2-day VTI r
```



Autoregressive Strategy With Aggregated Stock Returns

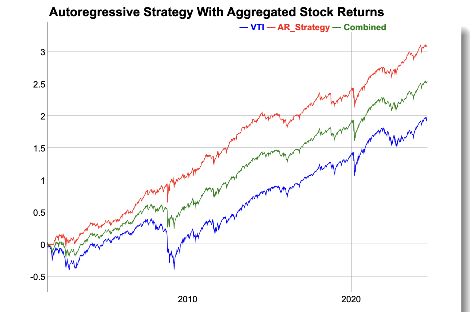
The autoregressive strategy with aggregated stock returns calculates the AR coefficients and the forecasts using stock returns aggregated over several days.

The strategy holds the stock position for several days, and rebalances the position at the end of the aggregation interval.

In this example, the returns are calculated over 2-day intervals, and the stock is held for 2 days. Effectively, there are two strategies, rebalancing every other day.

The aggregated autoregressive strategy has good performance both in periods of high volatility and in low volatility.

```
> # Define the response and predictor matrices
> reta <- rutils::diffit(log(pricev), lag=2)/2
> orderp <- 5
> predm <- lapply(2*(1:orderp), rutils::lagit, input=reta)
> predm <- rutils::do_call(cbind, predm)
> predm <- cbind(rep(1, NROW(reta)), predm)
> colnames(predm) <- c("phi0", paste0("lag", 1:orderp))
> # Calculate the AR coefficients
> predinv <- MASS::ginv(predm)
> coeff <- predinv %*% reta
> coeffn <- paste0("phi", 0:(NROW(coeff)-1))
> barplot(coeff ~ coeffn, xlab="", ylab="t-value", col="grey",
+   main="Coefficients of AR Forecasting Model")
> # Calculate the in-sample forecasts of VTI (fitted values)
> fcasts <- predm %*% coeff
> fcastv <- sqrt(HighFreq::run_var(fcasts, lambda=0.8)[, 2])
> fcastv[1:100] <- 1
> fcasts <- fcasts/fcastv
```




```
> # Calculate the autoregressive strategy PnLs
> pnls <- reta*fcasts
> pnls <- pnls*sd(reta[reta<0])/sd(pnls[pnls<0])
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(reta, pnls, (reta+pnls)/2)
> colnames(wealthv) <- c("VTI", "AR_Strategy", "Combined")
> cor(wealthv)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> sqrt(252)*sapply(wealthv["2010/"], function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the autoregressive strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Autoregressive Strategy With Aggregated Stock Returns") %>
+   dyOptions(colors=c("blue", "red", "green"), strokeWidth=1) %>%
+   dyLegend(show="always", width=300)
```

Daytime and Overnight Stock Strategies

The overnight stock strategy consists of holding a long position only overnight (buying at the market close and selling at the open the next day).

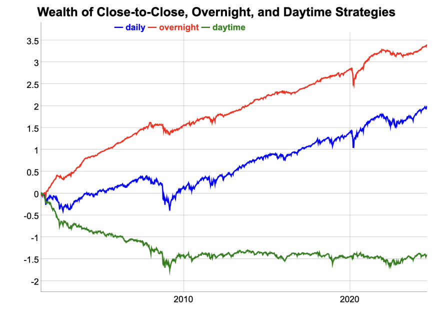
The daytime stock strategy consists of holding a long position only during the daytime (buying at the market open and selling at the close the same day).

The *Overnight Market Anomaly* is the consistent outperformance of overnight returns relative to the daytime returns.

The *Overnight Market Anomaly* has been observed for many decades for most stock market indices, but not always for all stock sectors.

The *Overnight Market Anomaly* is not as pronounced after the 2008–2009 financial crisis.

```
> # Calculate the log of OHLC VTI prices
> ohlc <- log(rutils::etfenv$VTI)
> nrows <- NROW(ohlc)
> openp <- quantmod::Op(ohlc)
> highp <- quantmod::Hi(ohlc)
> lowp <- quantmod::Lo(ohlc)
> closep <- quantmod::Cl(ohlc)
> # Calculate the close-to-close log returns,
> # the daytime open-to-close returns
> # and the overnight close-to-open returns.
> retp <- rutils::diffit(closep)
> colnames(retp) <- "daily"
> retd <- (closep - openp)
> colnames(retd) <- "daytime"
> reton <- (openp - rutils::lagit(closep, lagg=1, pad_zeros=FALSE))
> colnames(reton) <- "overnight"
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retp, reton, retd)
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of the Daytime and Overnight strategies
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Wealth of Close-to-Close, Overnight, and Daytime Strategie
+   dySeries(name="daily", strokeWidth=2, col="blue") %>%
+   dySeries(name="overnight", strokeWidth=2, col="red") %>%
+   dySeries(name="daytime", strokeWidth=2, col="green") %>%
+   dyLegend(width=600)
```

EMA Mean-Reversion Strategy For Daytime Returns

After the 2008−2009 financial crisis, the cumulative daytime stock index returns have been range-bound. So the daytime returns have more significant negative autocorrelations than overnight returns.

The *EMA* mean-reversion strategy holds stock positions equal to the *sign* of the trailing *EMA* of past returns.

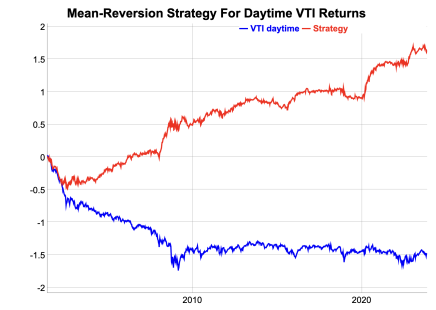
The strategy adjusts its stock position at the end of each day, just before the close of the market.

An alternative strategy holds positions proportional to minus of the sign of the trailing *EMA* of past returns.

The strategy takes very large positions in periods of high volatility, when returns are large and highly anti-correlated.

The limitation is that this strategy makes most of its profits in periods of high volatility, but otherwise it's profits are small.

```
> # Calculate the autocorrelations of daytime and overnight returns
> pacfl <- pacf(retd, lag.max=10, plot=FALSE)
> sum(pacfl$acf)
> pacfl <- pacf(reton, lag.max=10, plot=FALSE)
> sum(pacfl$acf)
> # Calculate the EMA returns recursively using C++ code
> retma <- HighFreq::run_mean(retd, lambda=0.4)
> # Calculate the positions and PnLs
> posv <- -rutils::lagit(sign(retma), lagg=1)
> pnls <- retd*posv
```




```
> # Calculate the pnls and the transaction costs
> bidask <- 0.0001 # The bid-ask spread is equal to 1 basis point
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI daytime", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+ + c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of crossover strategy
> endd <- rutils::calc_endpoints(wealthv, interval="weeks")
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Mean-Reversion Strategy For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Bollinger Strategy For Daytime Returns

The Bollinger *z-score* for daytime returns z_t, is equal to the difference between the cumulative returns $p_t = \sum r_t$ minus their trailing mean $\bar{p}_t$, divided by their volatility σ_t:

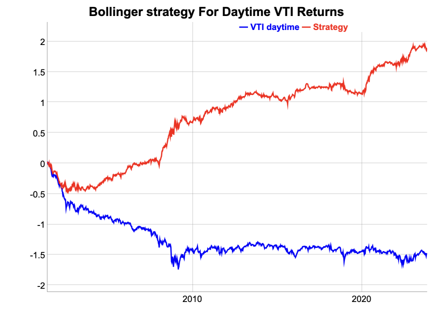
$$z_t = \frac{p_t - \bar{p}_t}{\sigma_t}$$

The Bollinger strategy switches to $1 dollar long stock if the *z-score* drops below the threshold of -1 (indicating the prices are cheap), and switches to -$1 dollar short if the *z-score* exceeds the threshold of 1 (indicating the prices are rich - expensive).

The Bollinger strategy is a *mean reverting* (contrarian) strategy because it bets on the cumulative returns reverting to their mean value.

The Bollinger strategy has performed well for daytime *VTI* returns because they exhibit significant mean-reversion.

```
> # Calculate the z-scores of the cumulative daytime returns
> retc <- cumsum(retd)
> lambdaf <- 0.24
> retm <- rutils::lagit(HighFreq::run_mean(retc, lambda=lambdaf))
> retv <- sqrt(rutils::lagit(HighFreq::run_var(retc, lambda=lambda
> zscores <- ifelse(retv > 0, (retc - retm)/retv, 0)
> # Calculate the positions from the Bollinger z-scores
> posv <- rep(NA_integer_, nrows)
> posv[1] <- 0
> posv <- ifelse(zscores > 1, -1, posv)
> posv <- ifelse(zscores < -1, 1, posv)
> posv <- zoo::na.locf(posv)
```




```
> # Calculate the pnls and the transaction costs
> pnls <- retd*posv
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
> # Calculate the Sharpe ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI daytime", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of daytime Bollinger strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+   main="Bollinger strategy For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Overnight Trend Strategy

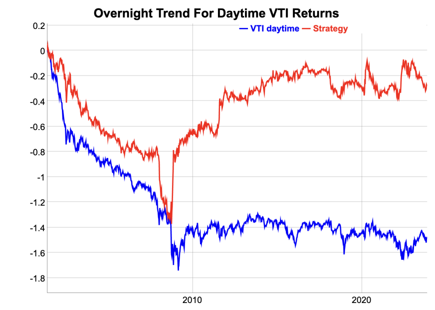
Analysts at *JPMorgan* and at the *Federal Reserve* have observed that there is a trend in the overnight returns.

Positive overnight returns are often followed by positive daytime returns, and vice versa.

If the overnight returns were positive, then the strategy buys \$1 dollar of stock at the market open and sells it at the market close, or if the overnight returns were negative then it shorts -\$1 dollar of stock.

The strategy has performed well immediately after the 2008-2009 financial crisis, but it has waned in recent years.

```
> # Calculate the pnls and the transaction costs
> posv <- sign(reton)
> pnls <- posv*retd
> costv <- 0.5*bidask*abs(rutils::diffit(posv))
> pnls <- (pnls - costv)
```




```
> # Calculate the Sharpe and Sortino ratios
> wealthv <- cbind(retd, pnls)
> colnames(wealthv) <- c("VTI daytime", "Strategy")
> sqrt(252)*sapply(wealthv, function(x)
+   c(Sharpe=mean(x)/sd(x), Sortino=mean(x)/sd(x[x<0])))
> # Plot dygraph of crossover strategy
> dygraphs::dygraph(cumsum(wealthv)[endd],
+ main="Overnight Trend For Daytime VTI Returns") %>%
+   dyOptions(colors=c("blue", "red"), strokeWidth=2) %>%
+   dyLegend(show="always", width=300)
```

Homework Assignment

Required

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