9
Rolling Analysis of Time Series

9.1 Introduction

A rolling analysis of a time series model is often used to assess the model's stability over time. When analyzing financial time series data using a statistical model, a key assumption is that the parameters of the model are constant over time. However, the economic environment often changes considerably, and it may not be reasonable to assume that a model's parameters are constant. A common technique to assess the constancy of a model's parameters is to compute parameter estimates over a rolling window of a fixed size through the sample. If the parameters are truly constant over the entire sample, then the estimates over the rolling windows should not be too different. If the parameters change at some point during the sample, then the rolling estimates should capture this instability.

Rolling analysis is commonly used to backtest a statistical model on historical data to evaluate stability and predictive accuracy. Backtesting generally works in the following way. The historical data is initially split into an estimation sample and a prediction sample. The model is then fit using the estimation sample and h-step ahead predictions are made for the prediction sample. Since the data for which the predictions are made are observed h-step ahead prediction errors can be formed. The estimation sample is then rolled ahead a given increment and the estimation and prediction exercise is repeated until it is not possible to make any more h-step predictions. The statistical properties of the collection of h-step ahead pre-

diction errors are then summarized and used to evaluate the adequacy of the statistical model.

Moving average methods are common in rolling analysis, and these methods lie at the heart of the technical analysis of financial time series. Moving averages typically use either equal weights for the observations or exponentially declining weights. One way to think of these simple moving average models is that they are a "poor man's" time varying parameter model. Sometimes simple moving average models are not adequate, however, and a general time varying parameter model is required. In these cases, the state space models discussed in Chapter 14 should be used.

This chapter describes various types of rolling analysis of financial time series using `S-PLUS`. Section 9.2 covers rolling descriptive statistics for univariate and bivariate time series with an emphasis on moving average techniques, and Section 9.3 gives a brief review of technical analysis using `S+FinMetrics` functions. Section 9.4 discusses rolling regression using the `S+FinMetrics` function `rollOLS` and illustrates how `rollOLS` may be used for backtesting regression models. Section 9.5 describes rolling analysis of general models using the `S+FinMetrics` function `roll`.

Rolling analysis of financial time series is widely used in practice but the technique is seldom discussed in textbook treatments of time series analysis. Notable exceptions are Alexander (2001) and Dacorogna et. al. (2001). Rolling analysis techniques in finance are generally discussed in the technical analysis literature, but the statistical properties of backtesting technical analysis are rarely addressed. A comprehensive treatment of technical analysis indicators is given in Colby and Meyers (1988) and a critical evaluation of technical analysis is provided in Bauer and Dahlquist (1999). The econometric literature on evaluating the predictive accuracy of models through backtesting has matured over the last decade. The main reference is Diebold and Mariano (1995).

9.2 Rolling Descriptive Statistics

9.2.1 Univariate Statistics

Consider the analysis of a univariate time series y_t over a sample from $t = 1, \ldots, T$. Whether the mean and variance (or standard deviation) parameters of the distribution of y_t are constant over the entire sample is of interest. To assess parameter constancy, let n denote the width of a sub-sample or window and define the *rolling* sample means, variances and

standard deviations

$$\hat{\mu}_t(n) = \frac{1}{n}\sum_{i=0}^{n-1} y_{t-i} \tag{9.1}$$

$$\hat{\sigma}_t^2(n) = \frac{1}{n-1}\sum_{i=0}^{n-1}(y_{t-i} - \hat{\mu}_t(n))^2 \tag{9.2}$$

$$\hat{\sigma}_t(n) = \sqrt{\hat{\sigma}_t^2(n)} \tag{9.3}$$

for windows $t = n, \ldots T$. The rolling mean and variance estimates at time t with window width n are the usual sample estimates using the most recent n observations. Provided the windows are rolled through the sample one observation at a time, there will be $T - n + 1$ rolling estimates of each parameter. The rolling mean $\hat{\mu}_t(n)$ is sometime called a n-period *simple moving average*.

Computing Rolling Descriptive Statistics Using the S-PLUS Function aggregateSeries

Consider the monthly continuously compounded returns on Microsoft stock over the period February 1990 through January 2001 computed from the monthly closing prices in the S+FinMetrics "timeSeries" singleIndex.dat

```
> msft.ret = getReturns(singleIndex.dat[,"MSFT"])
> start(msft.ret)
[1] Feb 1990
> end(msft.ret)
[1] Jan 2001
> nrow(msft.ret)
[1] 132
```

24-month rolling mean and standard deviations may be computed easily using the S-PLUS function aggregateSeries[1]

```
> roll.mean = aggregateSeries(msft.ret,moving=24,adj=1,FUN=mean)
> roll.sd = aggregateSeries(msft.ret,moving=24,adj=1,FUN=stdev)
```

The arguments moving=24, adj=1 and FUN=mean(stdev) tell the S-PLUS function aggregateSeries to evaluate the mean (stdev) function on a rolling window of size 24 and to adjust the output positions to the end of each window. roll.mean and roll.sd are "timeSeries" objects containing 109 rolling estimates:

```
> class(roll.mean)
```

[1] aggregateSeries is the method function of the generic S-PLUS function aggregate for objects of class "timeSeries" and "signalSeries".

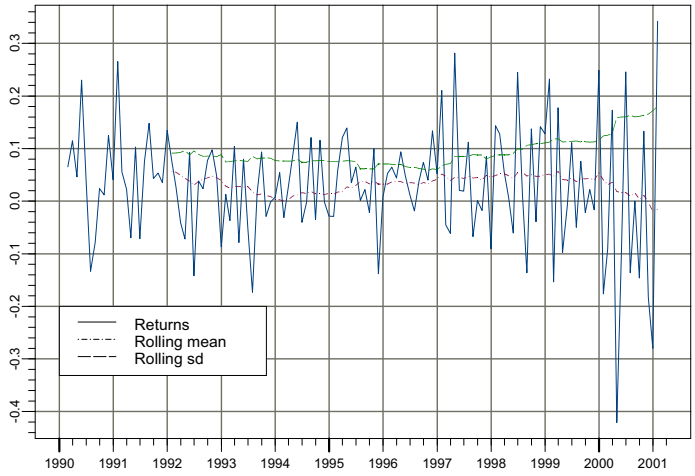

FIGURE 9.1. Monthly returns on Microsoft stock along with 24 month rolling means and standard deviations.

```
[1] "timeSeries"
> nrow(roll.mean)
[1] 109
> roll.mean[1:5]
 Positions       1
 Jan 1992  0.05671
 Feb 1992  0.05509
 Mar 1992  0.04859
 Apr 1992  0.04366
 May 1992  0.03795
```

The monthly returns along with the rolling means and standard deviations may be plotted together using

```
> plot(msft.ret,roll.mean,roll.sd,plot.args=list(lty=c(1,3,4)))
> legend(0,-0.2,legend=c("Returns","Rolling mean","Rolling sd"),
+ lty=c(1,3,4))
```

which is illustrated in Figure 9.1. The 24 month rolling estimates $\hat{\mu}_t(24)$ and $\hat{\sigma}_t(24)$ clearly vary over the sample. The rolling means start out around 5%, fall close to 0% in 1994, rise again to about 5% until 2000 and then fall below 0%. The rolling $\hat{\sigma}_t(24)$ values start out around 10%, fall slightly until 1997 and then begin to steadily rise for the rest of the sample. The

end of sample value of $\hat{\sigma}_t(24)$ is almost twice as big as the beginning of sample value.

The moving average estimates (9.1) - (9.3) are one-sided backward looking estimates. The S-PLUS function `aggregateSeries` may also compute two-sided asymmetric moving averages by specifying a value between 0 and 1 for the optional argument `adj`. For example, to compute a 24 month symmetric two-sided simple moving average set `adj=0.5`

```
> roll.mean.5 = aggregateSeries(msft.ret,moving=24,adj=0.5,
+ FUN=mean)
> roll.mean.5[1:5]
 Positions     MSFT
 Feb 1991  0.056708
 Mar 1991  0.055095
 Apr 1991  0.048594
 May 1991  0.043658
 Jun 1991  0.037950
```

Instead of computing the rolling means and standard deviations in separate calls to `aggregateSeries`, they can be computed in a single call by supplying a user-written function to `aggregateSeries` that simply returns the mean and standard deviation. One such function is

```
> mean.sd = function (x) {
>     tmp1 = mean(x)
>     tmp2 = stdev(x)
>     ans = concat(tmp1,tmp2)
>     ans
> }
```

The call to `aggregateSeries` to compute the rolling means and standard deviations is then

```
> roll.mean.sd = aggregateSeries(msft.ret,moving=24,adj=1,
+ FUN=mean.sd,colnames=c("mean","sd"))
> roll.mean.sd[1:5,]
 Positions    mean      sd
 Jan 1992  0.05671 0.09122
 Feb 1992  0.05509 0.09140
 Mar 1992  0.04859 0.09252
 Apr 1992  0.04366 0.09575
 May 1992  0.03795 0.08792
```

Notice that the column names of `roll.mean.sd` are specified using optional argument `colnames=c("mean","sd")` in the call to `aggregateSeries`.

Standard error bands around the rolling estimates of μ and σ may be computed using the asymptotic formulas

$$\widehat{\mathrm{SE}}(\hat{\mu}_t(n)) = \frac{\hat{\sigma}_t(n)}{\sqrt{n}},\ \widehat{\mathrm{SE}}(\hat{\sigma}_t(n)) = \frac{\hat{\sigma}_t(n)}{\sqrt{2n}}$$

Using the rolling estimates in `roll.mean.sd`, the `S-PLUS` commands to compute and plot the rolling estimates along with approximate 95% confidence bands are

```
> lower.mean = roll.mean.sd[,"mean"]-
+ 2*roll.mean.sd[,"sd"]/sqrt(24)
> upper.mean = roll.mean.sd[,"mean"]+
+ 2*roll.mean.sd[,"sd"]/sqrt(24)
> lower.sd = roll.mean.sd[,"sd"]-
+ 2*roll.mean.sd[,"sd"]/sqrt(2*24)
> upper.sd = roll.mean.sd[,"sd"]+
+ 2*roll.mean.sd[,"sd"]/sqrt(2*24)
> par(mfrow=c(2,1))
> plot(roll.mean.sd[,"mean"],lower.mean,upper.mean,
+ main="24 month rolling means",plot.args=list(lty=c(1,2,2)))
> plot(roll.mean.sd[,"sd"],lower.sd,upper.sd,
+ main="24 month rolling standard deviations",
+ plot.args=list(lty=c(1,2,2)))
```

Figure 9.2 shows the results. In general, the rolling $\hat{\sigma}_t(24)$ values are estimated much more precisely than the rolling $\hat{\mu}_t(24)$ values.

The rolling means, variances and standard deviations are not the only rolling descriptive statistics of interest, particularly for asset returns. For risk management purposes, one may be interested in extreme values. Therefore, one may want to compute rolling minima and maxima. These may be computed using `aggregateSeries` with `FUN=min` and `FUN=max`.

Computing Rolling Means, Variances, Maxima and Minima Using the S+FinMetrics Functions SMA, rollVar, rollMax and rollMin

The `S-PLUS` function `aggregateSeries` is extremely flexible but not efficient for computing rolling means, variances, maxima and minima. The `S+FinMetrics` functions `SMA` (simple moving average), `rollVar`, `rollMax` and `rollMin` implement efficient algorithms for computing rolling means, variances, maxima and minima. The arguments expected by these functions are

```
> args(SMA)
function(x, n = 9, trim = T, na.rm = F)
> args(rollVar)
function(x, n = 9, trim = T, unbiased = T, na.rm = F)
```

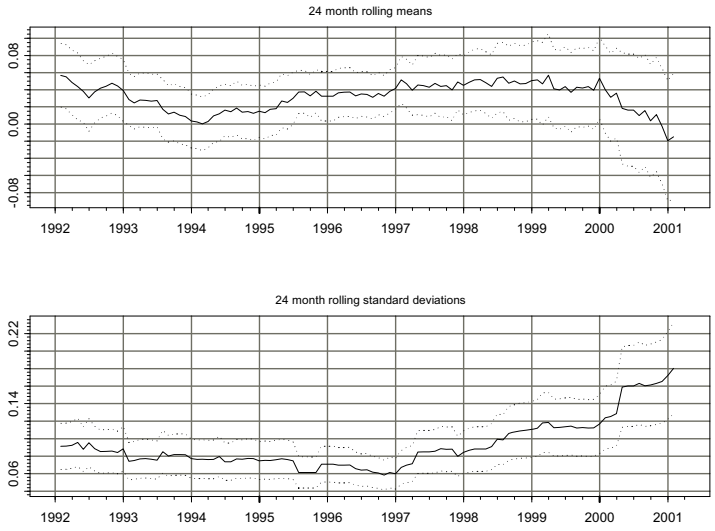

FIGURE 9.2. 24 month rolling estimates of $\hat{\mu}_t(24)$ and $\hat{\sigma}_t(24)$ for Microsoft with 95% confidence bands.

```
> args(rollMax)
function(x, n = 9, trim = T, na.rm = F)
> args(rollMin)
function(x, n = 9, trim = T, na.rm = F)
```

where `x` is a vector or univariate "`timeSeries`", `n` is the window width, `trim` determines if start-up values are trimmed from the output series and `na.rm` determines if missing values are to be removed. For `rollVar`, the option `unbiased=T` computes the unbiased variance estimator using $\frac{1}{n-1}$ as a divisor and `unbiased=F` computes the biased estimator using $\frac{1}{n}$.

To illustrate the use of `SMA`, `rollVar`, `rollMax` and `rollMin` 24 month rolling means, standard deviations, maxima and minima from the monthly returns on Microsoft are computed as

```
> roll2.mean = SMA(msft.ret,n=24)
> roll2.sd = sqrt(rollVar(msft.ret,n=24))
> roll.max = rollMax(msft.ret,n=24)
> roll.min = rollMin(msft.ret,n=24)
```

These estimates are identical to those computed using `aggregateSeries`, but the computation time required is much less. To compare the compu-

tation times in seconds within S-PLUS 6 for Windows the S-PLUS function dos.time may be used[2]

```
> dos.time(SMA(msft.ret,n=24))
[1] 0.05
> dos.time(aggregateSeries(msft.ret,moving=24,adj=1,FUN=mean))
[1] 4.23
> dos.time(sqrt(rollVar(msft.ret,n=24)))
[1] 0.06
> dos.time(aggregateSeries(msft.ret,moving=24,adj=1,FUN=stdev))
[1] 6.76
```

Example 49 *Computing rolling standard deviations from high frequency returns*

Rolling estimates of σ^2 and σ based on high frequency continuously compounded return data are often computed assuming the mean return is zero

$$\hat{\sigma}_t^2(n) = \frac{1}{n}\sum_{i=1}^{n} r_{t-i}^2$$

In this case the rolling estimates of σ^2 may be computed using the computationally efficient S+FinMetrics function SMA. For example, consider computing rolling estimates of σ based on the daily continuously compounded returns for Microsoft over the 10 year period from January 1991 through January 2001. The squared return data is computed from the daily closing price data in the S+FinMetrics "timeSeries" object DowJones30

```
> msft.ret2.d = getReturns(DowJones30[,"MSFT"],
+ type="continuous")^2
```

Rolling estimates of σ based on 25, 50 and 100 day windows are computed using SMA

```
> roll.sd.25 = sqrt(SMA(msft.ret2.d,n=25))
> roll.sd.50 = sqrt(SMA(msft.ret2.d,n=50))
> roll.sd.100 = sqrt(SMA(msft.ret2.d,n=100))
```

The rolling estimates $\hat{\sigma}_t(n)$ are illustrated in Figure 9.3 created using

```
> plot(roll.sd.25,roll.sd.50,roll.sd.100,
+ plot.args=(list(lty=c(1,3,4))))
> legend(0,0.055,legend=c("n=25","n=50","n=100"),
+ lty=c(1,3,4))
```

[2]The computations are carried out using S-PLUS 6 Professional Release 2 on a Dell Inspiron 3500 400MHz Pentium II with 96MB RAM.

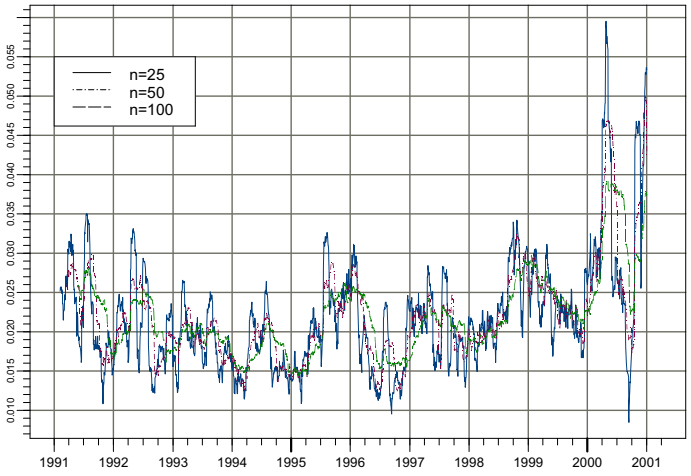

FIGURE 9.3. 25, 50 and 100 day rolling estimates of σ for the daily returns on Microsoft stock.

There is considerable variation in the rolling estimates of daily σ, and there appears to be a seasonal pattern with higher volatility in the summer months and lower volatility in the winter months.

9.2.2 Bivariate Statistics

Consider now the analysis of two univariate time series y_{1t} and y_{2t} over the sample from $t = 1, \ldots, T$. To assess if the covariance and correlation between y_{1t} and y_{2t} is constant over the entire sample the n-period rolling sample covariances and correlations

$$\begin{aligned}
\hat{\sigma}_{12,t}(n) &= \frac{1}{n-1}\sum_{i=0}^{n-1}(y_{1t-i} - \hat{\mu}_{1t}(n))(y_{2t-i} - \hat{\mu}_{2t}(n)) \\
\hat{\rho}_{12,t}(n) &= \frac{\hat{\sigma}_{12,t}(n)}{\hat{\sigma}_{1t}(n)\hat{\sigma}_{2t}(n)}
\end{aligned}$$

may be computed.

Example 50 *24 month rolling correlations between the returns on Microsoft and the S&P 500 index*

Consider the monthly continuously compounded returns on Microsoft stock and S&P 500 index over the period February 1990 through Jan-

uary 2001 computed from the monthly closing prices in the S+FinMetrics "timeSeries" object singleIndex.dat

```
> ret.ts = getReturns(singleIndex.dat,type="continuous")
> colIds(ret.ts)
[1] "MSFT"  "SP500"
```

The 24-month rolling correlations between the returns on Microsoft and S&P 500 index may be computed using the S-PLUS function aggregateSeries with a user specified function to compute the correlations. One such function is

```
> cor.coef = function(x) cor(x)[1,2]
```

The 24-month rolling correlations are then computed as

```
> smpl = positions(ret.ts)>=start(roll.cor)
> roll.cor = aggregateSeries(ret.ts,moving=24,together=T,
+ adj=1,FUN=cor.coef)
>  roll.cor[1:5]
 Positions      1
 Jan 1992  0.6549
 Feb 1992  0.6535
 Mar 1992  0.6595
 Apr 1992  0.6209
 May 1992  0.5479
```

In the call to aggregateSeries the argument together=T passes all of the columns of ret.ts to the function cor.coef instead of passing each column separately. The monthly returns on Microsoft and the S&P 500 index along with the rolling correlations are illustrated in Figure 9.4 which is created by

```
> par(mfrow=c(2,1))
> plot(ret.ts[smpl,],main="Returns on Microsoft and
+ S&P 500 index",
+ plot.args=list(lty=c(1,3)))
> legend(0,-0.2,legend=c("Microsoft","S&P 500"),
+ lty=c(1,3))
> plot(roll.cor,main="24-month rolling correlations")
```

At the beginning of the sample, the correlation between Microsoft and the S&P 500 is fairly high at 0.6. The rolling correlation declines steadily, hits

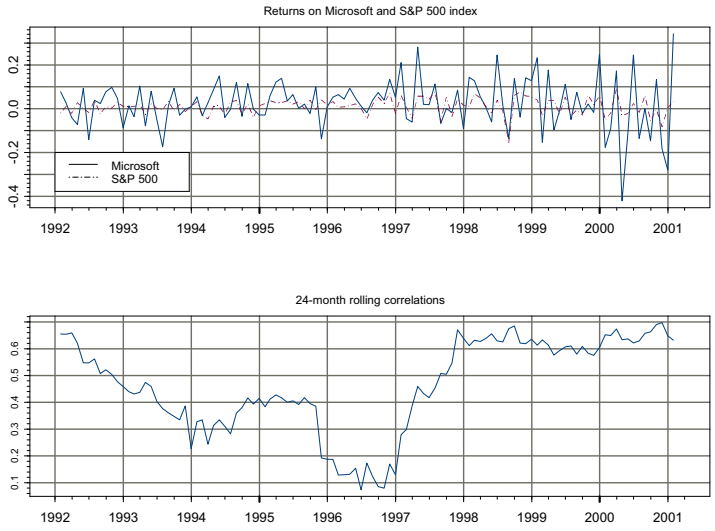

FIGURE 9.4. Returns on Microsoft and the S&P 500 index along with 24-month rolling correlations.

a low of about 0.1 at the beginning of 1997, then increases quickly to 0.6 and stabilizes at this value through the end of the sample[3].

9.2.3 Exponentially Weighted Moving Averages

The rolling descriptive statistics described in the previous sections are based on equally weighted moving averages of an observed time series y_t. Equally weighted moving averages are useful for uncovering periods of instability but may produce misleading results if used for short-term forecasting. This is because equally weighted averages are sensitive (not robust) to extreme values. To illustrate, consider $T = 100$ observations from a simulated time series $y_t \sim GWN(0, 1)$ with an outlier inserted at $t = 20$: i.e., $y_{20} = 10$. The data and rolling values $\hat{\mu}_t(10)$ and $\hat{\sigma}_t(10)$ are illustrated in Figure 9.5. Notice how the outlier at $t = 20$ inflates the rolling estimates $\hat{\mu}_t(10)$ and $\hat{\sigma}_t(10)$ for 9 periods.

[3]Approximate standard errors for the rolling correlations may be computed using

$$\widehat{SE}(\hat{\rho}_t(n)) = \sqrt{\frac{1 - \hat{\rho}_t(n)^2}{n}}$$

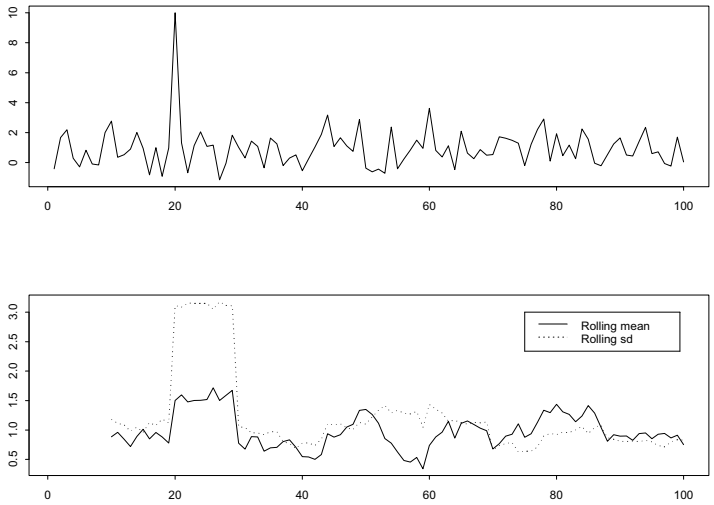

FIGURE 9.5. Effect of an outlier on equally weighted rolling estimates of μ and σ.

To mitigate the effects of extreme observations on moving average estimates the observations in a rolling window can be weighted differently. A common weighting scheme that puts more weight on the most recent observations is based on exponentially declining weights and the resulting weighted moving average is called an *exponentially weighted moving average* (EWMA). An n-period EWMA of a time series y_t is defined as

$$\tilde{\mu}_t(n) = \sum_{i=0}^{n-1} w_i \cdot y_{t-i}, \;\; w_i = \frac{\lambda^{i-1}}{\sum_{i=0}^{n-1} \lambda^{i-1}}$$

where $0 < \lambda < 1$ is the decay parameter. As $n \to \infty$, $\lambda^n \to 0$, $w_n \to 0$, and the EWMA converges to

$$\tilde{\mu}_t(\lambda) = (1 - \lambda) \sum_{i=0}^{\infty} \lambda^i y_{t-i} \tag{9.4}$$

so the EWMA may be defined independently of the window width n. The EWMA in (9.4) may be efficiently computed using the recursion

$$\tilde{\mu}_t(\lambda) = (1 - \lambda) y_t + \lambda \tilde{\mu}_{t-1}(\lambda) \tag{9.5}$$

From (9.5), it is clear that the closer λ is to one the more weight is put on the the previous period's estimate relative to the current period's observation.

Therefore, λ may be interpreted as a persistence parameter. The recursive formula (9.5) requires a starting value $\mu_0(\lambda)$. Common choices are the first observation and the average over a local window.

EWMA estimates of descriptive statistics for continuously compounded asset returns are usually computed using high frequency data with the assumption that the mean returns are zero. Accordingly, the EWMA estimates of σ^2 and σ_{12} are

$$\begin{aligned} \tilde{\sigma}_t^2(\lambda) &= (1-\lambda)r_t^2 + \lambda\tilde{\sigma}_{t-1}^2(\lambda) \\ \tilde{\sigma}_{12,t}(\lambda) &= (1-\lambda)r_{1t}r_{2t} + \lambda\tilde{\sigma}_{12,t-1}(\lambda) \end{aligned} \tag{9.6}$$

where r_t denotes the continuously compounded return on an asset. The EWMA estimate of volatility (9.6) is in the form of a IGARCH(1,1) model without an constant term.

Computing EWMA Estimates Using the `S+FinMetrics` Function `EWMA`

EWMA estimates based on (9.5) may be efficiently computed using the `S+FinMetrics` function `EWMA`. The arguments expected by `EWMA` are

```
> args(EWMA)
function(x, n = 9, lambda = (n - 1)/(n + 1), start =
"average", na.rm = F)
```

where `x` is the data input, `n` is a window width, `lambda` is the decay parameter and `start` specifies the starting value for the recursion (9.5). The implied default value for λ is 0.8. Valid choices for `start` are `"average"` and `"first"`. The use of `EWMA` is illustrated with the following examples.

Example 51 *Outlier example*

Consider again the outlier example data shown in Figure 9.5. EWMA estimates of μ for $\lambda = 0.95$, 0.75 and 0.5 are computed and plotted in Figure 9.6 using

```
> ewma95.mean = EWMA(e,lambda=0.95)
> ewma75.mean = EWMA(e,lambda=0.75)
> ewma50.mean = EWMA(e,lambda=0.5)
> tsplot(ewma95.mean,ewma75.mean,ewma50.mean)
> legend(60,4,legend=c("lamda=0.95","lamda=0.75",
+ "lamda=0.50"),lty=1:3)
```

Notice that the EWMA estimates with $\lambda = 0.95$, which put the most weight on recent observations, are only minimally affected by the one-time outlier whereas the EWMA estimates with $\lambda = 0.75$ and 0.5 increase sharply at the date of the outlier.

Example 52 *EWMA estimates of standard deviations and correlations from high frequency data*

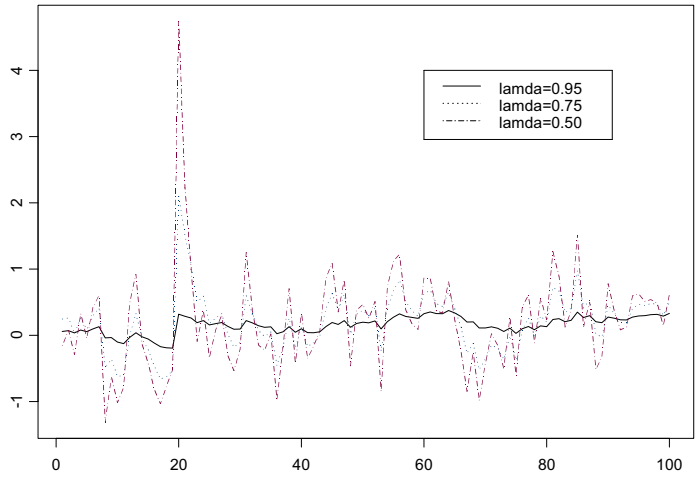

FIGURE 9.6. EWMA estimates of μ for outlier example data.

EWMA estimates of asset return standard deviations computed from high frequency data are commonly used as local or short-term estimates of volatility. Similarly, EWMA estimates of pairwise return correlations are often used to infer local interactions between assets. Indeed, J.P. Morgan's RiskMetrics® methodology is based on EWMA estimates of volatility and correlation. To illustrate, consider computing EWMA estimates of volatility and correlation with $\lambda = 0.95$ using daily closing price data on Microsoft and IBM stock over the five year period 1996-2000:

```
> smpl = (positions(DowJones30) >= timeDate("1/1/1996"))
> msft.ret.d = getReturns(DowJones30[smpl,"MSFT"])
> ibm.ret.d = getReturns(DowJones30[smpl,"IBM"])
> msft.ewma95.sd = sqrt(EWMA(msft.ret.d^2,lambda=0.95))
> ibm.ewma95.sd = sqrt(EWMA(ibm.ret.d^2,lambda=0.95))
> cov.ewma95 = EWMA(msft.ret.d*ibm.ret.d,lambda=0.95)
> cor.ewma95 = cov.ewma95/(msft.ewma95.sd*ibm.ewma95.sd)
> par(mfrow=c(2,1))
> plot(msft.ewma95.sd,ibm.ewma95.sd,
+ main="Rolling EWMA SD values",
+ plot.args=list(lty=c(1,3)))
> legend(0,0.055,legend=c("Microsoft","IBM"),lty=c(1,3))
> plot(cor.ewma95,main="Rolling EWMA correlation values")
```

Figure 9.7 shows the EWMA estimates of volatility and correlation. Daily

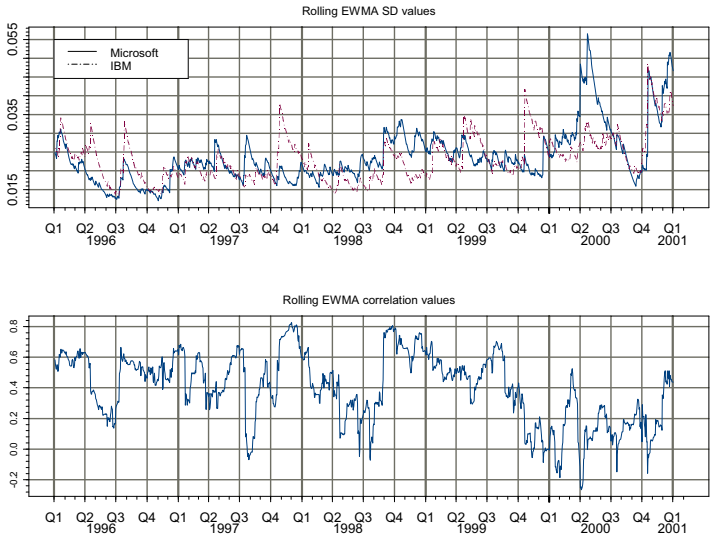

FIGURE 9.7. EWMA estimates of daily volatility and correlation with $\lambda = 0.95$.

volatility for Microsoft and IBM varies considerably, exhibiting apparent seasonality and an increasing trend. The daily correlations fluctuate around 0.5 for the first part of the sample and then drop to about 0.1 at the end of the sample.

9.2.4 Moving Average Methods for Irregularly Spaced High Frequency Data

The use of moving average and rolling methods on irregularly spaced or *inhomogeneous* high frequency time series data requires care. The moving average tools discussed so far are designed to work on regularly spaced or *homogeneous* time series data. Two approaches have been used to apply moving average methods to irregularly spaced data. The first approach converts the irregularly spaced data to regularly spaced data and then applies the tools for appropriate for regularly spaced data. The second approach, pioneered by Zumbach and Müller (2001), utilizes moving average methods specifically designed for irregularly spaced data.

Converting Inhomogeneous Time Series to Homogeneous Time Series

To illustrate the conversion of a inhomogeneous time series to a homogeneous time series, consider the transactions level data on 3M corporation

stock for December 1999 in the S+FinMetrics data frame highFreq3M.df. As in Chapter 2, a "timeSeries" object may be created using

```
> td = timeDate(julian=(highFreq3M.df$trade.day-1),
+ ms=highFreq3M.df$trade.time*1000,
+ in.origin=c(month=12,day=1,year=1999),zone="GMT")
> hf3M.ts = timeSeries(pos=td,data=highFreq3M.df)
> hf3M.ts[1:20,]
          Positions trade.day trade.time trade.price
 12/1/99 9:33:32 AM 1         34412      94.69
 12/1/99 9:33:34 AM 1         34414      94.69
 12/1/99 9:33:34 AM 1         34414      94.69
...
 12/1/99 9:34:45 AM 1         34485      94.69
 12/1/99 9:34:47 AM 1         34487      94.63
...
```

The trade time is measured in second from midnight. Notice that many of the first trades took place at the same price and that there are instances of multiple transactions at the same time. The analysis is limited to the first three trading days of December

```
> smpl = positions(hf3M.ts) < timeDate("12/4/1999")
> hf3M.ts = hf3M.ts[smpl,]
```

The data in hf3M.ts may be made homogeneous by use of an interpolation method to align the irregularly spaced time sequence and associated data to a regularly spaced time sequence. For example, consider creating a homogeneous time series of five minute observations. Since the data in hf3M.ts may not be recorded at all five minute intervals, some interpolation scheme must be used to create the data. Two common interpolation schemes are: *previous tick interpolation*, and *linear interpolation*. The former method uses the most recent values, and the latter method uses observations bracketing the desired time. The S-PLUS functions align may be used to perform these interpolation schemes.

The function align takes a "timeSeries" and a "timeDate" vector of new positions to align to. An easy way to create a "timeDate" sequence of five minute observations covering the trading hours for 3M stock is to use the S-PLUS function aggregateSeries as follows:

```
> tmp = aggregateSeries(hf3M.ts,by="minutes",k.by=5,FUN=mean)
```

The positions slot of "timeSeries" tmp contains the desired five minute "timeDate" sequence:

```
> positions(tmp[1:4])
 [1] 12/1/99 9:30:00 AM  12/1/99 9:35:00 AM
 [3] 12/1/99 9:40:00 AM  12/1/99 9:45:00 AM
```

To align the 3M price data to the five minute time sequence using previous tick interpolation use

```
> hf3M.5min = align(hf3M.ts[,"trade.price"], positions(tmp),
+ how="before")
> hf3M.5min[1:5,]
          Positions trade.price
 12/1/99 9:30:00 AM    NA
 12/1/99 9:35:00 AM 94.63
 12/1/99 9:40:00 AM 94.75
 12/1/99 9:45:00 AM 94.50
 12/1/99 9:50:00 AM 94.31
```

To align the price data using linear interpolation use

```
> hf3M.5min = align(hf3M.ts[,"trade.price"], positions(tmp),
+ how="interp")
> hf3M.5min[1:5,]
          Positions trade.price
 12/1/99 9:30:00 AM    NA
 12/1/99 9:35:00 AM 94.65
 12/1/99 9:40:00 AM 94.75
 12/1/99 9:45:00 AM 94.42
 12/1/99 9:50:00 AM 94.26
```

The usual methods for the analysis of homogeneous data may now be performed on the newly created data. For example, to compute and plot an EWMA of price with $\lambda = 0.9$ use

```
> hf3M.5min.ewma = EWMA(hf3M.5min,lambda=0.9,na.rm=T)
> plot(hf3M.5min.ewma)
```

The resulting plot is shown in Figure 9.8.

Inhomogeneous Moving Average Operators

Zumbach and Müller (2001) presented a general framework for analyzing inhomogeneous time series. A detailed exposition of this framework is beyond the scope of this book. Only a brief description of the most fundamental inhomogeneous time series operators is presented and reader is referred to Zumbach and Müller (2001) or Dacorogna et. al. (2001) for technical details and further examples.

Zumbach and Müller (2001) distinguished between *microscopic* and *macroscopic* operations on inhomogeneous time series. A microscopic operation depends on the actual sampling times of the time series, whereas a macroscopic operator extracts an average over a specified range. Macroscopic operations on high frequency inhomogeneous time series are advantageous because they are essentially immune to small variations in the individual

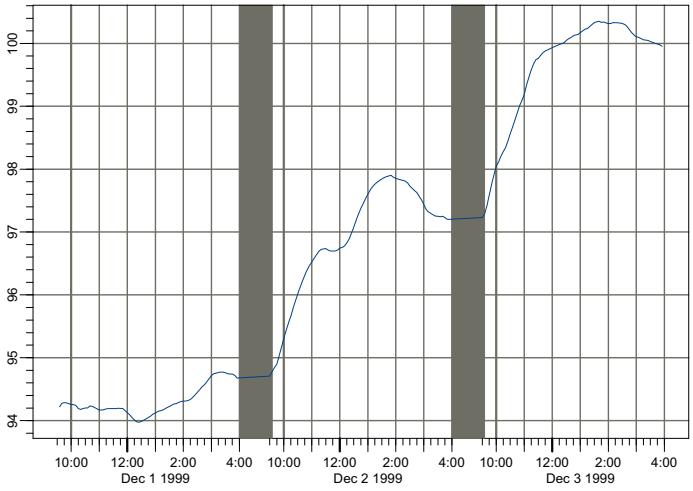

FIGURE 9.8. EWMA of five minute prices on 3M stock.

data observations and are better behaved and more robust than microscopic operations. The `S+FinMetrics` functions for analyzing inhomogeneous time series are based on a subset of the macroscopic operators discussed in Zumbach and Müller (2001). These functions are summarized in Table 9.1.

In general, given a continuous time signal $z(t)$, a macroscopic operator Ω can be defined as a *convolution* with a causal kernel $\omega(\cdot)$:

$$\Omega(t) = \int_0^t w(t-s)z(s)ds \tag{9.7}$$

for $t > 0$. Note that for a causal kernel $\omega(t) = 0$ for any $t < 0$, since future information cannot be utilized. In addition, it is usually required that

$$\int_0^\infty \omega(t)dt = 1$$

so that the operator can be interpreted as a weighted moving average of the signal $z(t)$. For example, the exponential moving average (EMA) operator is defined with an exponential kernel:

$$\omega(t) = \frac{e^{-t/\tau}}{\tau} \tag{9.8}$$

and it is easy to verify that

$$\int_0^\infty \frac{e^{-t/\tau}}{\tau} = 1$$

Function	Description
`iEMA`	Inhomogeneous EWMA
`iMA`	Inhomogeneous moving average
`iMNorm`	Inhomogeneous moving norm
`iMVar`	Inhomogeneous moving variance
`iMSD`	Inhomogeneous moving SD
`iMSkewness`	Inhomogeneous moving skewness
`iMKurtosis`	Inhomogeneous moving kurtosis
`iMCor`	Inhomogeneous moving correlation
`iDiff`	Inhomogeneous moving difference
`iEMA.kernel`	Kernel function for `iEMA`
`iMA.kernel`	Kernel function for `iMA`

TABLE 9.1. `S+FinMetrics` inhomogeneous time series function

The parameter τ can be shown to be the *range* of the EMA kernel.[4]

In reality, a time series signal is usually observed at discrete times. In addition, financial transactions level data are usually observed on irregular intervals. For the EMA operator, Zumbach and Müller suggest to use the following iterative formula to compute a discrete time approximation to (9.7):[5]

$$\mathrm{EMA}(t_n;\tau) = \mu\mathrm{EMA}(t_{n-1};\tau)+(1-\mu)z(t_n)+(\mu-\nu)[z(t_n)-z(t_{n-1})] \quad (9.9)$$

where

$$\mu = e^{-\alpha}, \nu = (1-\mu)/\alpha$$

and

$$\alpha = (t_n - t_{n-1}).$$

Note that when α is very small, $e^{\alpha} \approx 1+\alpha$ and it can be shown that $\mu \approx \nu$. In this case, the above formula reduces to the same iteration for evenly spaced EWMA operator.

Using the basic EMA operator, different operators can be constructed. For example, Zumbach and Müller suggested that the basic EMA operator can be iterated a finite number of times to obtain an operator with a different kernel, denoted $\mathrm{EMA}(\tau, k)$. The $\mathrm{EMA}(\tau, k)$ operator can be summed to obtain the analog of the moving average (MA) operator for inhomogeneous time series:

$$\mathrm{MA}(\tau, k) = \frac{1}{k}\sum_{i=1}^{k}\mathrm{EMA}(s, i)$$

where $s = 2\tau/(k+1)$ so that the range of $\mathrm{MA}(\tau, k)$ is equal to τ, independent of k.

[4]The range is defined as the first moment of the kernel, i.e., $\int_0^\infty \omega(t)tdt$.

[5]This formula is actually obtained by assuming linear interpolation between points. If previous tick interpolation is used, then $\nu = 1$.

The S+FinMetrics functions iEMA.kernel and iMA.kernel can be used to plot the kernel functions for EMA and MA operators, while iEMA and iMA can be used to compute the EMA and MA operator for inhomogeneous time series. For example, the following code plots the EMA(τ, k) and MA(τ, k) kernel functions for $\tau = 1$ and $k = 1, 2, \cdots, 10$:

```
> par(mfrow=c(2,1))
> knl = iEMA.kernel(1, 1)
> plot(knl, type="l", main="EMA Kernel")
> for(i in 2:10) {
>   knl = iEMA.kernel(1, i)
>   lines(knl, lty=i)
> }

> knl = iMA.kernel(1, 1)
> plot(knl, type="l", main="MA Kernel")
> for(i in 2:10) {
>   knl = iMA.kernel(1, i)
>   lines(knl, lty=i)
> }
> par(mfrow=c(1,1))
```

and the resulting plot is shown in Figure 9.9. From the figure, it can be seen that when $k = 1$, EMA(τ, k) and MA$(\tau, 1)$ are equivalent by definition. However, as k gets larger, the kernel function of EMA(τ, k) becomes flatter, while the kernel function of MA$(\tau, 1)$ becomes more like a rectangle. In fact, Zumbach and Müller showed that the range of EMA(τ, k) is $k\tau$, while the range of MA$(\tau, 1)$ becomes a constant for $t \leq 2\tau$ as $k \to \infty$. As a result, to obtain an MA operator with window width equal to 9 (which corresponds to a range of 8, i.e., using 8 observations in the past), one sets $\tau = 4$ and k to a large number:

```
> iMA(1:100, 4, iter=10)
  [1]  1.000  1.084  1.305  1.662  2.150  2.761  3.481
  [8]  4.289  5.166  6.091  7.047  8.024  9.011 10.005
 [15] 11.002 12.001 13.000 14.000 15.000 16.000 17.000
 [22] 18.000 19.000 20.000 21.000 22.000 23.000 24.000
 [29] 25.000 26.000 27.000 28.000 29.000 30.000 31.000
 [36] 32.000 33.000 34.000 35.000 36.000 37.000 38.000
 [43] 39.000 40.000 41.000 42.000 43.000 44.000 45.000
 [50] 46.000 47.000 48.000 49.000 50.000 51.000 52.000
 [57] 53.000 54.000 55.000 56.000 57.000 58.000 59.000
 [64] 60.000 61.000 62.000 63.000 64.000 65.000 66.000
 [71] 67.000 68.000 69.000 70.000 71.000 72.000 73.000
 [78] 74.000 75.000 76.000 77.000 78.000 79.000 80.000
 [85] 81.000 82.000 83.000 84.000 85.000 86.000 87.000
```

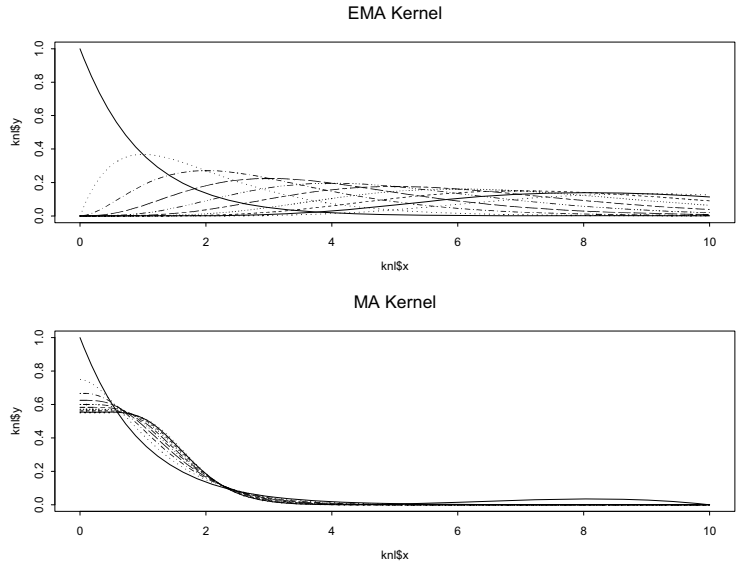

FIGURE 9.9. Kernel function for EMA and MA operators for inhomogeneous time series.

```
 [92] 88.000 89.000 90.000 91.000 92.000 93.000 94.000
 [99] 95.000 96.000
```

The `S+FinMetrics iMA` function requires at least two arguments: the first is the input series, and the second specifies the value for τ. In the above example, the optional argument `iter` is used to specify the number of iterations k; in addition, since the input series is not a "`timeSeries`" object, `iMA` treats it as evenly spaced and τ is in units of observations. If the input series is a "`timeSeries`" object, then τ should be specified in units of "business days". To illustrate the usage of `iMA` in this case, first create a "`timeSeries`" object representing the transaction price data from `hf3M.ts` created earlier[6]:

```
> smpl2 = positions(hf3M.ts) < timeDate("12/02/1999")
> hf3m.1min = aggregateSeries(hf3M.ts[smpl2,"trade.price"],
+ by="minutes", FUN=mean)
> hf3m.1min[103:110]
                Positions trade.price
 12/1/1999 11:28:00 AM 94.25000
```

[6]The `S-PLUS` function `aggregateSeries` is used to eliminate multiple transactions that occur at the same time. Currently, the `S+FinMetrics` inhomogeneous time series functions do not work if there are multiple observations with the same time stamp.

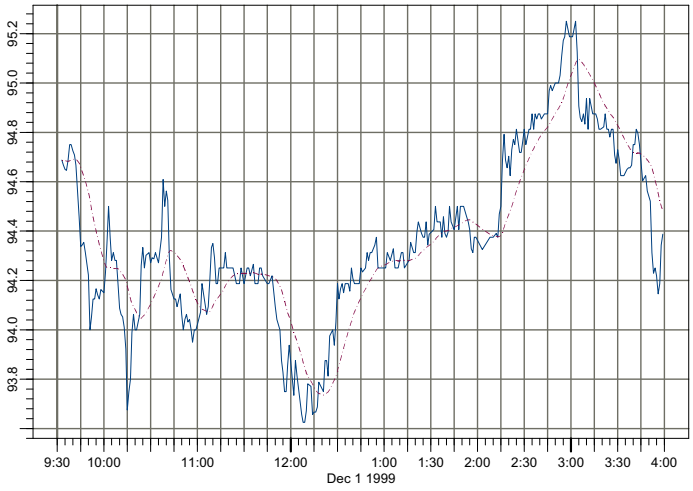

FIGURE 9.10. 20 minute moving average computed from `iMA` for 3M stock prices.

```
12/1/1999 11:30:00 AM 94.18750
12/1/1999 11:32:00 AM 94.25000
12/1/1999 11:33:00 AM 94.25000
12/1/1999 11:34:00 AM 94.21875
12/1/1999 11:36:00 AM 94.26563
12/1/1999 11:37:00 AM 94.18750
12/1/1999 11:39:00 AM 94.18750
```

Note that the data is not evenly spaced. To obtain a 20 minute moving average of `hf3m.1min`, set $\tau = 10/(6.5 * 60)$ because there are 6.5 hours for the default trading hours (from 9:30 AM to 4:00 PM):

```
> hf3m.ma = iMA(hf3m.1min, 10/(6.5*60), iter=10)
> plot(seriesMerge(hf3m.1min, hf3m.ma),
+ plot.args=list(lty=c(1,3)))
```

The original data and the 20 minutes moving average `hf3m.ma` are plotted together in Figure 9.10.

9.2.5 Rolling Analysis of Miscellaneous Functions

The standard analysis tools for time series require the data to be stationary. Rolling analysis of descriptive statistics can give an indication of structural change or instability in the moments of a time series. Level shifts and

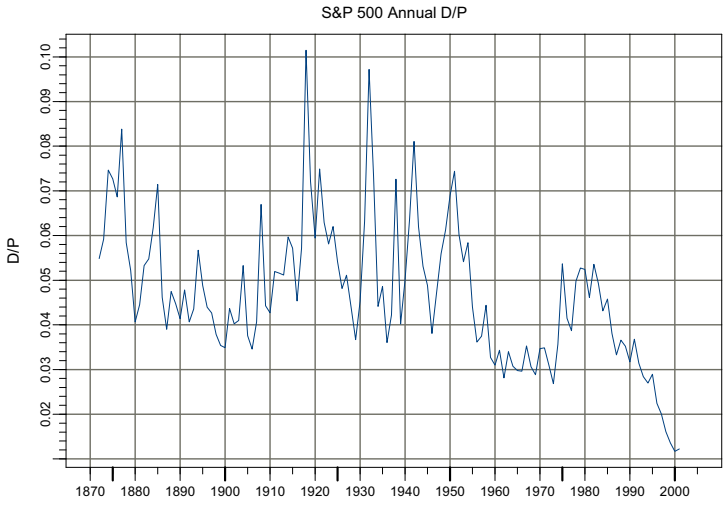

FIGURE 9.11. S & P 500 annual dividend/price ratio.

variance changes can usually be detected by rolling analyses. The `S-PLUS` function `aggregateSeries` may be used to perform rolling analysis with a variety of functions to uncover periods of instability and nonstationarity. The following example illustrates the use of `aggregateSeries` with the `S+FinMetrics` function `unitroot` to determine periods of unitroot nonstationarity of a time series.

Example 53 *Rolling unit root tests applied to annual dividend/price ratio*

Predictive regressions of asset returns on valuation ratios like dividend/price or earnings/price require the valuation ratios to be stationary or, more generally, $I(0)$, for the regressions to be statistically valid. Since asset returns are $I(0)$, if valuation ratios are $I(1)$ then the predictive regressions are unbalanced and the results will be nonsensical. To illustrate, consider the annual dividend-price (D/P) ratio on S&P 500 index taken from the `S+FinMetrics` "`timeSeries`" `shiller.annual`

```
> dp.ratio = shiller.annual[,"dp.ratio"]
> plot(dp.ratio,main="S&P 500 Annual D/P",ylab="D/P")
```

shown in Figure 9.11. For most of the sample the annual D/P looks to be $I(0)$ with mean near 5%. However, there are long periods when the ratio stays above or below 5% suggesting periods of non-mean reverting (nonstationary) behavior. Also, there is a clear drop in the ratio at the

end of the sample suggesting a fundamental change in the mean. Rolling unit root tests may be used to uncover periods of nonstationary behavior in D/P. To compute rolling ADF t-tests and normalized bias statistics using the S-PLUS function aggregateSeries create the following function adf.tests

```
> adf.tests = function(x, trend = "c", lags = 3)
> {
>     tmp1 = unitroot(x,trend=trend,lags=lags,statistic="t")
>     tmp2 = unitroot(x,trend=trend,lags=lags,statistic="n")
>     ans = concat(tmp1$sval,tmp2$sval)
>     ans
> }
```

The function adf.tests takes a time series x, passes it to the S+FinMetrics function unitroot twice and returns the ADF t-statistic and normalized bias statistic. Three lags are chosen for the tests based on a full sample analysis using the Ng-Perron backward selection procedure. Rolling unit root tests using a window of 50 years are then computed using the function aggregateSeries:

```
> roll.adf = aggregateSeries(dp.ratio,moving=50,adj=1,
+ FUN=adf.tests,colnames=c("t.test","norm.bias"))
```

The object roll.adf is a "timeSeries" containing the rolling unit root tests

```
> roll.adf[1:3,]
 Positions t.test norm.bias
 Dec 1920  -1.840 -13.24
 Dec 1921  -2.168 -15.24
 Dec 1922  -2.270 -16.03
```

Figure 9.12 is created using

```
> cvt.05 = qunitroot(0.05,trend="c",n.sample=50)
> cvn.05 = qunitroot(0.05,trend="c",statistic="n",
+ n.sample=50)
> par(mfrow=c(2,1))
> plot(roll.adf[,"t.test"], reference.grid=F,
+ main="Rolling ADF t-statistics")
> abline(h=cvt.05)
> plot(roll.adf[,"norm.bias"], reference.grid=F,
+ main="Rolling ADF normalized bias")
> abline(h=cvn.05)
```

and shows the rolling unit root tests along with 5% critical values. The results indicate that D/P is stationary mainly in the middle of the sample and becomes nonstationary toward the end of the sample. However, some

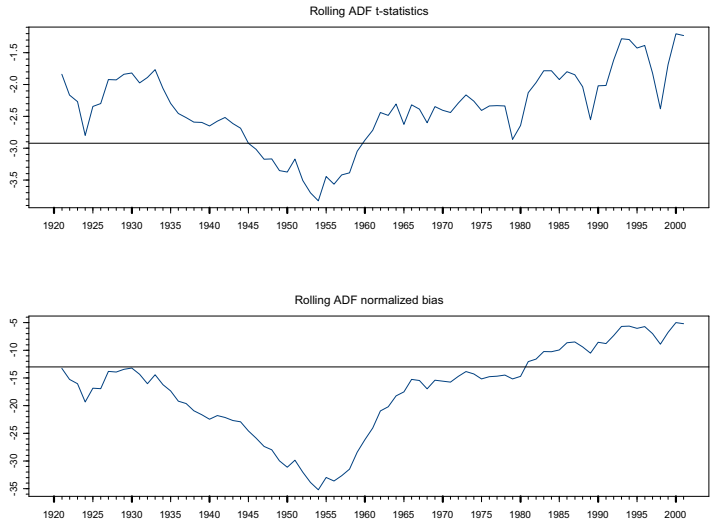

FIGURE 9.12. 50 year rolling ADF t-statistics and normalized bias statistics for the S&P 500 dividend-price ratio.

care must be used when interpreting the significance of the rolling unit root tests. The 5% critical values shown in the figures are appropriate for evaluating a single test and not a sequence of rolling tests. Critical values appropriate for rolling unit root tests are given in Banerjee, Lumsdaine and Stock (1992).

9.3 Technical Analysis Indicators

Technical analysis is, perhaps, the most widely used method for analyzing financial time series. Many of the most commonly used technical indicators are based on moving average techniques so it is appropriate to include a discussion of them here. A comprehensive survey of technical analysis is beyond the scope of this book. Useful references are Colby and Meyers (1988) and Bauer and Dahlquist (1999). The `S+FinMetrics` technical analysis indicators are implemented using the definitions in Colby and Meyers (1988). Broadly, the main technical indicators can be classified into four categories: price indicators, momentum indicators and oscillators, volatility indicators and volume indicators.

Function	Description
TA.Bollinger	Bollinger band
TA.medprice	Median price
TA.typicalPrice	Typical price
TA.wclose	Weighted close

TABLE 9.2. S+FinMetrics price indicators

9.3.1 Price Indicators

The S+FinMetrics price indicator functions are summarized in Table 9.2. To illustrate the use of these functions, consider the calculation of the typical daily price, which is defined to be the average of the highest, lowest and closing prices during the day, using the S+FinMetrics function TA.typicalPrice. The arguments expected by TA.typicalPrice are

```
> args(TA.typicalPrice)
function(high, low, close)
```

In order to compute this indicator, a data set with high, low and close prices is required. To compute the typical price for the Dow Jone Industrial Average over the period January 1, 1990 to February 20, 1990 using the S-PLUS "timeSeries" djia, use

```
> smpl = positions(djia) >= timeDate("1/1/1990")
> dj = djia[smpl,]
> tp.dj = TA.typicalPrice(dj[,"high"],
+ dj[,"low"],dj[,"close"])
> class(tp.dj)
[1] "timeSeries"
```

The typical price along with the high, low, open and close prices may be plotted together using

```
> plot.out = plot(dj[,1:4],plot.type="hloc")
> lines.render(positions(tp.dj),seriesData(tp.dj),
+ x.scale=plot.out$scale)
```

and the resulting plot is shown in Figure 9.13.

9.3.2 Momentum Indicators and Oscillators

The S+FinMetrics *momentum indicator and oscillator functions* are summarized in Table 9.3. For example, consider the popular *moving average convergence divergence* (MACD) indicator. This is an oscillator that represents the difference between two exponential moving averages. A signal line is computed as the exponential moving average of MACD. When the oscillator crosses above the signal line, it indicates a buy signal; when

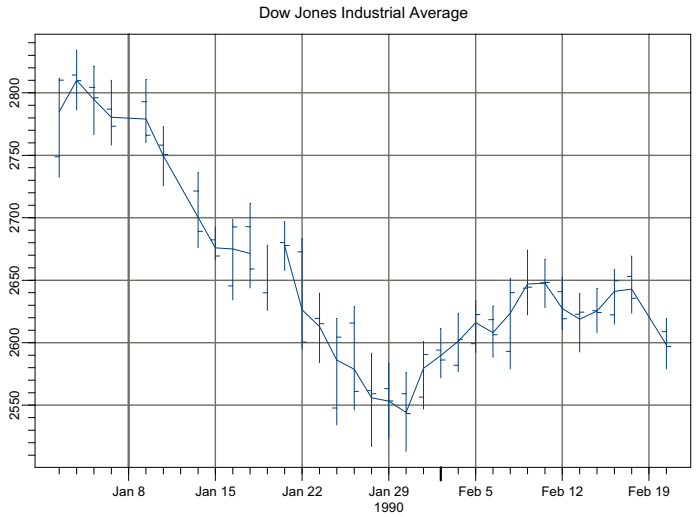

FIGURE 9.13. Typical price along with high, low, open and close prices for the Dow Jones Industrial Average.

the oscillator crosses below the signal line, it indicates a sell signal. The `S+FinMetrics` function `TA.macd` computes the MACD and has arguments

```
> args(TA.macd)
function(x, n.short = 12, n.long = 26, n.signal = 9, start
= "average", na.rm = F)
```

where `x` is a price series, `n.short` is a positive integer specifying the number of periods to be used for calculating the short window EWMA, `n.long` is a positive integer specifying the number of periods to be used for the calculating the long window EWMA, and `n.signal` is a positive integer

Function	Description
`TA.accel`	Acceleration
`TA.momentum`	Momentum
`TA.macd`	Moving average convergence divergence
`TA.roc`	Price rate of change
`TA.rsi`	Relative strength index
`TA.stochastic`	Stochastic Oscillator
`TA.williamsr`	Williams' %R
`TA.williamsad`	Williams' accumulation distribution

TABLE 9.3. `S+FinMetrics` momentum indicators

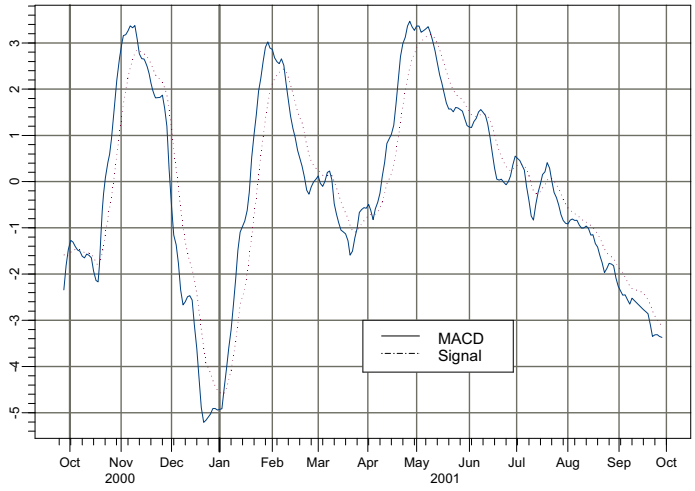

FIGURE 9.14. MACD and signal for daily closing prices on Microsoft stock.

Function	Description
`TA.adoscillator`	accumulation/distribution oscillator
`TA.chaikinv`	Chaikin's volatility
`TA.garmanKlass`	Garman-Klass estimator of volatility

TABLE 9.4. `S+FinMetrics` volatility indicator functions

giving the number of periods for the signal line. To compute and plot the MACD using daily closing prices on Microsoft use

```
> msft.macd = TA.macd(msft.dat[,"Close"])
> colIds(msft.macd) = c("MACD","Signal")
> plot(msft.macd,plot.args=list(lty=c(1:3)))
> legend(0.5,-3,legend=colIds(msft.macd),
+ lty=c(1,3))
```

Figure 9.14 shows the plot of the MACD and signal.

9.3.3 Volatility Indicators

The `S+FinMetrics` *volatility indicator functions* are summarized in Table 9.4. These functions compute estimates of volatility based on high, low, open and close information. For example, consider Chaikin's volatility indicator computed using the `S+FinMetrics` function `TA.chaikin`. It com-

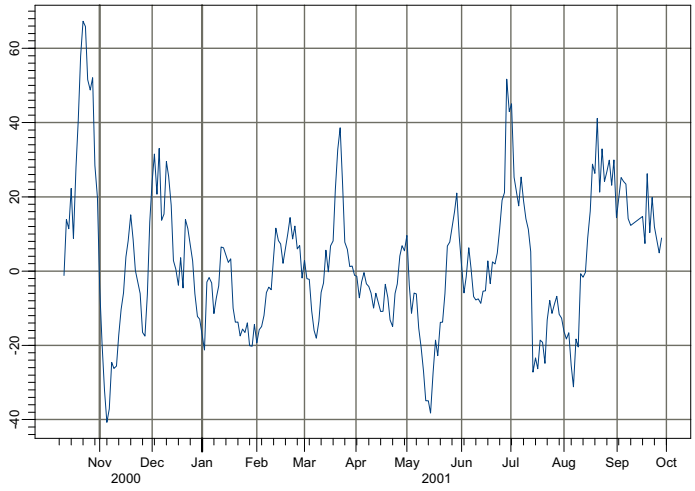

FIGURE 9.15. Chainkin's volatility estimate using the daily prices on Microsoft stock.

pares the spread between a security's high and low prices and quantifies volatility as a widening of the range between the high and low price. Let h_t and l_t represent the highest and lowest price for period t, respectively. Chaikin's volatility is calculated as the percentage change in the EWMA of $r_t = h_t - l_t$:

$$\frac{r_t - r_{t-nc}}{r_{t-nc}} \cdot 100$$

where nc is a positive number specifying the number of periods to use for computing the percentage change. To compute and plot Chaikin's volatility with $nc = 10$ and a ten day EWMA for the daily high and low price for Microsoft stock use

```
> msft.cv = TA.chaikinv(msft.dat[,"High"],
+ msft.dat[,"Low"],n.range=10,n.change=10)
> plot(msft.cv)
```

Figure 9.15 shows the estimated Chaikin volatility.

9.3.4 Volume Indicators

The `S+FinMetrics` *volume indicator functions* are summarized in Table 9.5. These indicators relate price movements with volume movements. To illustrate, consider the `S+FinMetrics` function `TA.adi` which computes the

Function	Description
TA.adi	Accumulation/distribution indicator
TA.chaikino	Chaikin oscillator
TA.nvi	Negative volume index
TA.pvi	Positive volume index
TA.obv	On balance volume
TA.pvtrend	Price-volume trend

TABLE 9.5. S+FinMetrics volume indicator functions

accumulations/distribution (A/D) indicator. This indicator associates price changes with volume as follows. Let c_t denote the closing price, h_t the highest price, l_t the lowest price, and v_t the trading volume for time t. The A/D indicator is the cumulative sum

$$AD_t = \sum_{i=1}^{t} \frac{c_i - l_i - (h_i - c_i)}{h_i - l_i} \cdot v_i$$

When AD_t moves up, it indicates that the security is being accumulated; when it moves down it indicates that the security is being distributed. To compute and plot the A/D indicator for Microsoft stock use

```
> msft.adi = TA.adi(msft.dat[,"High"],msft.dat[,"Low"],
+ msft.dat[,"Close"],msft.dat[,"Volume"])
> plot(msft.adi)
```

The resulting plot is shown in Figure 9.16.

9.4 Rolling Regression

For the linear regression model, rolling analysis may be used to assess the stability of the model's parameters and to provide a simple "poor man's" time varying parameter model. For a window of width $n < T$, the *rolling linear regression model* may be expressed as

$$\mathbf{y}_t(n) = \mathbf{X}_t(n)\boldsymbol{\beta}_t(n) + \boldsymbol{\varepsilon}_t(n),\ t = n, \ldots, T \tag{9.10}$$

where $\mathbf{y}_t(n)$ is an $(n \times 1)$ vector of observations on the response, $\mathbf{X}_t(n)$ is an $(n \times k)$ matrix of explanatory variables, $\boldsymbol{\beta}_t(n)$ is an $(k \times 1)$ vector of regression parameters and $\boldsymbol{\varepsilon}_t(n)$ is an $(n \times 1)$ vector of error terms. The n observations in $\mathbf{y}_t(n)$ and $\mathbf{X}_t(n)$ are the n most recent values from times $t - n + 1$ to t. It is assumed that $n > k$. The rolling least squares estimates

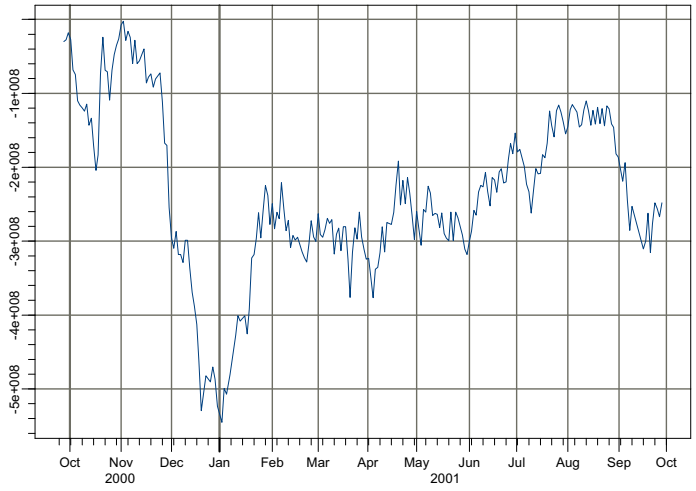

FIGURE 9.16. Accumulation/distribution indicator for Microsoft stock.

are

$$\begin{aligned}
\hat{\boldsymbol{\beta}}_t(n) &= \left[\mathbf{X}_t(n)'\mathbf{X}_t(n)\right]^{-1}\mathbf{X}_t(n)'\mathbf{y}_t(n) \\
\hat{\sigma}_t^2(n) &= \frac{1}{n-k}\hat{\boldsymbol{\varepsilon}}_t(n)'\hat{\boldsymbol{\varepsilon}}_t(n) \\
&= \frac{1}{n-k}\left[\mathbf{y}_t(n) - \mathbf{X}_t(n)\hat{\boldsymbol{\beta}}_t(n)\right]'\left[\mathbf{y}_t(n) - \mathbf{X}_t(n)\hat{\boldsymbol{\beta}}_t(n)\right] \\
\widehat{avar}(\hat{\boldsymbol{\beta}}_t(n)) &= \hat{\sigma}_t^2(n)\cdot\left[\mathbf{X}_t(n)'\mathbf{X}_t(n)\right]^{-1}
\end{aligned}$$

9.4.1 Estimating Rolling Regressions Using the S+FinMetrics Function rollOLS

The S+FinMetrics function rollOLS may be used to estimate general rolling regression models. rollOLS is based on the S+FinMetrics regression function OLS and implements efficient block updating algorithms for fast computation of rolling estimates. The arguments expected by rollOLS are

```
> args(rollOLS)
function(formula, data, subset, na.rm = F, method = "fit",
contrasts = NULL, start = NULL, end = NULL, width =
NULL, incr = 1, tau = 1e-010, trace = T, ...)
```

which are similar to those used by OLS. In particular, AR may be used in formulas to allow for lagged dependent variables and tslag and pdl may be used to allow for lagged independent variables. The argument width determines the rolling window width and the argument incr determines the increment size by which the windows are rolled through the sample. The output of rollOLS is an object of class "rollOLS" for which there are print, summary, plot and predict methods and extractor function coefficients. The use of rollOLS is illustrated with the following example.

Example 54 *Rolling estimation of CAPM for Microsoft*

Consider the estimation of the capital asset pricing model (CAPM) for an asset using rolling regression on the excess returns market model

$$r_t - r_{ft} = \alpha + \beta(r_{Mt} - r_{ft}) + \varepsilon_t,\ \varepsilon_t \sim WN(0, \sigma^2) \tag{9.11}$$

where r_t denotes the monthly return on an asset, r_{ft} denotes the 30 day T-bill rate, and r_{Mt} denotes the monthly return on a market portfolio proxy. The coefficient β measures the magnitude of market risk, and the CAPM imposes the restriction that $\alpha = 0$. Positive values of α indicate an average excess return above that predicted by the CAPM and negative values indicate an average return below that predicted by the CAPM. Rolling regression can be used to assess the stability of the CAPM regression over time and to uncover periods of time where an asset may have been overpriced or underpriced relative to the CAPM.

The monthly excess return data on Microsoft stock and S&P 500 index over the ten year period February 1990 through December 2000 are in the S+FinMetrics "timeSeries" object excessReturns.ts.

```
> colIds(excessReturns.ts)
[1] "MSFT"  "SP500"
> start(excessReturns.ts)
[1] Feb 1990
> end(excessReturns.ts)
[1] Dec 2000
```

The full sample CAPM estimates using the S+FinMetrics function OLS are

```
> ols.fit = OLS(MSFT~SP500,data=excessReturns.ts)
> summary(ols.fit)

Call:
OLS(formula = MSFT ~SP500, data = excessReturns.ts)

Residuals:
     Min       1Q   Median       3Q      Max
 -0.3101 -0.0620 -0.0024   0.0581   0.2260
```

```
Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept) 0.0175 0.0081     2.1654  0.0322
      SP500 1.5677 0.2015     7.7788  0.0000

Regression Diagnostics:

         R-Squared 0.3193
Adjusted R-Squared 0.3140
Durbin-Watson Stat 2.1891

Residual standard error: 0.09095 on 129 degrees of freedom
Time period: from Feb 1990 to Dec 2000
F-statistic: 60.51 on 1 and 129 degrees of freedom, the p-va
lue is 2.055e-012
```

The estimated full sample β for Microsoft is 1.57, which indicates that Microsoft was riskier than the market. Also, the full sample estimate of α is significantly different from zero so, on average, the returns on Microsoft are larger than predicted by the CAPM.

Consider now the 24-month rolling regression estimates incremented by 1 month computed using `rollOLS`

```
> roll.fit = rollOLS(MSFT~SP500, data=excessReturns.ts,
+ width=24,incr=1)
Rolling Window #1: Total Rows = 24
Rolling Window #2: Total Rows = 25
Rolling Window #3: Total Rows = 26
...
Rolling Window #108: Total Rows = 131
```

To suppress the printing of the window count, specify `trace=F` in the call to `rollOLS`. The returned object `roll.fit` is of class "`rollOLS`" and has components

```
> names(roll.fit)
 [1] "width"     "incr"      "nwin"      "contrasts" "rdf"
 [6] "coef"      "stddev"    "sigma"     "terms"     "call"
[11] "positions"
```

The components `coef`, `stddev` and `sigma` give the estimated coefficients, standard errors, and residual standard deviations for each of the `nwin` regressions. The `positions` component gives the start and end date of the estimation sample.

The `print` method, invoked by typing the object's name, gives a brief report of the fit

```
> roll.fit

Call:
rollOLS(formula = MSFT ~SP500, data = excessReturns.ts,
width = 24, incr = 1)

Rolling Windows:
 number width increment
    108    24         1
Time period: from Feb 1990 to Dec 2000

Coefficients:
          (Intercept)  SP500
     mean 0.0221      1.2193
std. dev. 0.0120      0.4549

Coefficient Standard Deviations:
          (Intercept)  SP500
     mean 0.0177      0.5057
std. dev. 0.0034      0.1107

Residual Scale Estimate:
   mean std. dev.
 0.0827 0.0168
```

Regression estimates are computed for 108 rolling windows. The mean and standard deviation are computed for the estimates and for the estimated coefficient standard errors. The average and standard deviation of the $\hat{\alpha}$ values are 0.0221 and 0.0120, respectively, and the average and standard deviation of the $\text{SE}(\hat{\alpha})$ values are 0.0177 and 0.0034, respectively. Hence, most of the $\hat{\alpha}$ values appear to be not significantly different from zero as predicted by the CAPM. The average and standard deviation of the $\hat{\beta}$ values are 1.2193 and 0.4549, respectively. The $\hat{\beta}$ values are quite variable and indicate that amount of market risk in Microsoft is not constant over time.

The rolling coefficient estimates for each rolling regression may be viewed using `summary`

```
> summary(roll.fit)

Call:
rollOLS(formula = MSFT ~ SP500, data = excessReturns.ts,
width = 24, incr = 1)

Rolling Windows:
 number width increment
```

```
    108    24         1
Time period: from Feb 1990 to Dec 2000

Coefficient: (Intercept)
           Value Std. Error t value Pr(>|t|)
Jan 1992 0.05075    0.01551   3.271 0.003492
Feb 1992 0.04897    0.01559   3.141 0.004751
Mar 1992 0.04471    0.01561   2.863 0.009035
...
Coefficient: SP500
          Value Std. Error t value  Pr(>|t|)
Jan 1992 1.3545     0.3322   4.077 0.0004993
Feb 1992 1.3535     0.3337   4.056 0.0005260
Mar 1992 1.3735     0.3332   4.123 0.0004472
...
```

or by using the `coef` extractor function. Notice that the first 24-month rolling estimates are computed for January 1992, which is 24 months after the sample start date of February 1990. The rolling estimates, however, are best viewed graphically using `plot`

```
> plot(roll.fit)

Make a plot selection (or 0 to exit):

1: plot: All
2: plot: Coef Estimates
3: plot: Coef Estimates with Confidence Intervals
4: plot: Residual Scale Estimates
Selection:
```

Plot selections 3 and 4 are illustrated in Figures 9.17 and 9.18. From the graphs of the rolling estimate it is clear that $\hat{\alpha}$ is significantly positive only at the very beginning of the sample. The $\hat{\beta}$ values are near unity for most windows and increase sharply at the end of the sample. However, the large standard errors make it difficult to determine if β is really changing over time. The residual scale estimates, $\hat{\sigma}$, increase sharply after 1999. This implies that the magnitude of the non-market risk in Microsoft increased after 1999.

In `rollOLS`, the optional argument `incr` sets the number of observations between the rolling blocks of data of length determined by `width`. Therefore, rolling regressions may be computed for arbitrary overlapping and non-overlapping blocks of data. For example, consider computing the CAPM estimates for Microsoft over the two non-overlapping but adjacent subsamples, February 1990 - June 1995 and July 1996 - November 2000 using `rollOLS`:

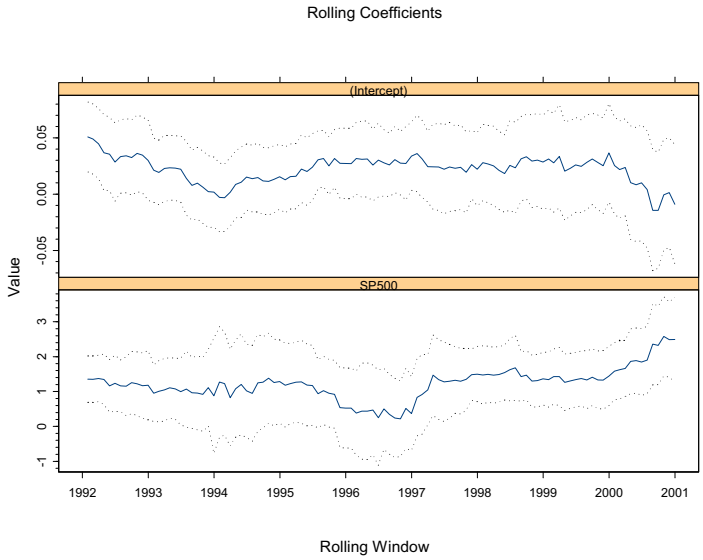

FIGURE 9.17. Rolling regression estimates of CAPM coefficients $\hat{\alpha}$ and $\hat{\beta}$ for Microsoft.

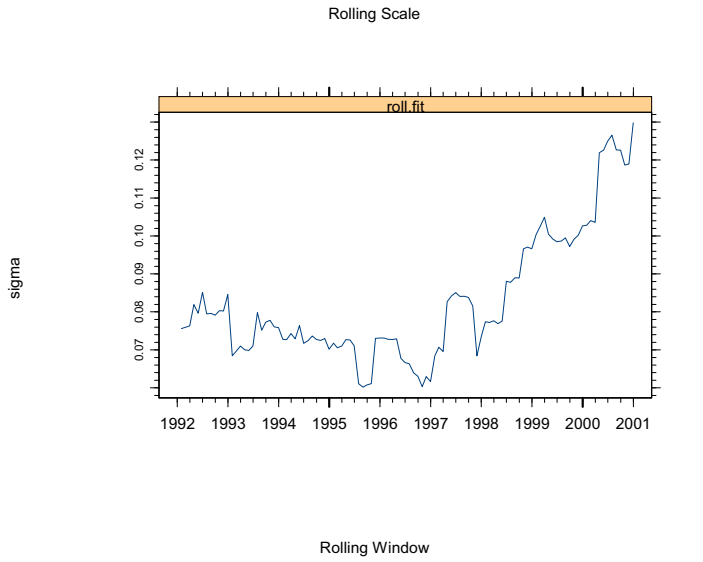

FIGURE 9.18. Rolling regression estimates of CAPM residual standard error for Microsoft.

```
> roll.fit2 = rollOLS(MSFT~SP500, data=excessReturns.ts,
+ width=65, incr=65)
Rolling Window #1: Total Rows = 65
Rolling Window #2: Total Rows = 130
> summary(roll.fit2)

Call:
rollOLS(formula = MSFT ~SP500, data = excessReturns.ts,
width = 65, incr = 65)

Rolling Windows:
 number width increment
      2    65        65
Time period: from Feb 1990 to Dec 2000

Coefficient: (Intercept)
           Value Std. Error t value Pr(>|t|)
Jun 1995 0.02765   0.009185  3.0100 0.003755
Nov 2000 0.01106   0.012880  0.8585 0.393851

Coefficient: SP500
         Value Std. Error t value   Pr(>|t|)
Jun 1995 1.339     0.2712   4.937 6.125e-006
Nov 2000 1.702     0.2813   6.050 8.739e-008
```

9.4.2 Rolling Predictions and Backtesting

Rolling regressions may be used to evaluate a model's predictive performance based on historical data using a technique commonly referred to as *backtesting*. To illustrate, consider the rolling regression model (9.10). The "out-of-sample" predictive performance of (9.10) is based on the rolling h-step predictions and prediction errors

$$\hat{y}_{t+h|t} = \mathbf{x}'_{t+h}\hat{\boldsymbol{\beta}}_t(n), \tag{9.12}$$

$$\hat{\varepsilon}_{t+h|t} = y_{t+h} - \hat{y}_{t+h|t} = y_{t+h} - \mathbf{x}'_{t+h}\hat{\boldsymbol{\beta}}_t(n) \tag{9.13}$$

The predictions are "out-of-sample" because $\hat{\boldsymbol{\beta}}_t(n)$ only uses data up to time t, whereas the predictions are for observations at times $t+h$ for $h > 0$. The rolling predictions are adaptive since $\hat{\boldsymbol{\beta}}_t(n)$ is updated when t is increased. When $h = 1$ there are $T - n$ rolling 1-step predictions $\{\hat{y}_{n+1|n}, \hat{y}_{n+2|n+1}, \ldots, \hat{y}_{T|T-1}\}$, when $h = 2$ there are $T - n - 1$ rolling 2-step predictions $\{\hat{y}_{n+2|n}, \hat{y}_{n+3|n+1}, \ldots, \hat{y}_{T|T-2}\}$ and so on.

Forecast Evaluation Statistics

The rolling forecasts (9.12) may be evaluated by examining the properties of the rolling forecast errors (9.13). Common evaluation statistics are

$$\begin{aligned}
\text{ME} &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}\hat{\varepsilon}_{t+h|t} \qquad (9.14)\\
\text{MSE}(h) &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}\hat{\varepsilon}^2_{t+h|t}\\
\text{RMSE}(h) &= \sqrt{MSE(h)}\\
\text{MAE}(h) &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}|\hat{\varepsilon}_{t+h|t}|\\
\text{MAPE}(h) &= \frac{1}{T-n-h+1}\sum_{t=n}^{T-h}\left|\frac{\hat{\varepsilon}_{t+h|t}}{y_{t+h}}\right|
\end{aligned}$$

The first measure evaluates the bias of the forecasts, and the other measures evaluate bias and precision.

Example 55 *Backtesting the CAPM*

Consider again the estimation of the CAPM (9.11) for Microsoft using rolling regression. The rolling regression information is contained in the "`rollOLS`" object `roll.fit`. The rolling h-step predictions (9.12) may be computed using the generic `predict` method. For example, the rolling 1-step forecasts are computed as

```
> roll.pred = predict(roll.fit,n.step=1)
> class(roll.pred)
[1] "listof"
> names(roll.pred)
[1] "1-Step-Ahead Forecasts"
```

The argument `n.step` determines the step length of the predictions. The object `roll.pred` is of class "`listof`" whose list component is a "`timeSeries`" object containing the rolling 1-step predictions:

```
> roll.pred[[1]]
 Positions           1
 Feb 1992   0.05994784
 Mar 1992   0.01481398
...
 Dec 2000  -0.00049354
```

The prediction errors (9.13) are then computed as

```
ehat.1step = excessReturns.ts[,"MSFT"]-roll.pred[[1]]
```

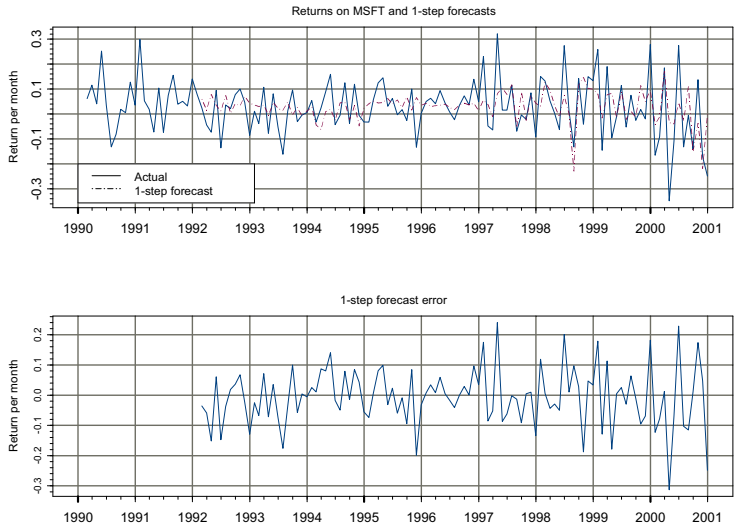

FIGURE 9.19. Monthly returns on Microsoft, 1-step rolling forecasts and forecast errors.

The monthly returns on Microsoft, 1-step forecasts and 1-step forecast errors are shown in Figure 9.19 created by

```
> par(mfrow=c(2,1))
> plot(excessReturns.ts[,"MSFT"], roll.pred[[1]],
+ main="Returns on MSFT and 1-step forecasts",
+ plot.args=list(lty=c(1,3)))
> legend(0, -0.2, legend=c("Actual","1-step forecast"),
+ lty=c(1,3))
> plot(ehat.1step,main="1-step forecast error")
```

The forecast evaluation statistics (9.14) may be computed as

```
> me.1step = mean(ehat.1step)
> mse.1step = as.numeric(var(ehat.1step))
> rmse.1step = sqrt(mse.1step)
> mae.1step = mean(abs(ehat.1step))
> mape.1step = mean(abs(ehat.1step/excessReturns.ts[,"MSFT"]),
+ na.rm=T)
```

To compute just the 2-step forecasts, specify `n.step=2` in the call to `predict`. To compute the 1-step and 2-step forecasts specify `n.step=1:2`

```
> roll.pred.12 = predict(roll.fit,n.steps=1:2)
```

```
> names(roll.pred.12)
[1] "1-Step-Ahead Forecasts" "2-Step-Ahead Forecasts"

> roll.pred.12[[1]]
 Positions            1
 Feb 1992   0.05994784
 Mar 1992   0.01481398
...
 Dec 2000  -0.00049354

> roll.pred.12[[2]]
 Positions           1
 Mar 1992   0.0165764
 Apr 1992   0.0823867
...
 Dec 2000  -0.0025076
```

Since the 1-step and 2-step predictions are components of the list object `roll.pred.12`, the `S-PLUS` function `lapply` may be used to simplify the computation of the forecast errors and evaluation statistics. To illustrate, supplying the user-defined function

```
> make.ehat = function(x,y) {
+    ans = y - x
+    ans[!is.na(ans),]
+ }
```

to `lapply` creates a "`named`" object containing the 1-step and 2-step forecasts errors as components:

```
> ehat.list = lapply(roll.pred.12, make.ehat,
+ excessReturns.ts[,"MSFT"])
> names(ehat.list)
[1] "1-Step-Ahead Forecasts" "2-Step-Ahead Forecasts"
```

The forecast evaluation statistics (9.14) may be computed for each component of `ehat.list` using `lapply` with the following user-defined function

```
> make.errorStats = function(x){
+    me = mean(x)
+    mse = as.numeric(var(x))
+    rmse = sqrt(mse)
+    mae = mean(abs(x))
+    ans = list(ME=me,MSE=mse,RMSE=rmse,MAE=mae)
+    ans
+ }

> errorStat.list = lapply(ehat.list,make.errorStats)
```

```
> unlist(errorStat.list)
 1-Step-Ahead Forecasts.ME 1-Step-Ahead Forecasts.MSE
                 -0.006165                    0.009283

 1-Step-Ahead Forecasts.RMSE 1-Step-Ahead Forecasts.MAE
                     0.09635                    0.07207

 2-Step-Ahead Forecasts.ME 2-Step-Ahead Forecasts.MSE
                 -0.007269                    0.009194

 2-Step-Ahead Forecasts.RMSE 2-Step-Ahead Forecasts.MAE
                     0.09589                    0.07187
```

The S-PLUS function `sapply` may be used instead of `lapply` to summarize the forecast error evaluation statistics:

```
> sapply(ehat.list,make.errorStats)
     1-Step-Ahead Forecasts 2-Step-Ahead Forecasts
[1,] -0.006165              -0.007269
[2,] 0.009283               0.009194
[3,] 0.09635                0.09589
[4,] 0.07207                0.07187
```

Comparing Predictive Accuracy

Backtesting is often used to compare the forecasting accuracy of two or more competing models. Typically, the forecast evaluation statistics (9.14) are computed for each model, and the model that produces the smallest set of statistics is judged to be the better model. Recently, Diebold and Mariano (1995) proposed a simple procedure using rolling h-step forecast errors for statistically determining if one model's forecast is more accurate than another's. Let $\hat{\varepsilon}^1_{t+h|t}$ and $\hat{\varepsilon}^2_{t+h|t}$ denote the h-step forecast errors from two competing models, and let N denote the number of h-step forecasts. The accuracy of each forecast is measured by a particular forecast evaluation or *loss function*

$$L(\hat{\varepsilon}^i_{t+h|t}),\ i = 1, 2$$

Two popular loss functions are the *squared error loss* $L(\hat{\varepsilon}^i_{t+h|t}) = \left(\hat{\varepsilon}^i_{t+h|t}\right)^2$ and *absolute error loss* $L(\hat{\varepsilon}^i_{t+h|t}) = \left|\hat{\varepsilon}^i_{t+h|t}\right|$. To determine if one model forecasts better than another Diebold and Mariano (1995) suggested computing the loss differential

$$d_t = L(\hat{\varepsilon}^1_{t+h|t}) - L(\hat{\varepsilon}^2_{t+h|t})$$

and testing the null hypothesis of equal forecasting accuracy

$$H_0 : E[d_t] = 0$$

The Diebold-Mariano test statistic is the simple ratio

$$\text{DM} = \frac{\bar{d}}{\widehat{\text{lrv}}(\bar{d})^{1/2}} \tag{9.15}$$

where

$$\bar{d} = \frac{1}{N}\sum_{t=1}^{N} d_t$$

is the average loss differential, and $\widehat{\text{lrv}}(\bar{d})$ is a consistent estimate of the long-run asymptotic variance of $\bar{d}$. Diebold and Mariano suggest computing $\widehat{\text{lrv}}(\bar{d})$ using the Newey-West nonparametric estimator with a rectangular weight function and a lag truncation parameter equal to the forecast step length, h, less one. Diebold and Mariano showed that under the null hypothesis of equal predictive accuracy the DM statistic is asymptotically distributed $N(0,1)$.

Example 56 *Backtesting regression models for predicting asset returns*

To illustrate model comparison and evaluation by backtesting, consider the problem of predicting the annual real return on S&P 500 index using two different valuation ratios. The regression model is of the form

$$r_t = \alpha + \beta x_{t-1} + \varepsilon_t \tag{9.16}$$

where r_t denotes the natural logarithm of the annual real total return on S&P 500 index and x_t denotes the natural logarithm of a valuation ratio. The first valuation ratio considered is the dividend/price ratio and the second ratio is the earning/price ratio. The data are constructed from the `S+FinMetrics` "`timeSeries`" `shiller.annual` as follows:

```
> colIds(shiller.annual)
 [1] "price"         "dividend"      "earnings"
 [4] "cpi"           "real.price"    "real.dividend"
 [7] "real.earnings" "pe.10"         "dp.ratio"
[10] "dp.yield"
> # compute log of real data
> ln.p = log(shiller.annual[,"real.price"])
> colIds(ln.p) = "ln.p"
> ln.dpratio = log(dp.ratio)
> colIds(ln.dpratio) = "ln.dpratio"
> ln.epratio = -log(shiller.annual[,"pe.10"])
> ln.epratio = ln.epratio[!is.na(ln.epratio),]
> colIds(ln.epratio) = "ln.epratio"
> # compute cc real total returns - see CLM pg. 261
> ln.r = diff(ln.p) + log(1+exp(ln.dpratio[-1,]))
> colIds(ln.r) = "ln.r"
```

```
> stock.ts = seriesMerge(ln.p,ln.d,ln.dpratio,
+ ln.epratio,ln.r,pos=positions(ln.epratio))
> start(stock.ts)
[1] Dec 1881
> end(stock.ts)
[1] Dec 2000
```

Rolling regression estimates of (9.16) with the two valuation ratios using a 50 year window incremented by 1 year are computed as

```
> roll.dp.fit = rollOLS(ln.r~tslag(ln.dpratio),data=stock.ts,
+ width=50,incr=1)
Rolling Window #1: Total Rows = 50
Rolling Window #2: Total Rows = 51
...
> roll.ep.fit = rollOLS(ln.r~tslag(ln.epratio),data=stock.ts,
+ width=50,incr=1)
Rolling Window #1: Total Rows = 50
Rolling Window #2: Total Rows = 51
...
Rolling Window #70: Total Rows = 119
```

Figures 9.20 and 9.21 show the rolling coefficient estimates from the two models along with standard error bands. The rolling estimates of β for the two models are similar. For both models, the strongest evidence for return predictability occurs between 1960 and 1990. The value of β for the earning/price model appears to be different from zero during more periods than the value of β for the dividend/price model.

The rolling h-step predictions for $h = 1, \ldots, 5$ and prediction errors are

```
> roll.dp.pred = predict(roll.dp.fit,n.steps=1:5)
> roll.ep.pred = predict(roll.ep.fit,n.steps=1:5)
> ehat.dp.list = lapply(roll.dp.pred,make.ehat,
+ stock.ts[,"ln.r"])
> ehat.ep.list = lapply(roll.ep.pred,make.ehat,
+ stock.ts[,"ln.r"])
```

The forecast evaluation statistics are

```
> errorStats.dp.list = lapply(ehat.dp.list,make.errorStats)
> errorStats.ep.list = lapply(ehat.ep.list,make.errorStats)
> tmp = cbind(unlist(errorStats.dp.list),
+ unlist(errorStats.ep.list))
> colIds(tmp) = c("D/P","E/P")
> tmp
numeric matrix: 20 rows, 2 columns.
                                 D/P     E/P
  1-Step-Ahead Forecasts.ME 0.03767 0.01979
```

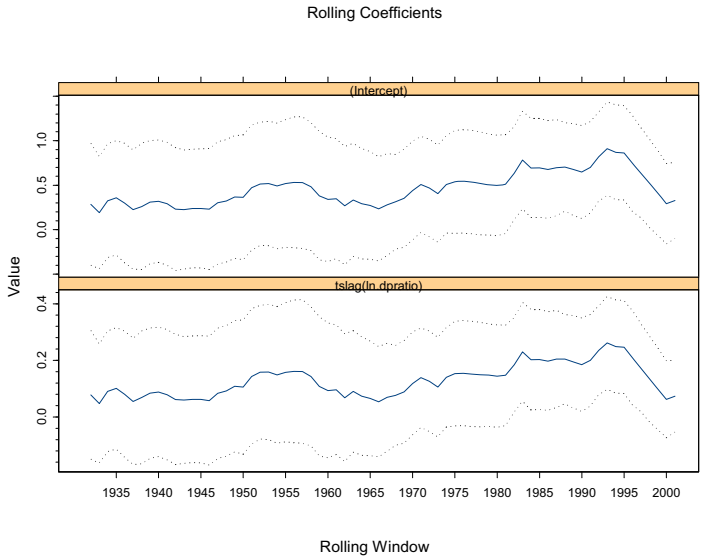

FIGURE 9.20. 50 year rolling regression estimates of (9.16) using dividend/price ratio.

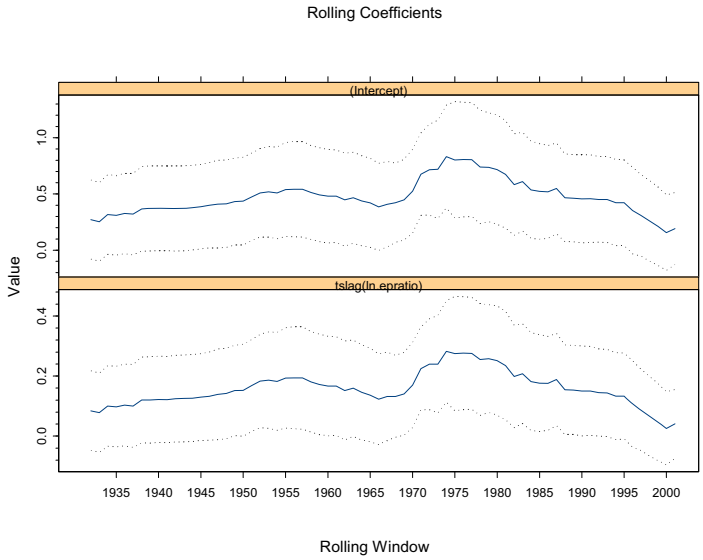

FIGURE 9.21. 50 year rolling regression estimates of (9.16) using earning/price.

```
 1-Step-Ahead Forecasts.MSE 0.03150 0.03139
1-Step-Ahead Forecasts.RMSE 0.17749 0.17718
 1-Step-Ahead Forecasts.MAE 0.14900 0.14556
  2-Step-Ahead Forecasts.ME 0.04424 0.02334
 2-Step-Ahead Forecasts.MSE 0.03223 0.03205
2-Step-Ahead Forecasts.RMSE 0.17952 0.17903
 2-Step-Ahead Forecasts.MAE 0.15206 0.14804
  3-Step-Ahead Forecasts.ME 0.04335 0.02054
 3-Step-Ahead Forecasts.MSE 0.03203 0.03180
3-Step-Ahead Forecasts.RMSE 0.17898 0.17832
 3-Step-Ahead Forecasts.MAE 0.14993 0.14731
  4-Step-Ahead Forecasts.ME 0.04811 0.02397
 4-Step-Ahead Forecasts.MSE 0.03292 0.03248
4-Step-Ahead Forecasts.RMSE 0.18143 0.18022
 4-Step-Ahead Forecasts.MAE 0.15206 0.14855
                               D/P     E/P
  5-Step-Ahead Forecasts.ME 0.04707 0.02143
 5-Step-Ahead Forecasts.MSE 0.03339 0.03255
5-Step-Ahead Forecasts.RMSE 0.18272 0.18043
 5-Step-Ahead Forecasts.MAE 0.15281 0.14825
```

The forecast evaluation statistics are generally smaller for the model using the earning/price ratio. The Diebold-Mariano statistics based on squared error and absolute error loss functions may be computed using

```
> for (i in 1:5) {
+   d.mse[,i] = ehat.dp.list[[i]]^2 - ehat.ep.list[[i]]^2
+   DM.mse[i] = mean(d.mse[,i])/sqrt(asymp.var(d.mse[,i],
+     bandwidth=i-1,window="rectangular"))
+   d.mae[,i] = abs(ehat.dp.list[[i]]) - abs(ehat.ep.list[[i]])
+   DM.mae[i] = mean(d.mae[,i])/sqrt(asymp.var(d.mae[,i],
+     bandwidth=i-1,window="rectangular"))
+ }
> names(DM.mse) = names(ehat.dp.list)
> names(DM.mae) = names(ehat.dp.list)
> cbind(DM.mse,DM.mae)
                        DM.mse  DM.mae
1-Step-Ahead Forecasts 0.07983 0.07987
2-Step-Ahead Forecasts 0.09038 0.08509
3-Step-Ahead Forecasts 0.07063 0.05150
4-Step-Ahead Forecasts 0.08035 0.06331
5-Step-Ahead Forecasts 0.07564 0.06306
```

Since the DM statistics are asymptotically standard normal, one cannot reject the null hypothesis of equal predictive accuracy at any reasonable

significance level based on the 1-step through 5-step forecast errors for the two models.

9.5 Rolling Analysis of General Models Using the S+FinMetrics Function roll

The S-PLUS aggregateSeries function is appropriate for rolling analysis of simple functions and the S+FinMetrics function rollOLS handles rolling regression. The S+FinMetrics function roll is designed to perform rolling analysis of general S-PLUS modeling functions that take a formula argument describing the relationship between a response and explanatory variables and where the data, usually a data frame or "timeSeries" object with a data frame in the data slot, is supplied explicitly in a data argument. The arguments expected by roll are

```
> args(roll)
function(FUN, data, width, incr = 1, start = NULL, end =
NULL, na.rm = F, save.list = NULL, arg.data =
"data", trace = T, ...)
```

where FUN is the S-PLUS modeling function to be applied to each rolling window, data is the data argument to FUN which must be either a data frame or a "timeSeries" with a data frame in the data slot,width specifies the width of the rolling window and incr determines the increment size by which the windows are rolled through the sample. The argument save.list specifies the components of the object returned by FUN to save in the object returned by roll. If FUN requires more arguments in addition to data, for example a formula relating a response to a set of explanatory variables, then these arguments should be supplied in place of .... The use of roll is illustrated with the following examples.

Example 57 *Rolling regression*

In this example, the 24-month rolling regression estimation of the CAPM for Microsoft using the "timeSeries" excessReturns.ts is repeated using the S+FinMetrics function roll with FUN=OLS. OLS requires a formula argument for model specification and a data frame or "timeSeries" in data argument. The 24 month rolling CAPM estimates using roll are

```
> roll.fit = roll(FUN=OLS, data=excessReturns.ts,
+ width=24, incr=1, formula=MSFT~SP500)
Rolling Window #1: Total Rows = 24
Rolling Window #2: Total Rows = 25
...
Rolling Window #108: Total Rows = 131
```

```
> class(roll.fit)
[1] "roll"
```

The return `roll.fit` is an object of class "`roll`" for which there are no specific method functions. Since the `data` argument `excessReturns.ts` is a "`timeSeries`", the default components of `roll.fit` are the positions of the rolling windows and "`timeSeries`" objects containing the components that are produced by `OLS` for each of the windows:

```
> names(roll.fit)
 [1] "R"         "coef"      "df.resid"  "fitted"
 [5] "residuals" "assign"    "contrasts" "ar.order"
 [9] "terms"     "call"      "positions"
> class(roll.fit$coef)
[1] "timeSeries"
> nrow(roll.fit$coef)
[1] 108
> class(roll.fit$residuals)
[1] "timeSeries"
> nrow(roll.fit$residuals)
[1] 108
```

The first column of the "`timeSeries`" `roll.fit$coef` contains the rolling intercept estimates, and the second column contains the rolling slope estimates.

```
> roll.fit$coef[1:2,]
 Positions       1     2
 Jan 1992  0.05075 1.354
 Feb 1992  0.04897 1.353
```

The rows of the "`timeSeries`" `roll.fit$residuals` contain the residuals for the OLS fit on each 24-month window

```
> roll.fit$residuals[1:2,]
 Positions        1       2       3        4         5
 Jan 1992  0.007267 0.04021 0.03550 0.085707  0.004596
 Feb 1992  0.042014 0.03726 0.08757 0.006368 -0.164498

 ...

       24
  0.05834
 -0.03393
```

If only some of the components of `OLS` are needed for each rolling window, these components may be specified in the optional argument `save.list`. For example, to retain only the components `coef` and `residuals` over the

rolling windows specify `save.list=c("coef","residuals")` in the call to `roll`:

```
> roll.fit = roll(FUN=OLS, data=excessReturns.ts,
+ width=24, incr=1, formula=MSFT~SP500,
+ save.list=c("coef","residuals"), trace=F)
> names(roll.fit)
[1] "coef"      "residuals" "call"      "positions"
```

9.6 References

Alexander, C. (2001). *Market Models: A Guide to Financial Data Analysis*. John Wiley & Sons, Chichester, UK.

Bauer, R.J. and J.R. Dahlquist (1999). *Techincal Market Indicators: Analysis & Performance*. John Wiley & Sons, New York.

Banerjee, A. R. Lumsdaine and J.H. Stock (1992). "Recursive and Sequential Tests of the Unit Root and Trend Break Hypothesis: Theory and International Evidence," *Journal of Business and Economic Statistics*, 10(3), 271-288.

Colby, R.W. and T.A Meyers (1988). *The Encyclopedia of Technical Market Indicators*. McGraw-Hill, New York.

Dacorogna, M.M., R. Gençay, U.A. Müller, R.B. Olsen, and O.V. Pictet (2001). *An Introduction to High-Frequency Finance*. Academic Press, San Diego.

Diebold, F.X. and R.S. Mariano (1995). "Comparing Predictive Accuracy," *Journal of Business and Economic Statistics*, 13, 253-263.

Shiller, R. (1998). *Irrational Exuberance*. Princeton University Press, Princeton, NJ.

Zumbach, G.O., and U.A. Müller (2001). "Operators on Inhomogeneous Time Series," *International Journal of Theoretical and Applied Finance*, 4, 147-178.

http://www.springer.com/978-0-387-27965-7

Modeling Financial Time Series with S-PLUS®
Zivot, E.; Wang, J.
2006, XXII, 998 p. 270 illus., Softcover
ISBN: 978-0-387-27965-7