Applied Nonparametric Econometrics, 2013 Fall
Project 2

Kun Ren*

December 13, 2013

Abstract

This project report is divided into two sections. In the first section, we conduct simulations to address the differences of performance achieved by various kernel density estimations under different settings featured by sample distribution, sample size, type of kernel, and bandwidth selection method. Then, additional simulations are conducted to compare Parzen-Rosenblatt kernel density estimation with local density method. In the second section, we perform a brief empirical study on 3-month US Treasury bill by applying Ljung-Box test, inspecting the autocorrelation, and estimating the density functions, of the up-to-date data.

1 Simulations

We conduct two groups of simulations. The first group examines the Parzen-Rosenblatt method and compares the performance of kernel density estimations under different settings. The second group compares the Parzen-Rosenblatt method with local density method.

1.1 Parzen-Rosenblatt method

This simulation exhausts all combinations of the settings featured by random sample distribution, sample size, kernel, bandwidth selection method, as summarized by Table 1.

*Kun Ren<renkun@outlook.com>, Student ID: 27720121152622.

Table 1: Elementary settings

Distribution	Sample Size	Kernel	Bandwidth Selection Method
Normal (0,1)	200	Gaussian	Scott (2009)
Exponential (1)	800	Epanechnikov	Sheather & Jones (1991)
Uniform (0,1)		Rectangular	Biased cross-validation
Beta (5,1)		Triangular	Unbiased cross-validation
Gamma (1,1)		Cosine	
		Optimal Cosine	
		Biweight	

Before we start the simulation, we set a seed to ensure the simulation is perfectly reproducible[1].

```
set.seed(123)
```

Then we specify the settings according to Table 1 so that a data frame containing all possible combinations of the settings could be expanded from the vectors.

```
samples <- c("norm","exp","unif","beta","gamma")
sample.settings <- list(norm=list(mean=0,sd=1),
                        exp=list(rate=1),
                        unif=list(min=0,max=1),
                        beta=list(shape1=2,shape2=5),
                        gamma=list(shape=9,scale=0.5))
sizes <- c(200,800)
kernels <- c("gaussian","epanechnikov","rectangular",
             "triangular","cosine","optcosine","biweight")
bws <- c("nrd0","nrd","sj","ucv","bcv")
test.table <- expand.grid(sample=samples,size=sizes,
                          kernel=kernels,bw=bws,
                          stringsAsFactors=FALSE)
```

[1]Since the report uses parallel computing to boost the computation, the random seed may not be set in each sub-environment of each parallel R session. Therefore, each node of the local computing cluster needs to evaluate a fixed random seed to ensure the reproducibility.

The data frame `test.table` contains all combinations of the vectors of settings. We may take a preview of the data frame where each row represents a particular set of settings.

```
head(test.table)
```

```
##   sample size   kernel   bw
## 1   norm  200 gaussian nrd0
## 2    exp  200 gaussian nrd0
## 3   unif  200 gaussian nrd0
## 4   beta  200 gaussian nrd0
## 5  gamma  200 gaussian nrd0
## 6   norm  800 gaussian nrd0
```

Now, we define a function that performs a simulation for once and calculates the mean absolute deviation error between the probability distribution function and kernel-estimated distribution function at each point an estimation is performed. The mean absolute deviation error is calculated by

$$\text{MADE} = \frac{1}{N}\sum_{k=1}^{N}\left|\hat{f}\left(u_k\right) - f\left(u_k\right)\right|$$

where $\hat{f}\left(\cdot\right)$ is the nonparametric estimate of $f\left(\cdot\right)$ and $\{u_k\}$ are the grid points used by the kernel density estimation function.

```
kest <- function(i,args,sample.settings) {
  settings <- sample.settings[[args$sample]]
  sample <- do.call(paste("r",args$sample,sep=""),
                    c(list(n=args$size),settings))
  kden <- density(sample,bw=args$bw,kernel=args$kernel)
  den <- do.call(paste("d",args$sample,sep=""),
                 c(list(x=kden$x),settings))
  transform(args,made=mean(abs(den-kden$y)))
}
```

Then we import `parallelMap` package to conduct simulations by parallel computing.

```
require(parallelMap)

## Loading required package:  parallelMap

parallelStartSocket(cpus=4,show.info=FALSE)

## Loading required package:  parallel

test.result <- do.call("rbind",
                       lapply(1:nrow(test.table),
                              function(i,fun,...) {
  do.call("rbind",parallelLapply(1:1000,fun,args=test.table[i,],
                                 sample.settings=sample.settings))
  },fun=kest,test.table=test.table,sample.settings=sample.settings))
parallelStop()
```

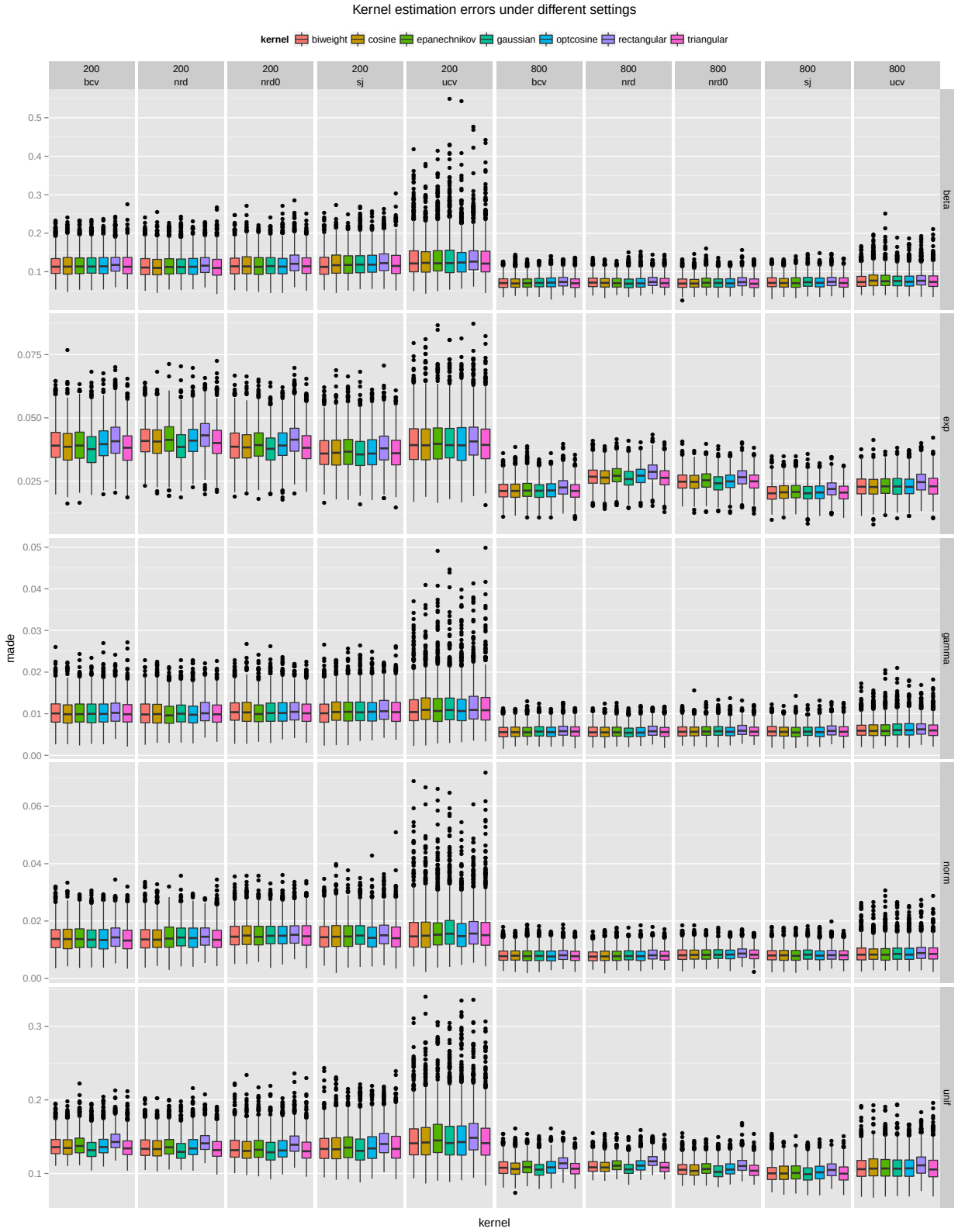
Then the test.table contains a stack of errors produced under different groups of settings represented by each row.

```
head(test.table)

##   sample size   kernel   bw
## 1   norm  200 gaussian nrd0
## 2    exp  200 gaussian nrd0
## 3   unif  200 gaussian nrd0
## 4   beta  200 gaussian nrd0
## 5  gamma  200 gaussian nrd0
## 6   norm  800 gaussian nrd0
```

Then we make an aggregate plot where each row represents a kernel and each column represents a combination of sample size and bandwidth selection method.

```
require(ggplot2)

## Loading required package:  ggplot2
## Loading required package:  methods
```

```r
ggplot(data=test.result,aes(x=kernel,y=made,fill=kernel))+
  geom_boxplot()+
  facet_grid(sample~size+bw,scales="free_y")+
  ggtitle("Kernel estimation errors under different settings")+
  scale_x_discrete(breaks=NULL)+
  theme(legend.position="top",legend.direction="horizontal")
```


Kernel estimation errors under different settings
kernel
biweight
cosine
epanechnikov
gaussian
optcosine
rectangular
triangular
200 bcv
200 nrd
200 nrd0
200 sj
200 ucv
800 bcv
800 nrd
800 nrd0
800 sj
800 ucv
beta
exp
gamma
norm
unif
made
kernel

After careful inspection of the diagram, we observe the following facts:

- All combination of kernels and bandwidth selection methods perform better under large sample size (800) than under small sample size (200) in that the variance of the distribution of mean absolute errors shrinks as the sample size gets larger.
- No combination of kernel and bandwidth selection method appears superior to other combinations for each random distribution used in the simulation. However, unbiased cross-validation appears to produce more upper outliers than other bandwidth selection methods do.
- For exponential distribution and uniform distribution, the rectangular kernel is likely to perform worse than other kernels under each bandwidth selection method.

When we use kernels to estimate the density function, we may use this diagram to help decide which kernel we should try.

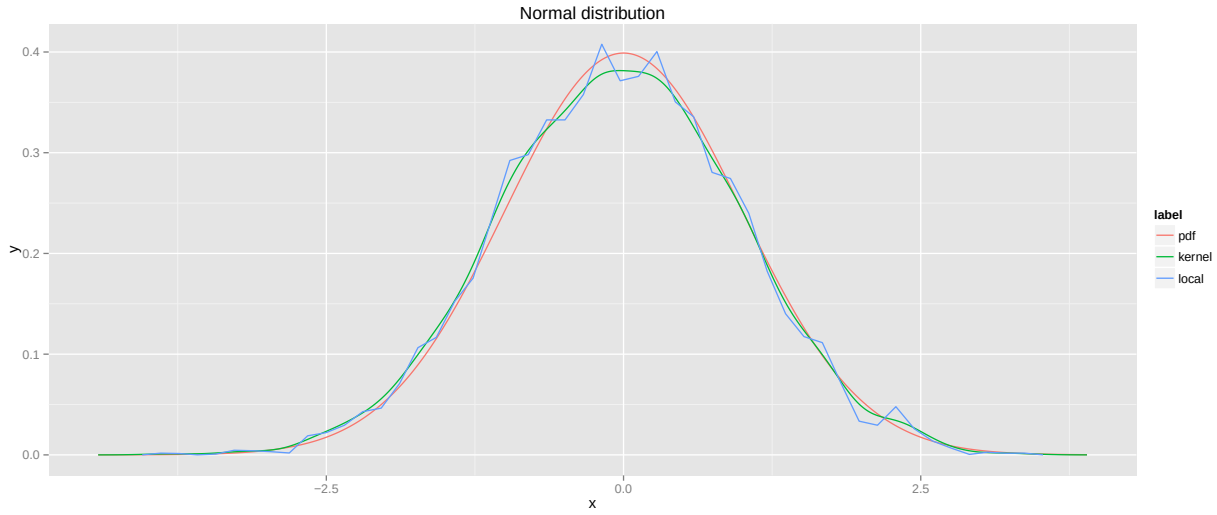
1.2 Local density method

It is reported that the Parzen-Rosenblatt kernel density estimation suffers from the boundary effect whereas the local density method may largely avoid this effect. Here we conduct some simulations to make a contrast between these two methods by estimating the density function of distributions that have bounded supports, e.g. Uniform distribution, and Exponential distribution.

In R, `locfit` package provides local density estimation functions. Now we conduct a benchmark simulation where a large sample of normally distributed numbers are estimated by kernel density and local density methods respectively.

```
require(locfit)

## Loading required package:  locfit
## locfit 1.5-9.1 2013-03-22

require(ggplot2)
x.norm <- rnorm(2000)
x.norm.density <- density(x.norm)
```

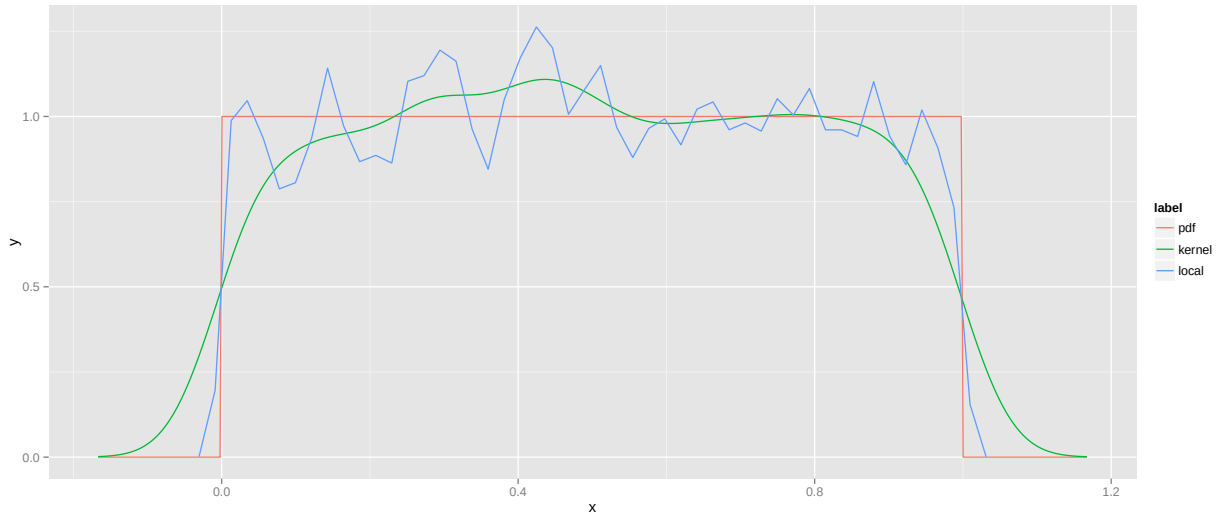
```
x.norm.local <- density.lf(x.norm)
x.norm.df <- rbind(data.frame(label="pdf",
                              x=x.norm.density$x,
                              y=dnorm(x.norm.density$x)),
                   data.frame(label="kernel",
                              x=x.norm.density$x,
                              y=x.norm.density$y),
                   data.frame(label="local",
                              x=x.norm.local$x,
                              y=x.norm.local$y))
qplot(x,y,data=x.norm.df,color=label,fill=label,geom="line")+
  ggtitle("Normal distribution")
```



From the figure above, we find that the result of local density estimation is quite close to that of kernel density estimation. The normal distribution has a non-bounded support of $\mathbb{R}$, which does not distinguish local density from kernel density. Now we apply the same simulation to uniform distribution and see what might be the effect of the existence of boundaries in the support of the data generating distribution.

```
x.unif <- runif(2000)
x.unif.density <- density(x.unif)
x.unif.local <- density.lf(x.unif)
x.unif.df <- rbind(data.frame(label="pdf",
```

```
                                x=x.unif.density$x,
                                y=dunif(x.unif.density$x)),
                    data.frame(label="kernel",
                                x=x.unif.density$x,
                                y=x.unif.density$y),
                    data.frame(label="local",
                                x=x.unif.local$x,
                                y=x.unif.local$y))
qplot(x,y,data=x.unif.df,color=label,geom="line")
```



Here we find that the uniform distribution we use to generate random sample has a support of $[0, 1]$, but both kernel density estimation and local density estimation assign positive probabilities out of the range, which contributes to the errors measured by mean absolute deviation errors. To better evaluate the error produced by boundary effects, we conduct a group of simulations.

First, we define `ade` function to calculate the mean and sum of absolute deviation errors both within $\mathbb{R}$ and/or within specified support.

```
ade <- function(x,y,range=c(0,1),bounded=FALSE) {
  if(bounded) {index <- x>=range[1] & x<=range[2]}
    else {index <- 1:length(x)}
  xs <- x[index]
  ys <- y[index]
```

```
  yp <- dunif(xs,min=range[1],max=range[2])
  list(mean=mean(abs(yp-ys)),sum=sum(abs(yp-ys)))
}
```

Then we define `simulate` function to do the simulation for one uniformly distributed sample and evaluate the boundary effects measured by

$$\text{Boundary Effect} = \frac{\sum_{u_k \in \mathbb{R}} \left| \hat{f}(u_k) - f(u_k) \right| - \sum_{u_k \in \Omega} \left| \hat{f}(u_k) - f(u_k) \right|}{\sum_{u_k \in \mathbb{R}} \left| \hat{f}(u_k) - f(u_k) \right|}$$

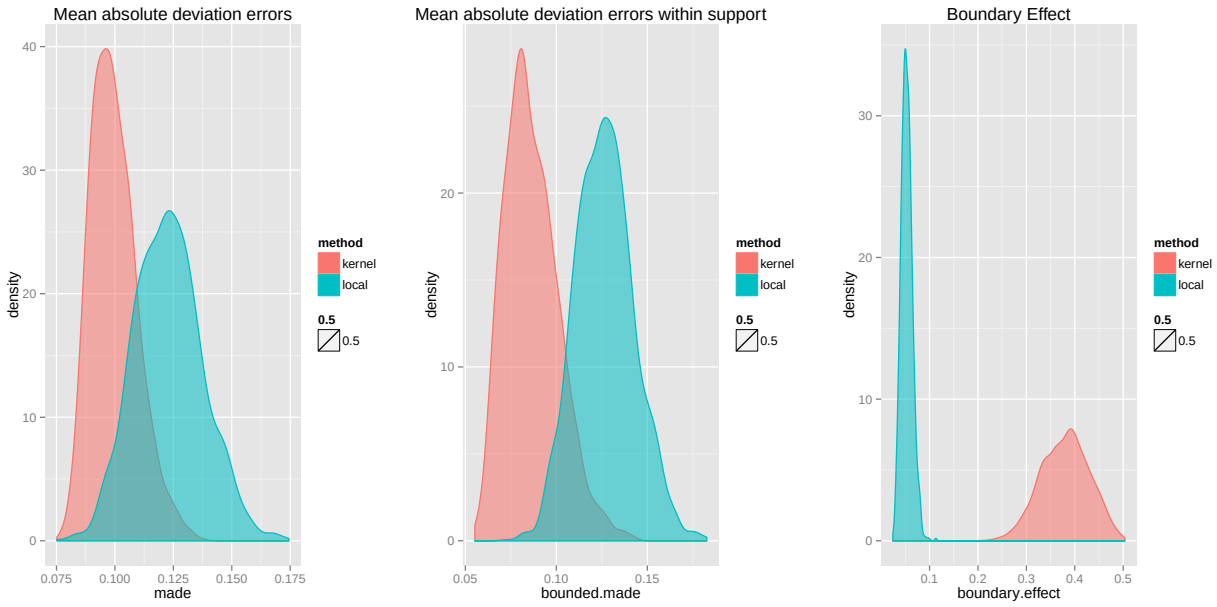
where Ω is the support of the uniform distribution, here by default $[0, 1]$. In other words, the boundary effect is measured by the percentage of absolute deviation errors produced by x out of the support of the random distribution. To implement the function, we have the following code ready for iteratively sampling.

```
simulate <- function(i,ade,range=c(0,1),size=1000) {
  require(locfit)
  sample <- runif(size,min=range[1],max=range[2])
  sample.density <- density(sample)
  sample.local <- density.lf(sample)
  sample.density.ade <- with(sample.density,ade(x,y,range,FALSE))
  sample.local.ade <- with(sample.local,ade(x,y,range,FALSE))
  sample.density.bounded.ade <- with(sample.density,ade(x,y,range,TRUE))
  sample.local.bounded.ade <- with(sample.local,ade(x,y,range,TRUE))
  rbind(data.frame(method="kernel",
                   min=range[1],max=range[2],sample.size=size,
                   made=sample.density.ade$mean,
                   bounded.made=sample.density.bounded.ade$mean,
                   boundary.effect=(sample.density.ade$sum-
                                      sample.density.bounded.ade$sum)/
                                    sample.density.ade$sum),
        data.frame(method="local",
                   min=range[1],max=range[2],sample.size=size,
                   made=sample.local.ade$mean,
                   bounded.made=sample.local.bounded.ade$mean,
```

```
                    boundary.effect=(sample.local.ade$sum-
                                       sample.local.bounded.ade$sum)/
                      sample.local.ade$sum))
}
```

To evaluate the boundary effect when we use kernel density estimation and local density estimation method respectively, we need sufficient number of replications.

```
parallelStartSocket(4)

## Starting parallelization in mode=socket with cpus=4.

boundary.effect.df <- do.call("rbind",parallelLapply(1:2000,simulate,ade=ade))

## Doing a parallel mapping operation.

parallelStop()

## Stopped parallelization.  All cleaned up.

require(grid)

## Loading required package:  grid

vp <- function(x,y) viewport(layout.pos.row=x,layout.pos.col=y)
plot1 <- qplot(made,data=boundary.effect.df,
      group=method,color=method,fill=method,geom="density",alpha=0.5)+
  ggtitle("Mean absolute deviation errors")
plot2 <- qplot(bounded.made,data=boundary.effect.df,
      group=method,color=method,fill=method,geom="density",alpha=0.5)+
  ggtitle("Mean absolute deviation errors within support")
plot3 <- qplot(boundary.effect,data=boundary.effect.df,
      group=method,color=method,fill=method,geom="density",alpha=0.5)+
  ggtitle("Boundary Effect")
grid.newpage()
pushViewport(viewport(layout=grid.layout(1,3)))
print(plot1,vp=vp(1,1))
print(plot2,vp=vp(1,2))
print(plot3,vp=vp(1,3))
```

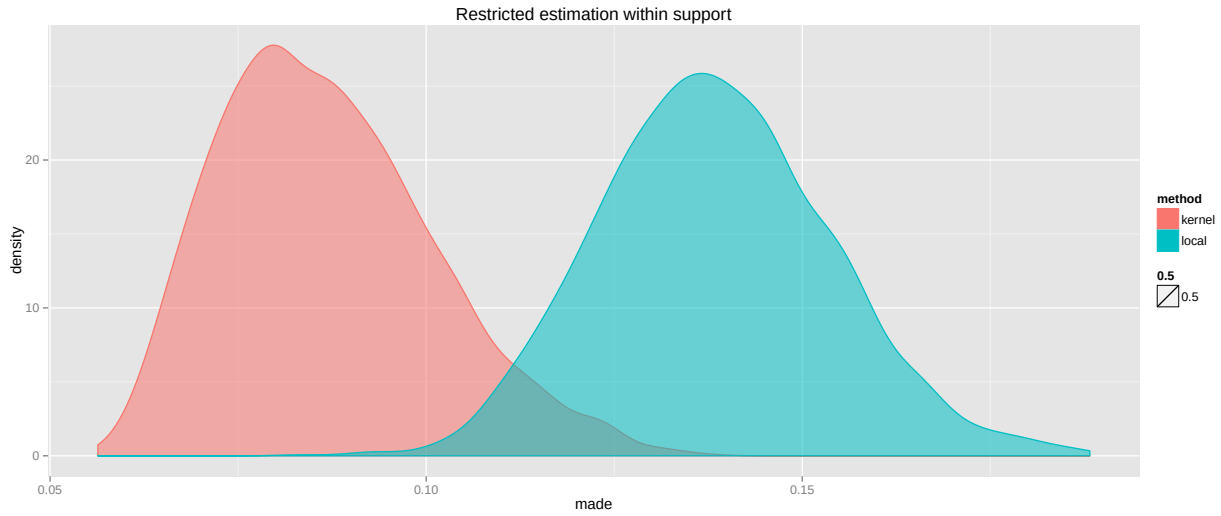


From the figures above, we find that although kernel density estimation performs better than local density estimation in aggregate level, local density suffers much less boundary effect than does kernel density.

However, sometimes we know the underlying distribution of certain data set to be bounded, we may restrict the density estimation method to restrict its support within given range. Here we simulate a benchmark where both kernel density estimation and local density estimation are restricted to only estimate the density within support $[0, 1]$.

```
simulate.bounded <- function(i,ade,range=c(0,1),size=1000) {
  require(locfit)
  sample <- runif(size,min=range[1],max=range[2])
  sample.density <- density(sample,from=range[1],to=range[2])
  sample.local <- density.lf(sample,from=range[1],to=range[2])
  sample.density.made <- with(sample.density,ade(x,y,range,FALSE))$mean
  sample.local.made <- with(sample.local,ade(x,y,range,FALSE))$mean
  rbind(data.frame(method="kernel",
                   min=range[1],max=range[2],sample.size=size,
                   made=sample.density.made),
        data.frame(method="local",
                   min=range[1],max=range[2],sample.size=size,
                   made=sample.local.made))
```

```
}
parallelStartSocket(4)

## Starting parallelization in mode=socket with cpus=4.

dlf.df <- do.call("rbind",parallelLapply(1:2000,simulate.bounded,ade=ade))

## Doing a parallel mapping operation.

parallelStop()

## Stopped parallelization.  All cleaned up.

qplot(made,data=dlf.df,
      group=method,color=method,fill=method,geom="density",alpha=0.5)+
  ggtitle("Restricted estimation within support")
```



Still, we observe that kernel density estimation performs better than local density estimation both restricted to the support $[0, 1]$.

2 Empirical Study

The up-to-date data for 3-month US Treasury bill (Secondary Market Rate) is retrieved from Federal Reserve Bank of Saint Louis on December 11, 2013. First, we load the data into the environment.

```
DTB3 <- read.delim("data/DTB3.txt",na.strings=".")
WTB3MS <- read.delim("data/WTB3MS.txt",na.strings=".")
TB3MS <- read.delim("data/TB3MS.txt",na.strings=".")
```

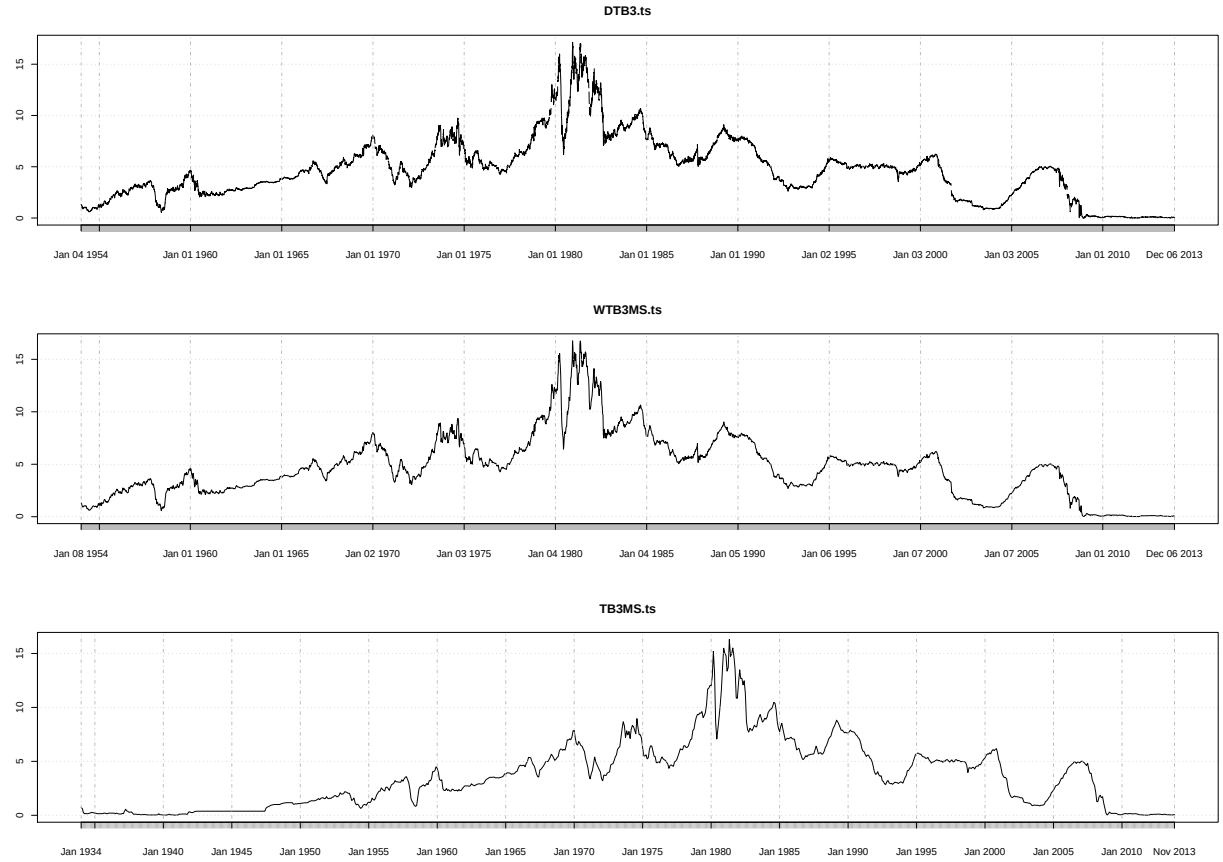
In order to better handle the time series data, we convert the data frame to extensible time series objects.

```
require(xts)

## Loading required package:  xts
## Loading required package:  zoo
##
## Attaching package:  'zoo'
##
## The following objects are masked from 'package:base':
##
##    as.Date, as.Date.numeric

DTB3.ts <- xts(DTB3$DTB3,order.by=as.Date(DTB3$DATE))
WTB3MS.ts <- xts(WTB3MS$WTB3MS,order.by=as.Date(WTB3MS$DATE))
TB3MS.ts <- xts(TB3MS$TB3MS,order.by=as.Date(TB3MS$DATE))
```

Here we need to notice that the data file contains some missing values that require special treatment in the following analysis.

First, we plot the time series data and get a rough impression on what the data looks like.

```
par(mfrow=c(3,1))
plot(DTB3.ts,type="l")
plot(WTB3MS.ts,type="l")
plot(TB3MS.ts,type="l")
```



Then we apply Ljung-Box test and see whether the series is autocorrelated.

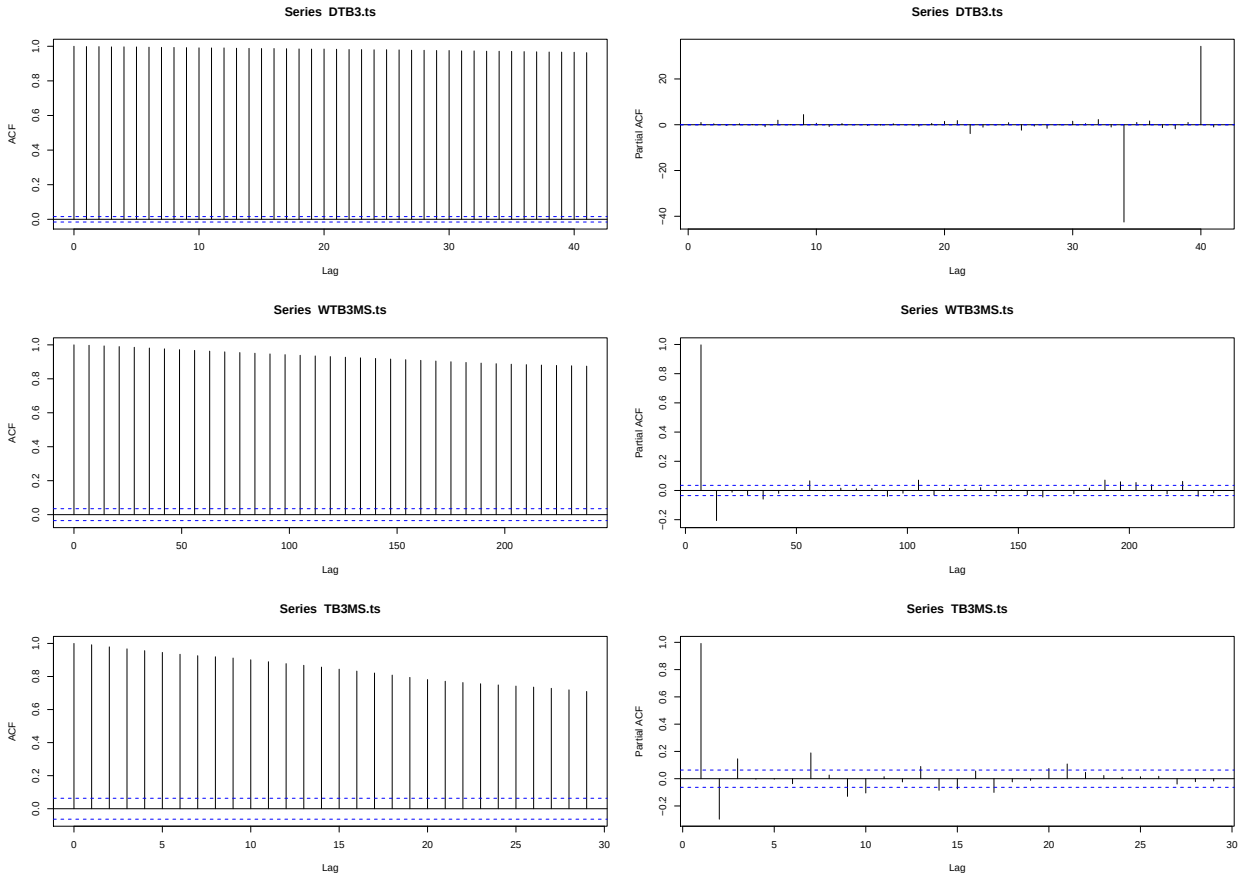
```
Box.test(DTB3.ts,type="Ljung-Box")

##
##  Box-Ljung test
##
## data:  DTB3.ts
## X-squared = 14927, df = 1, p-value < 2.2e-16

Box.test(WTB3MS.ts,type="Ljung-Box")

##
##  Box-Ljung test
##
## data:  WTB3MS.ts
## X-squared = 3114, df = 1, p-value < 2.2e-16
```

```r
Box.test(TB3MS.ts,type="Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  TB3MS.ts
## X-squared = 946.5, df = 1, p-value < 2.2e-16
```

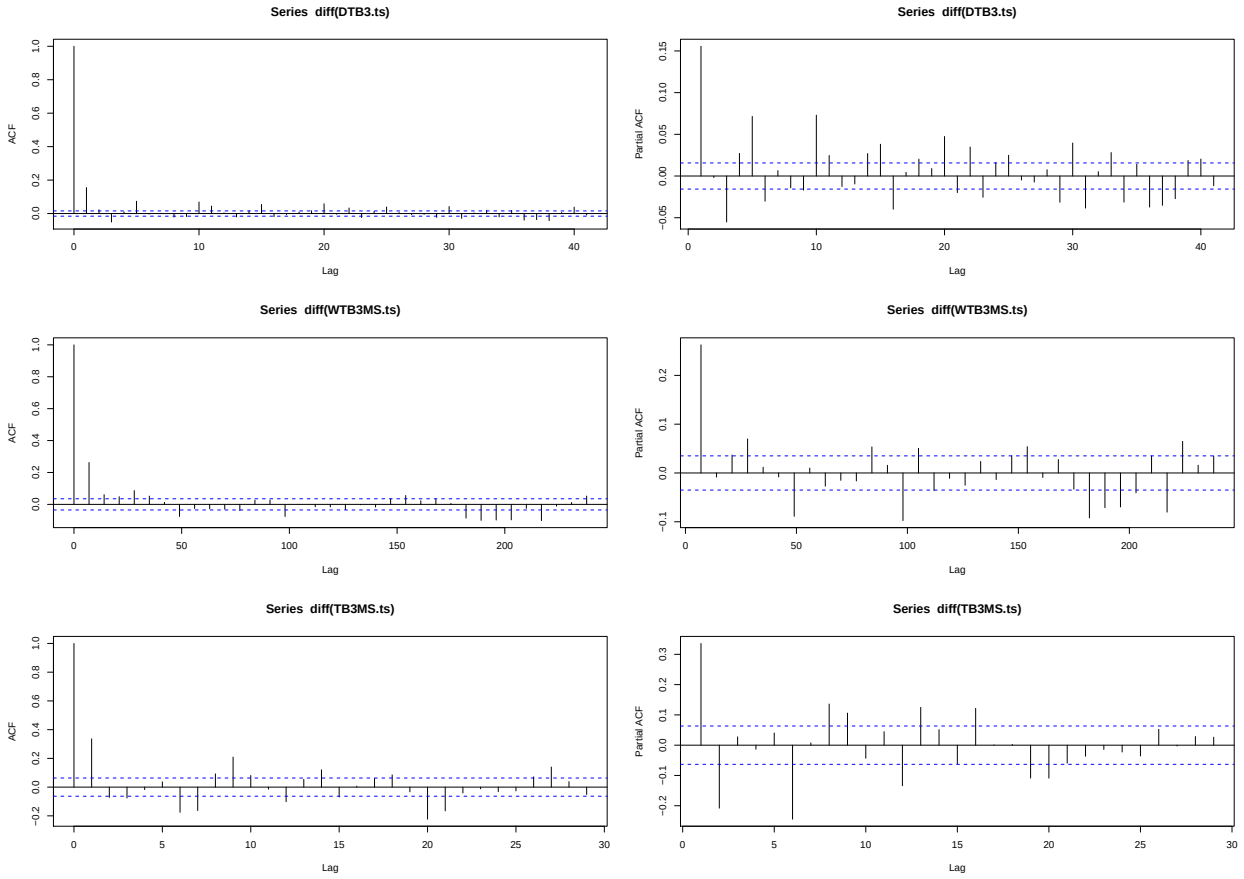
All the test results strongly suggest that the time series of the secondary market rate of daily, weekly, and monthly 3-month treasury bill is autocorrelated, which can be clearly illustrated by autocorrelation and partial autocorrelation plots as following:

```r
par(mfrow=c(3,2))
acf(DTB3.ts,na.action=na.pass)
pacf(DTB3.ts,na.action=na.pass)
acf(WTB3MS.ts,na.action=na.pass)
pacf(WTB3MS.ts,na.action=na.pass)
acf(TB3MS.ts,na.action=na.pass)
pacf(TB3MS.ts,na.action=na.pass)
```



Here we choose to pass the missing values in the data and ignore possible effects of such treatments. The autocorrelation function demonstrates that the series has very long memory and the partial autocorrelation function exhibits that it also appears to have long-term partial autocorrelation in high lags. Therefore, it may be more useful to look at the autocorrelation and partial autocorrelation of its difference sequence, as performed below.

```
par(mfrow=c(3,2))
acf(diff(DTB3.ts),na.action=na.pass)
pacf(diff(DTB3.ts),na.action=na.pass)
acf(diff(WTB3MS.ts),na.action=na.pass)
pacf(diff(WTB3MS.ts),na.action=na.pass)
acf(diff(TB3MS.ts),na.action=na.pass)
pacf(diff(TB3MS.ts),na.action=na.pass)
```

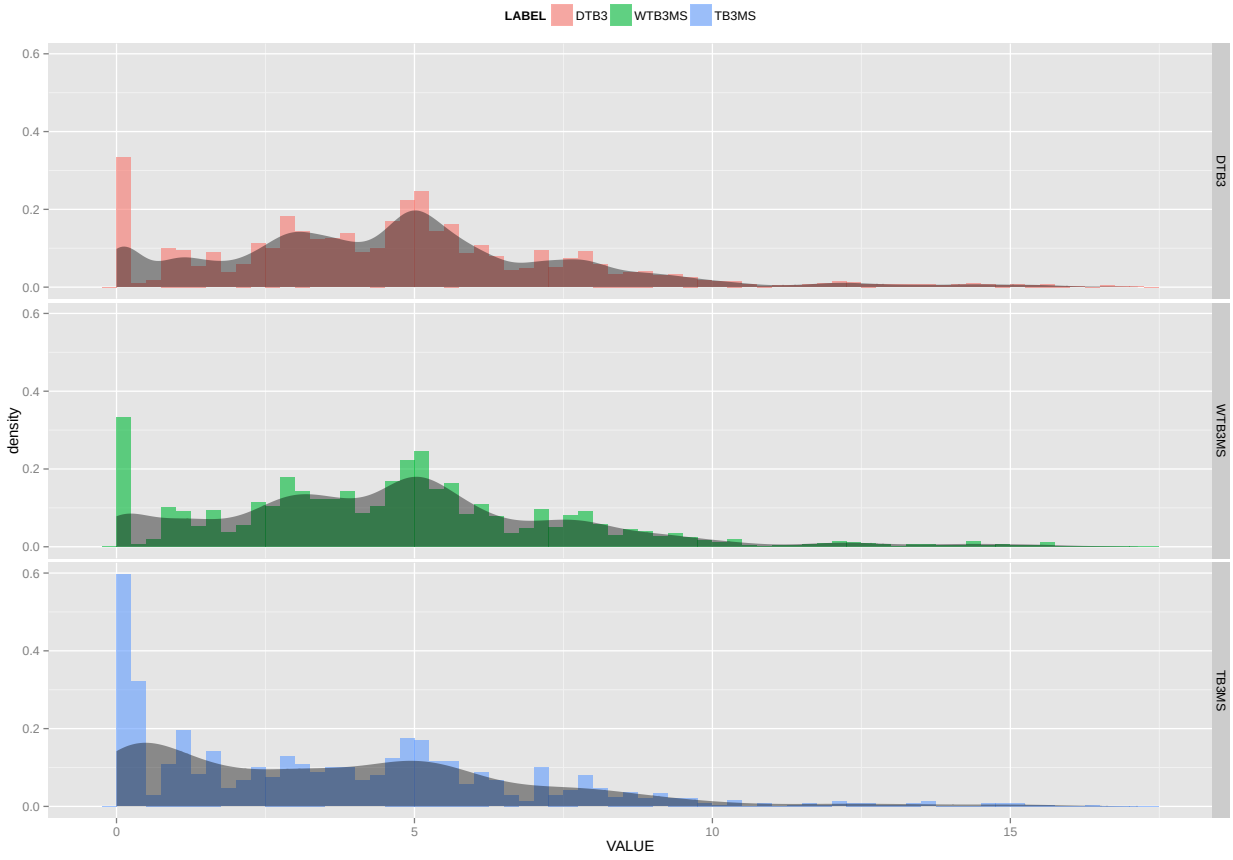


The difference sequence also exhibits long memory. However, the longer-term the data is collected, the less autocorrelation the time series exhibits. In other words, weekly secondary market rates of 3-month US T-bill exhibit higher autocorrelation than do daily series; monthly series of autocorrelation exhibit higher autocorrelation than do weekly series.

Now we estimate the kernel density function. We plot a histogram of the time series and then we apply the kernel density estimation to estimate the probability density function of the data.

```
labels <- c("LABEL","DATE","VALUE")
DTB3.df <- transform(DTB3,LABEL="DTB3",VALUE=DTB3)[,labels]
WTB3MS.df <- transform(WTB3MS,LABEL="WTB3MS",VALUE=WTB3MS)[,labels]
TB3MS.df <- transform(TB3MS,LABEL="TB3MS",VALUE=TB3MS)[,labels]
TB3 <- rbind(DTB3.df,WTB3MS.df,TB3MS.df)
ggplot(TB3,aes(x=VALUE,fill=LABEL))+
  geom_histogram(aes(y=..density..),alpha=0.6,binwidth=0.25)+
```

```
  geom_density(alpha=0.4,color=0,fill="black")+
  facet_grid(LABEL~.)+
  theme(legend.position="top",legend.direction="horizontal")
```



From the histograms and density plots above, we observe that it is more likely to see low return in monthly scale than in daily or weekly scale. Compared with the density estimation by Stanton (1997), the shape of the distribution is largely alike except that the latest data are more likely to locate at the lowest quantile so that the kernel estimate at nearly zero point is much higher than presented in the literature in 1997.

References

Scott, David W. 2009. *Multivariate density estimation: theory, practice, and visualization*. Vol. 383. Wiley. com.

Sheather, Simon J., & Jones, Michael C. 1991. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Methodological)*, 683–690.

Stanton, Richard. 1997. A nonparametric model of term structure dynamics and the market price of interest rate risk. *The Journal of Finance*, **52**(5), 1973–2002.