Time series analysis of Nordic spot electricity market data.

Bachelor thesis

School of mathematical sciences
Queen Mary University of London

Nikola Mihailova

May 7, 2021

Chapter 1

Introduction

The behaviour of financial time series is the subject of quite large number of studies (Johnson et al. [2003]). Due to their complexity linear stochastic models are not able to capture the properties of these series. This confirms that financial markets are nonlinear stochastic, chaotic or a combination of both (Brock et al. [1991],Osborne [1959]).

The Nordic electricity market, known as Nord Pool (http://www.nordpool.no) was created in 1993 and is owned by the two national grid companies, Statnett SF in Norway (50%) and Affarverket Svensa Kraftnat in Sweden (50%). The market was established as a consequence of the decision in 1991 by the Norwegian parliament to deregulate the market for power trading. Therefore, between 1992 and 1995 only Norway contributed to the market, in 1996 a joint Norwegian-Swedish power exchange was started-up and the power exchange was renamed Nord Pool ASA. Finland started a power exchange market of its own, EL-EX, in 1996 and joined Nord Pool in 1997. Beginning of 15th June 1998, Finland became an independent price area on the Nord Pool Exchange. The western part of Denmark (Jutland and Funen) has been part of the Nordic electric power market since 1 July 1999, whereas the eastern part of Denmark entered after 1st October 2000.

The spot market operated by Nord Pool is an exchange market where partici-

pants trade power contracts for physical delivery the next day. The spot market is based on an auction with bids for purchase and sale of power contracts of one hour duration covering the 24 hours of the following day. At the deadline for the collection of all buy and sell orders the information is gathered into overall supply and demand curves for each power-delivery hour. From these supply and demand curves the equilibrium spot prices are calculated.

We analyze the timeseries, which represents energy spot prices of the Nord Pool spot market. The data set consists of 70752 data points with hourly prices being expressed in EUR/MWh and covers the range from 1st January 1999 to 26th January 2007. Our goal is to analyse basic properties of the timeseries, like evolution of mean and variance with respect of time and try to see, if there are seasonal effects or global trends, which can be extracted and quantified. We also focus on fitting the distribution of the returns. The code, used for analysis is also presented.

Chapter 2

Basic analysis

We start with plotting the daily average prices, as the trends we want to analyse are on the daily scale. As one can see the averaged daily spot price is quite irregular with occasional spikes. The most noticeable one being in the late 2002 and beginning of 2003. This is attributed to dry and warm summer and autumn of 2002, which resulted in substantial lower hydro power generation in Sweden and Norway (Flatabo et al. [2003],Wang [2006]). Due to the low hydro power generation rate, more expensive types of power generation plant were taken into service to meet the public demands thus leading to historically high prices. It might also be affected to the transition to the deregulated market, happening at the same time, but the actual reasons for the price spikes are out of the scope of this work.

For a given set of data $(x_1, \ldots, x_n)$ be mean we denote

$$\hat{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

The running mean is presented on the Figure 2.1. As we can see from late of 1999 the prices have been steadily increasing. One can notice, that the mean is not affected by the spikes as much, as one would have thought. This is explained by the fact, that metric as the timespan expands becomes less sensitive to the spikes, which suggests using moving averages, which we will do latter.

The plot below also contains running standard deviation, which for dataset $(x_1, \ldots, x_n)$ is defined as

$$\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \hat{x})}{n}}.$$

Note, that the standard deviation is somewhat stable in regions between 1999 to 2002 and 2003 to 2007.

[28]:
```
from numpy import loadtxt
arr = loadtxt("TimeSeries2.txt") # load the data
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
plt.close("all")

# create a panda series with data sampled daily
y = arr[np.arange(0, len(arr), 1)]
date = pd.to_datetime("1st of January, 1999")
date2 = date + pd.to_timedelta(np.arange(len(y)), 'H')
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()

# plot the daily price with mean and standard deviation
plt.figure(figsize=([30, 10]))
series.plot(label=r'Daily price',linewidth=3, clip_on=False)
series.expanding().mean().plot(label=r'Running cumulative mean',␣
 ↪linewidth=3, clip_on=False)
series.expanding().std().plot(label=r'Running cumulative standard␣
 ↪deviation',linewidth=3, clip_on=False)
plt.xlabel('t [years]')
plt.legend(loc='upper left', fontsize=22)
```

```
plt.xticks(fontsize=22)
plt.yticks(fontsize=22)
plt.show()
```

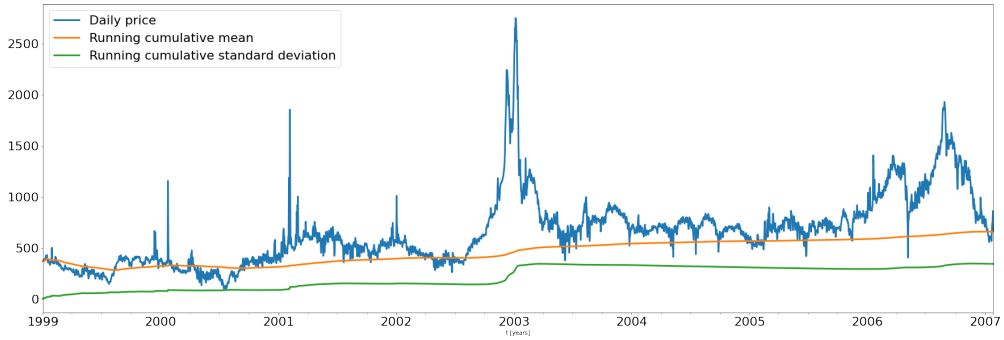


Figure 2.1: Daily returns with cumulative mean and standard deviation

We see, that the mean price increases, which leads us to believe, that no stationarity is present.

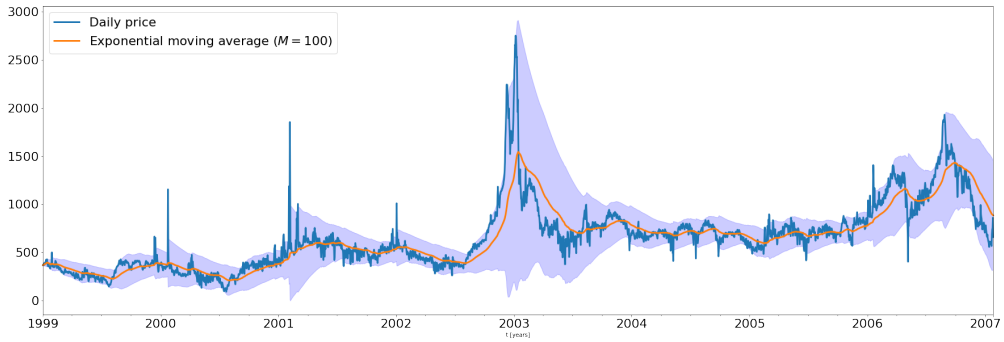
As we have seen, the simple average is not the best way, to track the mean as they put equal weights for all observations. The better way to track the mean is to use Exponential moving average (EMA), which is defined for a window size M as

$$\mathrm{EMA}(n) = (1-\alpha)\cdot \mathrm{EMA}(n-1) + \alpha \cdot x_n$$
$$\mathrm{EMA}(1) = x_0,$$

where $\alpha = \frac{2}{M-1}$ is the decay parameter and quantifies how quickly the effect of the particular datapoint is "forgotten". We choose $M = 100$, as it is a usual choice in finance for long term strategies, using moving averages. We also add the 95% confidence interval, which seems to capture the price changes very well.

[29]:
```
plt.figure(figsize=([30, 10]))
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()
```

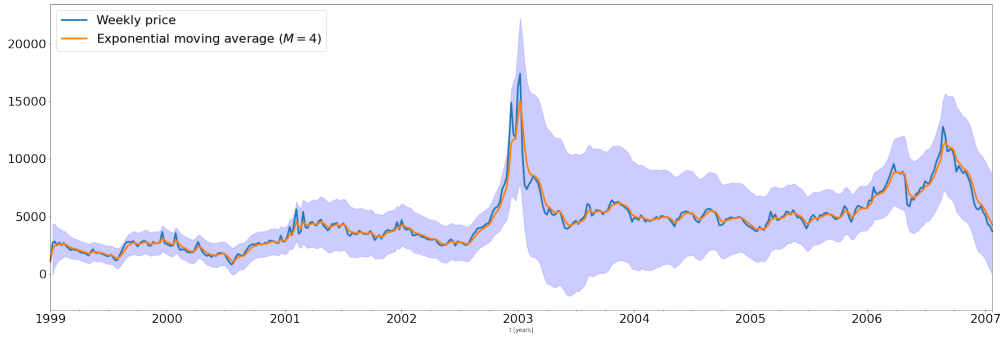
```
series.plot(label=r'Daily price',linewidth=3, clip_on=False)
series.ewm(span=100,adjust=True).mean().plot(label=r'Exponential␣
 ↪moving average ($M=100$)', linewidth=3, clip_on=False)

# plot the 95% confidence interval
upper = series.ewm(span=100,adjust=True).mean()+1.96*series.
 ↪ewm(span=100,adjust=True).std();
lower = series.ewm(span=100,adjust=True).mean()-1.96*series.
 ↪ewm(span=100,adjust=True).std();
plt.fill_between(series.index, upper, lower, color='b', alpha=.2)
plt.xlabel('t [years]')
plt.legend(loc='upper left', fontsize=22)
plt.xticks(fontsize=22)
plt.yticks(fontsize=22)
plt.show()
```



We do the same plot as above, but using resampled data on a weekly basis. This plot suggests a weak increasing trend. Again a huge increase around 2003, though this can be explained by the climatic factors.

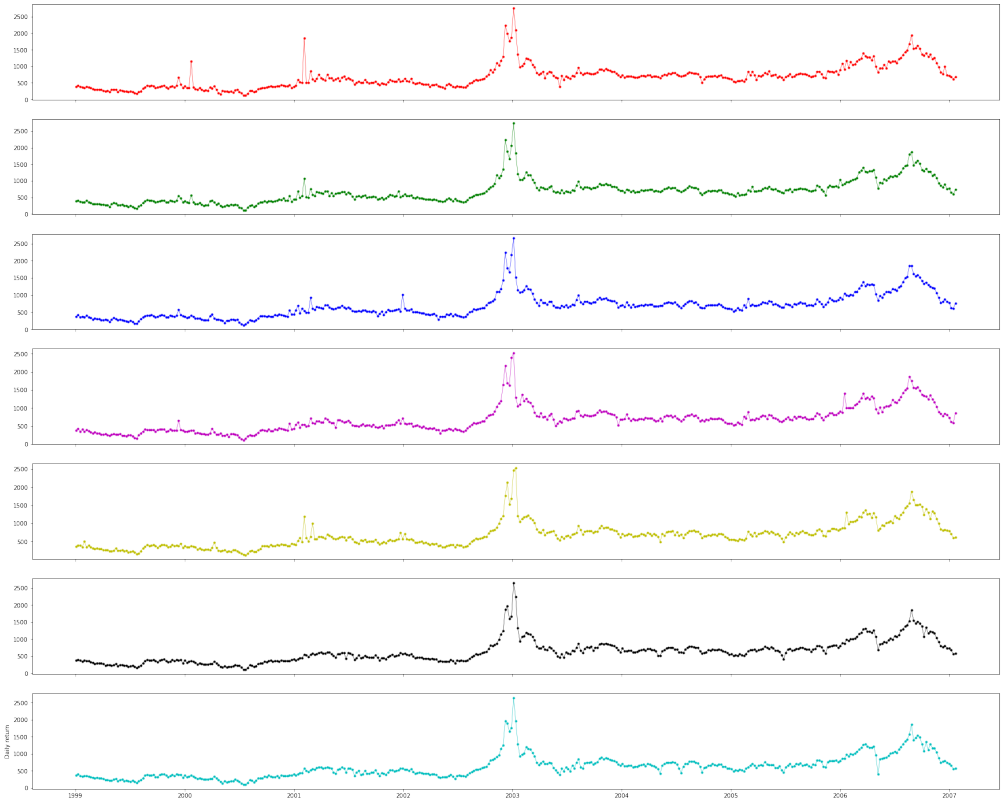
[37]:
```
plt.figure(figsize=([30, 10]))
ts = pd.Series(y, index = date2)
series = ts.resample('1W').sum()
```

```
series.plot(label=r'Weekly price',linewidth=3, clip_on=False)
series.ewm(span=4,adjust=True).mean().plot(label=r'Exponential␣
 ↪moving average ($M=4$)', linewidth=3, clip_on=False)

# plot the 95% confidence interval
upper = series.ewm(span=4,adjust=True).mean()+1.96*series.
 ↪ewm(span=100,adjust=True).std();
lower = series.ewm(span=4,adjust=True).mean()-1.96*series.
 ↪ewm(span=100,adjust=True).std();
plt.fill_between(series.index, upper, lower, color='b', alpha=.2)
plt.xlabel('t [years]')
plt.legend(loc='upper left', fontsize=22)
plt.xticks(fontsize=22)
plt.yticks(fontsize=22)
plt.show()
```



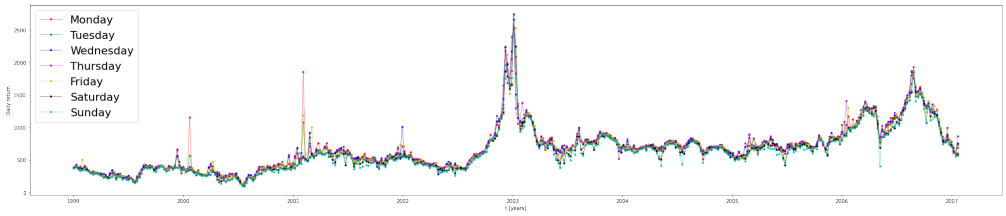
Before we move forward, lets do the same analysis as above, but for specific days of the week. It doesn't seem, that a specific day affects in some way the spot prices in terms of trends. We show them on separate plots and on an aggregate plot as well. Though we can see, that Sunday spot prices are in general lower, than the weekdays one, which suggests a possibility for separate analysis of returns for the weekends

[3]:
```
y = arr[np.arange(0, len(arr), 1)]
x = np.arange(0, len(y))
date = pd.to_datetime("1st of January, 1999")
date2 = date + pd.to_timedelta(np.arange(len(y)), 'H')
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()
s = pd.date_range('1999-01-01', '2007-01-26', freq='D').
 ↪to_series()
weekDaysArray = s.dt.dayofweek.array
sWeeks = pd.date_range('1999-01-01', '2007-01-26', freq='W').
 ↪to_series()
values = series.array
fig, ax = plt.subplots(7, sharex=True, figsize=(30,25))
colours = (['r','g','b','m','y','k','c'])
for j,i in enumerate(['Monday', 'Tuesday',␣
 ↪'Wednesday','Thursday', 'Friday', 'Saturday','Sunday']):
    arr0 = values[weekDaysArray == j]
    ts0 = pd.Series(arr0[np.arange(len(sWeeks.array))], index =␣
 ↪sWeeks)
    ax[j].plot(ts0, marker='.', linestyle='-', linewidth=0.5,␣
 ↪label=i, color = colours[j])
    plt.ylabel('Daily return')

plt.show()
```



[4]:
```
y = arr[np.arange(0, len(arr), 1)]
x = np.arange(0, len(y))
date = pd.to_datetime("1st of January, 1999")
date2 = date + pd.to_timedelta(np.arange(len(y)), 'H')
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()
s = pd.date_range('1999-01-01', '2007-01-26', freq='D').
 ↪to_series()
weekDaysArray = s.dt.dayofweek.array
sWeeks = pd.date_range('1999-01-01', '2007-01-26', freq='W').
 ↪to_series()
values = series.array
```

```
fig, ax = plt.subplots(1,1, sharex=True, figsize=(35,7))
colours = (['r','g','b','m','y','k','c'])
for j,i in enumerate(['Monday', 'Tuesday',␣
 ↪'Wednesday','Thursday', 'Friday', 'Saturday','Sunday']):
    arr0 = values[weekDaysArray == j]
    ts0 = pd.Series(arr0[np.arange(len(sWeeks.array))], index =␣
 ↪sWeeks)
    ax.plot(ts0, marker='.', linestyle='-', linewidth=0.5,␣
 ↪label=i, color = colours[j])
    plt.ylabel('Daily return')
plt.legend(loc='upper left', fontsize=22)
plt.xlabel('t [years]')
plt.ylabel('Daily return')

plt.show()
```



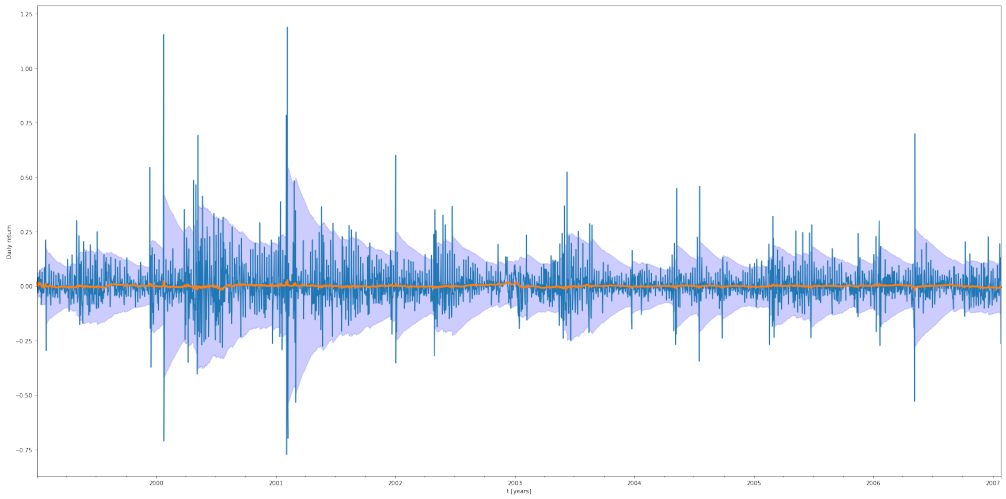
We have considered the time series x_t as well as the corresponding returns over the time horizon δ, defined as

$$r_t = \ln(x(t)) - \ln(x(t-1)).$$

Note, that here x_t stands fr the averaged hourly spot prices within a day t. We also plot the 100 day EMA and a 95% confidence interval as well. The following plot suggests, that the expectation should be around zero and we might expect a univariate distribution for the returns, though heavier tails and skew are ex-

pected, as we observe large deviations for returns and a slight skew towards negative side.

[21]:
```
plt.figure(figsize=([30, 15]))
L = len(series)
#transform the prices to the returns
z = np.log(series.array[np.arange(1, L, 1)]) - np.log(series.
 ↪array[np.arange(0, L-1, 1)])

date3 = date + pd.to_timedelta(np.arange(L), 'D')
ks = pd.Series(z, index = date3[np.arange(1, L, 1)])
ks.plot()
ks.ewm(span=100,adjust=True).mean().plot(label=r'Exponential␣
 ↪moving average ($M=100$)', linewidth=3, clip_on=False)
upper = ks.ewm(span=100,adjust=True).mean()+1.96*ks.
 ↪ewm(span=100,adjust=True).std();
lower = ks.ewm(span=100,adjust=True).mean()-1.96*ks.
 ↪ewm(span=100,adjust=True).std();
plt.fill_between(ks.index, upper, lower, color='b', alpha=.2)

plt.xlabel('t [years]')
plt.ylabel('Daily return')

plt.show()
```



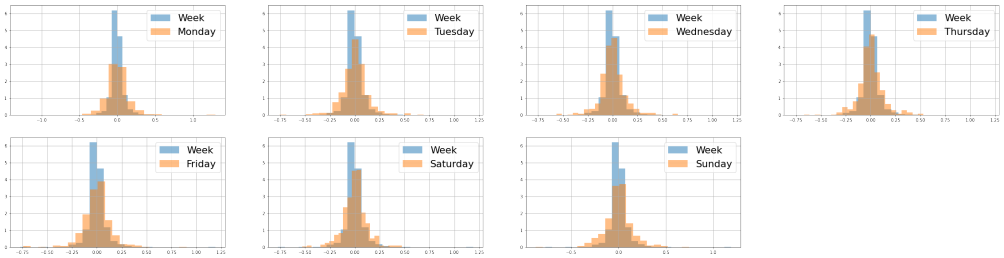
Now we analyse the mean, median and variance for the returns. We estimate them for each day separately and then for the all of them combined. The results are summarised in the Table below.

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Week
Mean	0.00137	0.00155	0.00168	0.00189	0.00138	0.00105	0.00101	0.0001
Median	-0.00371	0.00235	-0.00516	-0.00376	0.00566	0.00323	0.00308	-0.0058
Variance	0.18276	0.13589	0.13067	0.13056	0.13897	0.12089	0.14525	0.0948

As we have expected, there is a slight negative skew for the complete data and a slight decrease in the mean during the weekends.

[24]:
```
import statistics
y = arr[np.arange(0, len(arr), 1)]
date = pd.to_datetime("1st of January, 1999")
date2 = date + pd.to_timedelta(np.arange(len(y)), 'H')
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()
L = len(series)
```

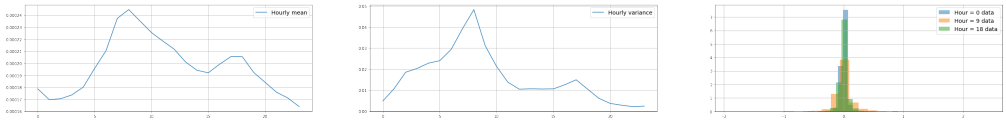
```
z = np.log(series.array[np.arange(1, L, 1)]) - np.log(series.
 ↪array[np.arange(0, L-1, 1)])
s = pd.date_range('1999-01-01', '2007-01-26', freq='D').
 ↪to_series()
weekDaysArray = s.dt.dayofweek.array
values = series.array
plt.figure(figsize=([40, 10]))

for j,i in enumerate(['Monday', 'Tuesday',␣
 ↪'Wednesday','Thursday', 'Friday', 'Saturday','Sunday']):
    arr0 = values[weekDaysArray == j]
    L = len(values)
    z = np.log(values[np.arange(1, L, 1)]) - np.log(values[np.
 ↪arange(0, L-1, 1)])
    plt.subplot(2, 4, j+1)
    plt.hist(z, bins= int(np.ceil(1+ 3.322*np.log(len(z)))),␣
 ↪alpha=0.5, density = True, label=r'Week')
    L = len(arr0)
    z = np.log(arr0[np.arange(1, L, 1)]) - np.log(arr0[np.
 ↪arange(0, L-1, 1)])
    plt.hist(z, bins= int(np.ceil(1+ 3.322*np.log(len(z)))),␣
 ↪alpha=0.5, density = True, label=i)
    plt.legend(loc='upper right', fontsize=22)
    plt.grid()

plt.show()
```



Now let's resample the data that we have for every hour. We estimate the mean and the variance and plot histograms for the density of the returns. Now we see, that the returns are higher in the mornings, which does agree with the expectation. Moreover, increase in expectation in this case will cause the increase in variance as well, as we are dealing with the spot prices.

[7]:
```
from numpy import loadtxt
arr = loadtxt("TimeSeries2.txt")
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statistics

plt.close("all")
y = arr[np.arange(0, len(arr), 1)]
date = pd.to_datetime("1st of January, 1999")
date2 = date + pd.to_timedelta(np.arange(len(y)), 'H')
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()
L = len(series)
hourofDayArray = date2.hour.array

# select a given hour of the day between 0 and 23
m=[]
v=[]
plt.figure(figsize=([45, 5]))
for hourNum in range(0, 24):
    arrN = y[hourofDayArray == hourNum]
    L2 = len(arrN)
    z = np.log(arrN[np.arange(1, L2, 1)]) - np.log(arrN[np.
 ↪arange(0, L2-1, 1)])
```

```
    m.append(statistics.mean(z))
    v.append(statistics.variance(z))
    if hourNum % 9 == 0:
        plt.subplot(1,3,3)
        plt.hist(z, bins= int(np.ceil(1+ 3.322*np.log(len(z)))),␣
 ↪alpha=0.5, density = True, label='Hour = %s data' % hourNum)
plt.legend(loc='upper right', fontsize='x-large')
plt.grid()
plt.subplot(1,3,1)
plt.plot(range(0, 24),m, label=r'Hourly mean')
plt.legend(loc='upper right', fontsize='x-large')
plt.grid()
plt.subplot(1,3,2)
plt.plot(range(0, 24),v,label=r'Hourly variance')
plt.legend(loc='upper right', fontsize='x-large')
plt.grid()
plt.show()
```



2.0.1 Kernel density approximation

Kernel density estimation is the method for estimating density function using a kernel function $K(x)$. Histograms, we have used previously essentially count number of datapoints in a region, which is influenced by the data range and number of bins, which is specified by the user. On the other hand a kernel density estimate is a function defined as the sum of a kernel function on every data point. The kernel function $K(x)$ usually satisfies the following properties:

1. Symmetricity: $K(x) = K(-x)$ 2. Normalization: $\int_{-\infty}^{\infty} K(x)\,dx = 1$ 3. Non-negativity: $K(x) \geq 0$, $\forall x$ 4. Nonincresing for $x \geq 0$ (and nondecreasing for $x \leq 0$)

There are several types of kernels, which satisfy these criteria, among the most popular being (with symbol $\propto$ meaning that there is a normalisation constant, such that $\int_{-\infty}^{\infty} K(x,h)\,dx = 1$) 1. gaussian: $K(x,h) \propto \exp(-\frac{x^2}{2h^2})$ 2. tophat: $K(x,h) \propto 1$, $-h \leq x \leq h$ 3. epanechnikov: $K(x,h) \propto 1 - \frac{x^2}{h^2}$ 4. exponential: $K(x,h) \propto \exp(-x/h)$ 5. linear: $K(x,h) \propto 1 - x/h$, if $0 < x < h$ 6. cosine: $K(x,h) \propto \cos(\frac{\pi x}{2h})$, if $0 < x < h$.

The parameter h is called a bandwidth. The forms of the kernels above is presented on a Figure below.

Now given the kernel form $K(x)$ and a bandwidth h, the estimate of the density $f(x)$ at a point x given observation $y_1, \ldots, y_n$ is of the form:

$$\hat{f}(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - y_i}{h}\right).$$

Let's introduce the concept of the Mean Square Error (MSE):

$$\begin{aligned} \mathbb{E}\left[\left(\hat{f} - f\right)^2\right] &= \mathbb{E}\left[\left(\hat{f} + \mathbb{E}\hat{f} - \mathbb{E}\hat{f} - f\right)^2\right] \\ &= \mathbb{E}\left[\left(\hat{f} - \mathbb{E}\hat{f}\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}\hat{f} - f\right)^2\right] \end{aligned} \tag{2.1}$$

As one can see the MSE has been decomposed into two errors: 1. $\left(\mathbb{E}\hat{f} - f\right)^2$ — squared bias error; 2. $\mathbb{E}\left[\left(\hat{f} - \mathbb{E}\hat{f}\right)^2\right]$ — Monte Carlo variance.

The Monte Carlo variance is proportional to $N^{-1}h^{-}1$ as

$$\mathbb{V}\hat{f} = \mathbb{V}\left(\frac{1}{Nh}\sum_{i=1}^{N}\hat{f}\right) = \frac{1}{N^2h^2}\mathbb{V}\left(\sum_{i=1}^{N}\hat{f}\right) = \frac{1}{Nh^2}\mathbb{V}\left(\hat{f}\right).$$

Now as

$$\mathbb{V}\left(\hat{f}\right) \leq \mathbb{E}\hat{f}^2 \approx h,$$

we have that the variance based on N observations with kernel bandwith h is proportional to $\frac{1}{Nh}$.

Now for the bias we can, using Taylors decomposition, quite easily see that

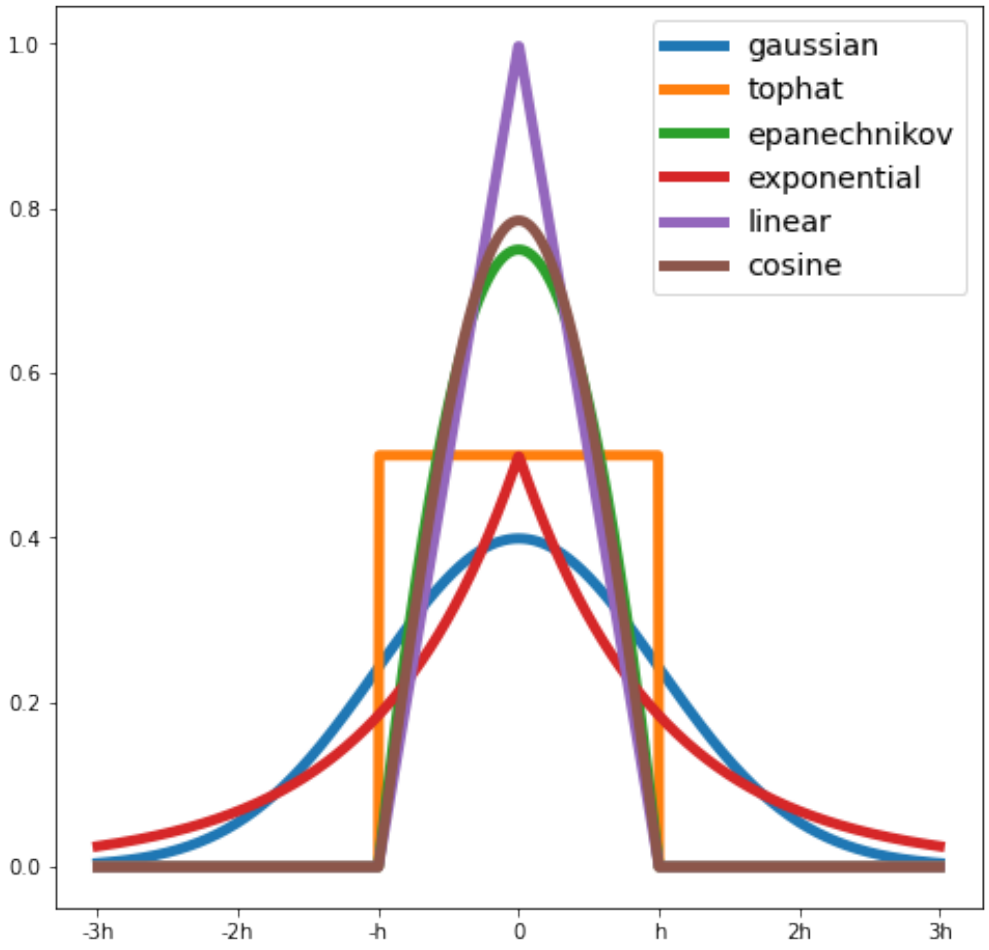
$$\frac{1}{h}\int K\left(\frac{x-y}{h}\right) f(x)dx - f(y) = \int K(x)f(y-hx)dx - f(y)$$
$$= f(y)\int K(x)f(x)dx + hf'(y)\int xK(x)f(x)dx + f''h^2\int x^2K(x)f(x)dx + o(h^2) - f(y)$$
$$\preceq ch^2 + o(h^2),$$

where c is the constant depending on the choice of kernel and the second order derivative of the underlying density. Now we can see, that the choice of bandwidth h is quite important, as there is a bias-variance tradeoff. Whenever we increase h. we increase bias and decrease variance and vice versa.

[25]:
```
from scipy.stats import norm
from sklearn.neighbors import KernelDensity
from sklearn.utils.fixes import parse_version

# Plot example of kernels
X_plot = np.linspace(-3, 3, 1000)[:, None]
X_src = np.zeros((1, 1))

plt.figure(figsize=([8, 8]))

for i, k in enumerate(['gaussian', 'tophat', 'epanechnikov',
                       'exponential', 'linear', 'cosine']):
    log_dens = KernelDensity(kernel=k).fit(X_src).
↪score_samples(X_plot)
    plt.plot(X_plot[:, 0], np.exp(log_dens), label=k, linewidth=5)
```

```
plt.xticks(np.arange(-3,4,1), ['-3h',␣
 ↪'-2h','-h','0','h','2h','3h'])
plt.legend(loc='upper right', fontsize='x-large')
```

[25]: <matplotlib.legend.Legend at 0x1aee2b7fac0>



Each optimal bandwidth value is unique to the choice of kernel. So for each kernel we run a grid search algorithm. Essentially for 4000 log-equidistantly spaced values of bandwidth between 10^{-2} and 10^{2} we compute the goodness of fit and choose the best one. Need to rerun the simulations on the finer grid

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Gaussian	0.06400318	0.06519339	0.05664848	0.05037042	0.07910876	0.04376836
Tophat	0.16722678	0.30294957	0.21248674	0.15463127	0.49822367	0.08774771
Epanechnikov	0.17510995	0.30434827	0.21445335	0.16080575	0.49937248	0.10262492
Exponential	0.04267343	0.04247731	0.03829533	0.03412976	0.04520264	0.03381678
Linear	0.17836631	0.30505004	0.21544347	0.16266826	0.49937248	0.10895805
Cosine	0.17632406	0.30434827	0.21445335	0.16154818	0.49937248	0.10309873

though

[9]:
```
# standalone script for estimating the optimal bandwidths
from sklearn.neighbors import KernelDensity
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import LeaveOneOut
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from numpy import loadtxt
arr = loadtxt("TimeSeries2.txt")
y = arr[np.arange(0, len(arr), 1)]
date = pd.to_datetime("1st of January, 1999")
date2 = date + pd.to_timedelta(np.arange(len(y)), 'H')
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()
L = len(series)
z = np.log(series.array[np.arange(1, L, 1)]) - np.log(series.
 ↪array[np.arange(0, L-1, 1)])

s = pd.date_range('1999-01-01', '2007-01-26', freq='D').
 ↪to_series()
weekDaysArray = s.dt.dayofweek.array
values = series.array
optimal_params=np.zeros((6,7))
```

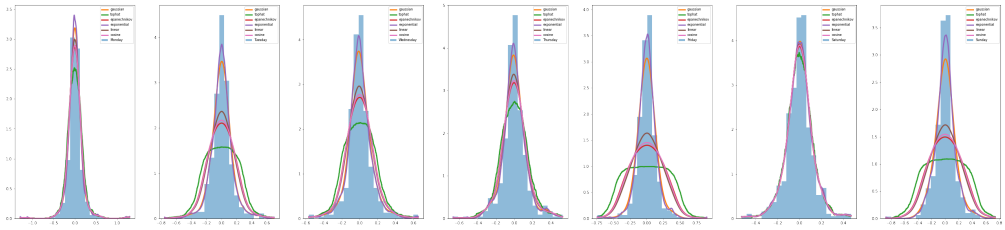
```
bandwidths = 10 ** np.linspace(-2, 2, 4000)
print(bandwidths)
for j, k in enumerate(['gaussian', 'tophat', 'epanechnikov',
                           'exponential', 'linear', 'cosine']):
    for i in range(0,7):
        arr0 = values[weekDaysArray == i]
        L = len(arr0)
        z = np.log(arr0[np.arange(1, L, 1)]) - np.log(arr0[np.
 ↪arange(0, L-1, 1)])
        z1 = np.array(z[:])
        grid = GridSearchCV(KernelDensity(kernel=k),
                        {'bandwidth': bandwidths})
        grid.fit(z1[:, None])
        optimal_params[j,i] = float(grid.
 ↪best_params_['bandwidth'])

print(optimal_params)
```

```
[1.00000000e-02 1.00230582e-02 1.00461695e-02 ... 9.95404271e+01
 9.97699489e+01 1.00000000e+02]
[[0.06400318 0.06519339 0.05664848 0.05037042 0.07910876 0.04376836
  0.08132565]
 [0.16722678 0.30294957 0.21248674 0.15463127 0.49822367 0.08774771
  0.45437492]
 [0.17510995 0.30434827 0.21445335 0.16080575 0.49937248 0.10262492
  0.45647275]
 [0.04267343 0.04247731 0.03829533 0.03412976 0.04520264 0.03381678
  0.04657652]
 [0.17836631 0.30505004 0.21544347 0.16266826 0.49937248 0.10895805
  0.45647275]
 [0.17632406 0.30434827 0.21445335 0.16154818 0.49937248 0.10309873
  0.45647275]]
```

Now we will plot the results on the datasets and see, how good do the choice of parameters fit the distribution of the returns for each day of the week. As we can see, the kernels approximate the data set with a various degree of success. We will stick to the exponential kernel from now on.

[10]:
```
from sklearn.neighbors import KernelDensity
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy import loadtxt
arr = loadtxt("TimeSeries2.txt")
y = arr[np.arange(0, len(arr), 1)]
date = pd.to_datetime("1st of January, 1999")
date2 = date + pd.to_timedelta(np.arange(len(y)), 'H')
ts = pd.Series(y, index = date2)
series = ts.resample('24H').sum()

z = np.log(series.array[np.arange(1, L, 1)]) - np.log(series.
 ↪array[np.arange(0, L-1, 1)])
plt.figure(figsize=([45, 10]))
s = pd.date_range('1999-01-01', '2007-01-26', freq='D').
 ↪to_series()
weekDaysArray = s.dt.dayofweek.array
values = series.array
for j,i in enumerate(['Monday', 'Tuesday',␣
 ↪'Wednesday','Thursday', 'Friday', 'Saturday','Sunday']):
    plt.subplot(1, 7, j+1)
    arr0 = values[weekDaysArray == j]
    L = len(arr0)
    z = np.log(arr0[np.arange(1, L, 1)]) - np.log(arr0[np.
 ↪arange(0, L-1, 1)])
    z1 = np.array(z[:])
```

```
    plt.hist(z, bins= int(np.ceil(1+ 3.322*np.log(len(z)))),␣
↪alpha=0.5, density = True, label=i)
    for l, k in enumerate(['gaussian', 'tophat',␣
↪'epanechnikov','exponential', 'linear', 'cosine']):
        kde = KernelDensity(kernel=k,␣
↪bandwidth=optimal_params[l,j]).fit(z1[:, np.newaxis])
        X_plot = np.linspace(min(z1), max(z1), 1000)[:, np.
↪newaxis]
        log_dens = kde.score_samples(X_plot)
        plt.plot(X_plot[:, 0], np.exp(log_dens), lw=3,␣
↪linestyle='-', label=k)
        plt.legend(loc='upper right', fontsize=8)
        plt.grid()

plt.show()
```



Bibliography

William A Brock, David Arthur Hsieh, Blake Dean LeBaron, William E Brock, et al. *Nonlinear dynamics, chaos, and instability: statistical theory and economic evidence*. MIT press, 1991.

N. Flatabo, G. Doorman, O.S. Grande, H. Randen, and I. Wangensteen. Experience with the nord pool design and implementation. *IEEE Transactions on Power Systems*, 18(2):541–547, 2003. doi: 10.1109/TPWRS.2003.810694.

Neil F Johnson, Paul Jefferies, Pak Ming Hui, et al. Financial market complexity. *OUP Catalogue*, 2003.

Maury FM Osborne. Brownian motion in the stock market. *Operations research*, 7(2):145–173, 1959.

Yan Wang. Renewable electricity in sweden: an analysis of policy and regulations. *Energy policy*, 34(10):1209–1220, 2006.