Time Series Analysis and Forecasting 2019

Assignment

Problem 1

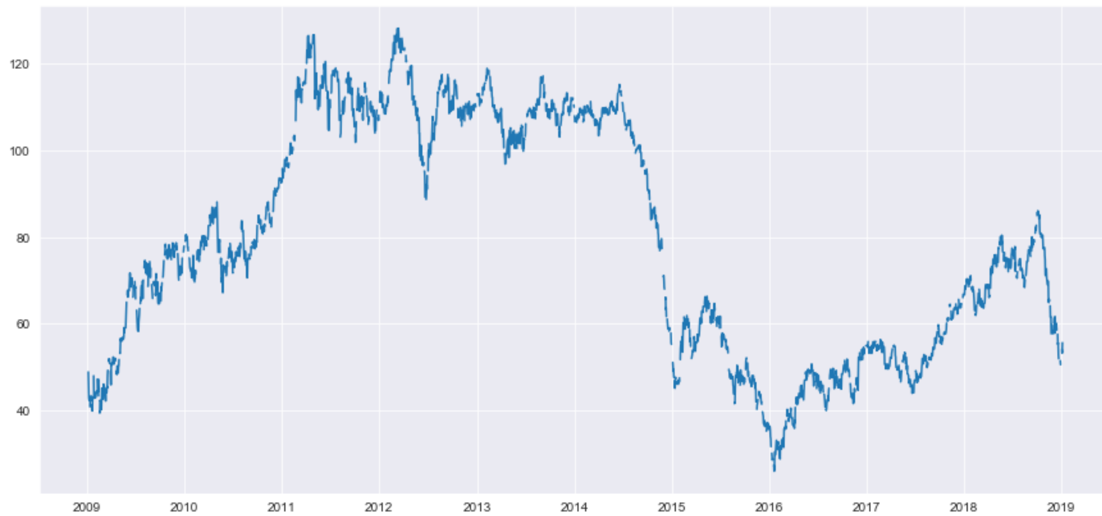
Solutions are by Yaroslava Lochman

In [3]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
%load_ext rpy2.ipython

from os.path import join as pjoin
import numpy as np
from numpy import hstack as stack
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

import seaborn as sns
sns.set_style('darkgrid')
from matplotlib import pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (15,10)
mpl.rcParams['image.cmap'] = 'inferno'


from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error as MSE
from statsmodels.tsa.stattools import acf

from utils import *
```

Trend extraction for non-seasonal time series of the crude oil price

In [7]:
```
mpl.rcParams['figure.figsize'] = (15,7)

CrudeOil = pd.read_csv('data/CrudeOil.csv', header=0, index_col=0,
```

```
                             squeeze=True)
N = len(CrudeOil)
dateIndex = pd.DatetimeIndex(start=CrudeOil.index[0], end=CrudeOil.index[-1],
                                   freq='D')
train = pd.Series(data=[np.nan]*len(dateIndex), index=dateIndex)
train.loc[CrudeOil.index.astype('datetime64[ns]')] = CrudeOil.values
plt.plot(train)
plt.show()
```



Before extracting the trend of this TS first we need to deal with missing values. Suggestion: interpolation with splines.

```
In [13]: train = train.interpolate(method='spline', order=3)

         mpl.rcParams['figure.figsize'] = (15,7)
         show_series(train, x_freq=400)
         plt.show()
         print('Closer look:')
         show_series(train[:300], x_freq=100)
         plt.show()
```

Closer look:

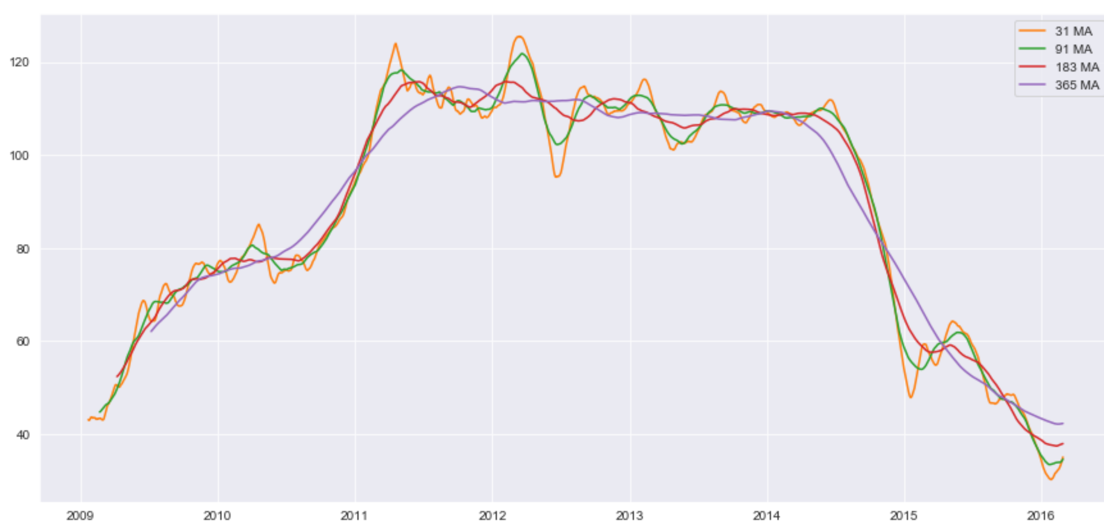
(a) Try several moving average techniques to extract the trend (ma in R). Which orders/form of moving averages do provide the best results? Use just gures of the trend and the dierence Y_tT_t.

```
In [14]: trends = {'simple': {}, 'centered': {}, 'double': {}}
         for i, p in enumerate([31,91,183,365]):
             trends['simple'][p] = trend_MA(train, p, 'simple')
```

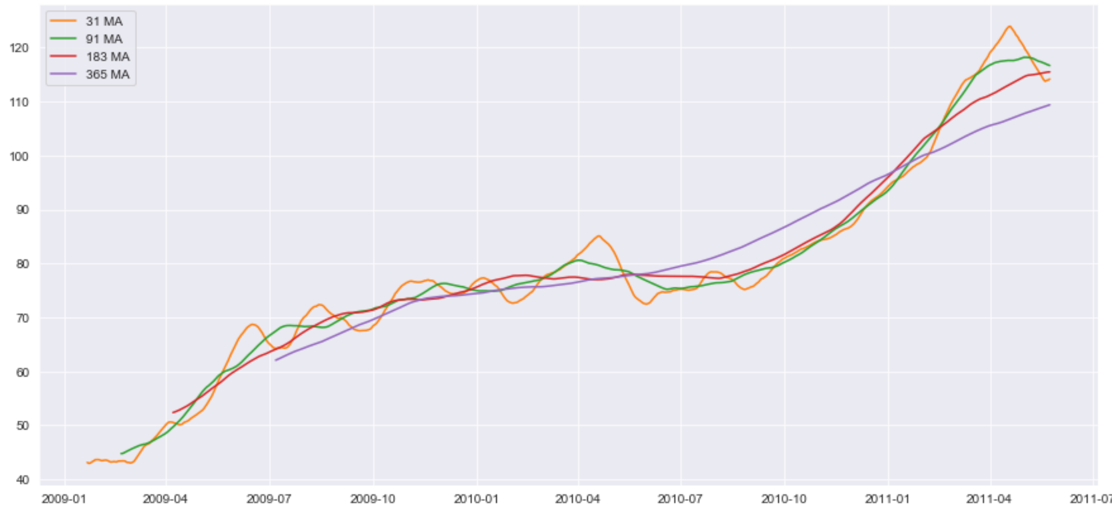
```
for i, p in enumerate([30,92,182,366]):
    trends['centered'][p] = trend_MA(train, p, 'centered')

for i, p in enumerate([31,91,183,365]):
    trends['double'][p] = trend_MA(train, p, 'double')
```

```
In [15]: mpl.rcParams['figure.figsize'] = (15,7)
         print('Simple Moving Average')
         for k in [N, N//3]:
             if k != N:
                 print('Closer:')

             for i, p in enumerate(trends['simple'].keys()):
                 trend = trends['simple'][p]
                 plt.plot(trend[(p - 1)// 2:k], color='C{}'.format((i+1)%10),
                          label='{} MA'.format(p))
             plt.legend()
             plt.show()

         print('Differences:')
         for i, p in enumerate(trends['simple'].keys()):
             trend = trends['simple'][p]
             plt.plot(train - trend[(p - 1)// 2:], color='C{}'.format((i+1)%10),
                      label='{} MA'.format(p))
         plt.legend()
         plt.show()
```

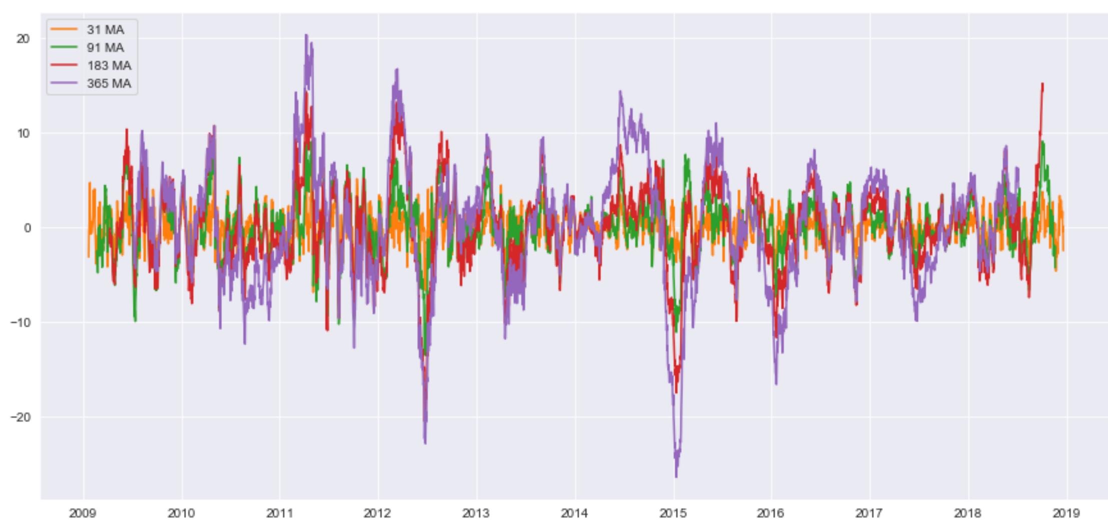
```
Simple Moving Average
```



Closer:



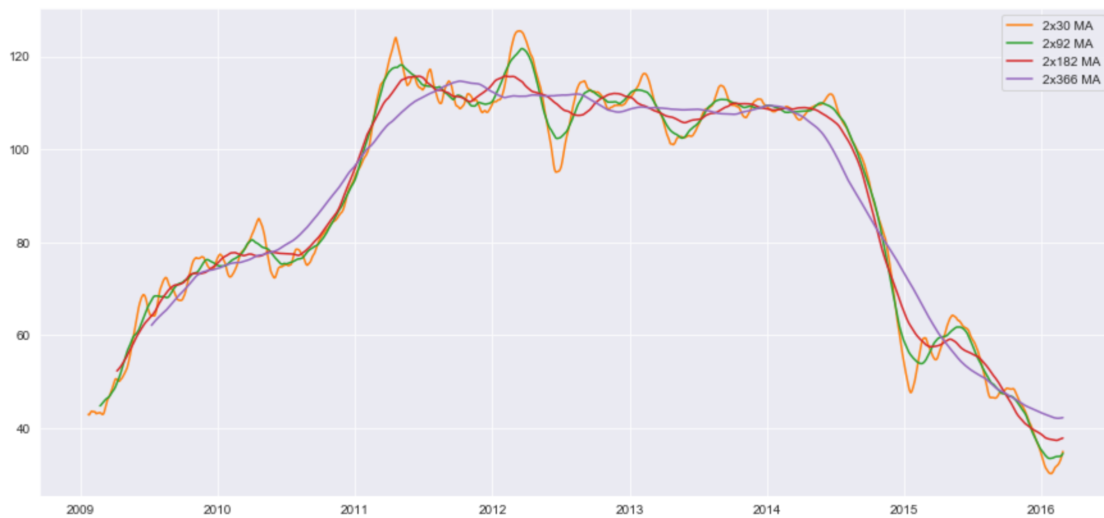
Differences:



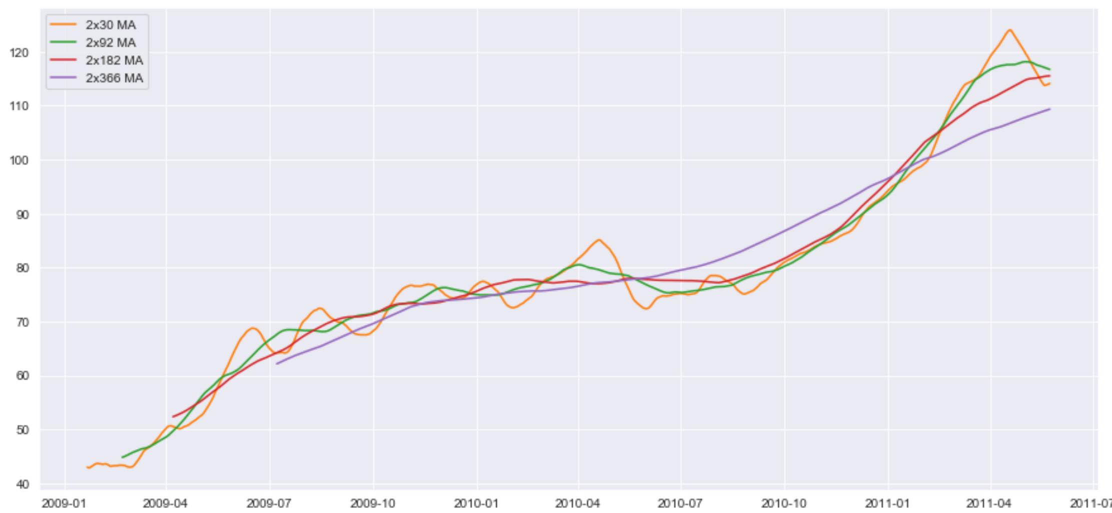
In [16]:
```
print('Centered Moving Average')
for k in [N, N//3]:
    if k != N:
        print('Closer:')

    for i, p in enumerate(trends['centered'].keys()):
```

```
            trend = trends['centered'][p]
            plt.plot(trend[p//2:k], color='C{}'.format((i+1)%10),
                     label='2x{} MA'.format(p))
        plt.legend()
        plt.show()

    print('Differences:')
    for i, p in enumerate(trends['simple'].keys()):
        trend = trends['simple'][p]
        plt.plot(train - trend[p// 2:], color='C{}'.format((i+1)%10),
                 label='2x{} MA'.format(p))
    plt.legend()
    plt.show()
```

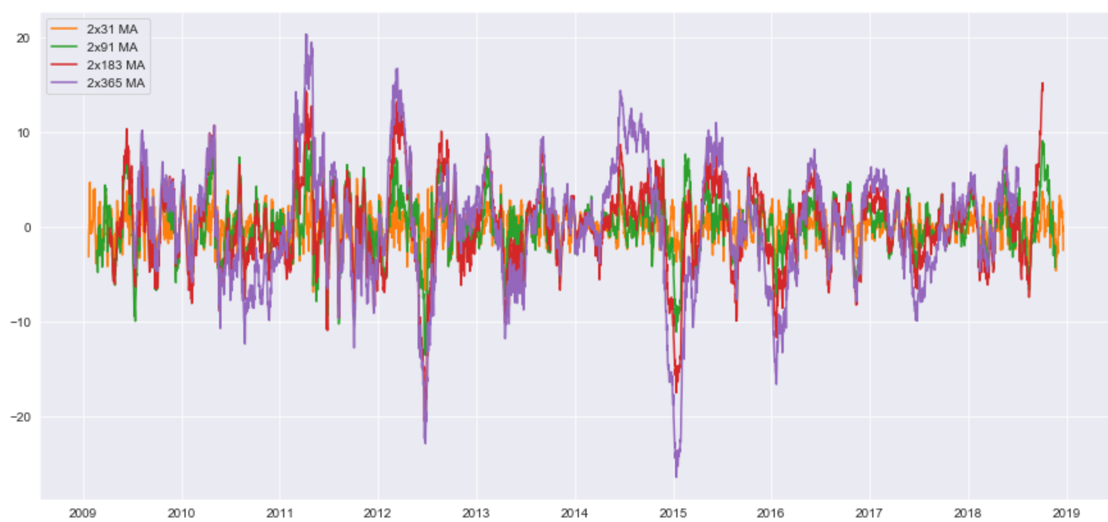
```
Centered Moving Average
```




```
Closer:
```



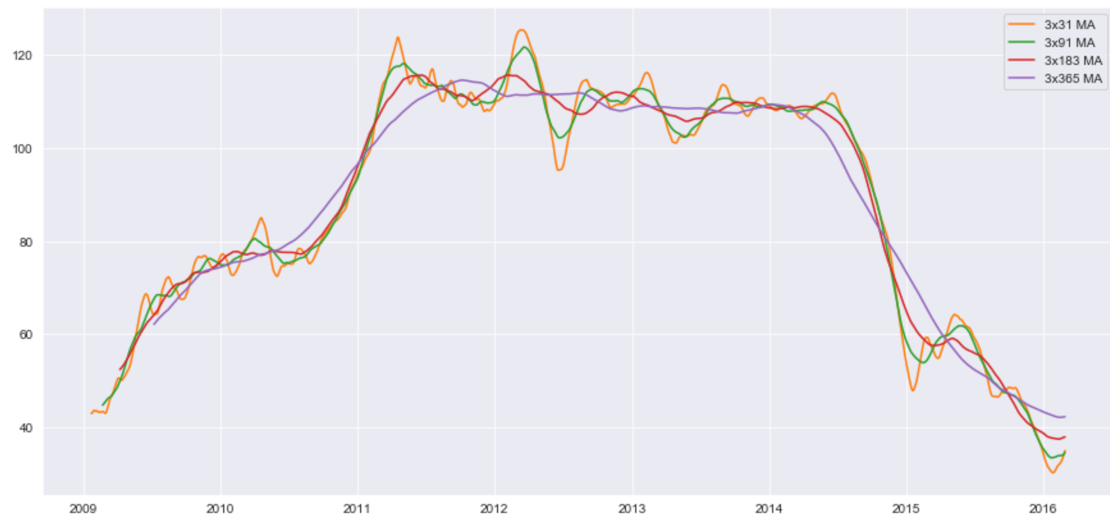
Differences:



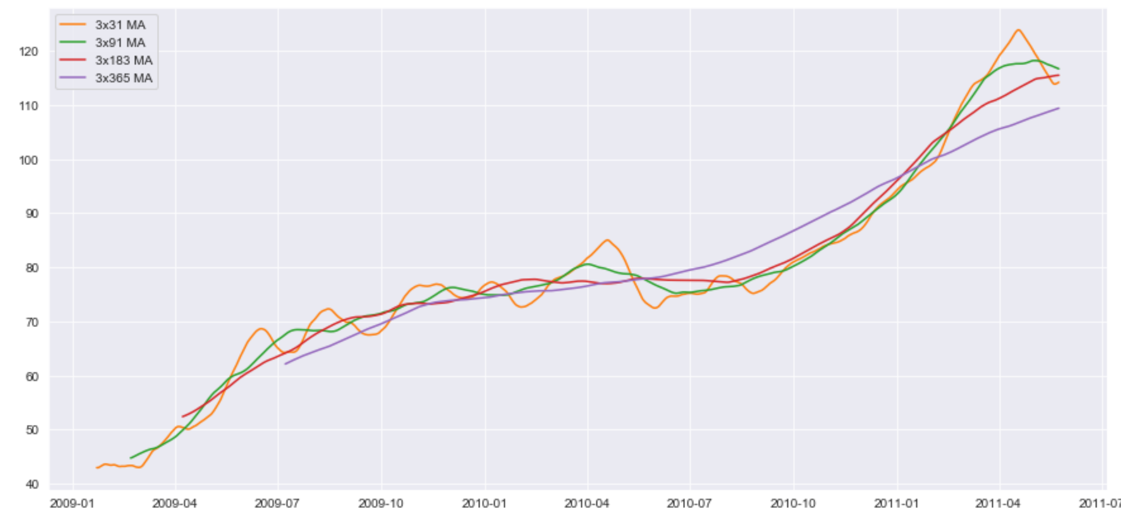

```
In [17]: print('Double Moving Average')
         for k in [N, N//3]:
             if k != N:
                 print('Closer:')

             for i, p in enumerate(trends['double'].keys()):
                 trend = trends['double'][p]
                 plt.plot(trend[(p - 1)// 2:k], color='C{}'.format((i+1)%10),
```

```
                        label='3x{} MA'.format(p))
        plt.legend()
        plt.show()

    print('Differences:')
    for i, p in enumerate(trends['simple'].keys()):
        trend = trends['simple'][p]
        plt.plot(train - trend[(p - 1)// 2:], color='C{}'.format((i+1)%10),
                 label='3x{} MA'.format(p))
    plt.legend()
    plt.show()
```

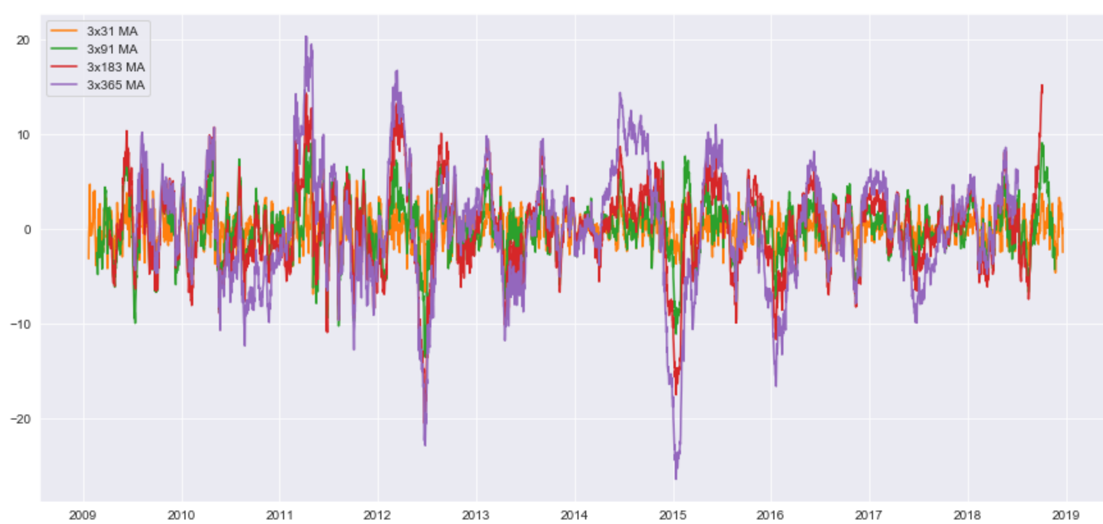
Double Moving Average



Closer:



`Differences:`



Almost all the residuals look stationary. The bigger is order of moving average, the higher is residual variability, but the smoother is the trend model. However we may be interested in different periods changes. I would say, two models look very good: 3-month moving average removes the irregular component and preserves the tendency over quarters, and 1-year moving average eliminates most of fluctuations and explains changes over years.

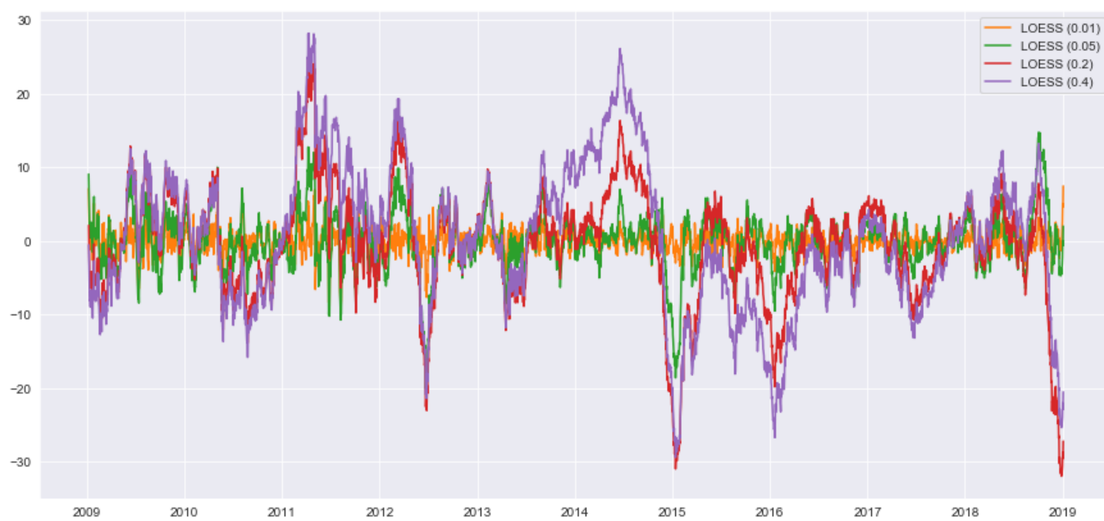
(b) Does it make sense to apply centered moving average (2 x k MA) smoothing? Explain and motivate your answer.

We have daily observations, therefore we may apply moving average for e.g. 2 weeks (14 days), month (30 days), 3 months (92 days), half-year (182 days) etc. So it makes sense to apply centered moving average. And simple / double as well (for 31, 91, 183, 365 days corresponding to a month, quater, half-year and year).

(c) Since the time series exhibits a very irregular trend apply and visualize the local polynomial regression. In your particular implementation verify the functional form of the weights used for trend extraction.

```
In [18]: train_ = train.copy()
         train_.index = np.arange(len(train))
```

```
In [79]: mpl.rcParams['figure.figsize'] = (15,7)
         from statsmodels.nonparametric.smoothers_lowess import lowess

         show_series(train, x_freq=400)
         trends['LOESS'] = {}
         fracs = [0.01, 0.05, 0.2, 0.4]
         for i, frac in enumerate(fracs):
             trend = lowess(train_.values, train_.index, frac=frac,
                            missing='drop', return_sorted=False)
             trend = pd.Series(trend, index=train.index)
             trends['LOESS'][frac] = trend
             plt.plot(trend, color='C{}'.format(i+1), label='LOESS ({})'.format(frac))
         plt.legend()
         plt.show()

         for i, frac in enumerate(fracs):
             trend = trends['LOESS'][frac]
             plt.plot(train - trend, color='C{}'.format((i+1)%10),
                      label='LOESS ({})'.format(frac))
         plt.legend()
         plt.show()
```



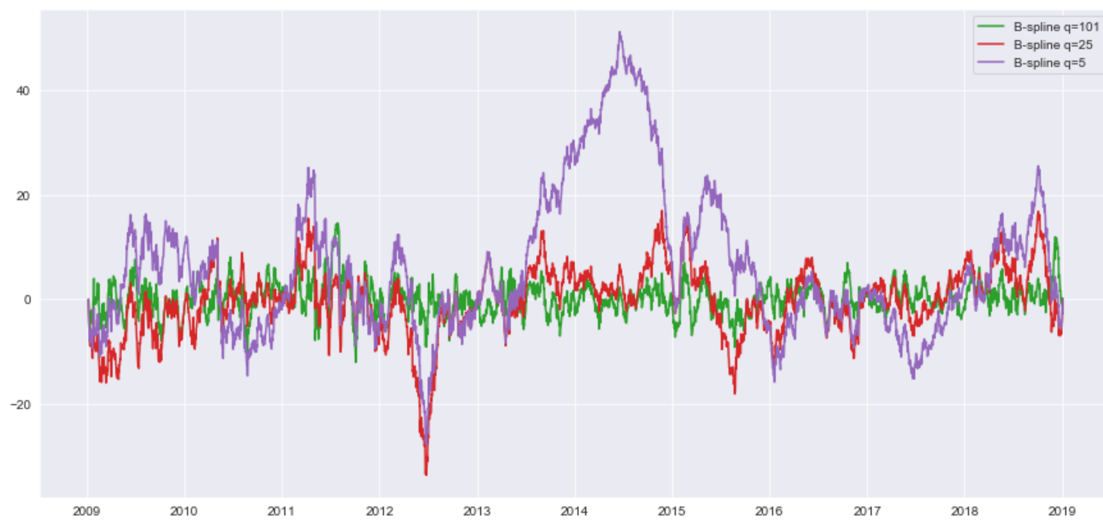
In this implementation the weight function is a tricube function: $w(u) = (1 - |u|^3)^3$ if $|u| \leq 1$ otherwise 0.

The fraction used for local estimate is chosen to be 5%, 20%, 40% (corresponding to different the bandwidth parameter values). The model with fraction 20% looks like the best for trend extracting.

(d) Apply the B-spline approach to extract the trend. Explain precisely how many splines you use, the underlying time grid and the order of polynomials.

```
In [92]: mpl.rcParams['figure.figsize'] = (15,7)
         import scipy.interpolate as interpolate
```

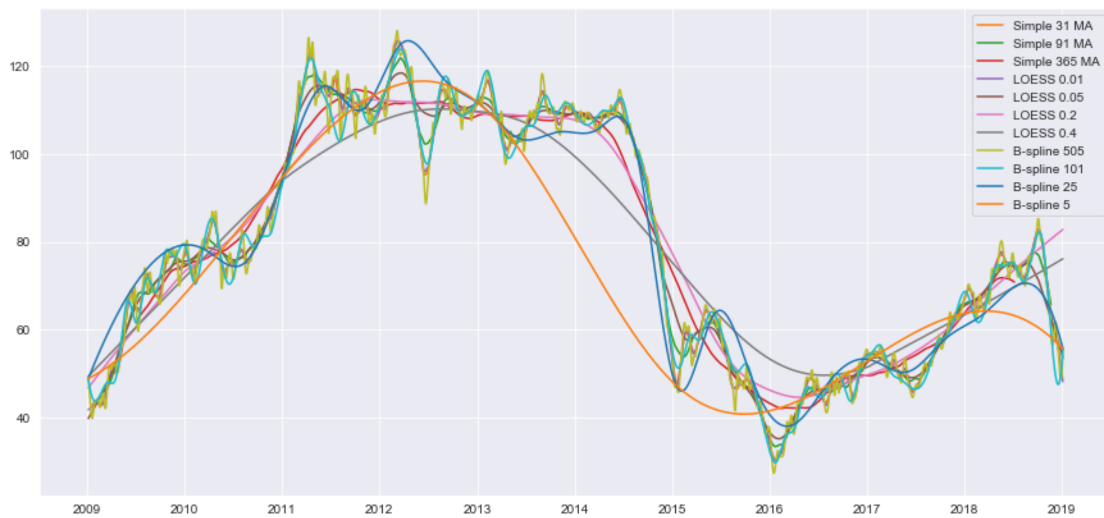
```python
import matplotlib.pyplot as plt

X = []
for date in CrudeOil.dropna().index.astype('datetime64[ns]').date:
    X.append(train.index.get_loc(date))
X = np.array(X)
Y = CrudeOil.dropna().values
full_N = len(X)

show_series(train)

trends['spline'] = {}
for i, freq in enumerate(divisors(full_N-1)[1:]):
    q = (full_N-1) // freq
    x = X[:(full_N-1)+freq:freq]
    y = Y[:(full_N-1)+freq:freq]

    t, c, k = interpolate.splrep(x, y, k=3)
    spline = interpolate.BSpline(t, c, k, extrapolate=False)
    spline_model = pd.Series(spline(train_.index), index=train.index)
    trends['spline'][q] = spline_model
    if q != 505:
        plt.scatter(train.index[x], y, color='C{}'.format(i+1))
        plt.plot(spline_model, color='C{}'.format(i+1),
                 label='B-spline q={}, k={}'.format(q, k))
        plt.legend()
plt.show()

for i, q in enumerate(trends['spline'].keys()):
    if q != 505:
        trend = trends['spline'][q]
        plt.plot(train - trend, color='C{}'.format((i+1)%10),
                 label='B-spline q={}'.format(q))
plt.legend()
plt.show()
```

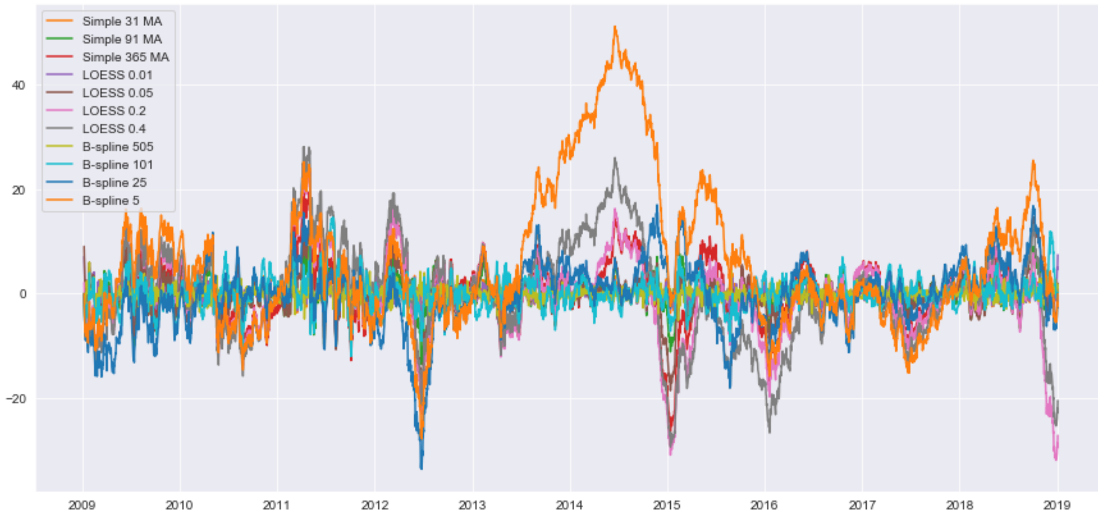


The q parameter was chosen with respect to the length of TS s.t. to get equally spaced time grid, and more specifically: $q_1 = 101$ (time grid width is 25), $q_2 = 25$ (time grid width is 101), $q_3 = 5$ (time grid width is 505). The lower values were not considered due to inadequate results. The same reason is for values of $q > 101$ – such interpolations keep the irregular components/seasonalities. The degree (order) of polynomials for all the three cases is chosen to be $k = 3$ since it is recommended to use it, and other degree values give much worse results.

(e) For every of the above approaches extract the irregular component (there is no seasonal component) and plot its autocorrelation.

In [83]:
```
T = {}
T['Simple 31 MA'] = trends['simple'][31]
T['Simple 91 MA'] = trends['simple'][91]
T['Simple 365 MA'] = trends['simple'][365]
T['LOESS 0.01'] = trends['LOESS'][0.01]
T['LOESS 0.05'] = trends['LOESS'][0.05]
T['LOESS 0.2'] = trends['LOESS'][0.2]
T['LOESS 0.4'] = trends['LOESS'][0.4]
T['B-spline 505'] = trends['spline'][505]
T['B-spline 101'] = trends['spline'][101]
T['B-spline 25'] = trends['spline'][25]
T['B-spline 5'] = trends['spline'][5]


mpl.rcParams['figure.figsize'] = (15,7)
for i, p in enumerate(T.keys()):
    plt.plot(T[p], color='C{}'.format((i+1)%10), label='{}'.format(p))
plt.legend()
plt.show()

for i, p in enumerate(T.keys()):
    plt.plot(train-T[p], color='C{}'.format((i+1)%10), label='{}'.format(p))
plt.legend()
plt.show()
```

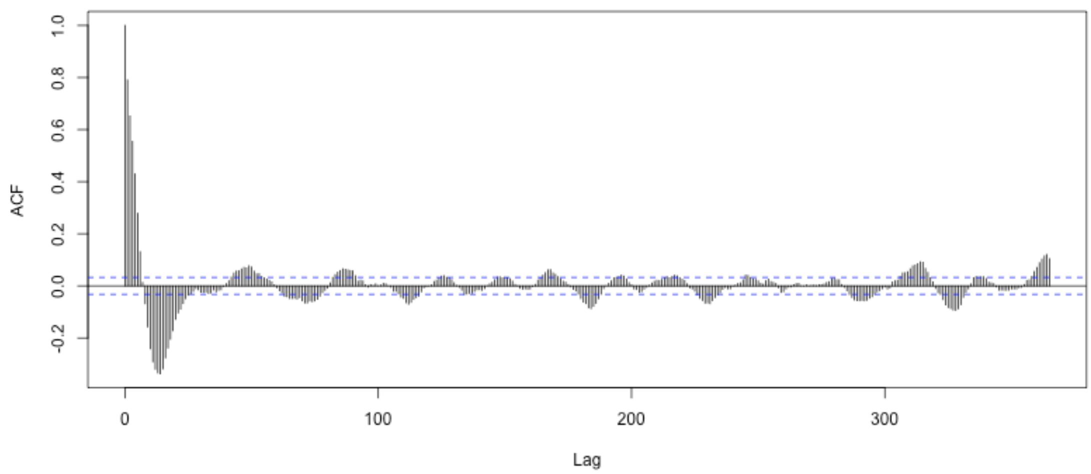




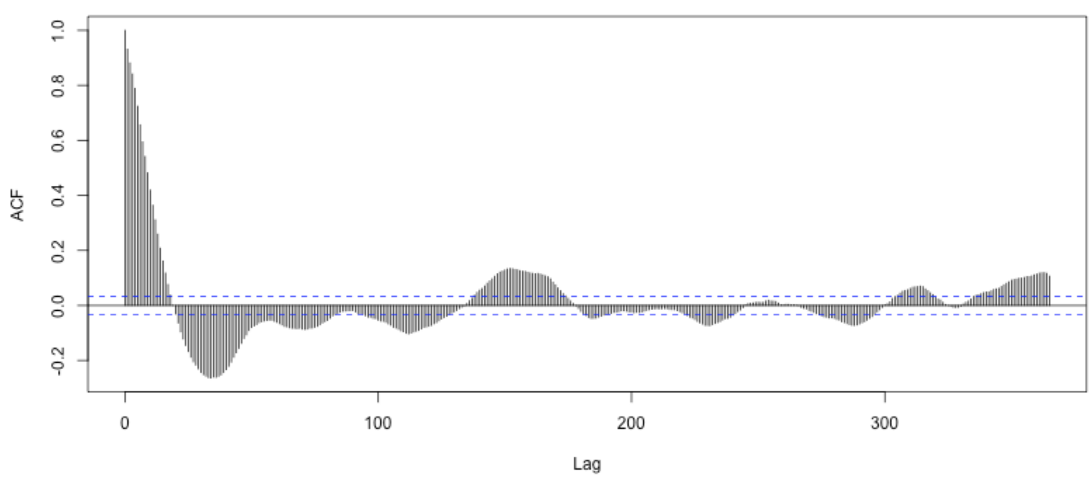
In [84]:
```
Model, Res = [], []
for p in T.keys():
    Model.append(p)
    Res.append(list((T[p] - train).dropna()))
Res = np.array(Res)
```

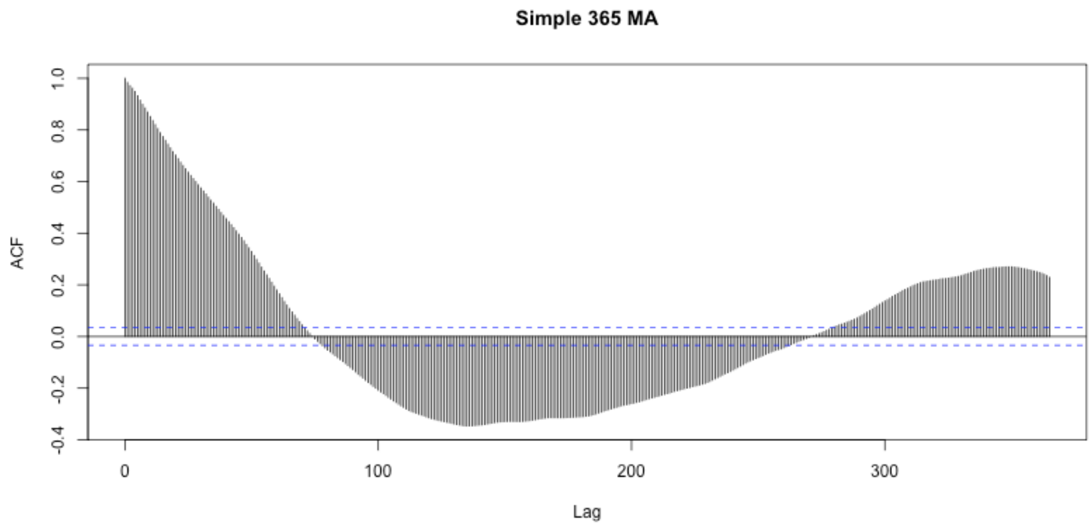
In [85]:
```
%%R -i Res -i Model -w 800 -h 400 -u px
library('forecast')
library('tseries')

names(Res) <- Model
for (name in names(Res))
{
AutoCorrelation <- acf(as.numeric(Res[[name]]), lag.max=365, plot = FALSE)
plot(AutoCorrelation, main = name)
}
```

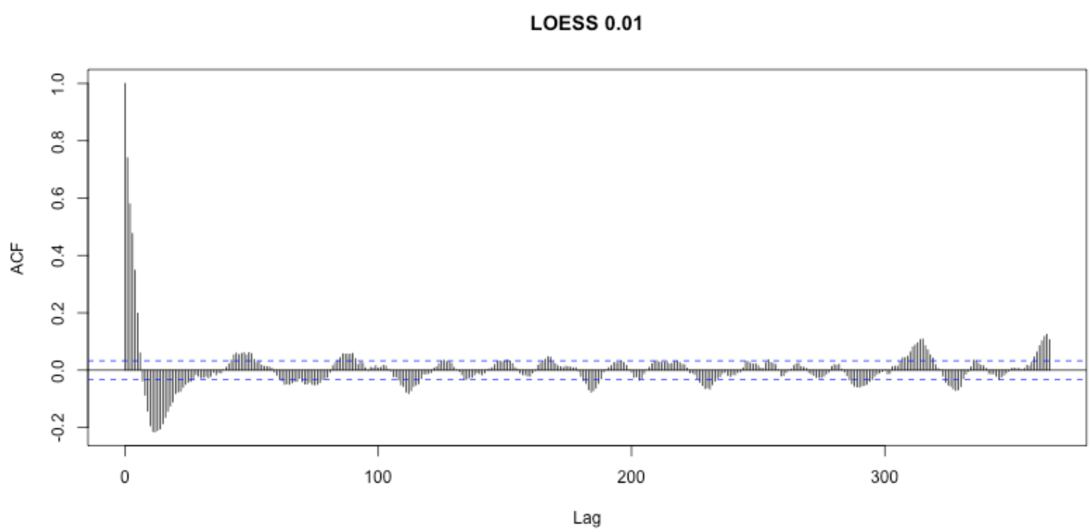
Simple 31 MA

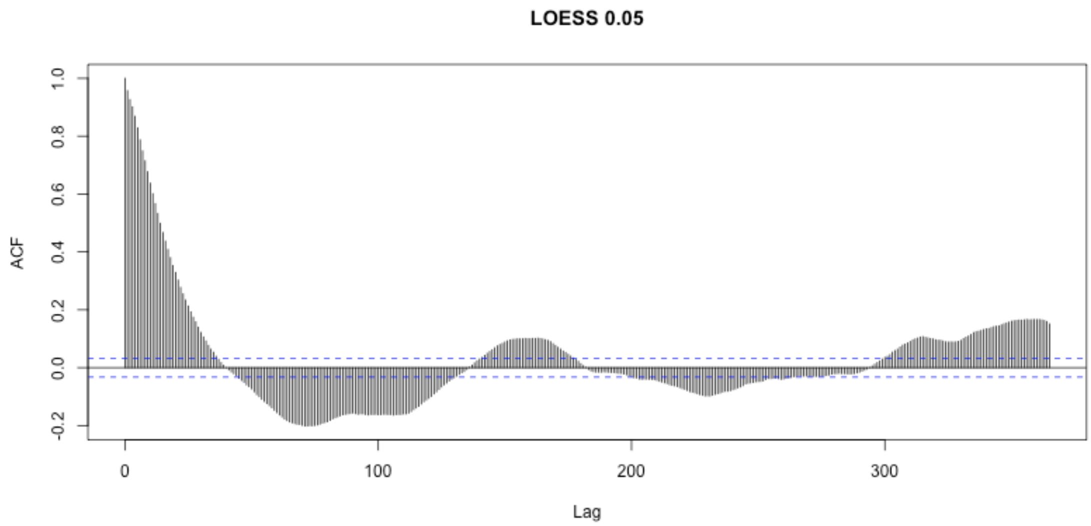


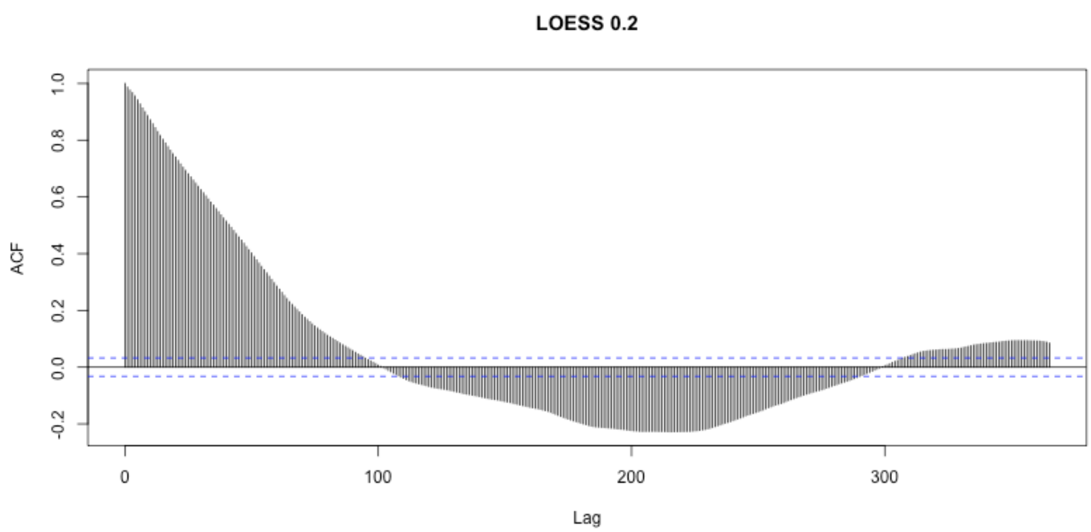
Simple 91 MA

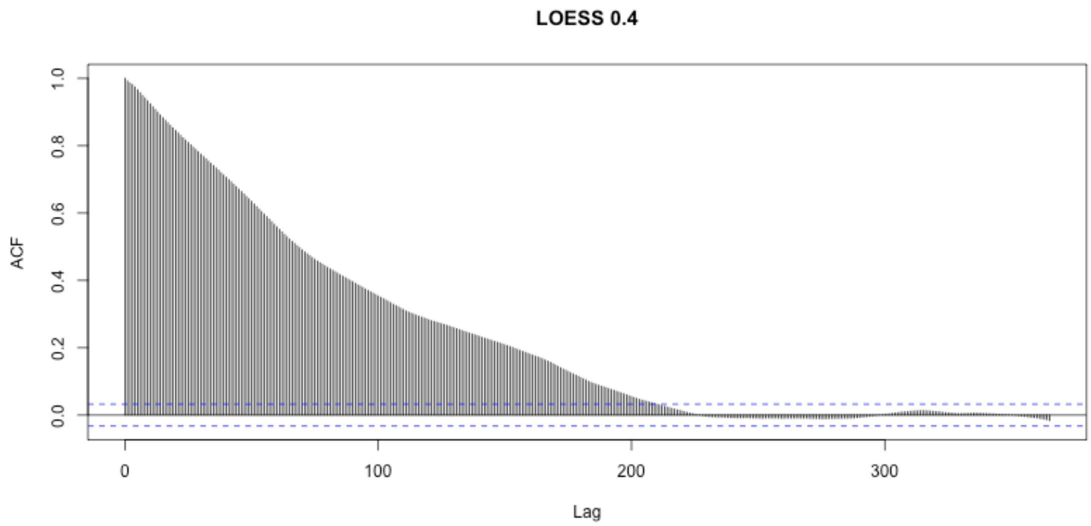


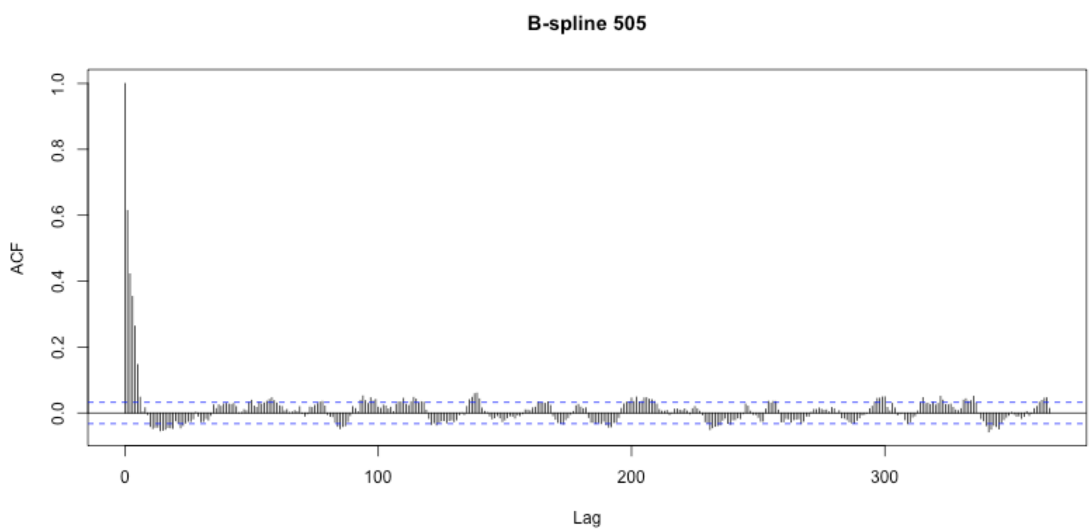

Simple 365 MA
ACF
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
0
100
200
300
Lag


LOESS 0.01
ACF
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
100
200
300
Lag

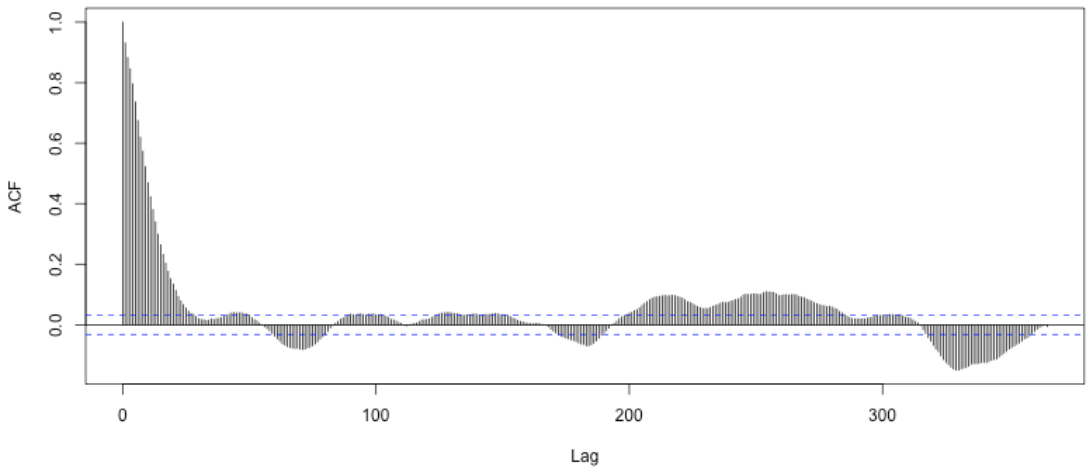

LOESS 0.05
ACF
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
100
200
300
Lag


LOESS 0.2
ACF
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
0
100
200
300
Lag

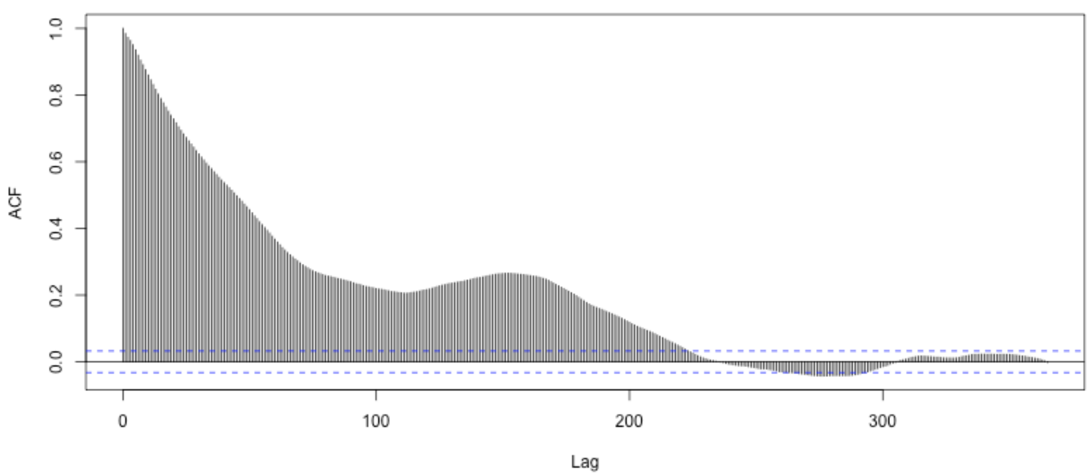

LOESS 0.4
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
Lag

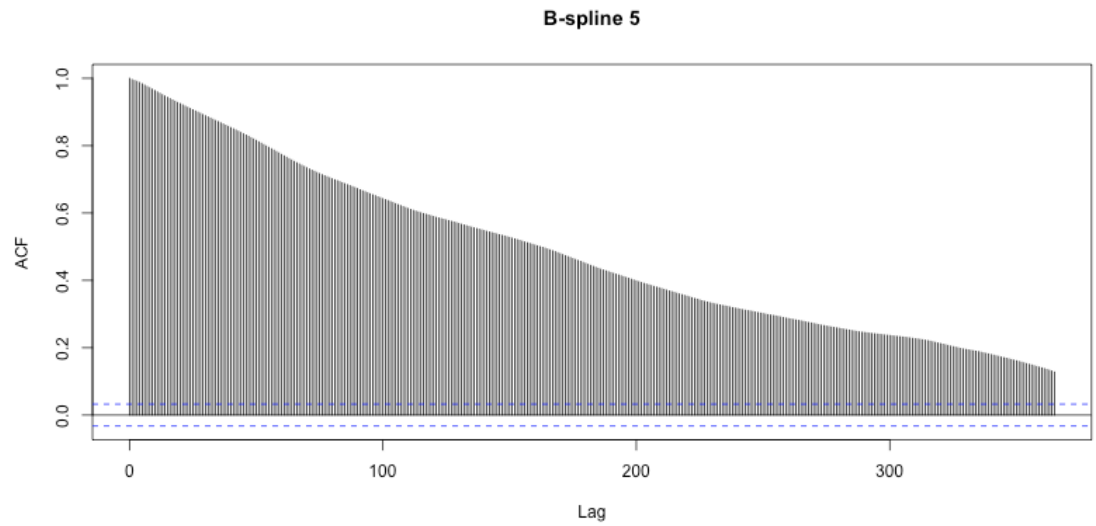

B-spline 505
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
Lag

B-spline 101



B-spline 25





One can notice that for the models extracting trend very locally the ACF of residual component quickly falls and has merely no significant lags. The more global model is, the more slowly falls the ACF, which is logical since such trends smooth the TS very much and leave a lot of information in the residuals.

If there are significant lags, is this good or bad for further analysis?

In case of existing significant lags the residual component should have some pattern / indicate having some memory, and, in fact, is not an irregular component. It may also be decomposed (depending on the ACF looking; if it is slowly falling down, we can extract the trend once again), or may be modeled somehow further. So I would say, this is neither good nor bad, it just tells that further steps have to be taken to extract useful information from the residuals.

Time Series Analysis and Forecasting 2019

Assignment

Problem 3

Solutions are by Yaroslava Lochman

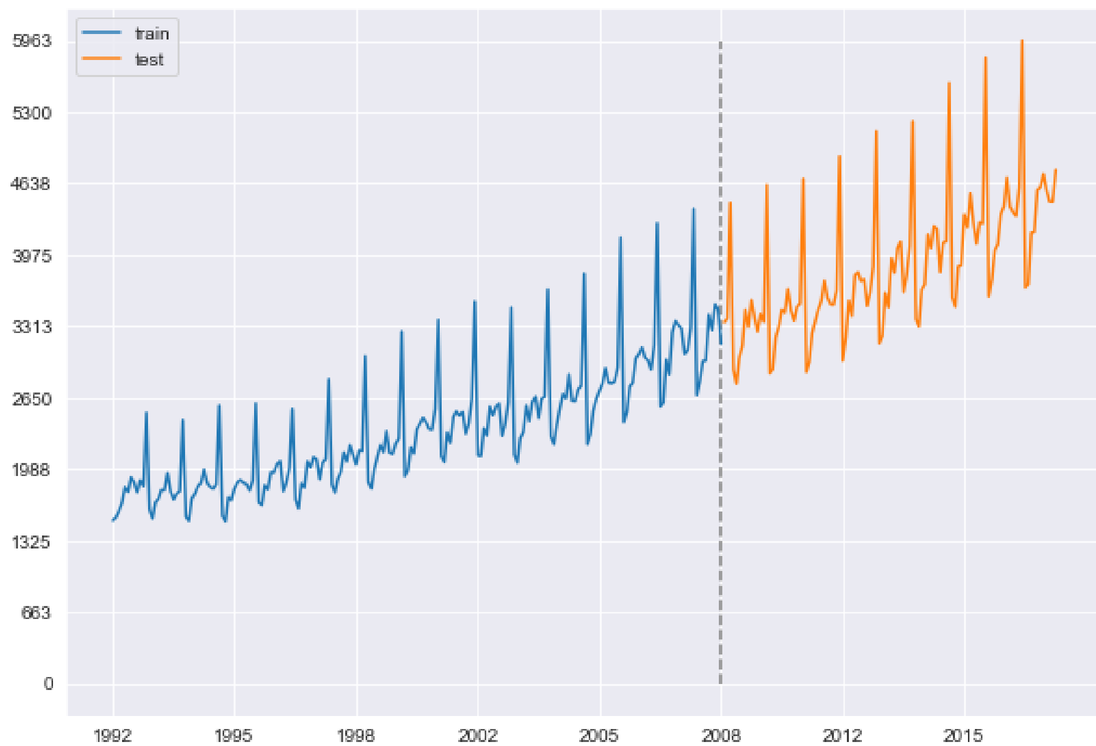
```
In [27]: %matplotlib inline
         %load_ext autoreload
         %autoreload 2
         %load_ext rpy2.ipython

         import numpy as np
         from numpy import hstack as stack
         import pandas as pd

         import warnings
         warnings.filterwarnings("ignore")

         import seaborn as sns
         sns.set_style('darkgrid')
         from matplotlib import pyplot as plt
         import matplotlib as mpl
         mpl.rcParams['figure.figsize'] = (15,10)
         mpl.rcParams['image.cmap'] = 'inferno'


         from statsmodels.tsa.holtwinters import ExponentialSmoothing
         from sklearn.metrics import mean_squared_error as MSE
         from statsmodels.tsa.stattools import acf

         from utils import *
```

Simple forecasting and forecasting with exponential smoothing with `BeerWineUS.csv`. Begin the forecasting starting with the observation 201. Thus the out-of-sample performance of the below methods will be assessed using the forecasts for the periods 201 to 311.

```
In [30]: # Retail sales of beer, wine and liquor in the U.S.
         # Monthly data from January 1992 till November 2017
```

```
BeerWine = pd.read_csv('data/BeerWineUS.csv', header=0, index_col=0,
                       squeeze=True)

mpl.rcParams['figure.figsize'] = (10,7)
N = len(BeerWine)
BeerWine.index = BeerWine.index.astype('datetime64[ns]')
n_train = 201
train, test, n_test = split_ts(BeerWine, n_train)
```

```
311: 201 + 110
```



(a) Compute forecasts using simple EWMA, Holt and Holt-Winters forecasts with the smoothing parameters calibrated from the rst 200 observations.

```
In [33]: forecasts = {}

         # EWMA
         ewma = ExponentialSmoothing(train, trend=None, damped=False, seasonal=None)
         forecast_values = ewma.predict(ewma.fit().params,
                                        start=BeerWine.index[0], end='2017-11-01')
         ewma_result = pd.Series(forecast_values, index=BeerWine.index)
         forecasts['EWMA'] = ewma_result.iloc[n_train:]
```

```python
# Holt
holt = ExponentialSmoothing(train, trend='add', damped=False, seasonal=None)
forecast_values = holt.predict(holt.fit().params,
                               start='1992-01-01', end='2017-11-01')
holt_result = pd.Series(forecast_values, index=BeerWine.index)
forecasts['Holt'] = holt_result.iloc[n_train:]

# Holt-Winters
holtWinters = ExponentialSmoothing(train, trend='add', damped=False,
                                   seasonal='mul', seasonal_periods=12)
forecast_values = holtWinters.predict(holtWinters.fit().params,
                                      start='1992-01-01', end='2017-11-01')
holtWinters_result = pd.Series(forecast_values, index=BeerWine.index)
forecasts['Holt-Winters'] = holtWinters_result.iloc[n_train:]
```

In [34]:
```python
keys = ['EWMA', 'Holt', 'Holt-Winters']

show_series(train, test)

for i, k in enumerate(keys):
    plt.plot(forecasts[k], label='{} forecast'.format(k), color='C{}'.format(i+2))

plt.legend()
plt.show()
```

Visually it is clear that EWMA model is inadequate, and Holt forecast does not consider seasonality in contradiction to Holt-Wonters.

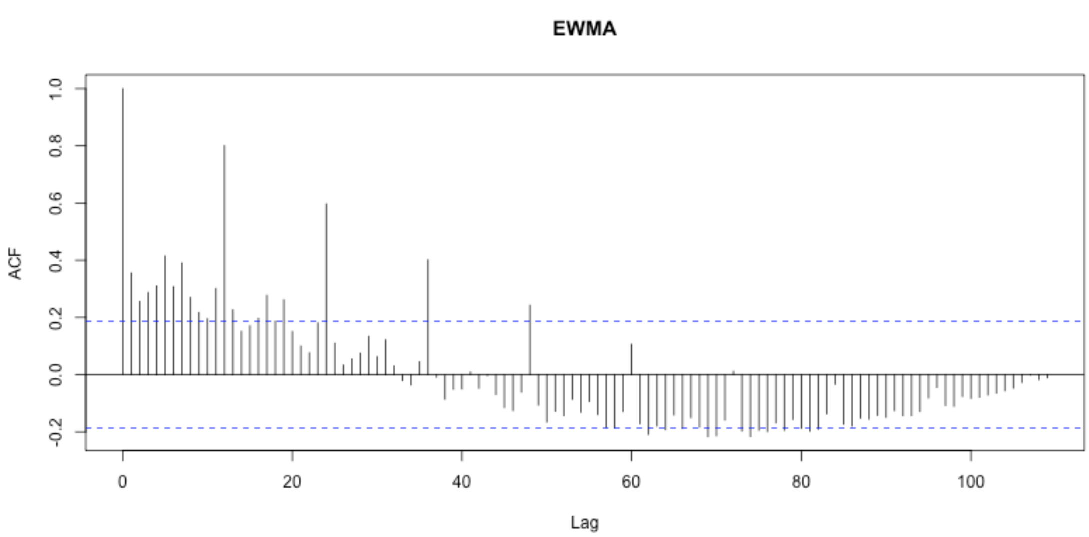
(b) Compute the corresponding MSE losses. Check the ACF of the forecast errors.

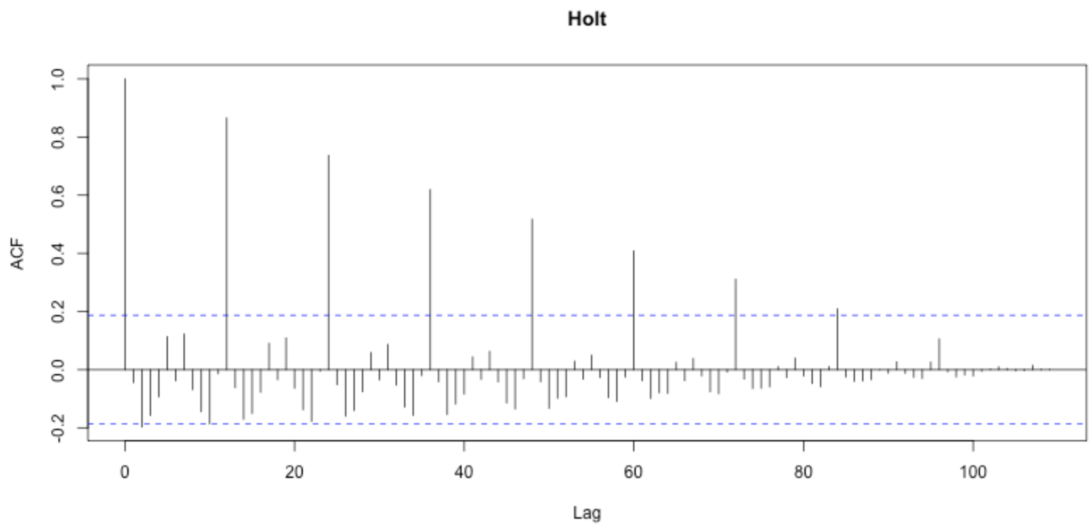
```
In [35]: print('\tEWMA\t\tHolt\t\tHolt-Winters')
         print('MSE:\t{:.1f}\t{:.1f}\t{:.1f}'.format(MSE(test, forecasts['EWMA']),
                                                    MSE(test, forecasts['Holt']),
                                                    MSE(test, forecasts['Holt-Winters'])))
```

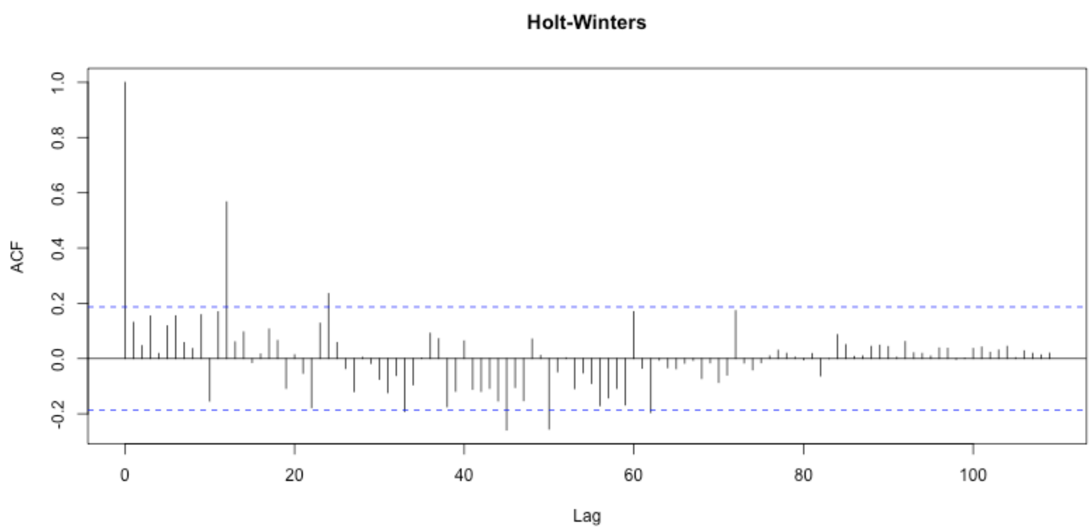
```
        EWMA                Holt                Holt-Winters
MSE:        791343.8        234909.3        38023.6
```

```
In [37]: Model, Err = [], []
         for k in keys:
             Model.append(k)
             Err.append(test-forecasts[k])
```

```
In [38]: %%R -i Err -i Model -w 800 -h 400 -u px
         library('forecast')
         library('tseries')

         names(Err) <- Model
         for (name in names(Err))
         {
         AutoCorrelation <- acf(as.numeric(Err[[name]]), lag.max=365, plot = FALSE)
         plot(AutoCorrelation, main = name)
         }
```







So yes, the same conclusions can be made by looking at the ACF of the the forecast errors.

(c) Compare the performance of the models using the three tests discussed in the lectures.

```
In [39]: from itertools import combinations
```

```python
print('Comparing the performance with 3 tests')
print('Diff. is [model 1] - [model 2]')
alpha = None
for (k1,k2) in combinations(keys, 2):
    f1, f2 = forecasts[k1], forecasts[k2]
    print('================================\nmodel 1: {}\nmodel 2: {}'.format(k1, k2)
    print('\n1. Sign test')
    EPA_sign(test, f1, f2, k1, k2, alpha)
    print('\n2. Wilcoxon sign test')
    EPA_Wilcoxon(test, f1, f2, alpha)
    print('\n3. Diebold-Mariano test')
    EPA_Diebold_Mariano(test, f1, f2, k1, k2, alpha)
```

```
Comparing the performance with 3 tests
Diff. is [model 1] - [model 2]
================================
model 1: EWMA
model 2: Holt

1. Sign test
p-value is 9.33E-07. For all  > 9.33E-07 we can reject H0.
Since the p-value is very small, we see that the Holt model is better than EWMA.

2. Wilcoxon sign test
p-value is 1.95E-08. For all  > 1.95E-08 we can reject H0.
The p-value is very small, so the second model is much better than None.

3. Diebold-Mariano test
p-value is 1.70E-08. For all  > 1.70E-08 we can reject H0.
The p-value is very small, so the Holt model is better than EWMA.
================================
model 1: EWMA
model 2: Holt-Winters

1. Sign test
p-value is 5.55E-16. For all  > 5.55E-16 we can reject H0.
Since the p-value is very small, we see that the Holt-Winters model is better than EWMA.

2. Wilcoxon sign test
p-value is 0.00E+00. For all  > 0.00E+00 we can reject H0.
The p-value is very small, so the second model is much better than None.

3. Diebold-Mariano test
p-value is 4.28E-09. For all  > 4.28E-09 we can reject H0.
The p-value is very small, so the Holt-Winters model is better than EWMA.
================================
model 1: Holt
model 2: Holt-Winters
```

1. Sign test
p-value is 6.59E-03. For all > 6.59E-03 we can reject H0.
Since the p-value is very small, we see that the Holt-Winters model is better than Holt.

2. Wilcoxon sign test
p-value is 7.93E-07. For all > 7.93E-07 we can reject H0.
The p-value is very small, so the second model is much better than None.

3. Diebold-Mariano test
p-value is 6.95E-06. For all > 6.95E-06 we can reject H0.
The p-value is very small, so the Holt-Winters model is better than Holt.

Time Series Analysis and Forecasting 2019

Assignment

Problem 4

Solutions are by Yaroslava Lochman

In [132]:
```
%matplotlib inline
%load_ext autoreload
%autoreload 2
%load_ext rpy2.ipython

import numpy as np
from numpy import hstack as stack
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

import seaborn as sns
sns.set_style('darkgrid')
from matplotlib import pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.figsize'] = (15,10)
mpl.rcParams['image.cmap'] = 'inferno'


from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error as MSE
from statsmodels.tsa.stattools import acf

from utils import *
```

ARMA modelling with `interestrate.csv`. Keep the last year for forecasting.

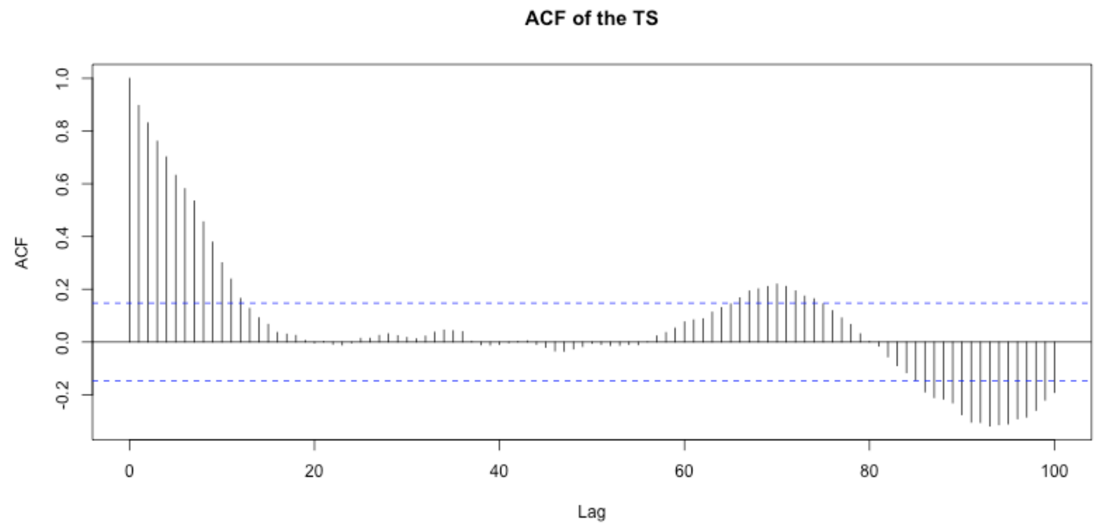
In [178]:
```
# Real (ination corrected) interest rate for bank deposits with
# investment durations between one and two years
interestrate = pd.read_csv('data/interestrate.csv', sep=';',
                           index_col=0, header=None).iloc[:,0]
```

```
interestrate = interestrate.map(lambda s: float(s.replace(',', '.')))
interestrate.index = interestrate.index.astype('datetime64[ns]')

mpl.rcParams['figure.figsize'] = (15,7)

N = len(interestrate)
n_train = N-1
train, test, n_test = split_ts(interestrate, n_train)
```

```
178: 177 + 1
```

(a) Check the ACF and decide about the strength of the memory in the time series using Box-Ljung/Pierce tests.

```
In [198]: %%R -i train -i test -w 800 -h 400 -u px
          train <- as.numeric(train)
          m <- 100
          AutoCorrelation <- acf(train, lag.max=m, plot = FALSE)
          plot(AutoCorrelation, main = 'ACF of the TS')
```




```
In [180]: %%R
          m <- 20
          print(Box.test(train, lag=m, type="Ljung-Box"))
          print(Box.test(train, lag=m, type="Box-Pierce"))
          alpha <- 0.05
          cat('Chi-squared with', m,'df at alpha', alpha, ':', pchisq(1-alpha, m))
```

```
Box-Ljung test

data:  train
X-squared = 760.51, df = 20, p-value < 2.2e-16


Box-Pierce test

data:  train
X-squared = 734.22, df = 20, p-value < 2.2e-16

Chi-squared with 20 df at alpha 0.05 : 1.047083e-10
```

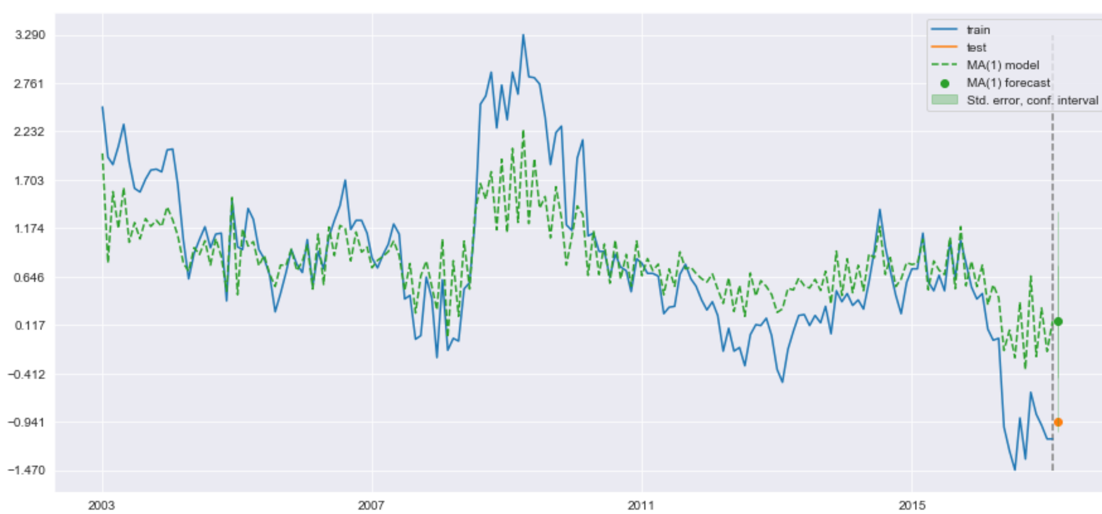
So the H0 (the process is not autocorrelated) can be rejected. There exists some memory in the time series.

(b) Try `MA(1)`, `AR(1)` and `ARMA(1,1)` processes and check the t (ACF of residuals, AIC, etc.)

```
In [380]: from statsmodels.tsa.arima_model import ARIMA
          Res, Model, AIC, BIC = [], [], [], []
```

```
model_names = ['MA(1)', 'AR(1)', 'ARMA(1,1)']
orders = [(0,0,1), (1,0,0), (1,0,1)]

for i, (model_name, order) in enumerate(zip(model_names, orders)):
    show_series(train, test)
    model_color = 'C{}'.format(i+2)
    print('\n{}:'.format(model_name))
    model = ARIMA(np.array(train), order=order).fit(method='css')
    plt.plot(train.index, model.predict(1,len(train)), '--',
             color=model_color, label='{} model'.format(model_name))
    forecast, stderr, confint = model.forecast(len(test), alpha=0.05)
    plt.scatter(test.index, forecast, color=model_color,
                label='{} forecast'.format(model_name))
    plt.fill_between(test.index, confint[:,0], confint[:,1],
                     color=model_color, alpha=0.3,
                     label='Std. error, conf. interval')
    plt.fill_between(test.index, forecast-stderr, forecast+stderr,
                     color=model_color, alpha=0.3)
    plt.legend()
    plt.show()
    print(model.summary())
    Res.append(train - model.predict(1,len(train)))
    Model.append(model_name)
    AIC.append(model.aic)
    BIC.append(model.bic)
```

```
MA(1):
```



```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  177
Model:                     ARMA(0, 1)   Log Likelihood                -165.311
Method:                           css   S.D. of innovations              0.616
Date:                Mon, 11 Feb 2019   AIC                            336.622
Time:                        11:35:55   BIC                            346.150
Sample:                             0   HQIC                           340.486

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.8242      0.078     10.539      0.000       0.671       0.977
ma.L1.y        0.6969      0.041     16.810      0.000       0.616       0.778
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1           -1.4348           +0.0000j            1.4348            0.5000
-----------------------------------------------------------------------------
```

```
AR(1):
```


```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  177
Model:                     ARMA(1, 0)   Log Likelihood                 -66.736
Method:                           css   S.D. of innovations              0.354
Date:                Mon, 11 Feb 2019   AIC                            139.472
```

```
Time:                        11:35:55   BIC                            148.983
Sample:                             1   HQIC                           143.330

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.5614      0.353      1.592      0.113      -0.130       1.253
ar.L1.y        0.9212      0.030     30.746      0.000       0.863       0.980
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0855           +0.0000j            1.0855            0.0000
-----------------------------------------------------------------------------
```

ARMA(1,1):


```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  177
Model:                     ARMA(1, 1)   Log Likelihood                 -64.607
Method:                           css   S.D. of innovations              0.349
Date:                Mon, 11 Feb 2019   AIC                            137.214
Time:                        11:35:55   BIC                            149.896
Sample:                             1   HQIC                           142.358

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

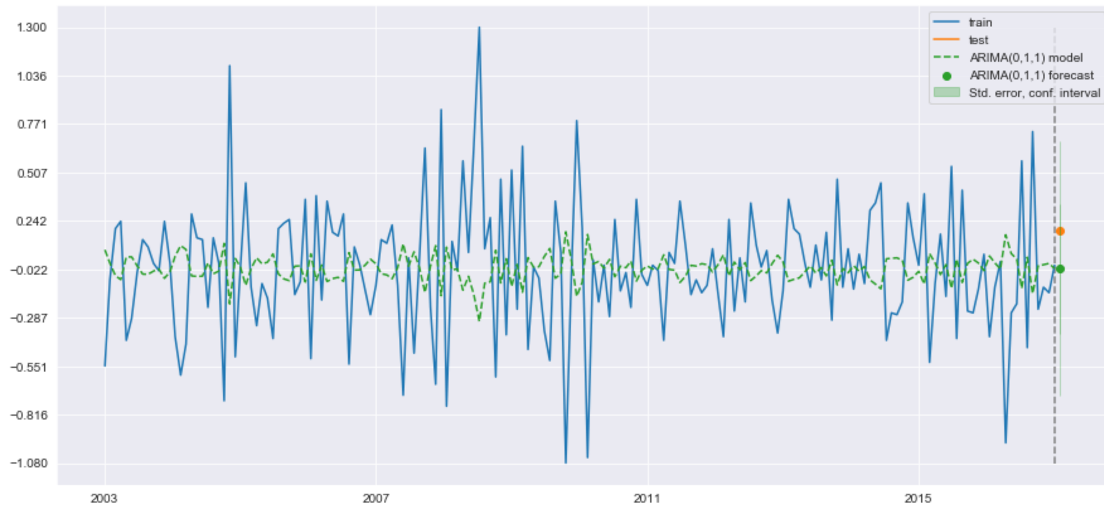
```
const          0.4429      0.460      0.964      0.337     -0.458       1.344
ar.L1.y        0.9470      0.027     34.718      0.000      0.894       1.000
ma.L1.y       -0.1625      0.077     -2.098      0.037     -0.314      -0.011
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.0560           +0.0000j            1.0560            0.0000
MA.1            6.1556           +0.0000j            6.1556            0.0000
-----------------------------------------------------------------------------
```

In the implementation of ARIMA the model is fitted by the maximization of conditional sum of squares likelihood.

(c) Try dierencing and subsequent application of MA(1), AR(1) and ARMA(1,1). Check again the processes and check the t (signicance, ACF, AIC, etc.) Decide which model is the best one.

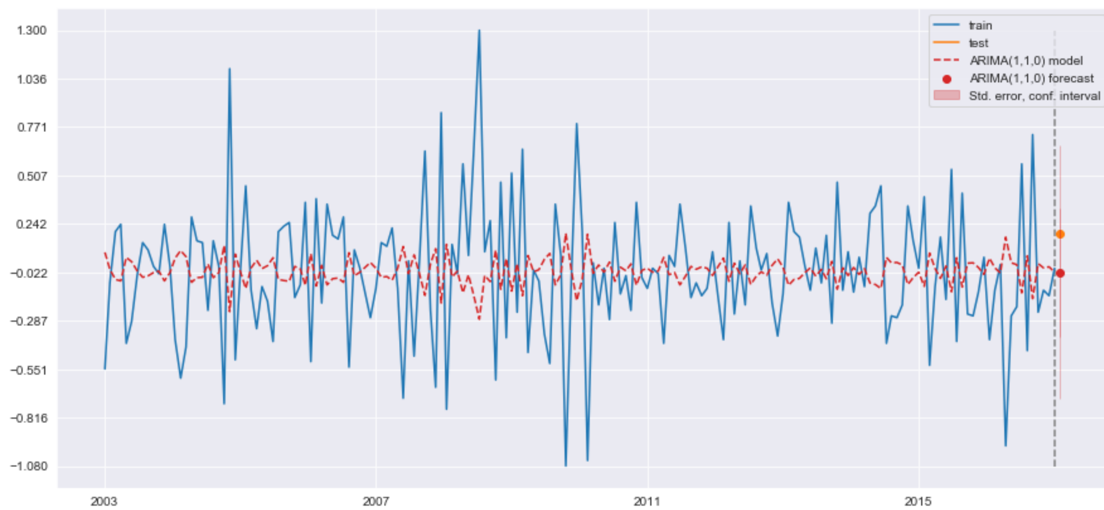
```
In [381]: train_d = (train - train.shift(1))[1:]
          test_d = test - train[-1]

          model_names = ['ARIMA(0,1,1)', 'ARIMA(1,1,0)', 'ARIMA(1,1,1)']
          orders = [(0,0,1), (1,0,0), (1,0,1)]

          for i, (model_name, order) in enumerate(zip(model_names, orders)):
              show_series(train_d, test_d)
              model_color = 'C{}'.format(i+2)
              print('\n{}:'.format(model_name))
              model = ARIMA(np.array(train_d), order=order).fit(method='css')
              plt.plot(train_d.index, model.predict(1,len(train_d)), '--',
                       color=model_color, label='{} model'.format(model_name))
              forecast, stderr, confint = model.forecast(len(test_d), alpha=0.05)
              plt.scatter(test_d.index, forecast, color=model_color,
                          label='{} forecast'.format(model_name))
              plt.fill_between(test_d.index, confint[:,0], confint[:,1],
                               color=model_color, alpha=0.3,
                               label='Std. error, conf. interval')
              plt.fill_between(test_d.index, forecast-stderr, forecast+stderr,
                               color=model_color, alpha=0.3)
              plt.legend()
              plt.show()
              print(model.summary())
              Res.append(train_d - model.predict(1,len(train_d)))
              Model.append(model_name)
              AIC.append(model.aic)
              BIC.append(model.bic)
```

ARIMA(0,1,1):




```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  176
Model:                     ARMA(0, 1)   Log Likelihood                 -66.682
Method:                           css   S.D. of innovations              0.353
Date:                Mon, 11 Feb 2019   AIC                            139.363
Time:                        11:35:58   BIC                            148.875
Sample:                             0   HQIC                           143.221

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0201      0.021     -0.939      0.349      -0.062       0.022
ma.L1.y       -0.1966      0.074     -2.670      0.008      -0.341      -0.052
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1            5.0857           +0.0000j            5.0857            0.0000
-----------------------------------------------------------------------------
```

ARIMA(1,1,0):




```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  176
Model:                     ARMA(1, 0)   Log Likelihood                 -65.614
Method:                           css   S.D. of innovations              0.352
Date:                Mon, 11 Feb 2019   AIC                            137.229
Time:                        11:35:58   BIC                            146.723
Sample:                             1   HQIC                           141.080

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0181      0.022     -0.815      0.416      -0.062       0.025
ar.L1.y       -0.1975      0.074     -2.682      0.008      -0.342      -0.053
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -5.0633           +0.0000j            5.0633            0.5000
-----------------------------------------------------------------------------
```

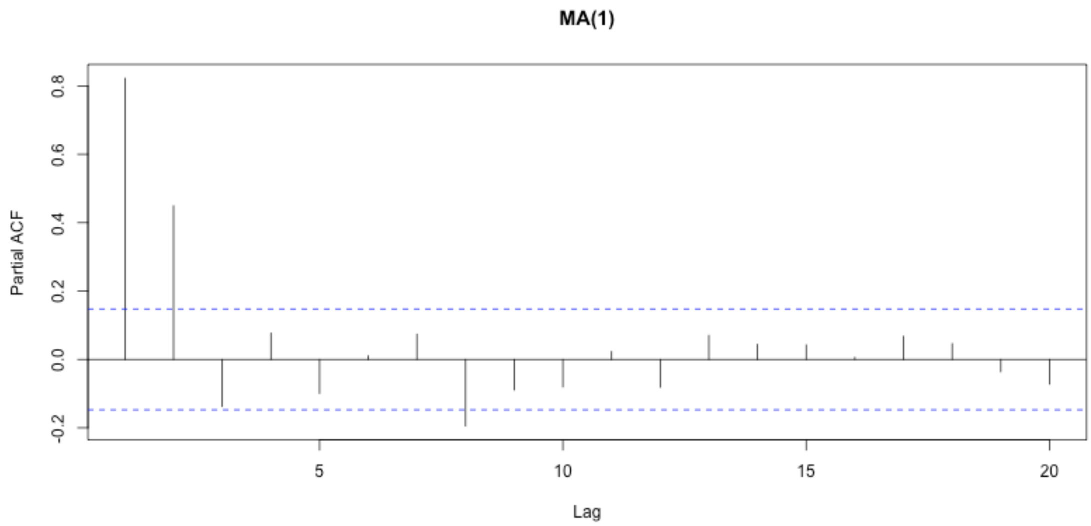
ARIMA(1,1,1):

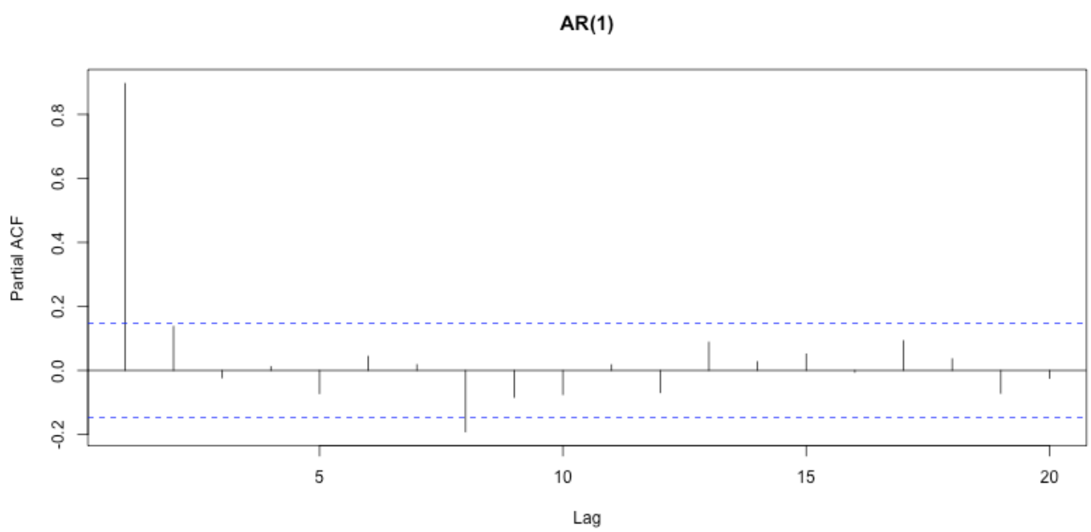

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  176
Model:                     ARMA(1, 1)   Log Likelihood                 -65.578
Method:                           css   S.D. of innovations              0.352
Date:                Mon, 11 Feb 2019   AIC                            139.157
Time:                        11:35:59   BIC                            151.816
Sample:                             1   HQIC                           144.292

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0178      0.022     -0.819      0.414      -0.061       0.025
ar.L1.y       -0.1002      0.368     -0.272      0.786      -0.821       0.621
ma.L1.y       -0.1012      0.369     -0.274      0.784      -0.825       0.623
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            -9.9847          +0.0000j            9.9847            0.5000
MA.1             9.8838          +0.0000j            9.8838            0.0000
-----------------------------------------------------------------------------
```

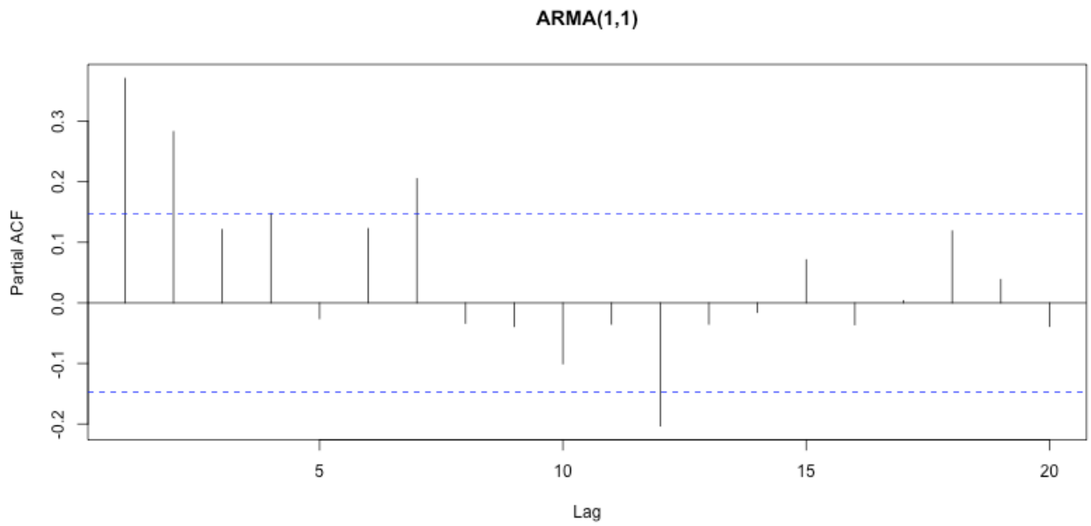
In [382]:
```
%%R -i Res -i Model -w 800 -h 400 -u px

cat('ACF of the residuals:')
names(Res) <- Model
for (name in names(Res))
{
```

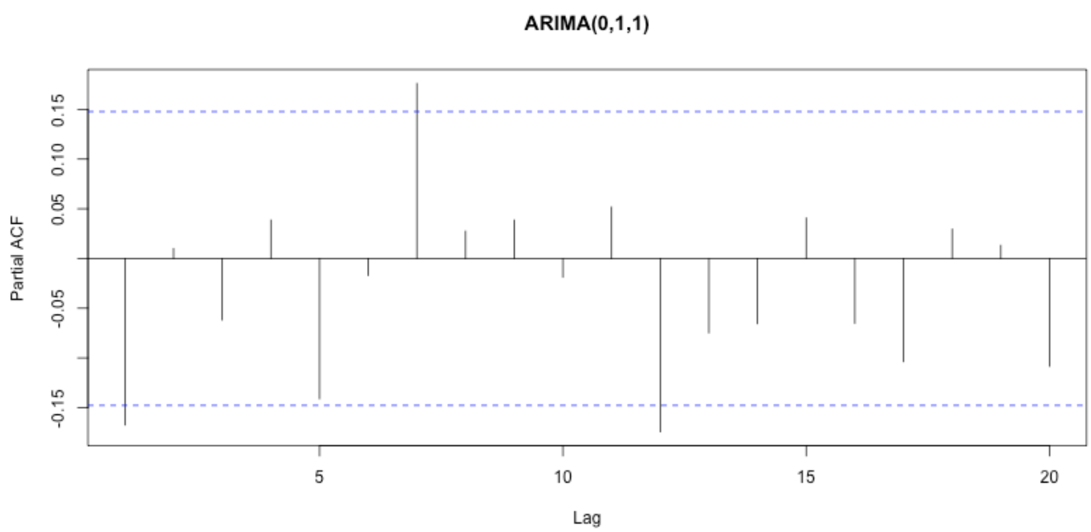
```
AutoCorrelation <- pacf(as.numeric(Res[[name]]), lag.max=20, plot = FALSE)
plot(AutoCorrelation, main = name)
}
```

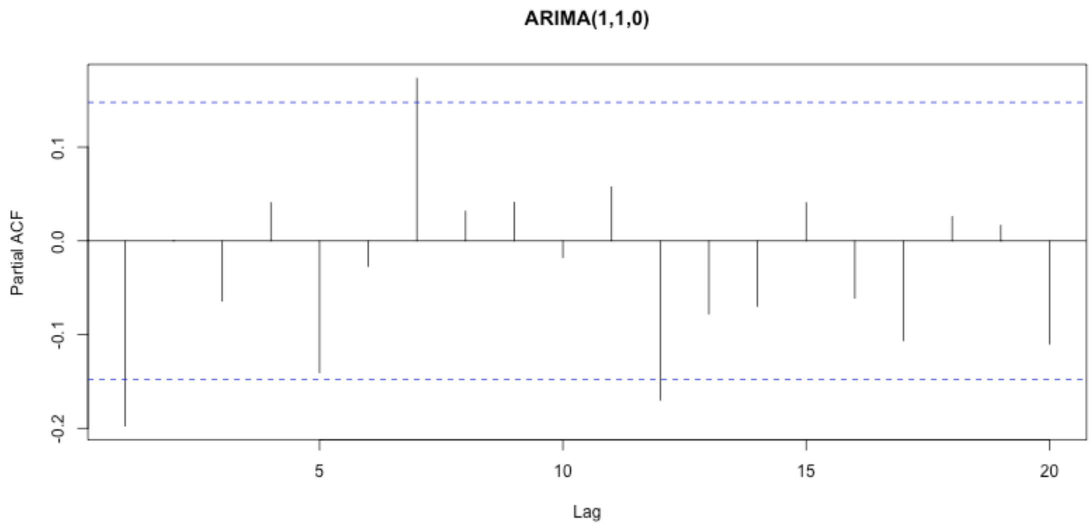
ACF of the residuals:

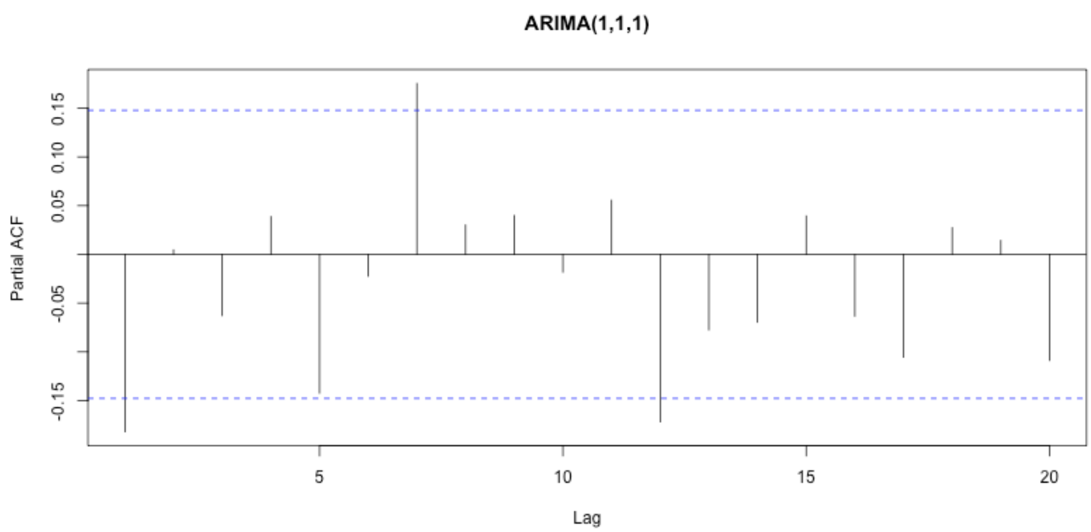





ARMA(1,1)
Partial ACF
Lag
0.3
0.2
0.1
0.0
-0.1
-0.2
5
10
15
20


ARIMA(0,1,1)
Partial ACF
Lag
0.15
0.10
0.05
-0.05
-0.15
5
10
15
20






```
In [383]: print(' Model\t\t AIC\tBIC')
          aic_min = np.argmin(AIC)
          bic_min = np.argmin(BIC)
          for i, (m, aic, bic) in enumerate(zip(Model, AIC, BIC)):
              print('{: <15} {:.2f}{}\t{:.2f}{}'.format(m, aic,
                                                        '*' if i == aic_min else ' ',
                                                        bic,
                                                        '*' if i == bic_min else ' '))
```

```
 Model                AIC        BIC
MA(1)           336.62         346.15
AR(1)           139.47         148.98
ARMA(1,1)       137.21*        149.90
ARIMA(0,1,1)    139.36         148.87
ARIMA(1,1,0)    137.23         146.72*
ARIMA(1,1,1)    139.16         151.82
```

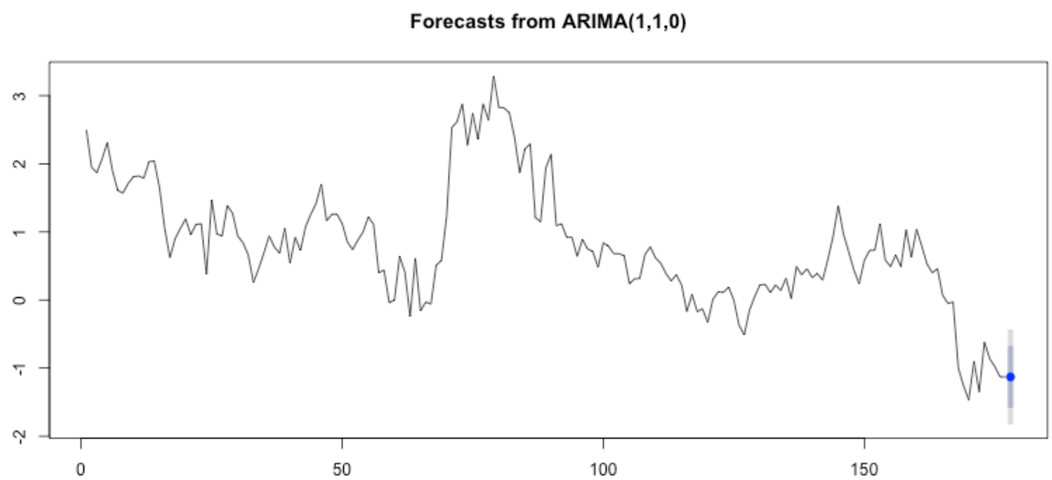
In terms of information criteria ARIMA(1,0,1) and ARIMA(1,1,0) are the best. By looking at the ACF one may notice that without differencing the autocorrelation at lag 1 remains high, but after differencing (autoregressive integrated moving average process) it becomes very low. Therefore ARIMA(1,1,0) is suggested as the best model.

(d) Try autoarima (in R) and compare the final model with the one you found in the previous step.

In [345]:
```
%%R -w 800 -h 400 -u px
library('forecast')

ARIMA <- auto.arima(train, trace=T)
plot(forecast(ARIMA, h=1))
summary(ARIMA)
```

```
 Fitting models using approximations to speed things up...

 ARIMA(2,1,2) with drift         : 146.49
 ARIMA(0,1,0) with drift         : 146.3579
 ARIMA(1,1,0) with drift         : 140.1561
 ARIMA(0,1,1) with drift         : 141.5407
 ARIMA(0,1,0)                    : 144.8869
 ARIMA(2,1,0) with drift         : 143.0287
 ARIMA(1,1,1) with drift         : 142.178
 ARIMA(2,1,1) with drift         : 144.7918
 ARIMA(1,1,0)                    : 138.751
 ARIMA(2,1,0)                    : 141.5415
 ARIMA(1,1,1)                    : 140.7557
 ARIMA(0,1,1)                    : 140.3444
 ARIMA(2,1,1)                    : 143.2891

 Now re-fitting the best model(s) without approximations...

 ARIMA(1,1,0)                    : 138.0862

 Best model: ARIMA(1,1,0)

Series: train
```

```
ARIMA(1,1,0)

Coefficients:
          ar1
      -0.1955
s.e.   0.0742

sigma^2 estimated as 0.1261:  log likelihood=-67.01
AIC=138.02   AICc=138.09   BIC=144.36

Training set error measures:
                      ME      RMSE       MAE  MPE MAPE      MASE         ACF1
Training set -0.02444446 0.3530476 0.2689919 -Inf  Inf 0.9906375 -0.002586706
```



The output of `autoarima` is also ARIMA(1,1,0) — the same as chosen previously.

(e) Compute the forecasts and forecast intervals using the nal model.

```
In [400]: from statsmodels.tsa.arima_model import ARIMA


          model_name = model_names[-2]
          order = orders[-2]

          show_series(train, test)
          model_color = 'C2'
          model = ARIMA(np.array(train), order=order).fit(method='css')
```

```
    print('\n{}'.format(model_name))
    print('alpha_1 = {:.4f}'.format(model.params[0]))
    print('Forecast: {:.4f}\n{}%-Confidence interval: ({:.4f}, {:.4f})'\
          .format(forecast[0], 95, *confint.squeeze()))
    forecast, stderr, confint = model.forecast(len(test), alpha=0.05)
    plt.scatter(test.index, forecast, color=model_color,
                label='{} forecast'.format(model_name))
    plt.fill_between(interestrate.index[-3:] + pd.offsets.MonthOffset(1),
                     confint[:,0], confint[:,1],
                     color=model_color, alpha=0.3,
                     label='Std. error, conf. interval')
    plt.fill_between(interestrate.index[-3:] + pd.offsets.MonthOffset(1),
                     forecast-stderr, forecast+stderr,
                     color=model_color, alpha=0.3)
    plt.legend()
    plt.show()
```

```
ARIMA(1,1,0)
alpha_1 = 0.5614
Forecast: -0.9968
95%-Confidence interval: (-1.6897, -0.3038)
```

(f) Explain why multi-step-ahead forecasts have wider forecast intervals than onestep-ahead-forecasts.

The variance of the multi-step-ahead forecast error depends on all the previous forecast errors variances, so the deeper we go with the forecast, the more intermediate forecast errors we get, and so the wider become the forecast intervals.

(g) Imagine an ACF with only the rst two correlations being signicant. Which process is suitable to model this and why?

For the moving average process the ACF disappears for lags greater than the order of the process. So for MA(2): $\gamma_1 = \sigma_u^2(\beta_1 + \beta_1\beta_2),\ \gamma_2 = \sigma_u^2\beta_2,$ the next autocovariances are zero. So in the modeling of the TS with described ACF the MA(2) should be included (if the first two correlations mean 1 & 2, otherwise if it means 0 & 1 then MA(1) may suit). The first several significant correlations may indicate a non-stationary process so one can also test the stationarity and may use integrated process.

(h) Imagine an ACF which consists only of positive values and quickly decays towards zero. Which process is suitable to model this and why?

The autoregressive process AR(1) $(1 - \alpha L)Y_t = u_t$ has $\rho_h = \alpha^{|h|}$ meaning that for positive small values of α the ACF would quickly decay towards zero. So AR(1) process is suitable for the TS with described ACF. If the falling doesn't look very consistently, AR(2) may also be considered. The parameters α_i of the process are likely to be small indicating quick memory lost.

(i) Consider an AR(1) process with parameter α_1 . Assume we have a shock to a time series (a large error term, unexpected event) at the time point t = 10. Which impact do you expect this shock to have on the observation at time point t = 15? Provide the formula and give formal motivation.

$Y_{10} = \alpha_1 Y_9 + u_{10} = ... = f_1(\alpha_1, u_9...u_1) + u_{10}$. We are interested in the error u_{10} since it is large, it might impact the next values. So $Y_{15} = \alpha_1 Y_{14} + u_{15} = ... = \alpha_1^5 Y_{10} + \alpha_1^4 u_{11} + ... + \alpha_1 u_{14} + u_{15} = \alpha_1^5 u_{10} + f_2(\alpha_1, u_{15}, ...u_{11}, u_9...u_1)$. So the impact on the observation in time point $t = 15$ would be the added $\alpha^5 u_{10}$.