01_stationarity_and_arima

September 29, 2021

1 Time Series Analysis and Univariate ARIMA Models

[1]:
```
import os
import sys
import warnings
from datetime import date
import pandas as pd
import pandas_datareader.data as web
import numpy as np
from numpy.linalg import LinAlgError

import statsmodels.tsa.api as tsa
from statsmodels.graphics.tsaplots import plot_acf, acf, plot_pacf, pacf
from statsmodels.tsa.stattools import acf, q_stat, adfuller
import statsmodels.api as sm
from scipy.stats import probplot, moment
from sklearn.metrics import mean_squared_error

import quandl
import matplotlib.pyplot as plt
import matplotlib as mpl
```

[2]:
```
%matplotlib inline
warnings.filterwarnings('ignore')
plt.style.use('ggplot')
```

[3]:
```
def plot_correlogram(x, lags=None, title=None):
    lags = min(10, int(len(x)/5)) if lags is None else lags
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14, 8))
    x.plot(ax=axes[0][0])
    q_p = np.max(q_stat(acf(x, nlags=lags), len(x))[1])
    stats = f'Q-Stat: {np.max(q_p):>8.2f}\nADF: {adfuller(x)[1]:>11.2f}'
    axes[0][0].text(x=.02, y=.85, s=stats, transform=axes[0][0].transAxes)
    probplot(x, plot=axes[0][1])
    mean, var, skew, kurtosis = moment(x, moment=[1, 2, 3, 4])
    s = f'Mean: {mean:>12.2f}\nSD: {np.sqrt(var):>16.2f}\nSkew: {skew:12.
↪2f}\nKurtosis:{kurtosis:9.2f}'
```

```
    axes[0][1].text(x=.02, y=.75, s=s, transform=axes[0][1].transAxes)
    plot_acf(x=x, lags=lags, zero=False, ax=axes[1][0])
    plot_pacf(x, lags=lags, zero=False, ax=axes[1][1])
    axes[1][0].set_xlabel('Lag')
    axes[1][1].set_xlabel('Lag')
    fig.suptitle(title, fontsize=20)
    fig.tight_layout()
    fig.subplots_adjust(top=.9)
```

1.1 Download Series

Load monthly industrial production and daily NASDAQ stock market index:

[4]:
```
industrial_production = web.DataReader('IPGMFN', 'fred', '1988', '2017-12').
 ↪squeeze().dropna()
nasdaq = web.DataReader('NASDAQCOM', 'fred', '1990', '2017-12-31').squeeze().
 ↪dropna()
```

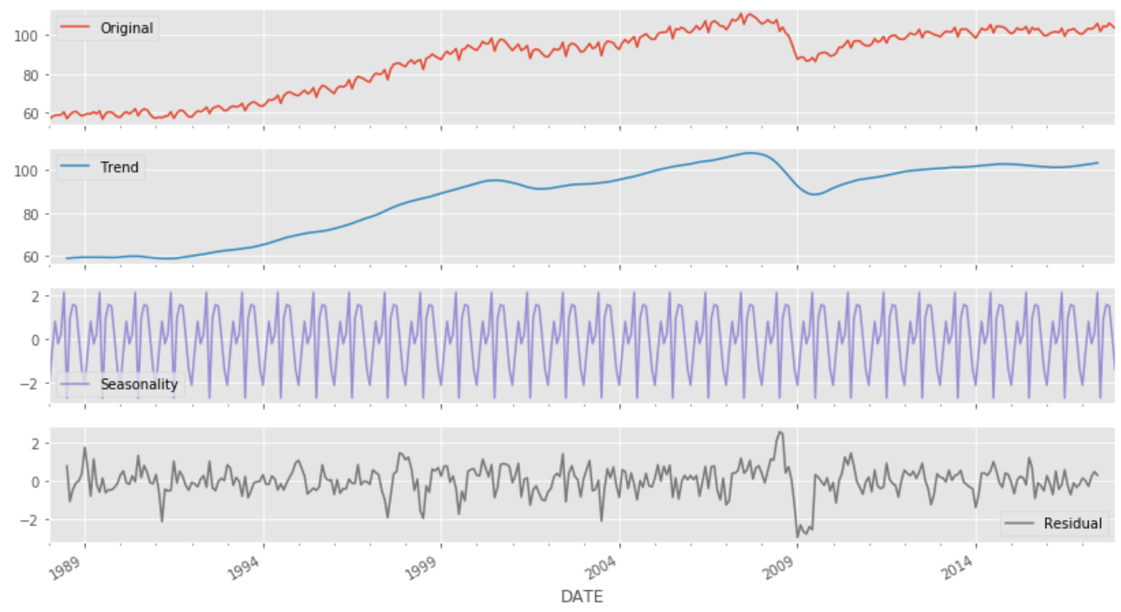
1.2 Additive Decomposition

Time series data typically contains a mix of various patterns that can be decomposed into several components, each representing an underlying pattern category. In particular, time series often consist of the systematic components trend, seasonality and cycles, and unsystematic noise. These components can be combined in an additive, linear model, in particular when fluctuations do not depend on the level of the series, or in a non-linear, multiplicative model.

These components can be split up automatically. statsmodels includes a simple method to split the time series into a trend, seasonal, and residual component using moving averages. We can apply it to monthly data on industrial manufacturing production with both a strong trend and seasonality component, as follows:

[5]:
```
components = tsa.seasonal_decompose(industrial_production, model='additive')
```

[6]:
```
ts = (industrial_production.to_frame('Original')
      .assign(Trend=components.trend)
      .assign(Seasonality=components.seasonal)
      .assign(Residual=components.resid))
ts.plot(subplots=True, figsize=(14, 8));
```



1.3 Time Series Stationarity

The statistical properties, such as the mean, variance, or autocorrelation, of a stationary time series are independent of the period, that is, they don't change over time. Hence, stationarity implies that a time series does not have a trend or seasonal effects and that descriptive statistics, such as the mean or the standard deviation, when computed for different rolling windows, are constant or do not change much over time. It reverts to its mean, and the deviations have constant amplitude, while short-term movements always look the same in the statistical sense.

More formally, strict stationarity requires the joint distribution of any subset of time series observations to be independent of time with respect to all moments. So, in addition to the mean and variance, higher moments such as skew and kurtosis, also need to be constant, irrespective of the lag between different observations. In most applications, we limit stationarity to first and second moments so that the time series is covariance stationary with constant mean, variance, and autocorrelation.

Note that we specifically allow for dependence between observations at different lags, just like we want the input data for linear regression to be correlated with the outcome. Stationarity implies that these relationships are stable, which facilitates prediction as the model can focus on learning systematic patterns that take place within stable statistical properties. It is important because classical statistical models assume that the time series input data is stationary.

To satisfy the stationarity assumption of linear time series models, we need to transform the original time series, often in several steps. Common transformations include the application of the (natural) logarithm to convert an exponential growth pattern into a linear trend and stabilize the variance. Deflation implies dividing a time series by another series that causes trending behavior, for example dividing a nominal series by a price index to convert it into a real measure.

1.3.1 Log Transformation

Double check fo zero values

```
[7]: (nasdaq == 0).any(), (industrial_production==0).any()
```

```
[7]: (False, False)
```

```
[8]: nasdaq_log = np.log(nasdaq)
     industrial_production_log = np.log(industrial_production)
```

1.3.2 Differencing

In many cases, de-trending is not sufficient to make the series stationary. Instead, we need to transform the original data into a series of period-to-period and/or season-to-season differences. In other words, we use the result of subtracting neighboring data points or values at seasonal lags from each other. Note that when such differencing is applied to a log-transformed series, the results represent instantaneous growth rates or returns in a financial context.

If a univariate series becomes stationary after differencing d times, it is said to be integrated of the order of d, or simply integrated if d=1. This behavior is due to so-called unit roots.

Differencing of log series produces instantaneous returns.

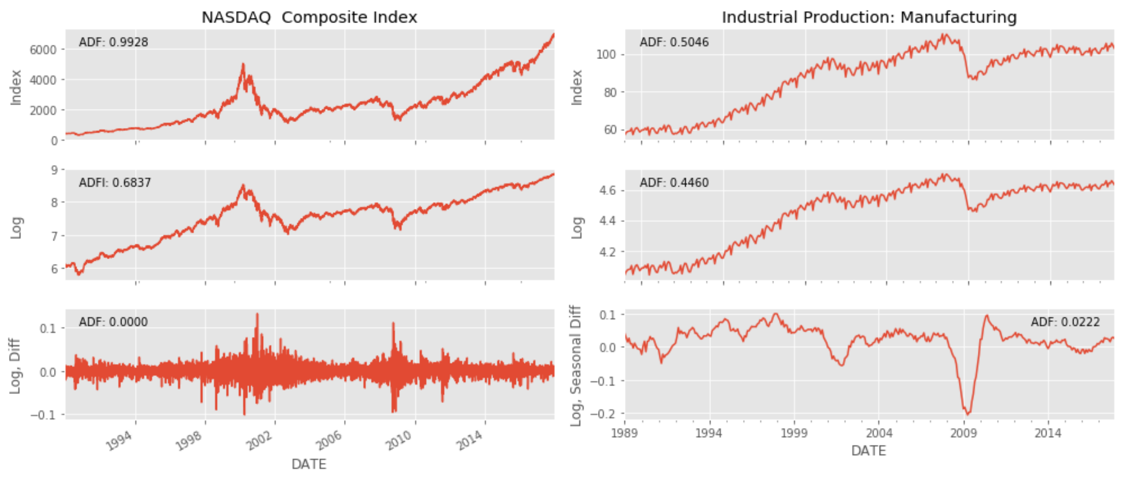
```
[9]: nasdaq_log_diff = nasdaq_log.diff().dropna()

     # seasonal differencing => yoy instantanteous returns
     industrial_production_log_diff = industrial_production_log.diff(12).dropna()
```

1.3.3 Plot Series

The following chart shows time series for the NASDAQ stock index and industrial production for the 30 years through 2017 in original form, as well as the transformed versions after applying the logarithm and subsequently applying first and seasonal differences (at lag 12), respectively. The charts also display the ADF p-value, which allows us to reject the hypothesis of unit-root non-stationarity after all transformations in both cases:

```
[10]: fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(14,6))

      nasdaq.plot(ax=axes[0][0], title='NASDAQ  Composite Index')
      axes[0][0].text(x=.03, y=.85, s=f'ADF: {tsa.adfuller(nasdaq.dropna())[1]:.4f}',␣
       ↪transform=axes[0][0].transAxes)
      axes[0][0].set_ylabel('Index')

      nasdaq_log.plot(ax=axes[1][0], sharex=axes[0][0])
      axes[1][0].text(x=.03, y=.85, s=f'ADFl: {tsa.adfuller(nasdaq_log.dropna())[1]:.
       ↪4f}', transform=axes[1][0].transAxes)
      axes[1][0].set_ylabel('Log')
```

```
nasdaq_log_diff.plot(ax=axes[2][0], sharex=axes[0][0])
axes[2][0].text(x=.03, y=.85, s=f'ADF: {tsa.adfuller(nasdaq_log_diff.
 ↪dropna())[1]:.4f}', transform=axes[2][0].transAxes)
axes[2][0].set_ylabel('Log, Diff')

industrial_production.plot(ax=axes[0][1], title='Industrial Production:␣
 ↪Manufacturing')
axes[0][1].text(x=.03, y=.85, s=f'ADF: {tsa.adfuller(industrial_production)[1]:.
 ↪4f}', transform=axes[0][1].transAxes)
axes[0][1].set_ylabel('Index')

industrial_production_log.plot(ax=axes[1][1], sharex=axes[0][1])
axes[1][1].text(x=.03, y=.85, s=f'ADF: {tsa.adfuller(industrial_production_log.
 ↪dropna())[1]:.4f}', transform=axes[1][1].transAxes)
axes[1][1].set_ylabel('Log')

industrial_production_log_diff.plot(ax=axes[2][1], sharex=axes[0][1])
axes[2][1].text(x=.83, y=.85, s=f'ADF: {tsa.
 ↪adfuller(industrial_production_log_diff.dropna())[1]:.4f}',␣
 ↪transform=axes[2][1].transAxes)
axes[2][1].set_ylabel('Log, Seasonal Diff')
fig.tight_layout()
fig.align_ylabels(axes);
```



1.4 Correlogram

Autocorrelation (also called serial correlation) adapts the concept of correlation to the time series context: just as the correlation coefficient measures the strength of a linear relationship between two variables, the autocorrelation coefficient, ρ_k, measures the extent of a linear relationship between time series values separated by a given lag, k.

Hence, we can calculate one autocorrelation coefficient for each of the T-1 lags in a time series; T is the length of the series. The autocorrelation function (ACF) computes the correlation coefficients as a function of the lag. The autocorrelation for a lag larger than 1 (that is, between observations more than one time step apart) reflects both the direct correlation between these observations and the indirect influence of the intervening data points. The partial autocorrelation removes this influence and only measures the linear dependence between data points at the given lag distance. The partial autocorrelation function (PACF) provides all the correlations that result once the effects of a correlation at shorter lags have been removed.

There are algorithms that estimate the partial autocorrelation from the sample autocorrelation based on the exact theoretical relationship between the PACF and the ACF.

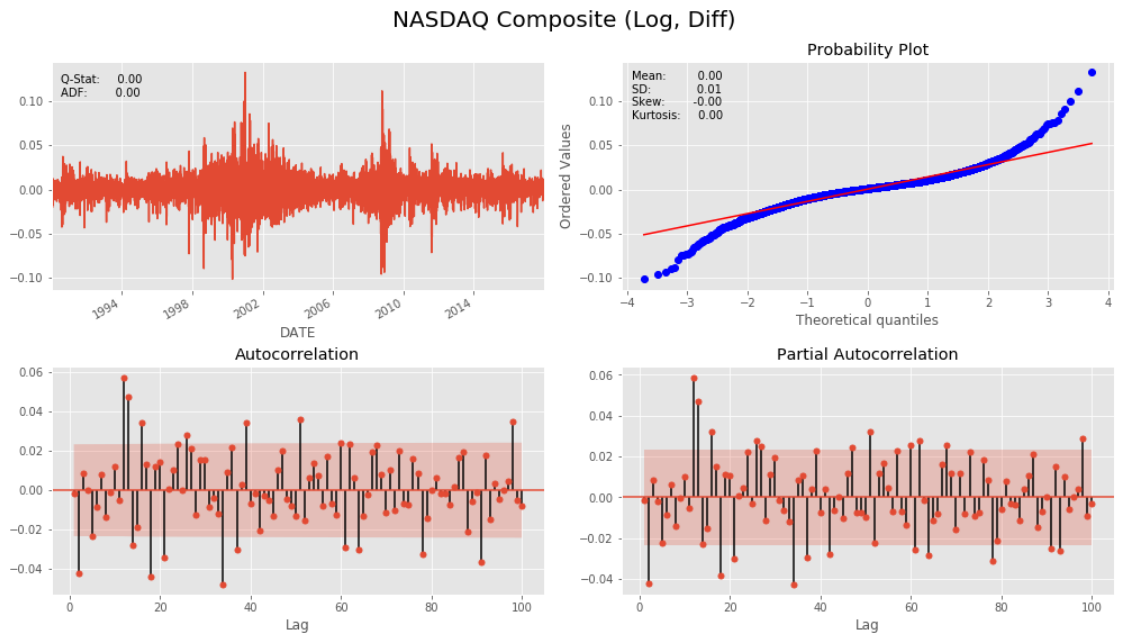
A correlogram is simply a plot of the ACF or PACF for sequential lags, k=0,1,...,n. It allows us to inspect the correlation structure across lags at one glance. The main usage of correlograms is to detect any autocorrelation after the removal of the effects of deterministic trend or seasonality. Both the ACF and the PACF are key diagnostic tools for the design of linear time series models and we will review examples of ACF and PACF plots in the following section on time series transformations.

1.4.1 NASDAQ (log, diff)

We can further analyze the relevant time series characteristics for the transformed series using a Q-Q plot that compares the quantiles of the distribution of the time series observation to the quantiles of the normal distribution and the correlograms based on the ACF and PACF.

For the NASDAQ plot, we notice that while there is no trend, the variance is not constant but rather shows clustered spikes around periods of market turmoil in the late 1980s, 2001, and 2008. The Q-Q plot highlights the fat tails of the distribution with extreme values more frequent than the normal distribution would suggest. The ACF and the PACF show similar patterns with autocorrelation at several lags appearing significant:

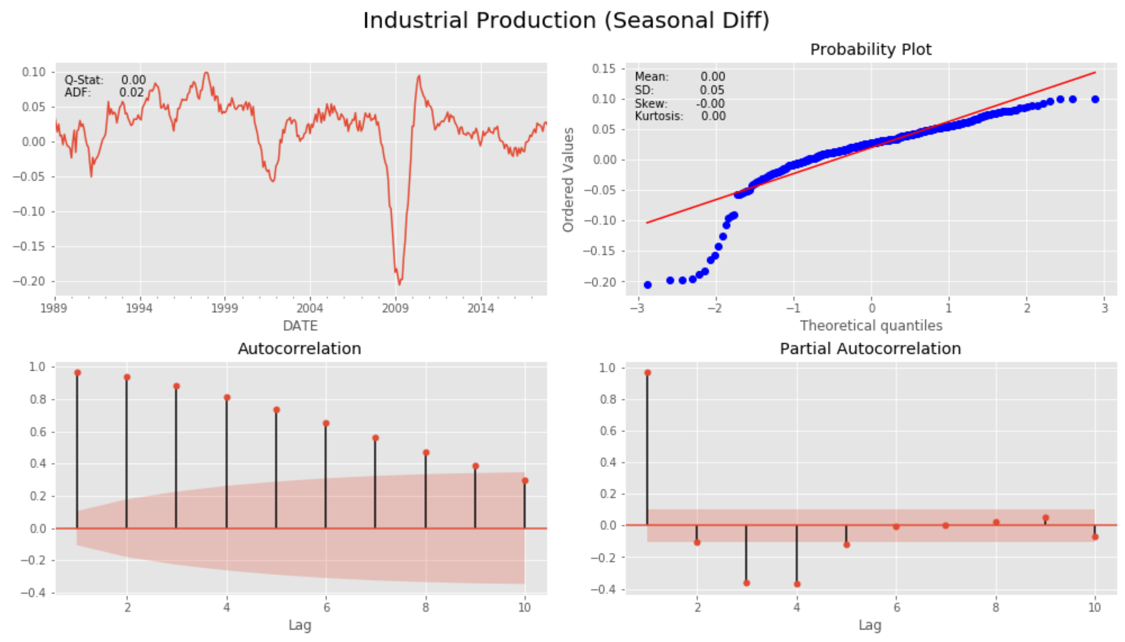
[11]:
```
plot_correlogram(nasdaq_log_diff, lags=100, title='NASDAQ Composite (Log,␣
↪Diff)')
```



1.4.2 Industrial Production (log, seasonl diff)

For the monthly time series on industrial manufacturing production, we notice a large negative outlier following the 2008 crisis as well as the corresponding skew in the Q-Q plot. The autocorrelation is much higher than for the NASDAQ returns and declines smoothly. The PACF shows distinct positive autocorrelation patterns at lag 1 and 13, and significant negative coefficients at lags 3 and 4:

```
[12]: plot_correlogram(industrial_production_log_diff, title='Industrial Production␣
       ↪(Seasonal Diff)')
```



1.5 Univariate Time Series Models

1.5.1 Autoregressive (AR) Model

Multiple linear-regression models expressed the variable of interest as a linear combination of predictors or input variables. Univariate time series models relate the value of the time series at the point in time of interest to a linear combination of lagged values of the series and possibly past disturbance terms. While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data. ARIMA(p, d, q) models require stationarity and leverage two building blocks: - Autoregressive (AR) terms consisting of p-lagged values of the time series - Moving average (MA) terms that contain q-lagged disturbances

Chapter 7 introduces the ARIMA building blocks, simple autoregressive (AR) and moving average (MA) models, and explains how to combine them in autoregressive moving-average (ARMA) models that may account for series integration as ARIMA models or include exogenous variables as AR(I)MAX models.

Furthermore, we will illustrate how to include seasonal AR and MA terms to extend the toolbox to also include SARMAX models.

1.5.2 ARMA vs ARIMA

The ARMA model of the undifferenced series produces the same result as the ARIMA model of the differenced series.

[13]:
```
model1 = tsa.ARMA(endog=nasdaq_log_diff, order=(2,2)).fit()
model2 = tsa.ARIMA(endog=nasdaq_log, order=(2,1,2)).fit()
```

```
[14]: model1.params.sort_index() == model2.params.sort_index().values
```

```
[14]: ar.L1.NASDAQCOM    True
      ar.L2.NASDAQCOM    True
      const              True
      ma.L1.NASDAQCOM    True
      ma.L2.NASDAQCOM    True
      dtype: bool
```

1.5.3 Seasonal differencing vs SARIMAX

Seasonal differencing has same effect as using SARIMAX w seasonal order (0,1,0,12).

```
[15]: model1 = tsa.statespace.SARIMAX(industrial_production_log, order=(2,0,2),␣
       ↪seasonal_order=(0,1,0,12)).fit()
      model2 = tsa.statespace.SARIMAX(industrial_production_log_diff, order=(2,0,2),␣
       ↪seasonal_order=(0,0,0,12)).fit()
```

```
[16]: model1.params.to_frame('SARIMAX').join(model2.params.to_frame('diff'))
```

```
[16]:          SARIMAX      diff
      ar.L1   1.761975  1.765978
      ar.L2  -0.784200 -0.788255
      ma.L1  -0.841204 -0.859608
      ma.L2   0.313330  0.353158
      sigma2  0.000101  0.000101
```

1.6 Finding the optimal ARMA lags

1.6.1 Run candidate models

We iterate over various (p, q) lag combinations and collect diagnostic statistics to compare the result.

```
[18]: train_size = 120
      test_results = {}
      y_true = industrial_production_log_diff.iloc[train_size:]
      for p in range(5):
          for q in range(5):
              aic, bic = [], []
              if p == 0 and q == 0:
                  continue
              print(p, q)
              convergence_error = stationarity_error = 0
              y_pred = []
              for T in range(train_size, len(industrial_production_log_diff)):
                  train_set = industrial_production_log_diff.iloc[T-train_size:T]
                  try:
```

```
                model = tsa.ARMA(endog=train_set, order=(p, q)).fit()
            except LinAlgError:
                convergence_error += 1
            except ValueError:
                stationarity_error += 1

            forecast, _, _ = model.forecast(steps=1)
            y_pred.append(forecast[0])
            aic.append(model.aic)
            bic.append(model.bic)

        result = (pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})
                  .replace(np.inf, np.nan)
                  .dropna())

        rmse = np.sqrt(mean_squared_error(
            y_true=result.y_true, y_pred=result.y_pred))

        test_results[(p, q)] = [rmse,
                                np.mean(aic),
                                np.mean(bic),
                                convergence_error,
                                stationarity_error]
```

```
0 1
0 2
0 3
0 4
1 0
1 1
1 2
1 3
1 4
2 0
2 1
2 2
2 3
2 4
3 0
3 1
3 2
3 3
3 4
4 0
4 1
4 2
4 3
```

```
4 4
```

[19]:
```
test_results = pd.DataFrame(test_results).T
test_results.columns = ['RMSE', 'AIC', 'BIC', 'convergence', 'stationarity']
test_results.index.names = ['p', 'q']
test_results.info()
```

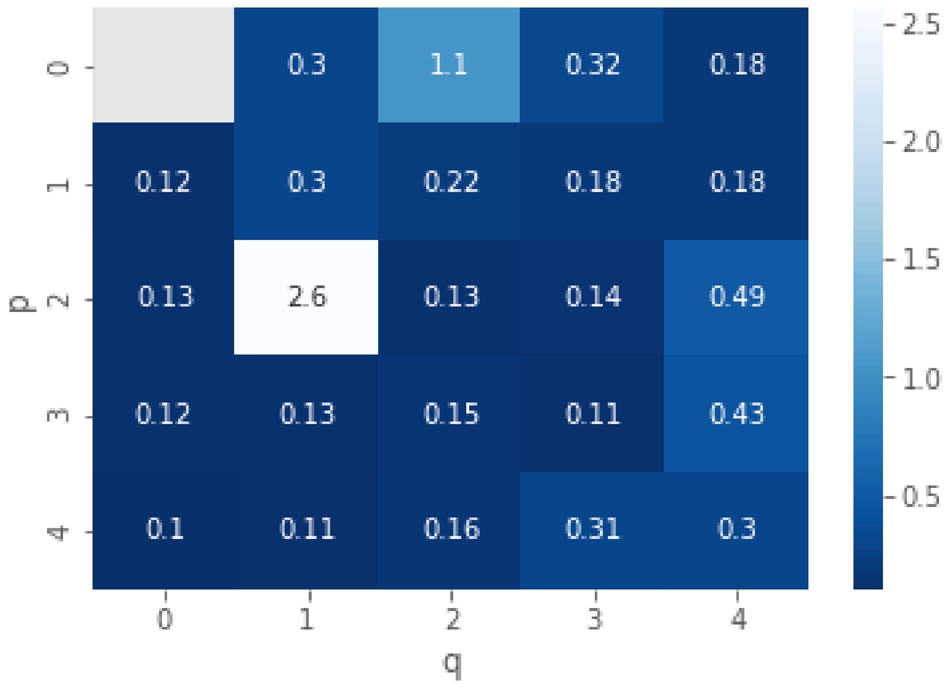
```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 24 entries, (0, 1) to (4, 4)
Data columns (total 5 columns):
RMSE            24 non-null float64
AIC             19 non-null float64
BIC             19 non-null float64
convergence     24 non-null float64
stationarity    24 non-null float64
dtypes: float64(5)
memory usage: 1.2 KB
```

[20]:
```
test_results.dropna()
```

[20]:
```
          RMSE         AIC         BIC  convergence  stationarity
p q
0 1  0.030183 -530.769444 -522.406969          0.0           0.0
  2  0.105961 -598.705151 -587.555184          0.0         107.0
  3  0.031667 -649.864725 -635.927266          0.0          93.0
  4  0.018419 -682.530641 -665.805690          0.0          93.0
1 0  0.011551 -727.389093 -719.026618          0.0           0.0
  1  0.029556 -732.051839 -720.901872          0.0          16.0
  2  0.021869 -743.997434 -730.059975          0.0          10.0
  3  0.018212 -747.753733 -731.028783          0.0           8.0
  4  0.018165 -753.080599 -733.568157          0.0           8.0
2 0  0.012655 -734.103589 -722.953622          6.0           0.0
  2  0.012650 -758.483844 -741.758894          2.0           4.0
  3  0.013529 -760.086676 -740.574233          6.0           3.0
  4  0.048993 -759.455313 -737.155379          3.0          24.0
3 0  0.012272 -747.431400 -733.493941          4.0           0.0
  1  0.012567 -754.539592 -737.814642          3.0           5.0
  4  0.043029 -762.632697 -737.545272          1.0          57.0
4 0  0.010092 -758.665355 -741.940405          0.0           0.0
  1  0.011080 -759.107983 -739.595541          1.0           2.0
  2  0.016428 -759.451378 -737.151444          7.0           4.0
```

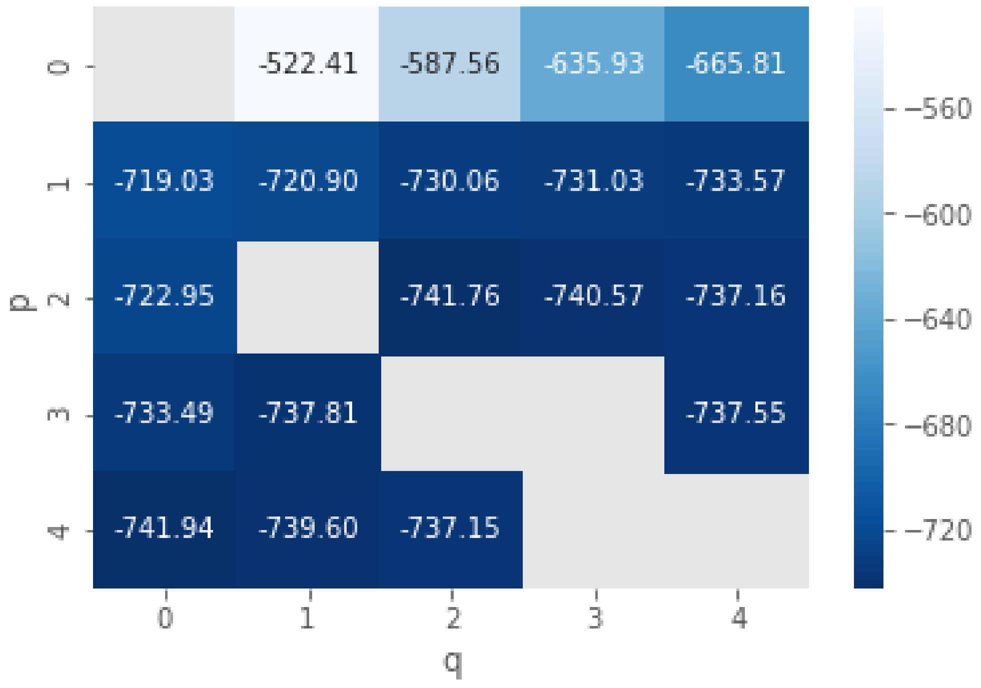
We aim to minimize RMSE:

[23]:
```
sns.heatmap(test_results.RMSE.unstack().mul(10), fmt='.2', annot=True,␣
 ↪cmap='Blues_r');
```



We also aim to minimize BIC:

```
[24]: sns.heatmap(test_results.BIC.unstack(), fmt='.2f', annot=True, cmap='Blues_r');
```



1.6.2 Estimating the best ARMA Model

ARMA(0,4) and ARMA(2,2) are close, with a slight edge for the former.

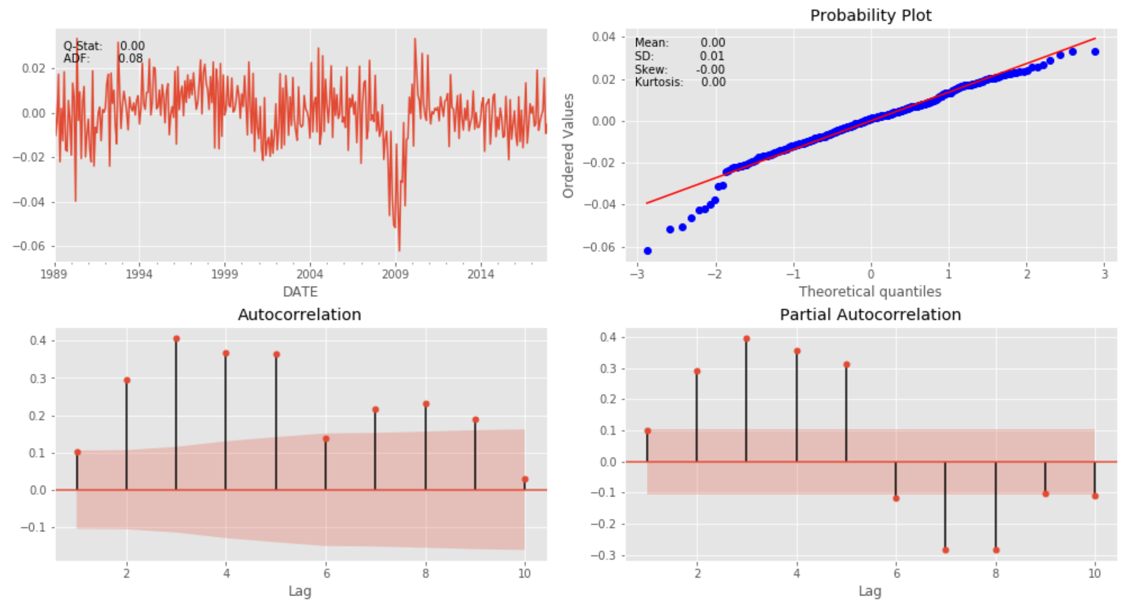
[25]:
```
model = tsa.ARMA(endog=industrial_production_log_diff, order=(0, 4)).fit()
print(model.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                 IPGMFN   No. Observations:                  348
Model:                     ARMA(0, 4)   Log Likelihood                 996.700
Method:                       css-mle   S.D. of innovations              0.014
Date:                Thu, 18 Apr 2019   AIC                          -1981.399
Time:                        23:09:24   BIC                          -1958.286
Sample:                    01-01-1989   HQIC                         -1972.197
                         - 12-01-2017
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const            0.0195      0.004      4.892      0.000       0.012       0.027
ma.L1.IPGMFN     1.2929      0.049     26.414      0.000       1.197       1.389
ma.L2.IPGMFN     1.4380      0.060     23.886      0.000       1.320       1.556
ma.L3.IPGMFN     1.1548      0.059     19.506      0.000       1.039       1.271
ma.L4.IPGMFN     0.5560      0.046     12.019      0.000       0.465       0.647
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
MA.1            0.0313           -1.0641j            1.0645           -0.2453
MA.2            0.0313           +1.0641j            1.0645            0.2453
MA.3           -1.0699           -0.6652j            1.2598           -0.4115
MA.4           -1.0699           +0.6652j            1.2598            0.4115
-----------------------------------------------------------------------------
```

1.6.3 Check Residual Correlogram

[26]:
```
plot_correlogram(model.resid)
```

None



1.7 SARIMAX

[48]:
```
model = tsa.SARIMAX(endog=industrial_production_log_diff.dropna(),
                    order=(2, 0, 2),
                    seasonal_order=(1, 0, 1, 12)).fit(start_params=[0, 0, 0, 0,␣
 ↪0, 0, 1])
print(model.summary())
```

```
                                 Statespace Model Results
================================================================================
==========
Dep. Variable:                              IPGMFN   No. Observations:
348
Model:             SARIMAX(2, 0, 2)x(1, 0, 1, 12)   Log Likelihood
1128.813
Date:                             Wed, 31 Oct 2018   AIC
-2243.626
Time:                                     16:28:47   BIC
-2216.661
Sample:                                 01-01-1989   HQIC
-2232.891
                                      - 12-01-2017
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
ar.L1          0.1105      0.643      0.172      0.864      -1.150       1.371
ar.L2          0.8825      0.640      1.380      0.168      -0.371       2.136
ma.L1          0.9857      0.637      1.547      0.122      -0.263       2.234
ma.L2          0.0813      0.083      0.975      0.330      -0.082       0.245
ar.S.L12      -0.0301      0.073     -0.413      0.680      -0.173       0.113
ma.S.L12      -0.7133      0.051    -14.111      0.000      -0.812      -0.614
sigma2      8.247e-05   5.02e-06     16.438      0.000    7.26e-05    9.23e-05
================================================================================
===
Ljung-Box (Q):                      95.86   Jarque-Bera (JB):
28.30
Prob(Q):                             0.00   Prob(JB):
0.00
Heteroskedasticity (H):              1.25   Skew:
-0.28
Prob(H) (two-sided):                 0.23   Kurtosis:
4.28
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```python
plot_correlogram(model.resid)
```

We will build a SARIMAX model for monthly data on an industrial production time series for the 1988-2017 period. As illustrated in the first section on analytical tools, the data has been log-transformed, and we are using seasonal (lag-12) differences. We estimate the model for a range of both ordinary and conventional AR and MA parameters using a rolling window of 10 years of training data, and evaluate the RMSE of the 1-step-ahead forecas

1.7.1 Finding the optimal number of lags

[ ]:
```python
train_size = 120 # 10 years of training data
test_results = {}
test_set = industrial_production_log_diff.iloc[train_size:]

for p1 in range(4):
    for q1 in range(4):
        for p2 in range(3):
            for q2 in range(3):
                preds = test_set.copy().to_frame('y_true').assign(y_pred=np.nan)
                aic, bic = [], []
                if p1 == 0 and q1 == 0:
                    continue
                print(p1, q1, p2, q2)
                convergence_error = stationarity_error = 0
                y_pred = []
                for i, T in enumerate(range(train_size,␣
↪len(industrial_production_log_diff))):
                    train_set = industrial_production_log_diff.
↪iloc[T-train_size:T]
                    try:
                        model = tsa.SARIMAX(endog=train_set,
                                            order=(p1, 0, q1),
                                            seasonal_order=(p2, 0, q2, 12)).
↪fit()
                    except LinAlgError:
                        convergence_error += 1
                    except ValueError:
                        stationarity_error += 1

                    preds.iloc[i, 1] = model.forecast(steps=1)[0]
                    aic.append(model.aic)
                    bic.append(model.bic)

                preds.dropna(inplace=True)
                mse = mean_squared_error(preds.y_true, preds.y_pred)
                test_results[(p1, q1, p2, q2)] = [np.sqrt(mse),
                                                  preds.y_true.sub(preds.
↪y_pred).std(),
                                                  np.mean(aic),
                                                  np.std(aic),
                                                  np.mean(bic),
                                                  np.std(bic),
                                                  convergence_error,
                                                  stationarity_error]
```

1.7.2 Compare model metrics

```
[ ]: df = pd.DataFrame(test_results).T
     df.columns = ['RMSE', 'RMSE_std', 'AIC', 'AIC_std', 'BIC', 'BIC_std',␣
      ↪'convergence', 'stationarity']
     df['CV'] = df.RMSE_std.div(df.RMSE)
     df.index.names = ['p1', 'q1', 'p2', 'q2']
     df.info()
```

```
[ ]: with pd.HDFStore('results/arima.h5') as store:
         store.put('arima_results/3', df)
         print(store.info())
```

```
[51]: with pd.HDFStore('results/arima.h5') as store:
          df = store.get('arima_results/3')
```

```
[52]: df.sort_values('RMSE').head(10)
```

```
[52]:                    RMSE  RMSE_std         AIC    AIC_std         BIC    BIC_std  \
      p1 q1 p2 q2
      2  3  1  0   0.009323  0.009335 -772.247023  20.106627 -752.734581  20.106627
      3  2  1  0   0.009467  0.009486 -768.844028  17.479101 -749.331586  17.479101
      2  2  1  0   0.009540  0.009559 -770.904835  17.162805 -754.179884  17.162805
         3  0  0   0.009773  0.009790 -760.248885  23.627233 -743.523935  23.627233
         2  0  0   0.009986  0.010005 -758.775827  20.636387 -744.838368  20.636387
      3  2  0  0   0.010066  0.010088 -757.467934  20.319519 -740.742984  20.319519
         3  1  0   0.010490  0.010457 -770.605332  19.677360 -748.305398  19.677360
            0  0   0.011543  0.011511 -758.301684  23.107904 -738.789241  23.107904
         2  2  0   0.011688  0.011617 -767.646566  28.365300 -745.199922  28.261192
      2  3  2  0   0.011909  0.011857 -770.765094  30.125115 -748.318450  30.041888

                   convergence  stationarity        CV
      p1 q1 p2 q2
      2  3  1  0           2.0           1.0  1.001263
      3  2  1  0           1.0           1.0  1.001913
      2  2  1  0           0.0           2.0  1.002046
         3  0  0           0.0           1.0  1.001768
         2  0  0           0.0           2.0  1.001922
      3  2  0  0           2.0           1.0  1.002172
         3  1  0           2.0           6.0  0.996836
            0  0           1.0           6.0  0.997194
         2  2  0           2.0          13.0  0.993956
      2  3  2  0           1.0          13.0  0.995627
```

We also collect the AIC and BIC criteria that show a very high rank correlation coefficient of 0.94, with BIC favoring models with slightly fewer parameters than AIC. The best five models by RMSE are:

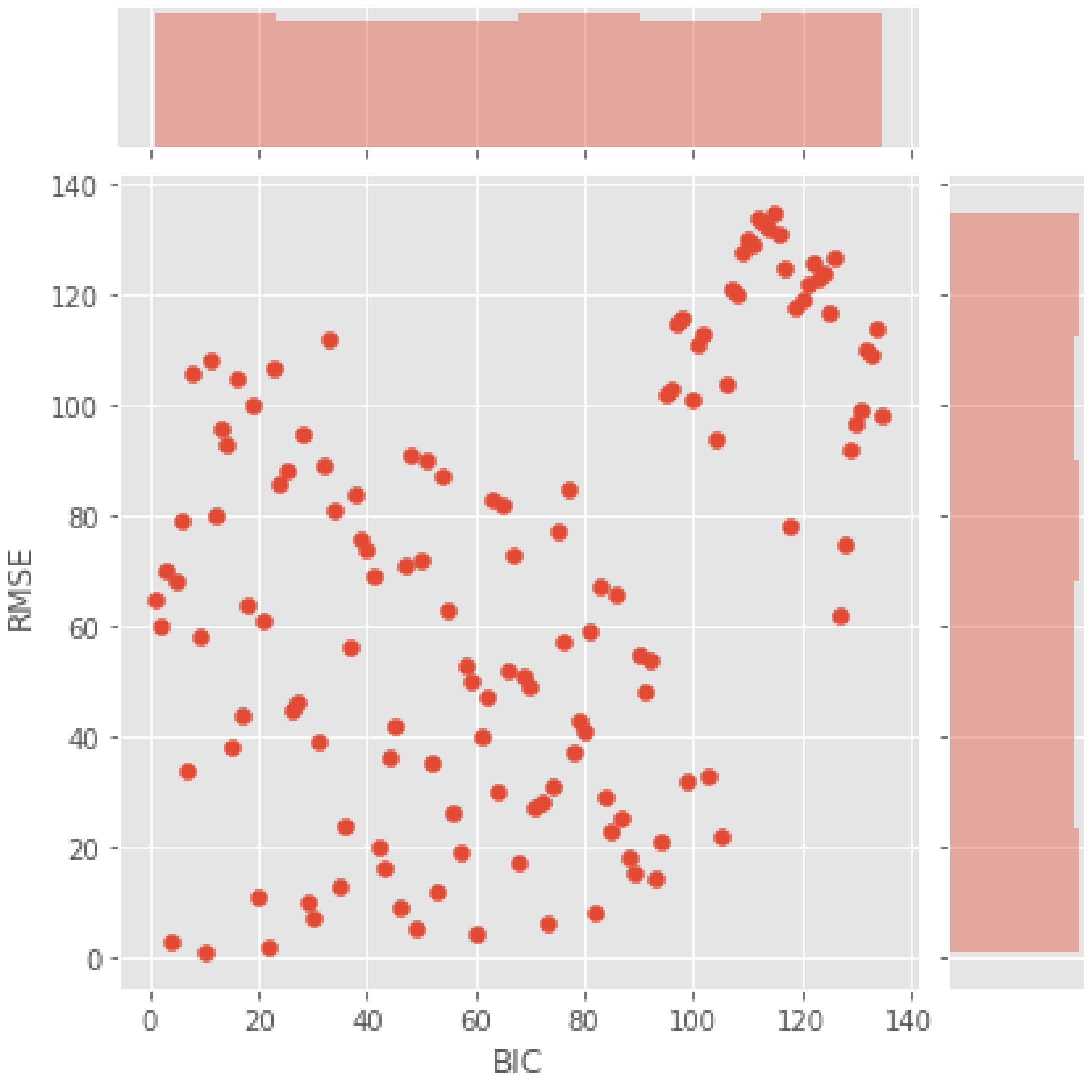
```
[53]: print(df[['RMSE', 'AIC', 'BIC']].sort_values('RMSE').head())
```

```
                   RMSE         AIC         BIC
p1 q1 p2 q2
2  3  1  0   0.009323 -772.247023 -752.734581
3  2  1  0   0.009467 -768.844028 -749.331586
2  2  1  0   0.009540 -770.904835 -754.179884
   3  0  0   0.009773 -760.248885 -743.523935
   2  0  0   0.009986 -758.775827 -744.838368
```

```
[54]: df[['RMSE', 'AIC', 'BIC']].corr('spearman')
```

```
[54]:          RMSE       AIC       BIC
      RMSE  1.000000  0.368545  0.417667
      AIC   0.368545  1.000000  0.940401
      BIC   0.417667  0.940401  1.000000
```

[56]:
```
sns.jointplot(y='RMSE', x='BIC', data=df[['RMSE', 'BIC']].rank());
```



[57]:
```
df[(df.RMSE<df.RMSE.quantile(.05))&(df.BIC<df.BIC.quantile(.1))]
```

[57]:
```
                    RMSE  RMSE_std         AIC    AIC_std         BIC    BIC_std  \
p1 q1 p2 q2
2  2  1  0    0.009540  0.009559 -770.904835  17.162805 -754.179884  17.162805
   3  1  0    0.009323  0.009335 -772.247023  20.106627 -752.734581  20.106627

              convergence  stationarity        CV
p1 q1 p2 q2
2  2  1  0            0.0           2.0  1.002046
   3  1  0            2.0           1.0  1.001263
```

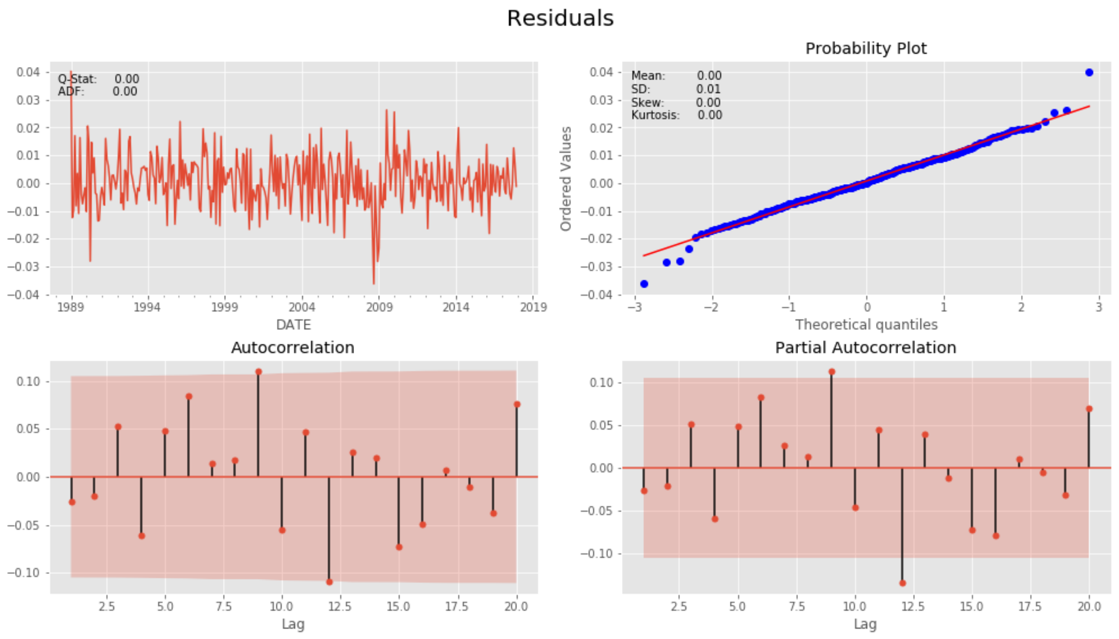
1.7.3 Train best model

[58]:
```
best_model = tsa.SARIMAX(endog=industrial_production_log_diff, order=(2, 0, 3),
                         seasonal_order=(1, 0, 0, 12)).fit()
print(best_model.summary())
```

```
                                 Statespace Model Results
================================================================================
==========
Dep. Variable:                             IPGMFN   No. Observations:
348
Model:               SARIMAX(2, 0, 3)x(1, 0, 0, 12)   Log Likelihood
1139.719
Date:                            Wed, 31 Oct 2018   AIC
-2265.438
Time:                                    22:58:00   BIC
-2238.472
Sample:                                01-01-1989   HQIC
-2254.702
                                     - 12-01-2017
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.4934      0.104     14.351      0.000       1.289       1.697
ar.L2         -0.5159      0.102     -5.083      0.000      -0.715      -0.317
ma.L1         -0.5499      0.114     -4.813      0.000      -0.774      -0.326
ma.L2          0.2872      0.062      4.662      0.000       0.166       0.408
ma.L3          0.1815      0.070      2.589      0.010       0.044       0.319
ar.S.L12      -0.4486      0.047     -9.533      0.000      -0.541      -0.356
sigma2      8.141e-05   5.65e-06     14.399      0.000    7.03e-05    9.25e-05
================================================================================
===
Ljung-Box (Q):                       61.58   Jarque-Bera (JB):
9.97
Prob(Q):                              0.02   Prob(JB):
0.01
Heteroskedasticity (H):               1.07   Skew:
-0.20
Prob(H) (two-sided):                  0.71   Kurtosis:
3.73
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

1.7.4 Check Residual Correlogram

[60]:
```
plot_correlogram(best_model.resid, lags=20, title='Residuals')
```


Residuals
Q-Stat: 0.00
ADF: 0.00
DATE
Probability Plot
Mean: 0.00
SD: 0.01
Skew: 0.00
Kurtosis: 0.00
Ordered Values
Theoretical quantiles
Autocorrelation
Lag
Partial Autocorrelation
Lag