SARIMAX: Introduction

This notebook replicates examples from the Stata ARIMA time series estimation and postestimation documentation.

First, we replicate the four estimation examples http://www.stata.com/manuals13/tsarima.pdf [http://www.stata.com/manuals13/tsarima.pdf]:

1. ARIMA(1,1,1) model on the U.S. Wholesale Price Index (WPI) dataset.
2. Variation of example 1 which adds an MA(4) term to the ARIMA(1,1,1) specification to allow for an additive seasonal effect.
3. ARIMA(2,1,0) x (1,1,0,12) model of monthly airline data. This example allows a multiplicative seasonal effect.
4. ARMA(1,1) model with exogenous regressors; describes consumption as an autoregressive process on which also the money supply is assumed to be an explanatory variable.

Second, we demonstrate postestimation capabilities to replicate http://www.stata.com/manuals13/tsarimapostestimation.pdf [http://www.stata.com/manuals13/tsarimapostestimation.pdf]. The model from example 4 is used to demonstrate:

1. One-step-ahead in-sample prediction
2. n-step-ahead out-of-sample forecasting
3. n-step-ahead in-sample dynamic prediction

```
[1]: %matplotlib inline
```

```
[2]: import numpy as np
import pandas as pd
from scipy.stats import norm
import statsmodels.api as sm
import matplotlib.pyplot as plt
from datetime import datetime
import requests
from io import BytesIO
# Register converters to avoid warnings
pd.plotting.register_matplotlib_converters()
plt.rc("figure", figsize=(16,8))
plt.rc("font", size=14)
```

ARIMA Example 1: Arima

As can be seen in the graphs from Example 2, the Wholesale price index (WPI) is growing over time (i.e. is not stationary). Therefore an ARMA model is not a good specification. In this first example, we consider a model where the original time series is assumed to be integrated of order 1, so that the difference is assumed to be stationary, and fit a model with one autoregressive lag and one moving average lag, as well as an intercept term.

The postulated data process is then:

$$\Delta y_t = c + \phi_1 \Delta y_{t-1} + \theta_1 \epsilon_{t-1} + \epsilon_t$$

where c is the intercept of the ARMA model, Δ is the first-difference operator, and we assume $\epsilon_t \sim N(0, \sigma^2)$. This can be rewritten to emphasize lag polynomials as (this will be useful in example 2, below):

$$(1-\phi_1 L)\Delta y_t = c + (1+\theta_1 L)\epsilon_t$$

where L is the lag operator.

Notice that one difference between the Stata output and the output below is that Stata estimates the following model:

$$(\Delta y_t - \beta_0) = \phi_1(\Delta y_{t-1} - \beta_0) + \theta_1 \epsilon_{t-1} + \epsilon_t$$

where β_0 is the mean of the process y_t. This model is equivalent to the one estimated in the statsmodels SARIMAX class, but the interpretation is different. To see the equivalence, note that:

$$\begin{aligned}(\Delta y_t - \beta_0) &= \phi_1(\Delta y_{t-1} - \beta_0) + \theta_1 \epsilon_{t-1} + \epsilon_t \\ \Delta y_t &= (1-\phi_1)\beta_0 + \phi_1 \Delta y_{t-1} + \theta_1 \epsilon_{t-1} + \epsilon_t\end{aligned}$$

so that $c = (1-\phi_1)\beta_0$.

[3]:

```
# Dataset
wpi1 = requests.get('https://www.stata-press.com/data/r12/wpi1.dta').content
data = pd.read_stata(BytesIO(wpi1))
data.index = data.t
# Set the frequency
data.index.freq="QS-OCT"

# Fit the model
mod = sm.tsa.statespace.SARIMAX(data['wpi'], trend='c', order=(1,1,1))
res = mod.fit(disp=False)
print(res.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    wpi   No. Observations:                  124
```

```
Model:               SARIMAX(1, 1, 1)   Log Likelihood                -135.351
Date:                Mon, 09 May 2022   AIC                            278.703
Time:                        08:24:51   BIC                            289.951
Sample:                    01-01-1960   HQIC                           283.272
                         - 10-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0943      0.068      1.389      0.165      -0.039       0.227
ar.L1          0.8742      0.055     16.028      0.000       0.767       0.981
ma.L1         -0.4120      0.100     -4.119      0.000      -0.608      -0.216
sigma2         0.5257      0.053      9.849      0.000       0.421       0.630
===================================================================================
Ljung-Box (L1) (Q):                   0.09   Jarque-Bera (JB):                 9.78
Prob(Q):                              0.77   Prob(JB):                         0.01
Heteroskedasticity (H):              15.93   Skew:                             0.28
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.26
===================================================================================

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

Thus the maximum likelihood estimates imply that for the process above, we have:

$$\Delta y_t = 0.0943 + 0.8742\Delta y_{t-1} - 0.4120\epsilon_{t-1} + \epsilon_t$$

where $\epsilon_t \sim N(0, 0.5257)$. Finally, recall that $c = (1 - \phi_1)\beta_0$, and here $c = 0.0943$ and $\phi_1 = 0.8742$. To compare with the output from Stata, we could calculate the mean:

$$\beta_0 = \frac{c}{1 - \phi_1} = \frac{0.0943}{1 - 0.8742} = 0.7496$$

Note: This value is virtually identical to the value in the Stata documentation, $\beta_0 = 0.7498$. The slight difference is likely down to rounding and subtle differences in stopping criterion of the numerical optimizers used.

ARIMA Example 2: Arima with additive seasonal effects

This model is an extension of that from example 1. Here the data is assumed to follow the process:

$$\Delta y_t = c + \phi_1 \Delta y_{t-1} + \theta_1 \epsilon_{t-1} + \theta_4 \epsilon_{t-4} + \epsilon_t$$

The new part of this model is that there is allowed to be a annual seasonal effect (it is annual even though the periodicity is 4 because the dataset is quarterly). The second difference is that this model uses the log of the data rather than the level.

Before estimating the dataset, graphs showing:

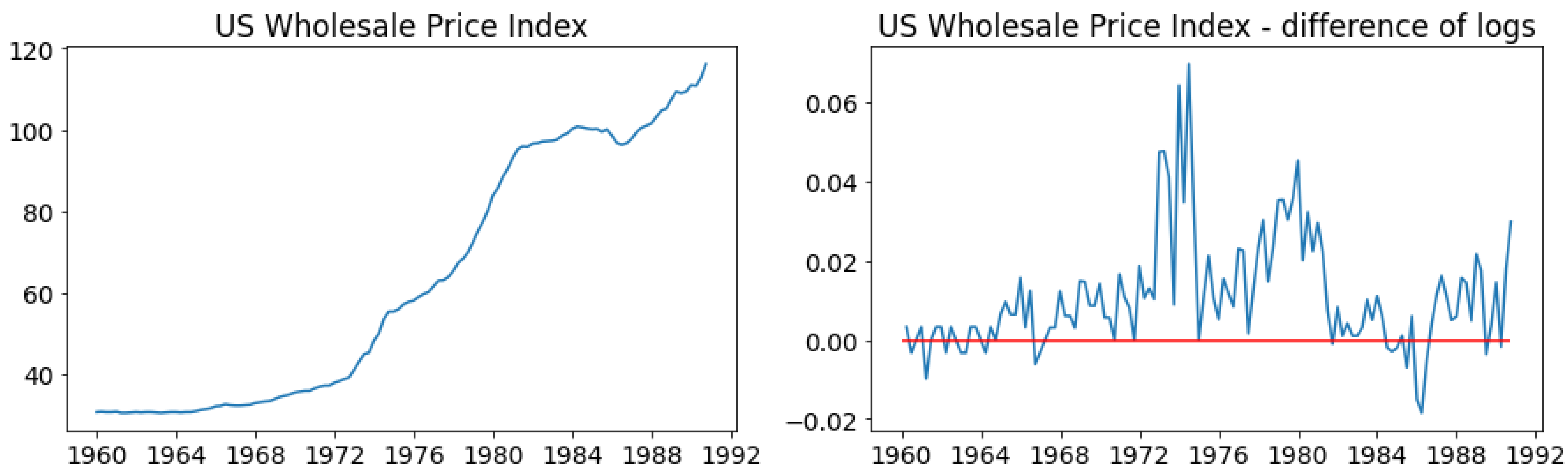
1. The time series (in logs)
2. The first difference of the time series (in logs)
3. The autocorrelation function
4. The partial autocorrelation function.

From the first two graphs, we note that the original time series does not appear to be stationary, whereas the first-difference does. This supports either estimating an ARMA model on the first-difference of the data, or estimating an ARIMA model with 1 order of integration (recall that we are taking the latter approach). The last two graphs support the use of an ARMA(1,1,1) model.

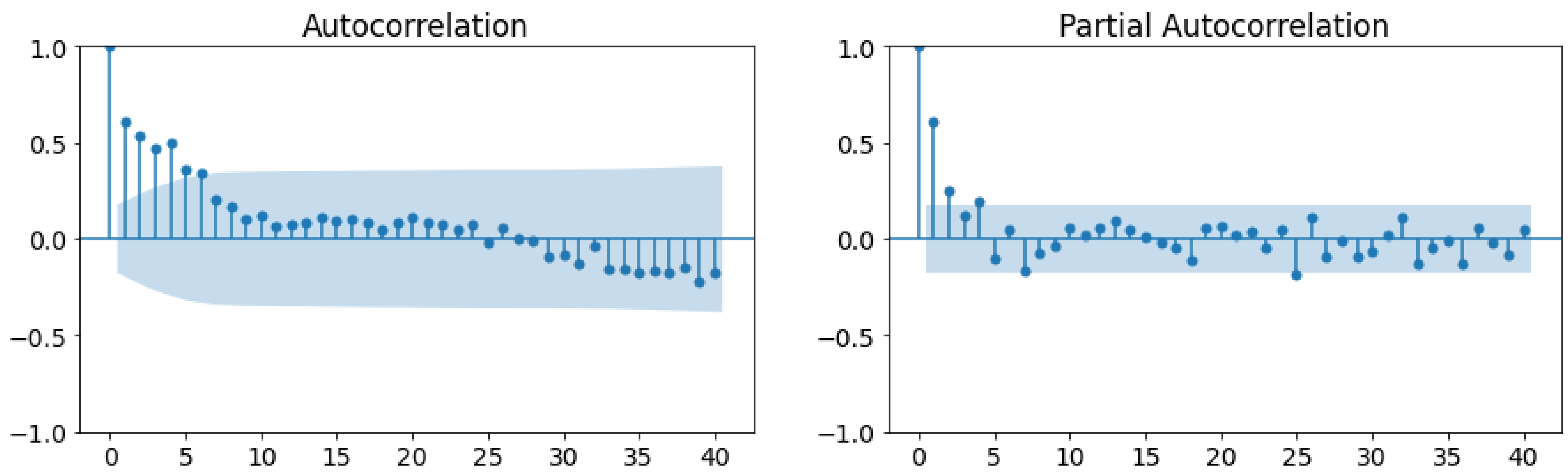
[4]:
```
# Dataset
data = pd.read_stata(BytesIO(wpi1))
```

```
data.index = data.t
data.index.freq="QS-OCT"

data['ln_wpi'] = np.log(data['wpi'])
data['D.ln_wpi'] = data['ln_wpi'].diff()
```

[5]:
```
# Graph data
fig, axes = plt.subplots(1, 2, figsize=(15,4))

# Levels
axes[0].plot(data.index._mpl_repr(), data['wpi'], '-')
axes[0].set(title='US Wholesale Price Index')

# Log difference
axes[1].plot(data.index._mpl_repr(), data['D.ln_wpi'], '-')
axes[1].hlines(0, data.index[0], data.index[-1], 'r')
axes[1].set(title='US Wholesale Price Index - difference of logs');
```



[6]:
```python
# Graph data
fig, axes = plt.subplots(1, 2, figsize=(15,4))

fig = sm.graphics.tsa.plot_acf(data.iloc[1:]['D.ln_wpi'], lags=40, ax=axes[0])
fig = sm.graphics.tsa.plot_pacf(data.iloc[1:]['D.ln_wpi'], lags=40, ax=axes[1])
```



To understand how to specify this model in statsmodels, first recall that from example 1 we used the following code to specify the ARIMA(1,1,1) model:

```python
mod = sm.tsa.statespace.SARIMAX(data['wpi'], trend='c', order=(1,1,1))
```

The `order` argument is a tuple of the form `(AR specification, Integration order, MA specification)`. The integration order must be an integer (for example, here we assumed one order of integration, so it was specified as 1. In a pure ARMA model where the underlying data is already stationary, it would be 0).

For the AR specification and MA specification components, there are two possibilities. The first is to specify the **maximum degree** of the corresponding lag polynomial, in which case the component is an integer. For example, if we wanted to specify an ARIMA(1,1,4) process, we would use:

```
mod = sm.tsa.statespace.SARIMAX(data['wpi'], trend='c', order=(1,1,4))
```

and the corresponding data process would be:

$$y_t = c + \phi_1 y_{t-1} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \theta_3 \epsilon_{t-3} + \theta_4 \epsilon_{t-4} + \epsilon_t$$

or

$$(1 - \phi_1 L)\Delta y_t = c + (1 + \theta_1 L + \theta_2 L^2 + \theta_3 L^3 + \theta_4 L^4)\epsilon_t$$

When the specification parameter is given as a maximum degree of the lag polynomial, it implies that all polynomial terms up to that degree are included. Notice that this is *not* the model we want to use, because it would include terms for ϵ_{t-2} and ϵ_{t-3}, which we do not want here.

What we want is a polynomial that has terms for the 1st and 4th degrees, but leaves out the 2nd and 3rd terms. To do that, we need to provide a tuple for the specification parameter, where the tuple describes **the lag polynomial itself**. In particular, here we would want to use:

```
ar = 1          # this is the maximum degree specification
ma = (1,0,0,1)  # this is the lag polynomial specification
mod = sm.tsa.statespace.SARIMAX(data['wpi'], trend='c', order=(ar,1,ma)))
```

This gives the following form for the process of the data:

$$\begin{aligned}\Delta y_t &= c + \phi_1 \Delta y_{t-1} + \theta_1 \epsilon_{t-1} + \theta_4 \epsilon_{t-4} + \epsilon_t \\ (1 - \phi_1 L)\Delta y_t &= c + (1 + \theta_1 L + \theta_4 L^4)\epsilon_t\end{aligned}$$

which is what we want.

```
[7]: # Fit the model
mod = sm.tsa.statespace.SARIMAX(data['ln_wpi'], trend='c', order=(1,1,(1,0,0,1)))
res = mod.fit(disp=False)
print(res.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                 ln_wpi   No. Observations:                  124
Model:           SARIMAX(1, 1, [1, 4])   Log Likelihood                 385.864
Date:                Mon, 09 May 2022   AIC                           -761.727
Time:                        08:24:52   BIC                           -747.666
Sample:                    01-01-1960   HQIC                          -756.016
                         - 10-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0029      0.002      1.665      0.096      -0.001       0.006
ar.L1          0.7369      0.097      7.582      0.000       0.546       0.927
ma.L1         -0.3456      0.123     -2.817      0.005      -0.586      -0.105
ma.L4          0.3438      0.115      2.996      0.003       0.119       0.569
sigma2         0.0001   9.95e-06     10.974      0.000    8.97e-05       0.000
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                45.38
Prob(Q):                              0.99   Prob(JB):                         0.00
Heteroskedasticity (H):               2.53   Skew:                             0.33
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.90
===================================================================================

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

ARIMA Example 3: Airline Model

In the previous example, we included a seasonal effect in an *additive* way, meaning that we added a term allowing the process to depend on the 4th MA lag. It may be instead that we want to model a seasonal effect in a multiplicative way. We often write the model then as an ARIMA $(p, d, q) \times (P, D, Q)_s$, where the lowercase letters indicate the specification for the non-seasonal component, and the uppercase letters indicate the specification for the seasonal component; s is the periodicity of the seasons (e.g. it is often 4 for quarterly data or 12 for monthly data). The data process can be written generically as:

$$\phi_p(L)\tilde{\phi}_P(L^s)\Delta^d\Delta_s^D y_t = A(t) + \theta_q(L)\tilde{\theta}_Q(L^s)\epsilon_t$$

where:

- $\phi_p(L)$ is the non-seasonal autoregressive lag polynomial
- $\tilde{\phi}_P(L^s)$ is the seasonal autoregressive lag polynomial
- $\Delta^d\Delta_s^D y_t$ is the time series, differenced d times, and seasonally differenced D times.
- $A(t)$ is the trend polynomial (including the intercept)
- $\theta_q(L)$ is the non-seasonal moving average lag polynomial
- $\tilde{\theta}_Q(L^s)$ is the seasonal moving average lag polynomial

sometimes we rewrite this as:

$$\phi_p(L)\tilde{\phi}_P(L^s)y_t^* = A(t) + \theta_q(L)\tilde{\theta}_Q(L^s)\epsilon_t$$

where $y_t^* = \Delta^d\Delta_s^D y_t$. This emphasizes that just as in the simple case, after we take differences (here both non-seasonal and seasonal) to make the data stationary, the resulting model is just an ARMA model.

As an example, consider the airline model ARIMA $(2,1,0) \times (1,1,0)_{12}$, with an intercept. The data process can be written in the form above as:

$$(1 - \phi_1 L - \phi_2 L^2)(1 - \tilde{\phi}_1 L^{12}) \Delta \Delta_{12} y_t = c + \epsilon_t$$

Here, we have:

- $\phi_p(L) = (1 - \phi_1 L - \phi_2 L^2)$
- $\tilde{\phi}_P(L^s) = (1 - \phi_1 L^1 2)$
- $d = 1, D = 1, s = 12$ indicating that y_t^* is derived from y_t by taking first-differences and then taking 12-th differences.
- $A(t) = c$ is the *constant* trend polynomial (i.e. just an intercept)
- $\theta_q(L) = \tilde{\theta}_Q(L^s) = 1$ (i.e. there is no moving average effect)

It may still be confusing to see the two lag polynomials in front of the time-series variable, but notice that we can multiply the lag polynomials together to get the following model:

$$(1 - \phi_1 L - \phi_2 L^2 - \tilde{\phi}_1 L^{12} + \phi_1 \tilde{\phi}_1 L^{13} + \phi_2 \tilde{\phi}_1 L^{14}) y_t^* = c + \epsilon_t$$

which can be rewritten as:

$$y_t^* = c + \phi_1 y_{t-1}^* + \phi_2 y_{t-2}^* + \tilde{\phi}_1 y_{t-12}^* - \phi_1 \tilde{\phi}_1 y_{t-13}^* - \phi_2 \tilde{\phi}_1 y_{t-14}^* + \epsilon_t$$

This is similar to the additively seasonal model from example 2, but the coefficients in front of the autoregressive lags are actually combinations of the underlying seasonal and non-seasonal parameters.

Specifying the model in statsmodels is done simply by adding the `seasonal_order` argument, which accepts a tuple of the form `(Seasonal AR specification, Seasonal Integration order, Seasonal MA, Seasonal periodicity)`. The seasonal AR and MA specifications, as before, can be expressed as a maximum polynomial degree or as the lag polynomial itself. Seasonal periodicity is an integer.

For the airline model ARIMA $(2,1,0) \times (1,1,0)_{12}$ with an intercept, the command is:

```
mod = sm.tsa.statespace.SARIMAX(data['lnair'], order=(2,1,0), seasonal_order=(1,1,0,12))
```

[8]:

```
# Dataset
air2 = requests.get('https://www.stata-press.com/data/r12/air2.dta').content
data = pd.read_stata(BytesIO(air2))
data.index = pd.date_range(start=datetime(data.time[0], 1, 1), periods=len(data), freq='MS')
data['lnair'] = np.log(data['air'])

# Fit the model
mod = sm.tsa.statespace.SARIMAX(data['lnair'], order=(2,1,0), seasonal_order=(1,1,0,12), simple_differencing=True)
res = mod.fit(disp=False)
print(res.summary())
```

```
/tmp/ipykernel_6125/3610926118.py:4: DeprecationWarning: an integer is required (got type numpy.float32).  Implicit conversion to integers using __int__ is deprecated, and may be removed in a future version of Python.
  data.index = pd.date_range(start=datetime(data.time[0], 1, 1), periods=len(data), freq='MS')
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                         D.DS12.lnair   No. Observations:              131
Model:             SARIMAX(2, 0, 0)x(1, 0, 0, 12)   Log Likelihood               240.821
Date:                            Mon, 09 May 2022   AIC                         -473.643
Time:                                    08:24:52   BIC                         -462.142
Sample:                                02-01-1950   HQIC                        -468.970
                                     - 12-01-1960
```

```
Covariance Type:                                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4057      0.080     -5.045      0.000      -0.563      -0.248
ar.L2         -0.0799      0.099     -0.809      0.419      -0.274       0.114
ar.S.L12      -0.4723      0.072     -6.592      0.000      -0.613      -0.332
sigma2         0.0014      0.000      8.403      0.000       0.001       0.002
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                 0.72
Prob(Q):                              0.91   Prob(JB):                         0.70
Heteroskedasticity (H):               0.54   Skew:                             0.14
Prob(H) (two-sided):                  0.04   Kurtosis:                         3.23
===================================================================================

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

Notice that here we used an additional argument `simple_differencing=True`. This controls how the order of integration is handled in ARIMA models. If `simple_differencing=True`, then the time series provided as `endog` is literally differenced and an ARMA model is fit to the resulting new time series. This implies that a number of initial periods are lost to the differencing process, however it may be necessary either to compare results to other packages (e.g. Stata's `arima` always uses simple differencing) or if the seasonal periodicity is large.

The default is `simple_differencing=False`, in which case the integration component is implemented as part of the state space formulation, and all of the original data can be used in estimation.

ARIMA Example 4: ARMAX (Friedman)

This model demonstrates the use of explanatory variables (the X part of ARMAX). When exogenous regressors are included, the SARIMAX module uses the concept of “regression with SARIMA errors” (see http://robjhyndman.com/hyndsight/arimax/ [http://robjhyndman.com/hyndsight/arimax/] for details of regression with ARIMA errors versus alternative specifications), so that the model is specified as:

$$
\begin{aligned}
y_t & = \beta_t x_t + u_t \\
\phi_p (L) \tilde \phi_P (L^s) \Delta^d \Delta_s^D u_t & = A(t) + \theta_q (L) \tilde \theta_Q (L^s) \epsilon_t
\end{aligned}
$$

Notice that the first equation is just a linear regression, and the second equation just describes the process followed by the error component as SARIMA (as was described in example 3). One reason for this specification is that the estimated parameters have their natural interpretations.

This specification nests many simpler specifications. For example, regression with AR(2) errors is:

$$
\begin{aligned}
y_t & = \beta_t x_t + u_t \\
(1 - \phi_1 L - \phi_2 L^2) u_t & = A(t) + \epsilon_t
\end{aligned}
$$

The model considered in this example is regression with ARMA(1,1) errors. The process is then written:

$$
\begin{aligned}
\text{consump}_t & = \beta_0 + \beta_1 \text{m2}_t + u_t \\
(1 - \phi_1 L) u_t & = (1 - \theta_1 L) \epsilon_t
\end{aligned}
$$

Notice that β_0 is, as described in example 1 above, *not* the same thing as an intercept specified by `trend='c'`. Whereas in the examples above we estimated the intercept of the model via the trend polynomial, here, we demonstrate how to estimate β_0 itself by adding a constant to the exogenous dataset. In the output, the $beta_0$ is called `const`, whereas above the intercept c was called `intercept` in the output.

[9]:
```
# Dataset
friedman2 = requests.get('https://www.stata-press.com/data/r12/friedman2.dta').content
data = pd.read_stata(BytesIO(friedman2))
data.index = data.time
```

```python
data.index.freq = "QS-OCT"

# Variables
endog = data.loc['1959':'1981', 'consump']
exog = sm.add_constant(data.loc['1959':'1981', 'm2'])

# Fit the model
mod = sm.tsa.statespace.SARIMAX(endog, exog, order=(1,0,1))
res = mod.fit(disp=False)
print(res.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                consump   No. Observations:                   92
Model:               SARIMAX(1, 0, 1)   Log Likelihood                -340.508
Date:                Mon, 09 May 2022   AIC                            691.015
Time:                        08:24:53   BIC                            703.624
Sample:                    01-01-1959   HQIC                           696.105
                         - 10-01-1981
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -36.0629     56.642     -0.637      0.524    -147.078      74.952
m2             1.1220      0.036     30.826      0.000       1.051       1.193
ar.L1          0.9348      0.041     22.717      0.000       0.854       1.016
ma.L1          0.3091      0.089      3.488      0.000       0.135       0.483
sigma2        93.2549     10.888      8.565      0.000      71.914     114.596
===================================================================================
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):                23.49
Prob(Q):                              0.84   Prob(JB):                         0.00
Heteroskedasticity (H):              22.51   Skew:                             0.17
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.45
===================================================================================
```

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

ARIMA Postestimation: Example 1 - Dynamic Forecasting

Here we describe some of the post-estimation capabilities of statsmodels' SARIMAX.

First, using the model from example, we estimate the parameters using data that *excludes the last few observations* (this is a little artificial as an example, but it allows considering performance of out-of-sample forecasting and facilitates comparison to Stata's documentation).

[10]:

```
# Dataset
raw = pd.read_stata(BytesIO(friedman2))
raw.index = raw.time
raw.index.freq = "QS-OCT"
data = raw.loc[:'1981']

# Variables
endog = data.loc['1959':, 'consump']
exog = sm.add_constant(data.loc['1959':, 'm2'])
nobs = endog.shape[0]

# Fit the model
mod = sm.tsa.statespace.SARIMAX(endog.loc[:'1978-01-01'], exog=exog.loc[:'1978-01-01'], order=(1,0,1))
fit_res = mod.fit(disp=False, maxiter=250)
print(fit_res.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                consump   No. Observations:                   77
```

```
Model:                 SARIMAX(1, 0, 1)   Log Likelihood                -243.316
Date:                  Mon, 09 May 2022   AIC                            496.633
Time:                          08:24:53   BIC                            508.352
Sample:                      01-01-1959   HQIC                           501.320
                           - 01-01-1978
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.6761     18.491      0.037      0.971     -35.565      36.917
m2             1.0379      0.021     50.331      0.000       0.997       1.078
ar.L1          0.8775      0.059     14.859      0.000       0.762       0.993
ma.L1          0.2771      0.108      2.571      0.010       0.066       0.488
sigma2        31.6981      4.683      6.769      0.000      22.520      40.877
===================================================================================
Ljung-Box (L1) (Q):                   0.32   Jarque-Bera (JB):                 6.05
Prob(Q):                              0.57   Prob(JB):                         0.05
Heteroskedasticity (H):               6.09   Skew:                             0.57
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.76
===================================================================================

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

Next, we want to get results for the full dataset but using the estimated parameters (on a subset of the data).

[11]:
```
mod = sm.tsa.statespace.SARIMAX(endog, exog=exog, order=(1,0,1))
res = mod.filter(fit_res.params)
```

The `predict` command is first applied here to get in-sample predictions. We use the `full_results=True` argument to allow us to calculate confidence intervals (the default output of `predict` is just the predicted values).

With no other arguments, `predict` returns the one-step-ahead in-sample predictions for the entire sample.

[12]:
```
# In-sample one-step-ahead predictions
predict = res.get_prediction()
predict_ci = predict.conf_int()
```

We can also get *dynamic predictions*. One-step-ahead prediction uses the true values of the endogenous values at each step to predict the next in-sample value. Dynamic predictions use one-step-ahead prediction up to some point in the dataset (specified by the `dynamic` argument); after that, the previous *predicted* endogenous values are used in place of the true endogenous values for each new predicted element.

The `dynamic` argument is specified to be an *offset* relative to the `start` argument. If `start` is not specified, it is assumed to be `0`.

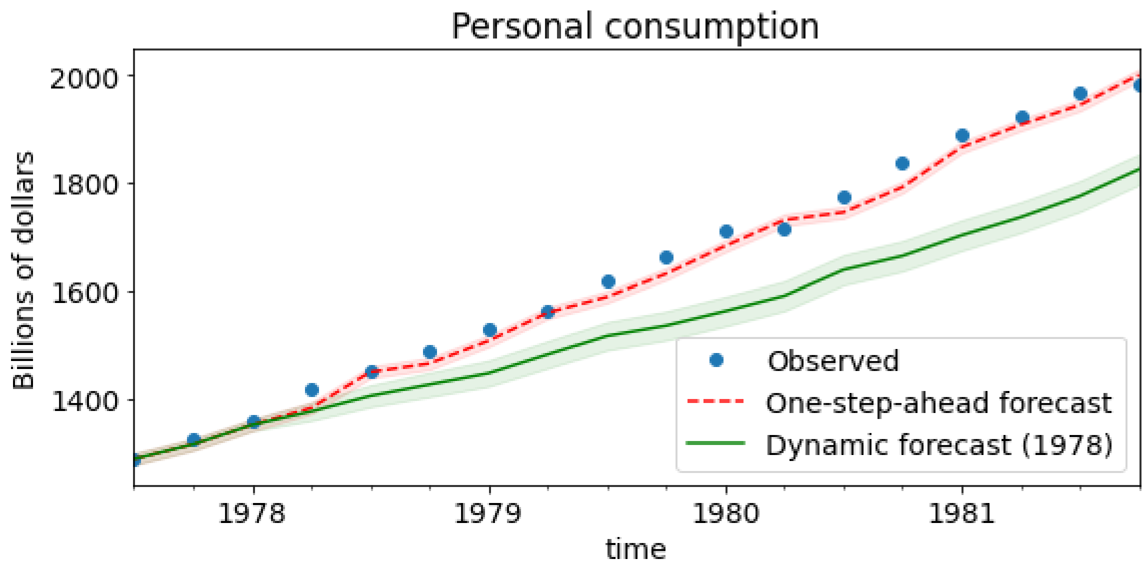
Here we perform dynamic prediction starting in the first quarter of 1978.

[13]:
```
# Dynamic predictions
predict_dy = res.get_prediction(dynamic='1978-01-01')
predict_dy_ci = predict_dy.conf_int()
```

We can graph the one-step-ahead and dynamic predictions (and the corresponding confidence intervals) to see their relative performance. Notice that up to the point where dynamic prediction begins (1978:Q1), the two are the same.

[14]:
```
# Graph
fig, ax = plt.subplots(figsize=(9,4))
npre = 4
ax.set(title='Personal consumption', xlabel='Date', ylabel='Billions of dollars')

# Plot data points
data.loc['1977-07-01':, 'consump'].plot(ax=ax, style='o', label='Observed')
```

```python
# Plot predictions
predict.predicted_mean.loc['1977-07-01':].plot(ax=ax, style='r--', label='One-step-ahead forecast')
ci = predict_ci.loc['1977-07-01':]
ax.fill_between(ci.index, ci.iloc[:,0], ci.iloc[:,1], color='r', alpha=0.1)
predict_dy.predicted_mean.loc['1977-07-01':].plot(ax=ax, style='g', label='Dynamic forecast (1978)')
ci = predict_dy_ci.loc['1977-07-01':]
ax.fill_between(ci.index, ci.iloc[:,0], ci.iloc[:,1], color='g', alpha=0.1)

legend = ax.legend(loc='lower right')
```

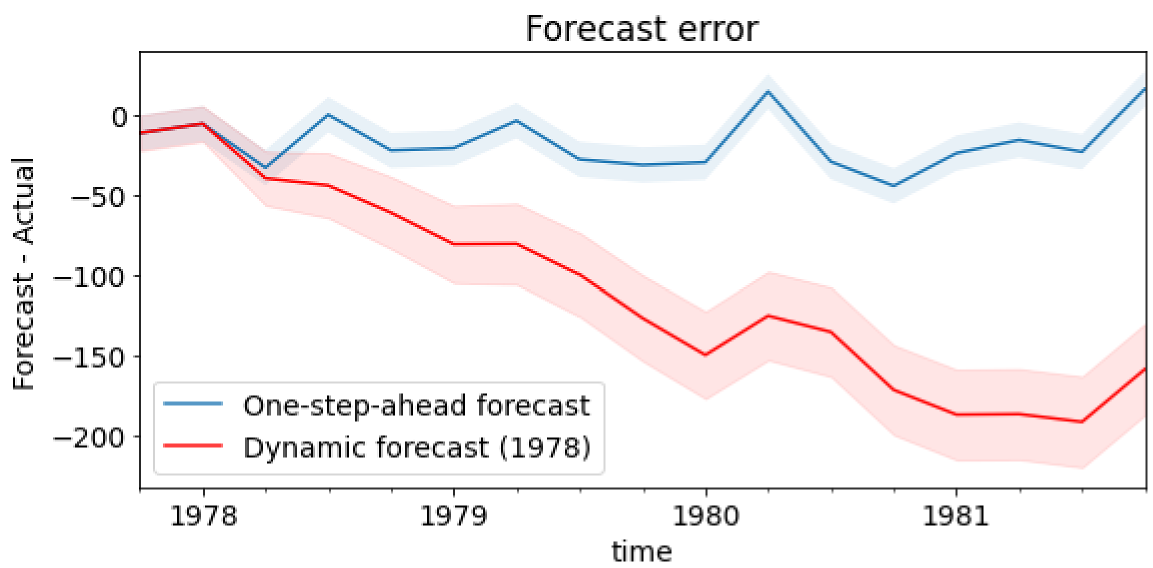


Finally, graph the prediction *error*. It is obvious that, as one would suspect, one-step-ahead prediction is considerably better.

[15]:

```python
# Prediction error
```

```python
# Graph
fig, ax = plt.subplots(figsize=(9,4))
npre = 4
ax.set(title='Forecast error', xlabel='Date', ylabel='Forecast - Actual')

# In-sample one-step-ahead predictions and 95% confidence intervals
predict_error = predict.predicted_mean - endog
predict_error.loc['1977-10-01':].plot(ax=ax, label='One-step-ahead forecast')
ci = predict_ci.loc['1977-10-01':].copy()
ci.iloc[:,0] -= endog.loc['1977-10-01':]
ci.iloc[:,1] -= endog.loc['1977-10-01':]
ax.fill_between(ci.index, ci.iloc[:,0], ci.iloc[:,1], alpha=0.1)

# Dynamic predictions and 95% confidence intervals
predict_dy_error = predict_dy.predicted_mean - endog
predict_dy_error.loc['1977-10-01':].plot(ax=ax, style='r', label='Dynamic forecast (1978)')
ci = predict_dy_ci.loc['1977-10-01':].copy()
ci.iloc[:,0] -= endog.loc['1977-10-01':]
ci.iloc[:,1] -= endog.loc['1977-10-01':]
ax.fill_between(ci.index, ci.iloc[:,0], ci.iloc[:,1], color='r', alpha=0.1)

legend = ax.legend(loc='lower left');
legend.get_frame().set_facecolor('w')
```


Forecast error
Forecast - Actual
0
−50
−100
−150
−200
One-step-ahead forecast
Dynamic forecast (1978)
1978
1979
1980
1981
time