TP 4 – AOS1
Time series analysis
Corrigé

In this practical work session, we will leverage the Python library Pandas to load a time series, model it and make predictions. We will then need the following imports:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

(1) Load the time series stored in the file `data/consumption.csv` with `pd.read_csv`. You can use the arguments `parse_dates` and `index_col`. Make sure your dataset is properly loaded by running the following checks :

```
assert(ts.shape == (164, 1))
assert(type(ts.index) is pd.core.indexes.datetimes.DatetimeIndex)
```

To avoid some future warnings you can also set the frequency of the time series by running

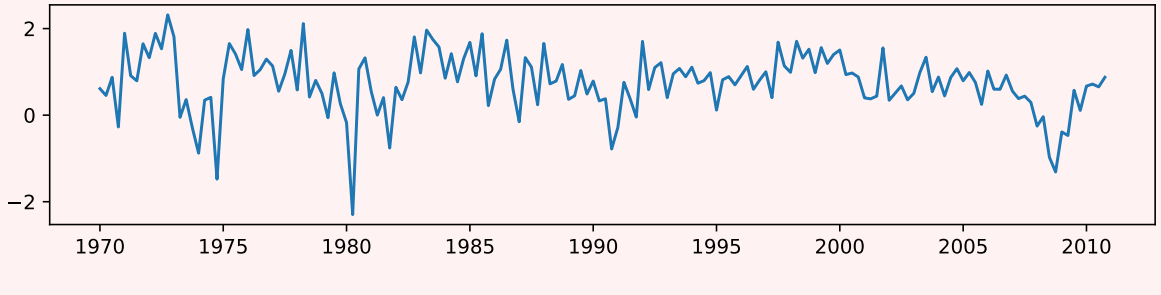
```
ts.index.freq = "QS"
```

Here the frequency is set to quarter start (`QS`) meaning that the time series is a quarterly one starting at january.

```
In [1]: ts = pd.read_csv("data/consumption.csv", parse_dates=True, index_col=0)
        ts.index.freq = "QS"
        assert(ts.shape == (164, 1))
        assert(type(ts.index) is pd.core.indexes.datetimes.DatetimeIndex)
```

(2) Plot the time series. Is the time series showing a trend, does it look stationary ?

```
In [2]: plt.plot(ts)
```



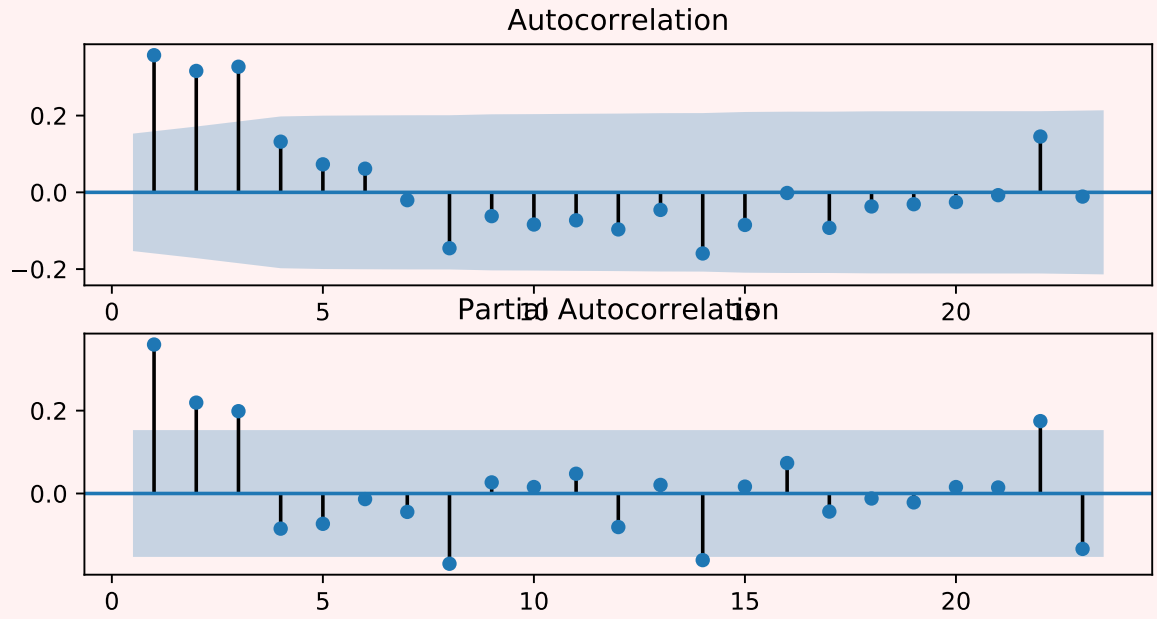
(3) Test the stationarity of the time series by applying an Augmented Dickey-Fuller statistical test (available in the `statsmodels.tsa.stattools` module). What is the p-value ? Is the time series stationary ?

```
In [3]: from statsmodels.tsa.stattools import adfuller
        test = adfuller(ts.V1, autolag='AIC')
        pvalue = test[1]
        print(pvalue)
Out [3]: 0.0005165365491351535
```

The p-value is less than 10^{-3}. The null hypothesis (a unit root is present) is rejected with strong confidence. The time series is then stationary.

(4) Plot the ACF and PACF of the time series. You can use the functions `plot_acf` and `plot_pacf` from the module `statsmodels.graphics.tsaplots`. What kind of models might explain the observed time series ?

```
In [4]: from statsmodels.graphics.tsaplots import plot_pacf
        from statsmodels.graphics.tsaplots import plot_acf
        fig, ax = plt.subplots(2, figsize=(8, 4))
        plot_acf(ts, zero=False, ax=ax[0])
        plot_pacf(ts, zero=False, ax=ax[1])
Out [4]: <Figure size 800x400 with 2 Axes>
In [5]: plt.show()
```



In both cases the first three lags are significant. This leads us to propose the models AR(3) and MA(3) models.

(5) Split the time series into two parts. The first part `ts_train` will be used to train the model and the second part `ts_test` will be used to test it.

```
In [6]:  ts_train = ts[:"1997-01-01"]
         ts_test = ts["1997-01-01":]
```

To build an ARIMA model, we will use the more general `SARIMAX` class that can also handle seasonality and exogenous variables.

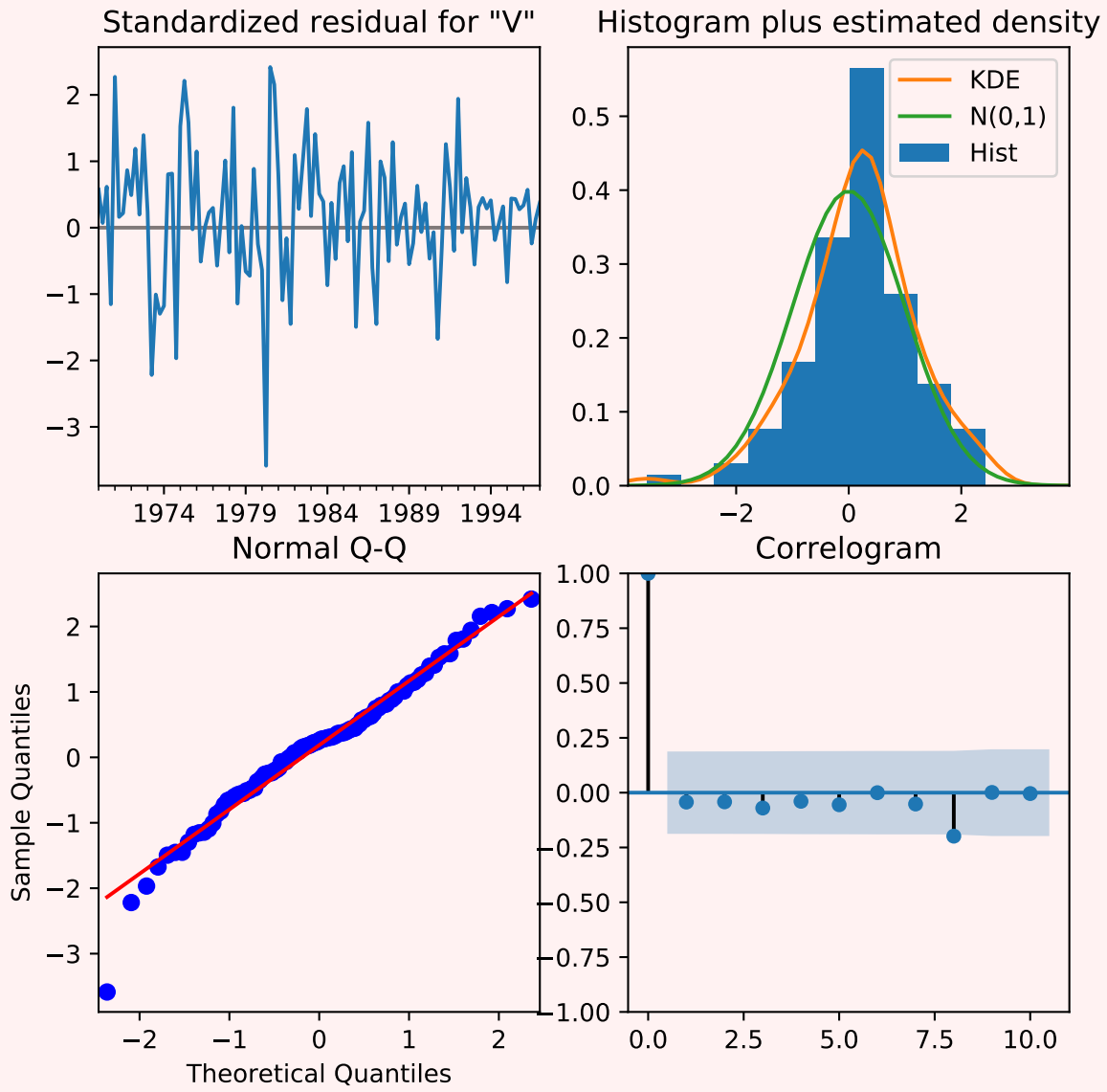
```
from statsmodels.tsa.statespace.sarimax import SARIMAX as sarimax
```

(6) Fit an ARIMA model on the train set with the parameters identified at question 4.

```
In [7]:  model = sarimax(ts_train, order=(3, 0, 0))
         model_fit = model.fit()
```

(7) Use the `plot_diagnostics` method on the fitted model to display various diagnostics plots. Check that the residuals are normally distributed and not correlated.

```
In [8]: model_fit.plot_diagnostics()
```



Looking at the histogram of residuals and a normal Q-Q-plot, we see that the residuals seem to be normally distributed. Plus, the correlogram shows that they are not correlated.

Now that the parameters of the model have been learned, we want to make predictions on the test set. To that end, we must first define the same model on the whole time series and give it the learned parameters. This can be done by the following code:

```
# Defining same model on whole time series
model_test = sarimax(ts, order=(p, d, q))

# Use the previouly learned parameters instead of learning them
model_test_res = model_test.filter(model_fit.params)
```

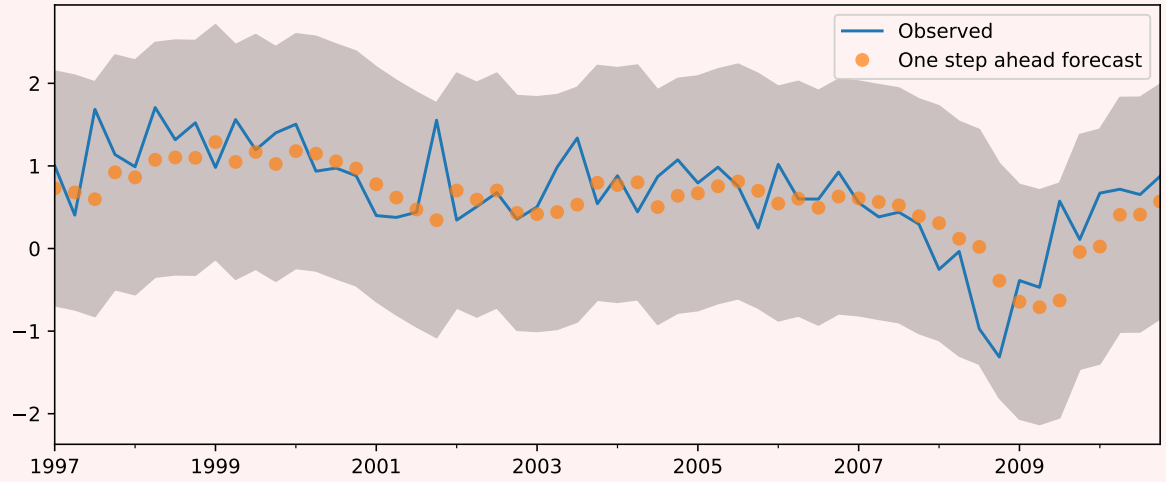
(8) Use `model_test_res` and its `get_predictions` method with the correct start argument

to make one step ahead forecast on the test set.

```
In [9]:  model_test = sarimax(ts, order=(3, 0, 0))
         model_test_res = model_test.filter(model_fit.params)
         pred = model_test_res.get_prediction(start="1997-01-01")
         pred_ci = pred.conf_int()
```

(9) Plot the observed test time series along with one step ahead predictions taken from `model_test_res`.

```
In [10]: ax = ts_test.plot(y="V1", label="Observed", figsize=(10, 4))
         pred.predicted_mean.plot(ax=ax, label="One step ahead forecast", alpha=0.7, linestyle='',
         ↪  marker='o')
         ax.fill_between(pred_ci.index, pred_ci['lower V1'], pred_ci['upper V1'], color="k",
         ↪  alpha=0.2)
         plt.legend()
```



(10) What is the mean square error of one step ahead predictions on the test set defined by

$$\text{MSE} = \sum_{h=1}^{H} \left(y_{T+h} - \widehat{y}_{T+h:T+h-1}\right)^2.$$

What is the error if we choose another potential model ?

```
In [11]: mse = np.mean((pred.predicted_mean - ts_test.V1)**2)
```

The mean square error of one step ahead prediction on the test set is : $0,1977168$.
The other potential model was MA(3).

```
In [12]: model = sarimax(ts_train, order=(0, 0, 3))
         model_fit = model.fit()
         model_test = sarimax(ts, order=(0, 0, 3))
         model_test_res = model_test.filter(model_fit.params)
         pred = model_test_res.get_prediction(start="1997-01-01")
         mse = np.mean((pred.predicted_mean - ts_test.V1)**2)
```

The MSE is $0,2845882$.

Problem

The dataset in the file `data/debitcards.csv` stores the monthly retail debit card usage in Iceland (million ISK) from january 2000 to december 2012. Give an estimate of the cumulated debit card usage during the 4 first months of 2013.

Particular care should be taken with the frequency of time series, its nature, the choice of the model and its validation.