Modeling and Analysis for Complex systems
Forecasting US Retail Food and Beverages

Emanuele Giuseppe Maria Maita 1000008891

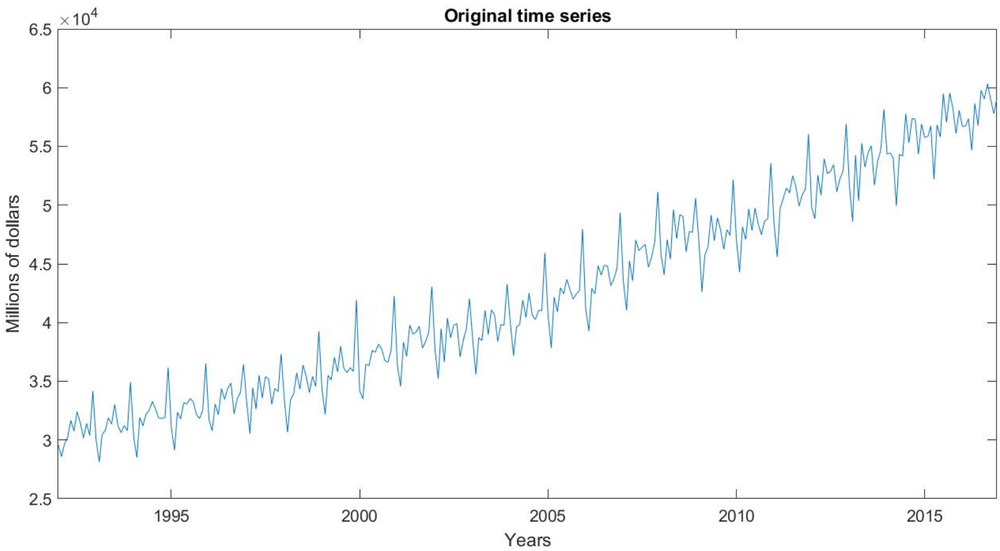
1 Static modeling

1.1 Evaluating stationarity

A time series is a series of data points indexed in time order. More commonly, and as for the time series object of this report, they are sequences taken at successively equally spaced points in time. It can be modeled as the realization of a stochastic process $\{y(t)\}$, i.e. a sequence of random variables ordered by an index (in this case t, referred to as time).

The following is the 300 samples-long monthly food and beverage retail time series in million of dollars recorded by the US Census Bureau from 1992 to 2016:

```
>> disp(head(Food))
        Time          Data
     ___________     _____
     01-Jan-1992     29589
     01-Feb-1992     28570
     01-Mar-1992     29682
     01-Apr-1992     30228
     01-May-1992     31677
     01-Jun-1992     30769
     01-Jul-1992     32402
     01-Aug-1992     31469
>> disp(numel(Food.Data))
   300
>> plot(Food.Time, Food.Data)
>> title("Original time series");
>> xlabel("Years");
>> ylabel("Millions of dollars");
```



A stationary time series is defined as a process which has constant mean function

$$m(t) = \mathbb{E}[y(t)] = m \ \forall t$$

i.e. the function of time around which the samples of $\{y(t)\}$ fluctuate, constant variance

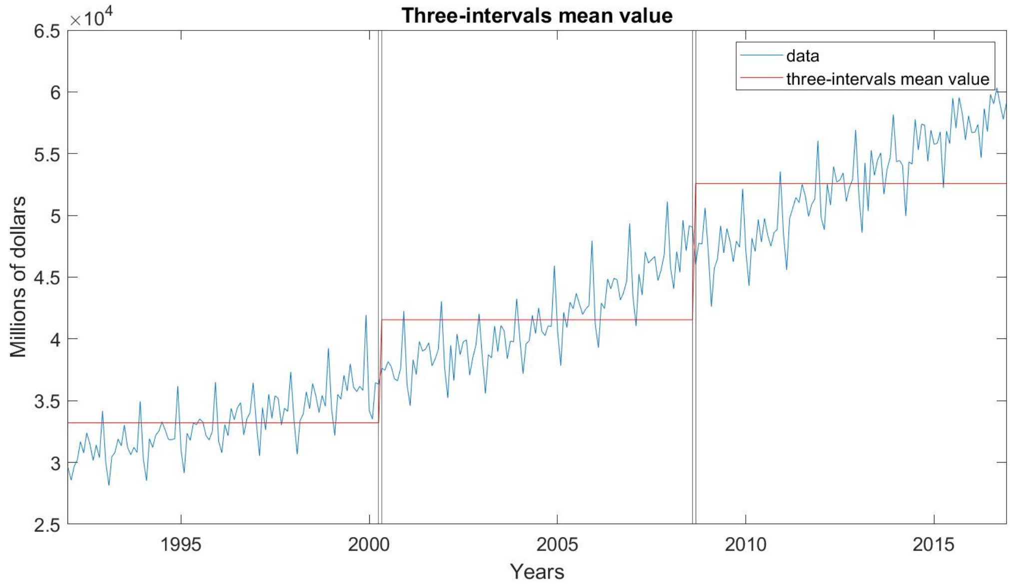
$$\mathrm{Var}(t) = \mathbb{E}[(y(t) - m)^2] = \gamma(0) \ \forall t$$

which is a measure of spread from the mean, and autocorrelation function (normalized covariance function), which captures the mutual dependence of two variables $y(t_1), y(t_2)$ extracted from the process, dependent on the lag $\tau = t_2 - t_1$ only:

$$\begin{aligned} \rho(t_1, t_2) &= \frac{\mathrm{Cov}(t_1, t_2)}{\sqrt{Var(t_1)}\sqrt{Var(t_2)}} = \frac{\mathbb{E}[(y(t_1) - m)(y(t_2) - m)]}{\sqrt{\gamma(0)}\sqrt{\gamma(0)}} \\ = \quad \rho(\tau) &= \frac{\mathbb{E}[(y(t) - m)(y(t+\tau) - m)]}{\gamma(0)} = \frac{\gamma(\tau)}{\gamma(0)} \end{aligned}$$

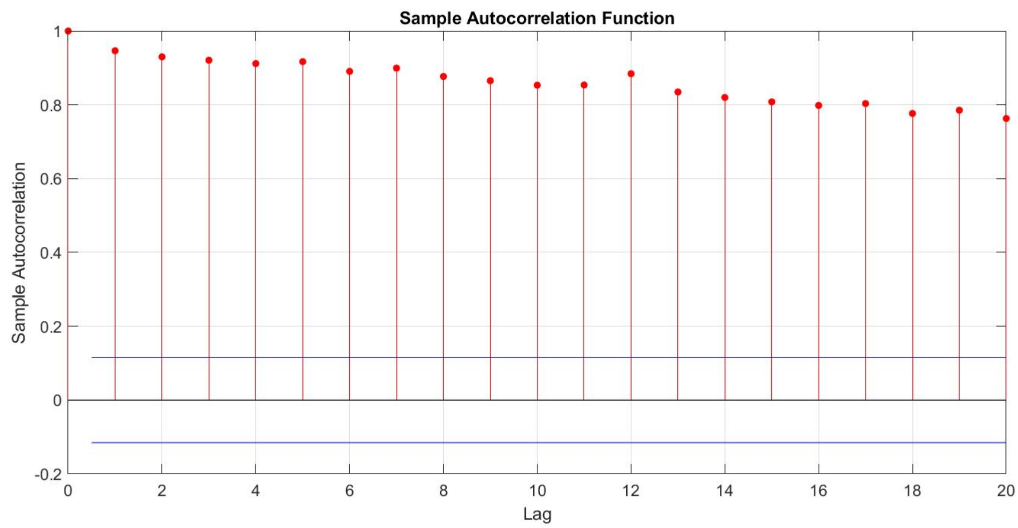
`Food` is clearly non-stationary: the mean changes over time. The average value over all 300 samples is about 42 millions.

```
>> mean_1 = mean(Food.Data(1:100));
>> mean_2 = mean(Food.Data(101:200));
>> mean_3 = mean(Food.Data(201:300));
>> mean_three_intervals = [repmat(mean_1,100), repmat(mean_2,100), repmat(mean_3,100)];
>> plot(Food.Time, Food.Data,Food.Time, mean_three_intervals, "r")
>> xline(Food.Time(100)); xline(Food.Time(101)); xline(Food.Time(200)); xline(Food.Time(201));
>> legend("data", "three-intervals mean value")
>> title("Three-intervals mean value")
>> xlabel("Years")
>> ylabel("Millions of dollars")
>> format bank; disp(mean(Food.Data));
     42446.46
```



Moreover, stationary time series usually have “short memory”, i.e. autocorrelation values tend to decrease fast, differently to what it’s possible to see in `Food` data:

```
autocorr(Food.Data);
```



1.2 Training/test split

In the following, the data will be split in training set (270 observations), where a series of models (both static and dynamic) will be fitted on, and test set (30 data points), used to evaluate their predictive power. In particular the normalized mean square error[1]

$$\text{NMSE} = 1 - \left(\frac{\|y(t) - \hat{y}(t)\|}{\|y(t) - m\|} \right)^2$$

between test data $y(t)^{\text{test}}$ and forecasted data $\hat{y}(t)^{\text{test}}$ will be the metric used to choose the best models.

```
>> train_data = Food.Data(1:270);
>> train_time = Food.Time(1:270);
>> test_data =  Food.Data(271:end);
>> test_time = Food.Time(271:end);
>> all_data = {train_data, train_time, test_data, test_time};
```

1.3 Trend

If deterministic, the non-stationary component (in the mean-sense) of the time series can often be deal with via the fitting and, then, the removal of a d-degree polynomial. i.e. a model which consider the series as

$$\hat{y}(t) = \beta_0 + \beta_1 t + \beta_2 t^2 + \cdots + \beta_d t^d$$

where $\beta_0, \beta_1, \cdots, \beta_d$ are the real coefficients which minimize the mean square error from the training series:

$$\min_{\boldsymbol{\beta}} \frac{1}{270} \sum_{t=1}^{270} (y(t) - \hat{y}(t))^2$$

```
function [train_pred, test_pred, params] = FitAndForecastTrend(all_data, degree)
train_data = all_data{1};
train_time = all_data{2};
test_data = all_data{3};
test_time = all_data{4};
% ======== initialization ===============================================
R = zeros(degree+1,degree+1);
B = zeros(degree+1,1);
train_pred = zeros(270,1);
test_pred = zeros(30,1);
% ======== training =====================================================
for t=1:270
    fi = power( [1 repmat(t, 1, degree)], 0:degree)';
    R = R + fi*fi';
    B = B + fi*train_data(t);
end
params = R\B;
```

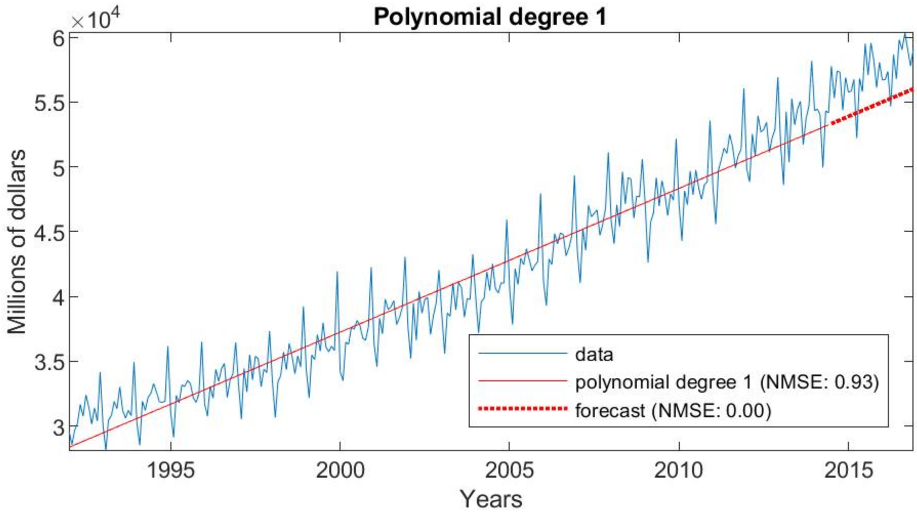
(... continue ...)

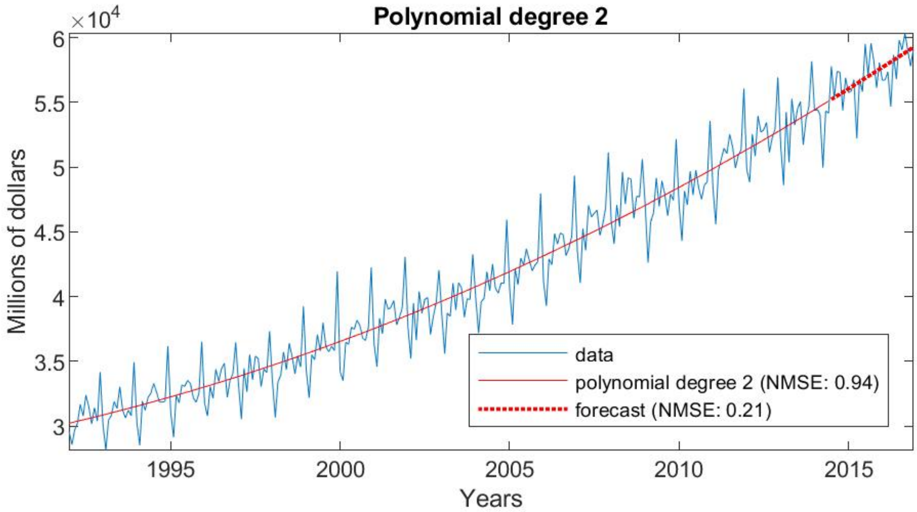
[1] in the actual implementation in this report, all negative values of the NMSE are equalled to zero.

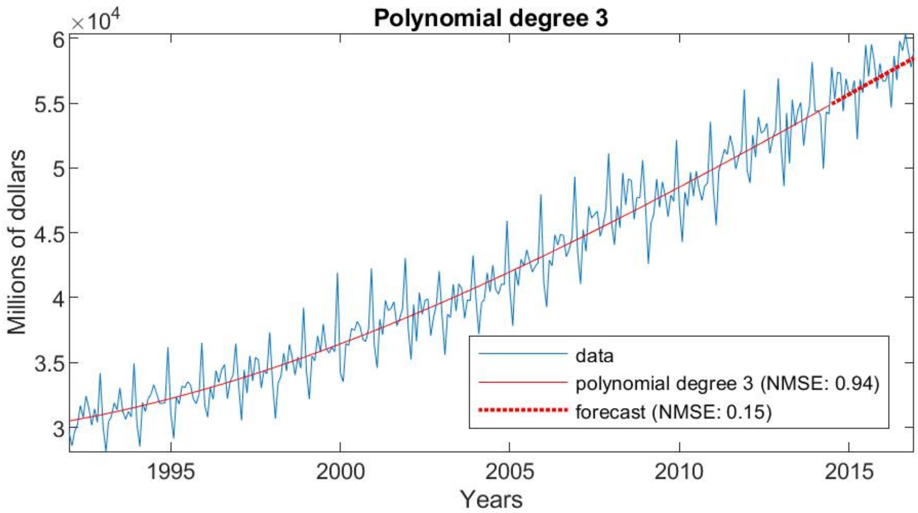
```
% ======== predict training data ==========================================
for t=1:270
    fi = power( [1 repmat(t, 1, degree)], 0:degree);
    train_pred(t) = fi*params;
end
% ======== predict test data ==============================================
for t=271:300
    fi = power( [1 repmat(t, 1, degree)], 0:degree);
    test_pred(t-270) = fi*params;
end
% ======== compute error train set ========================================
train_nmse = 1- min([1 ...
                   power( ...
                     norm( train_data - train_pred) / ...
                     norm( train_data - mean(train_data)) ...
                         ,2) ...
                ]);
% ======== compute error test set =========================================
test_nmse = 1- min([1 ...
                   power( ...
                     norm( test_data - test_pred) / ...
                     norm( test_data - mean(test_data)) ...
                         ,2) ...
                ]);
% ======== plot results ===================================================
plot([train_time; test_time], [train_data; test_data])
hold on
plot(train_time, train_pred,"r")
plot(test_time, test_pred, "r:",'LineWidth',1.8)
legend("data", ...
    sprintf('polynomial degree %d (NMSE: %.2f)', degree, train_nmse), ...
    sprintf("forecast (NMSE: %.2f)", test_nmse), ...
    'Location', "southeast")
title(sprintf("Polynomial degree %d", degree))
xlabel("Years")
ylabel("Millions of dollars")
hold off
```

Between polynomials of degrees $d = 1, 2, 3$, the best one according to forecast NMSE has $d = 2$.

```
>> figure()
>> for degree = 1:3, ...
    subplot(3,1,degree), ...
    FitAndForecastTrend(all_data, degree);
end
```

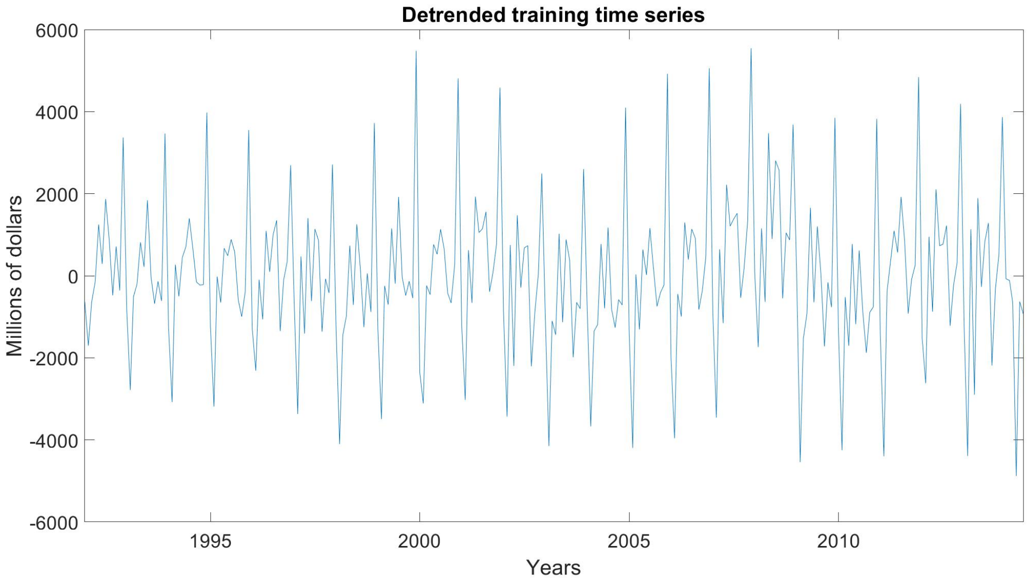

Polynomial degree 1
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
polynomial degree 1 (NMSE: 0.93)
forecast (NMSE: 0.00)


Polynomial degree 2
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
polynomial degree 2 (NMSE: 0.94)
forecast (NMSE: 0.21)


Polynomial degree 3
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
polynomial degree 3 (NMSE: 0.94)
forecast (NMSE: 0.15)

Subsequently, the detrended training time series dt_train_data is plotted:

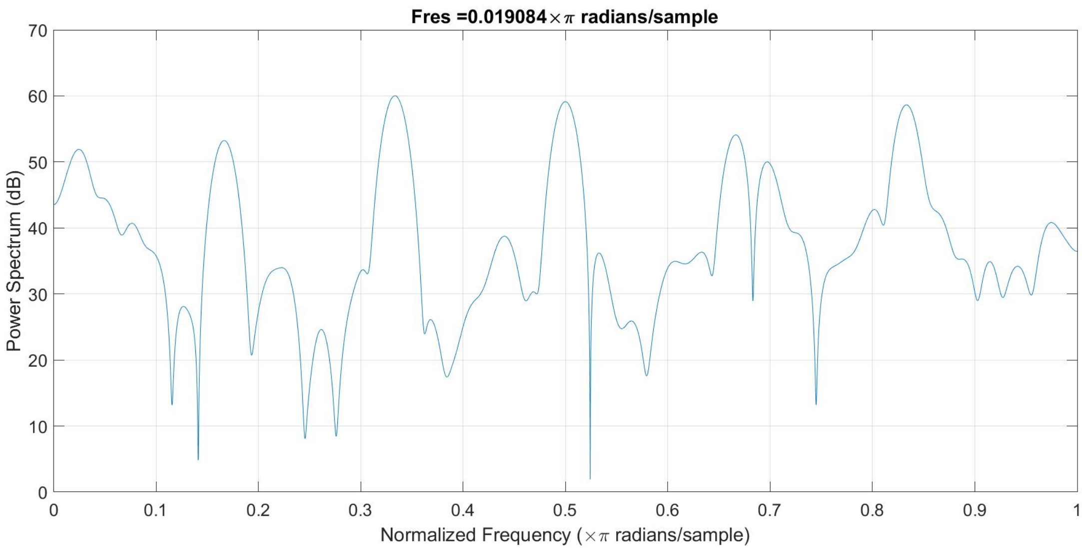
```
>> [train_trend, test_trend, params] = FitAndForecastTrend( ...
    all_data,2);
>> disp(params)
     30170.22
        50.70
         0.15
>> dt_train_data = train_data - train_trend;
>> dt_test_data = test_data - test_trend;
>> plot(train_time, dt_train_data);
>> title("Detrended time series");
>> xlabel("Years");
>> ylabel("Millions of dollars");
```



1.4 Seasonality

dt_train_Data clearly presents seasonality, both at high frequency and low frequency, as the spectrum shows:

```
>> pspectrum(dt_train_data)
```



Seasonality can be modeled estimating h sinusoidal components of period T, i.e. with a model of the type:

$$\hat{y}(t) = w_0 + w_{1c}\cos(\omega t) + w_{1s}\sin(\omega_i t) + \cdots + w_{hc}\cos(h\omega t) + w_{hs}\sin(h\omega t)$$
$$\text{where } [w_0\ w_{1c}\ w_{1s}\ \cdots\ w_{hc}\ w_{hc}] = \underset{\boldsymbol{w}\in\mathbb{R}^{2h+1}}{\operatorname{argmin}} \frac{1}{270}\sum_{t=1}^{270}(y(t)-\hat{y}(t))^2$$
$$\text{and } \omega = \frac{2\pi}{T}$$

To catch both long and short term seasonality, different periods (value of T) are tried, in particular from 1 to 10 years (i.e. $T = 12, 24, \cdots, 120$), with a number of sinusoidal components per period (h) ranging from 1 to 12. The seasonal model will be fitted on detrended data (`dt_train_data`) and then the estimated parameters are used to forecast test data, by summing the predicted seasonality with the priorly estimated 2-degree predicted trend.

```
function [train_pred, test_pred, train_seas_comp, test_seas_comp, ...
                    params, train_nmse, test_nmse] = FitAndForecastSeasons(T, n_components, train_data, ...
                                                                        test_data,trend_comp)
% =========================================================================
train_trend = trend_comp{1};
test_trend = trend_comp{2};
dt_train_data = train_data - train_trend;
% ======== initialization =================================================
R = zeros(2*n_components+1,2*n_components+1);
B = zeros(2*n_components+1,1);
train_seas_comp = zeros(270,1);
test_seas_comp = zeros(30,1);
% ======== training =======================================================
for t=1:270
    fi = [1];
    for s=1:n_components
        fi = [fi cos(2*s*pi/T*t) sin(2*s*pi/T*t)];
    end
    R = R + fi'*fi;
    B = B + fi'*dt_train_data(t);
end
params = R\B;
% ======== predict training data ==========================================
for t=1:270
    fi = [1];
    for s=1:n_components
        fi = [fi cos(2*s*pi/T*t) sin(2*s*pi/T*t)];
    end
    train_seas_comp(t) = fi*params;
end
train_pred = train_seas_comp + train_trend;
% ======== predict test data ==============================================
for t=271:300
    fi = [1];
    for s=1:n_components
        fi = [fi cos(2*s*pi/T*t) sin(2*s*pi/T*t)];
    end
    test_seas_comp(t-270) = fi*params;
end
test_pred = test_seas_comp + test_trend;
```

(... continue ...)

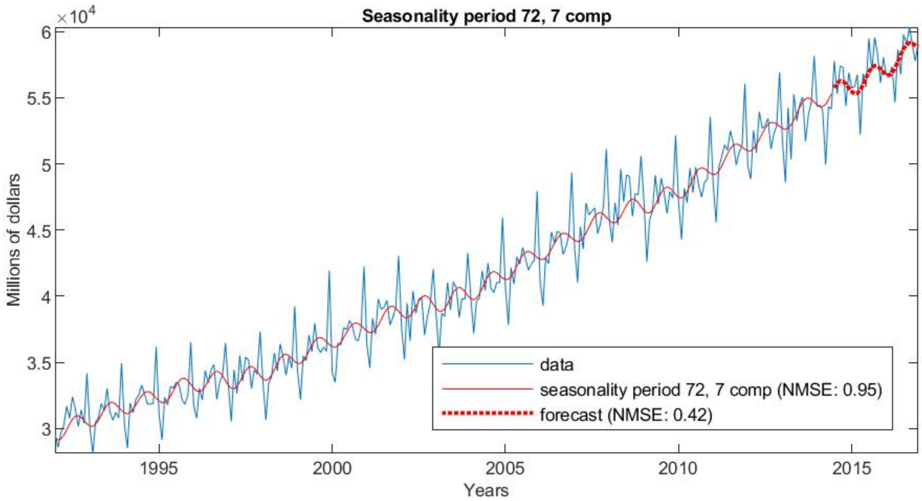
```
            % ======== compute error train set ========================================
            train_nmse = 1- min([1 ...
                               power( ...
                                 norm( train_data - train_pred) / ...
                                 norm( train_data - mean(train_data)) ...
                                     ,2) ...
                            ]);
            % ======== compute error test set =========================================
            test_nmse = 1- min([1 ...
                               power( ...
                                 norm( test_data - test_pred) / ...
                                 norm( test_data - mean(test_data)) ...
                                     ,2) ...
                            ]);



function [train_trendAndSeas, test_trendAndSeas, train_Seas, ...
                       test_Seas, Params] = FindBestSeasons(T_sequence, ...
                                             n_components_sequence, all_data, trend_comp)
% ==========================================================================
train_data = all_data{1};
train_time = all_data{2};
test_data = all_data{3};
test_time = all_data{4};
% ======== initialization ({:,4} position is the best test_nmse) =========
best_models = cell(6,9);
best_models(1:6,4) = num2cell(repmat(-1000, 6,1));
for T = T_sequence
    for n_components = n_components_sequence
        try
            % ======== fit model ==========================================
            [train_pred, test_pred, train_seas_comp, test_seas_comp, ...
                            params, train_nmse, test_nmse] = FitAndForecastSeasons(T, n_components, train_data, ...
                                                                       test_data,trend_comp);
            % === if better, add to best 6 ranking in respective position =
            if test_nmse>best_models{1,4}
                best_models = [ {train_pred, test_pred, train_nmse, test_nmse, ...
                                 T, n_components, train_seas_comp, test_seas_comp,params}; ...
                                 best_models(1:5, 1:9) ];
            elseif test_nmse>best_models{2,4}
                best_models = [ best_models(1, 1:9); {train_pred, test_pred, train_nmse, ...
                                test_nmse, T, n_components, train_seas_comp, test_seas_comp, params}; ...
                                best_models(2:5, 1:9) ];
            elseif test_nmse>best_models{3,4}
                best_models = [ best_models(1:2, 1:9); {train_pred, test_pred, train_nmse, ...
                                test_nmse, T, n_components, train_seas_comp, test_seas_comp, params}; ...
                                best_models(3:5, 1:9) ];
            elseif test_nmse>best_models{4,4}
               best_models = [ best_models(1:3, 1:9); {train_pred, test_pred, train_nmse, test_nmse, ...
                               T, n_components, train_seas_comp, test_seas_comp, params}; ...
                               best_models(4:5, 1:9) ];
            elseif test_nmse>best_models{5,4}
               best_models = [ best_models(1:4, 1:9); {train_pred, test_pred, train_nmse, test_nmse, ...
                               T, n_components, train_seas_comp, test_seas_comp, params}; ...
                               best_models(5, 1:9) ];
            elseif test_nmse>best_models{6,4}
               best_models = [ best_models(1:5, 1:9); {train_pred, test_pred, train_nmse, test_nmse, ...
                               T, n_components, train_seas_comp, test_seas_comp, params} ];
            end
        catch
            warning('Failed Seasonality estimate with period %d and number of components %d',T,n_components);
        end
    end
end
```

(... continue ...)

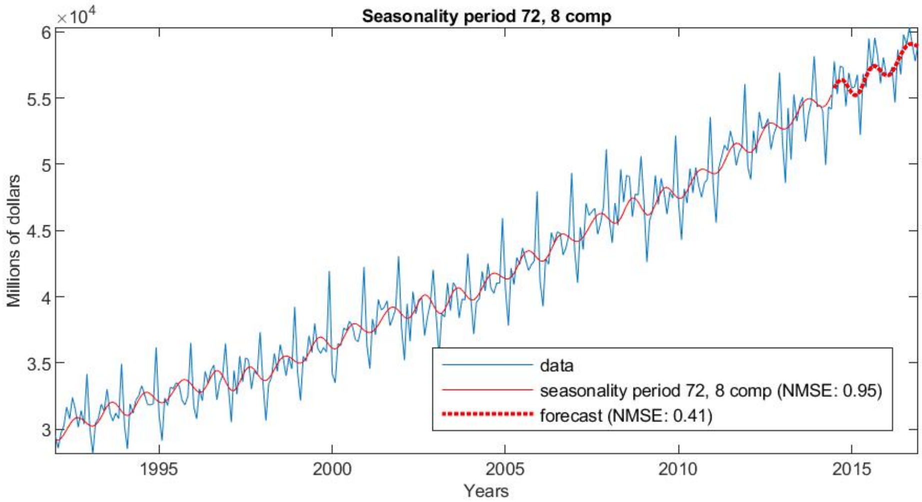
```
% ======== plot best models (2x (3x1) plots) ==============================
for model_index=1:6
    train_pred = best_models{model_index,1};
    test_pred = best_models{model_index, 2};
    train_nmse = best_models{model_index, 3};
    test_nmse = best_models{model_index, 4};
    T = best_models{model_index, 5};
    n_components = best_models{model_index, 6};
    train_seas_comp = best_models{model_index, 7};
    test_seas_comp = best_models{model_index, 8};
    params = best_models{model_index, 9};
    if model_index < 4
        if model_index==1
            % ========== save for output the first model only =============
            train_trendAndSeas = train_pred;
            test_trendAndSeas = test_pred;
            train_Seas = train_seas_comp;
            test_Seas = test_seas_comp;
            Params = params;
        end
        subplot(3,1,model_index)
    else
        if model_index == 4
            figure()
        end
        subplot(3,1,model_index-3)
    end
    plot([train_time; test_time], [train_data; test_data])
    hold on
    plot(train_time, train_pred, "r")
    plot(test_time, test_pred, "r:", 'LineWidth',1.8)
    hold off
    legend("data", ...
    sprintf('seasonality period %d, %d comp (NMSE: %.2f)', T, n_components, train_nmse), ...
    sprintf("forecast (NMSE: %.2f)", test_nmse), ...
    'Location', "southeast")
    title(sprintf('Seasonality period %d, %d comp', T, n_components));
    xlabel("Years");
    ylabel("Millions of dollars");
end
```

Below, the best six models are shown, in decreasing order by test NMSE. The best seasonal model has period 72 (6 years), with 7 components.

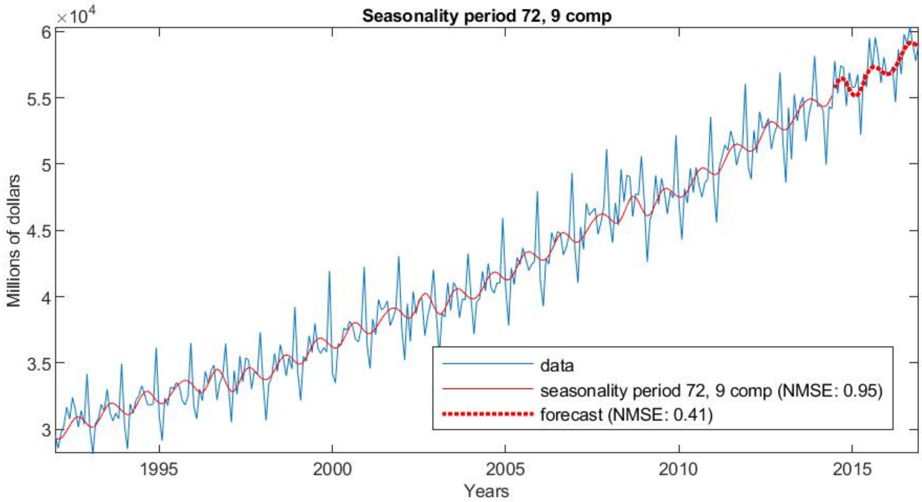
```
>> trend_comp = {train_trend, test_trend};
>> figure()
>> [train_trendAndSeas, test_trendAndSeas, ...
    train_seas, test_seas, params] = FindBestSeasons(...
          12:12:120, 1:12, all_data, trend_comp ...
);
```



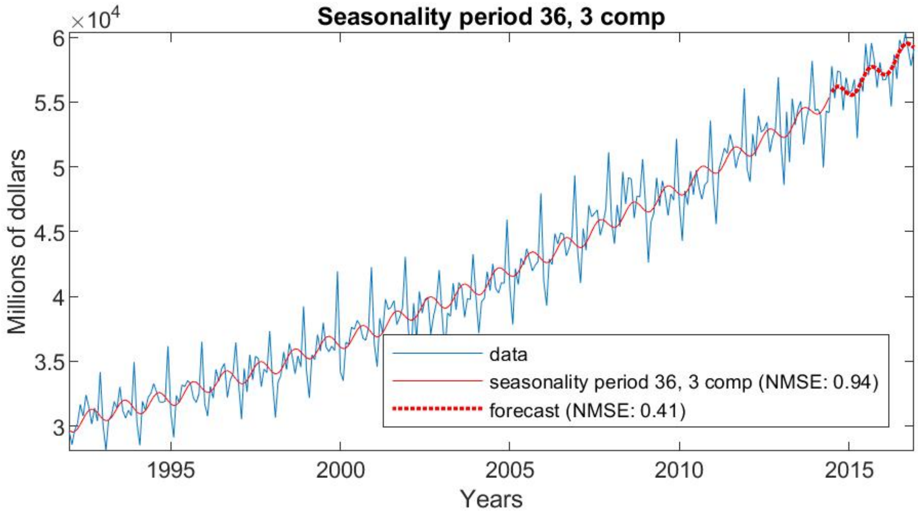
Seasonality period 72, 7 comp
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
seasonality period 72, 7 comp (NMSE: 0.95)
forecast (NMSE: 0.42)



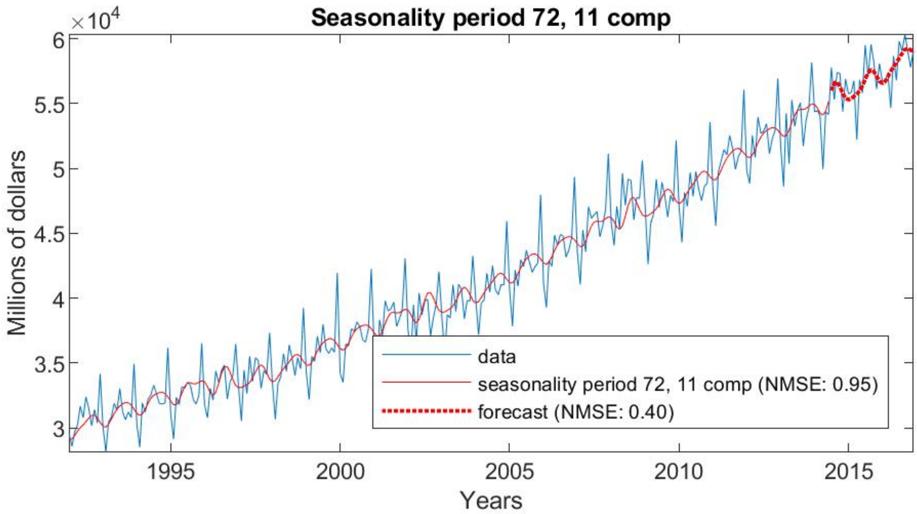
Seasonality period 72, 8 comp
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
seasonality period 72, 8 comp (NMSE: 0.95)
forecast (NMSE: 0.41)



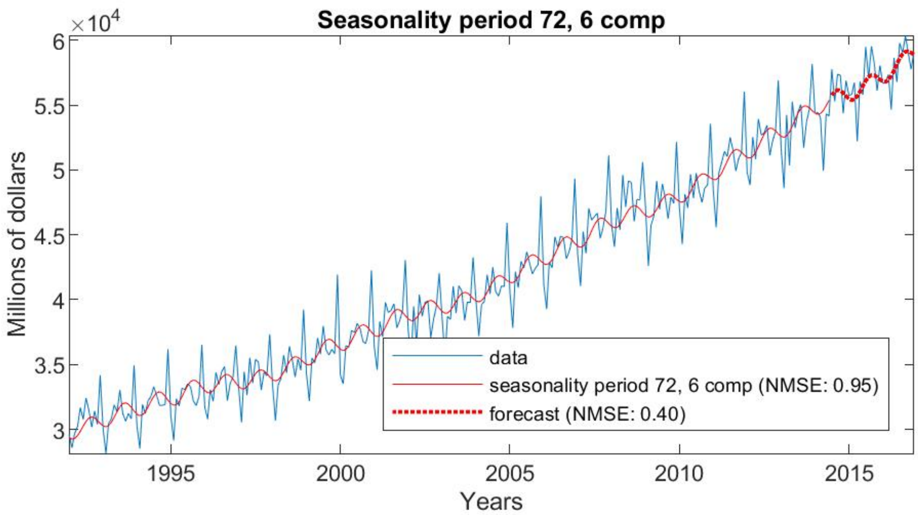
Seasonality period 72, 9 comp
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
seasonality period 72, 9 comp (NMSE: 0.95)
forecast (NMSE: 0.41)



Seasonality period 36, 3 comp
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
seasonality period 36, 3 comp (NMSE: 0.94)
forecast (NMSE: 0.41)

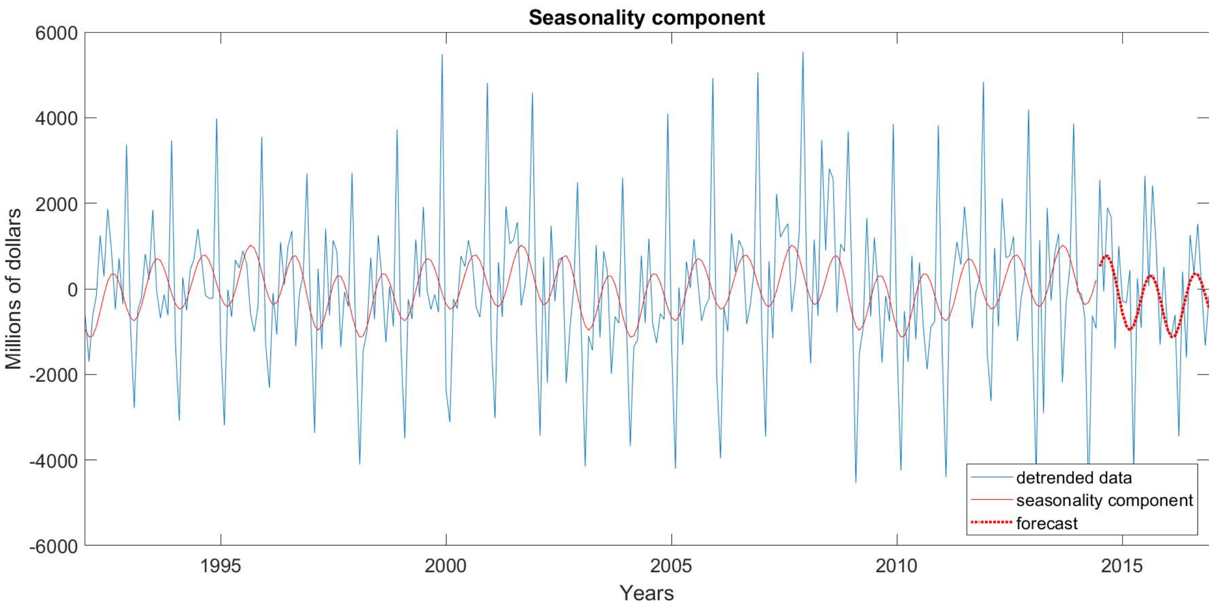


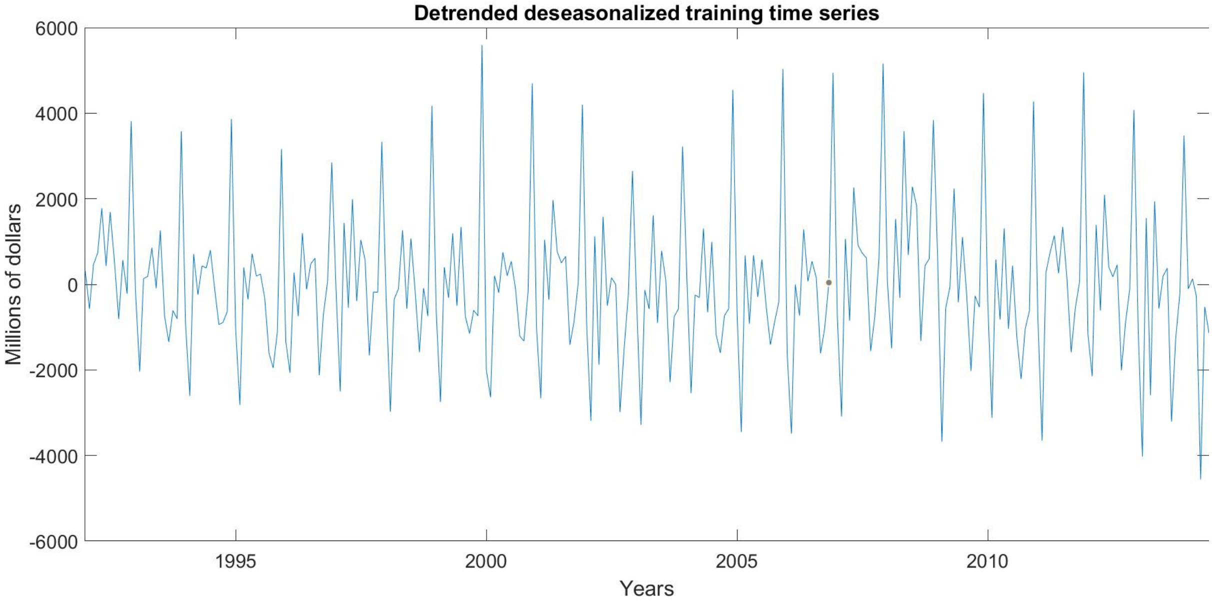
Seasonality period 72, 11 comp
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
seasonality period 72, 11 comp (NMSE: 0.95)
forecast (NMSE: 0.40)



Seasonality period 72, 6 comp
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
seasonality period 72, 6 comp (NMSE: 0.95)
forecast (NMSE: 0.40)

```
>> disp(params)
        -4.71
      -338.86
       -88.44
      -123.31
        38.37
        23.91
         3.00
         8.07
       -21.47
         6.65
        55.48
      -134.05
      -652.18
       -59.58
       -84.89
>> dtds_train_data = train_data - train_trendAndSeas;
>> figure()
>> subplot(2,1,1)
>> plot([train_time; test_time], [dt_train_data; dt_test_data])
>> hold on
>> plot(train_time, train_seas, "r")
>> plot(test_time, test_seas, "r:", 'LineWidth',1.8)
>> hold off
>> legend("detrended data", ...
    sprintf('seasonality component'), ...
    sprintf("forecast"), ...
    'Location', "southeast")
>> title('Seasonality component');
>> xlabel("Years");
>> ylabel("Millions of dollars");
>> subplot(2,1,2)
>> plot(train_time, dtds_train_data);
>> title("Detrended deseasonalized training time series");
>> xlabel("Years");
>> ylabel("Millions of dollars");
```





2 Dynamic modeling

2.1 Moving average model

It's possible to model the irregular component of the training series (`dtds_train_data`) using stationary processes, like the MA process; this model maps the series at time t as the linear combination[2] of a white noise at times t to $t-q$, where q is the order:

$$\hat{y}(t) = \xi(t) + c_1\xi(t-1) + \cdots + c_q\xi(t-q)$$

where $c_1, \cdots, c_q \in \mathbb{R}$, $\{\xi(t)\} \sim \mathrm{WN}(0, \lambda^2)$, i.e. a stationary process with $\gamma(0) = \lambda^2$, $\rho(\tau) = 0\ \forall\tau \neq 0$ which formalizes an unpredictable signal. In terms of a delay operator z, it can be written as:

$$\begin{aligned}\hat{y}(t) &= (1 + c_1 z^{-1} + \cdots + c_q z^{-q})\xi(t) \\ &= C(z)\xi(t)\end{aligned}$$

The mean value of the MA process is zero

$$\mathbb{E}[\hat{y}(t)] = m = 0$$

The variance is:

$$\mathrm{Var}[\hat{y}(t)] = \gamma(0) = (c_0^2 + \cdots + c_q^2)\lambda^2$$

and the autocorrelation function is equal to:

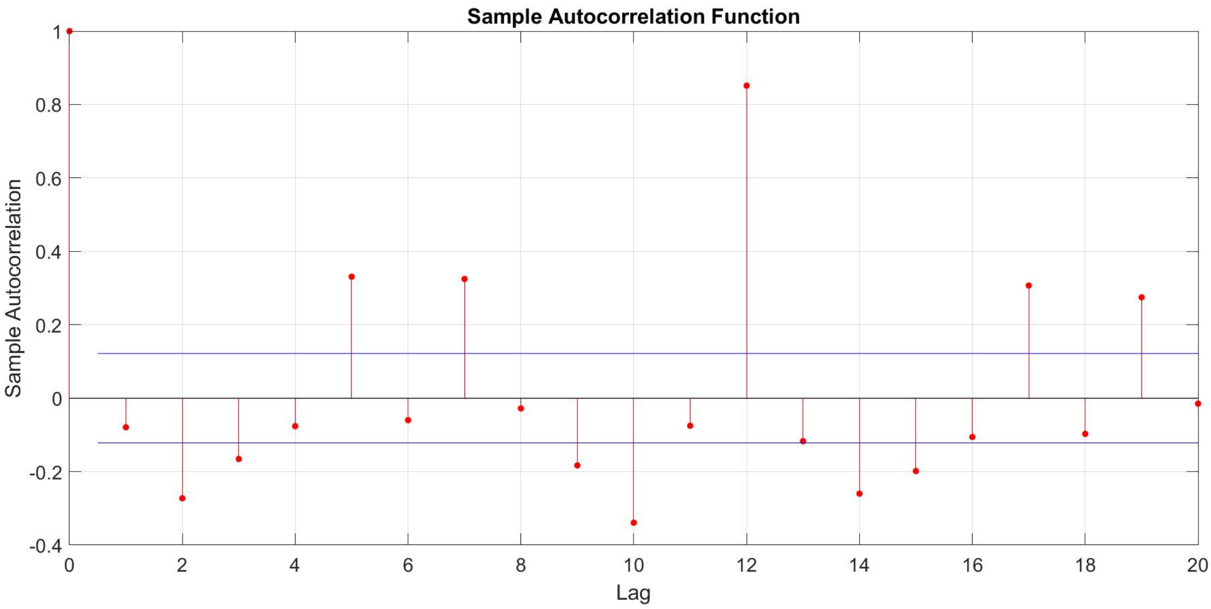
$$\rho(\tau) = \begin{cases} 1 & \tau = 0 \\ \dfrac{\left(\sum_{i=0}^{q-\tau} c_i c_{i-\tau}\right)\lambda^2}{\gamma(0)} = \dfrac{\sum_{i=0}^{q-\tau} c_i c_{i-\tau}}{\sum_{i=0}^{q} c_i^2} & \tau \leq q, \tau \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

[2] c_0 is equal to 1, as it is usually presented.

i.e. it is different from 0 solely for values of the lag less or equal than the order of the model. So, a possible way to identify the order of the MA model to be fitted on empirical series is to determine the latest lag before autocorrelation flattens to zero (i.e. it's not significantly different from 0).

The presence of residual seasonality make hard the order selection based on the autocorrelation plot.

```
>> autocorr(dtds_train_data)
```

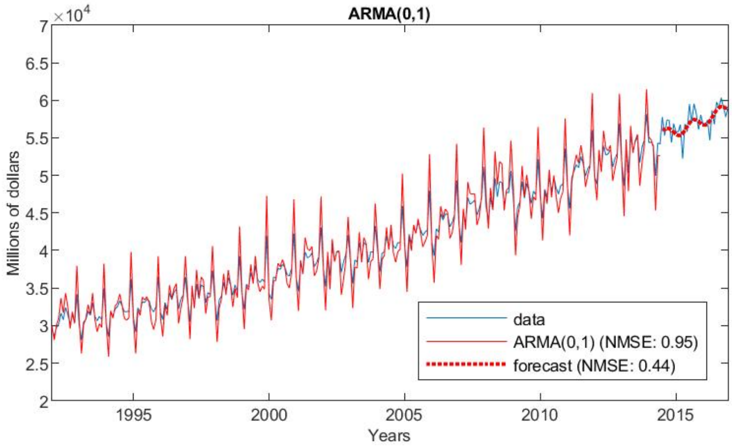


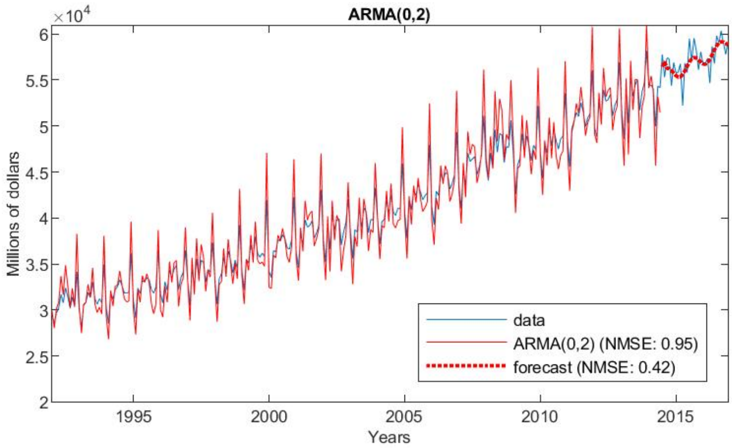
MA processes with orders from 1 to 12 will be fitted on `dtds_train_data`, and then their predictions added to priorly estimated (trend and seasonality) components for forecasting. The best six models are shown below[3].

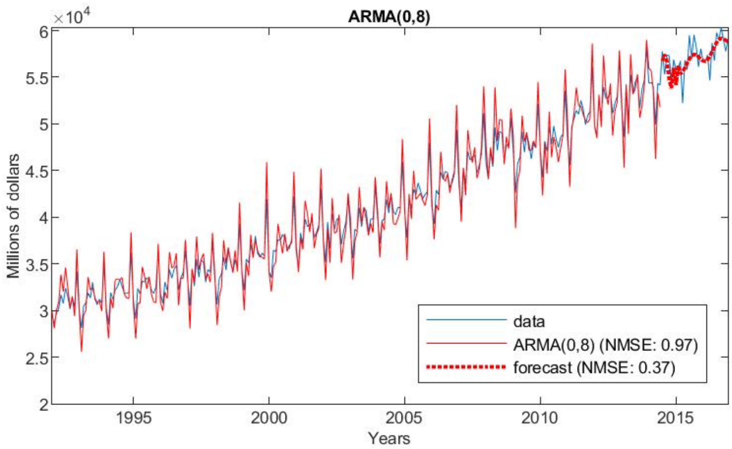
```
function [train_pred, test_pred, train_nmse, test_nmse] = FitAndForecastARMA(ar_order, ma_order, ...
                                                                    train_data, test_data, components)
% ==========================================================================
train_comp = components{1};
test_comp = components{2};
dtds_train_data = train_data - train_comp;
% ======== fit model ======================================================
model = estimate(arima(ar_order, 0, ma_order), dtds_train_data, 'Display', 'off');
residuals = infer(model, dtds_train_data);
train_pred = dtds_train_data + residuals + train_comp;
% ======== compute train error ============================================
train_nmse = 1- min([1 ...
                  power( ...
                    norm( train_data - train_pred ) / ...
                    norm( train_data - mean(train_data)) ...
                        ,2) ...
               ]);
% ======== forecast =======================================================
[test_pred, ~] = forecast(model, numel(test_data), dtds_train_data);
test_pred = test_pred + test_comp;
% ======== compute test error =============================================
test_nmse = 1- min([1 ...
                  power( ...
                    norm( test_data - test_pred) / ...
                    norm( test_data - mean(test_data)) ...
                        ,2) ...
               ]);
```

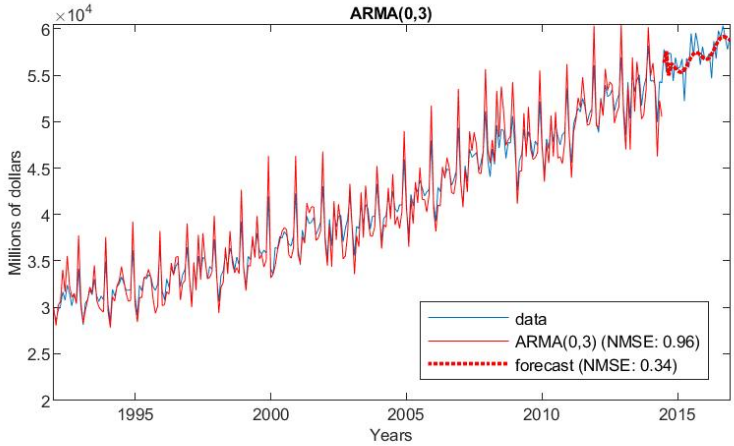
[3]the `findBestARMA` function, which works similarly to `findBestSEASONS`, can be found in the appendix.

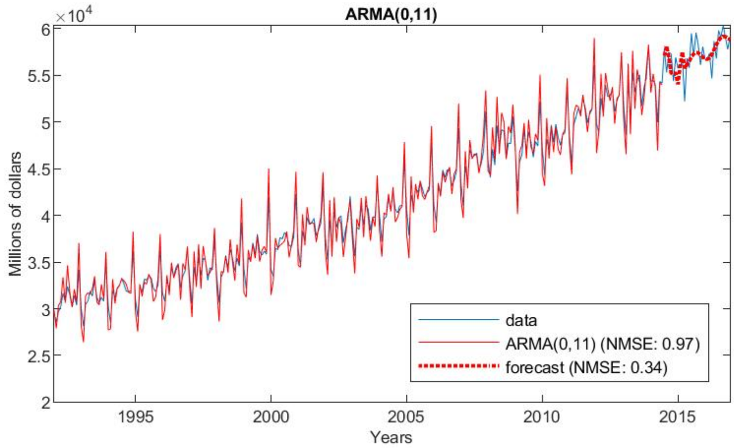
```
>> components = {train_trendAndSeas, test_trendAndSeas};
>> FindBestARMA(0, 1:12, all_data, components);
```

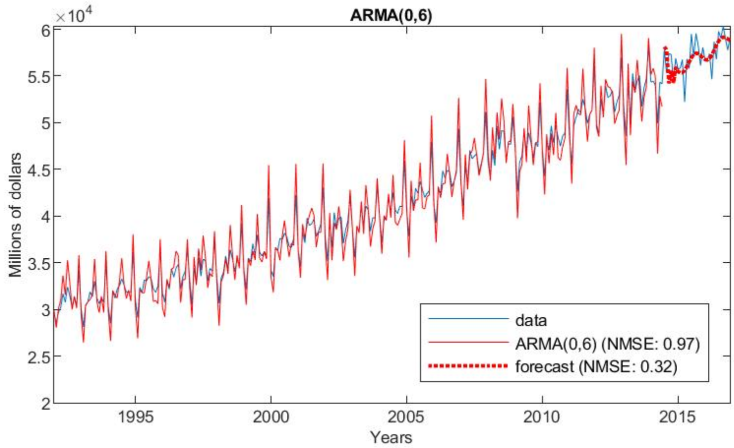












MA processes don't appear to be able to capture the irregular component of the data:

only the first model, MA(1), is able to get a slighly better performance (2% increase in test NMSE); all others contribute in no way or worsen the previous results.

2.2 Autoregressive model

An autoregressive model of order q is a stationary process defined by the linear combination of a white noise at time t and past values of the series $(y(t))$ from $t-1$ to $t-p$:

$$\begin{aligned} \hat{y}(t) &= a_1 y(t-1) + \cdots + a_p y(t-p) + \xi(t) \\ (1 - a_1 z^{-1} - \cdots - a_p z^{-p})\hat{y}(t) &= \xi(t) \\ A(t)\hat{y}(t) &= \xi(t) \end{aligned}$$

with $\{\xi(t)\} \sim \mathrm{WN}(0, \lambda^2)$.

It can be shown that the partial autocorrelation function $\alpha(k)$ of an AR process has value 0 for all lags greater than the order q. Partial autocorrelation can be found by:

- predicting $y(t)$ linearly based on $y(t-1), \cdots, y(t-k+1)$ for a given k and computing the simple errors:

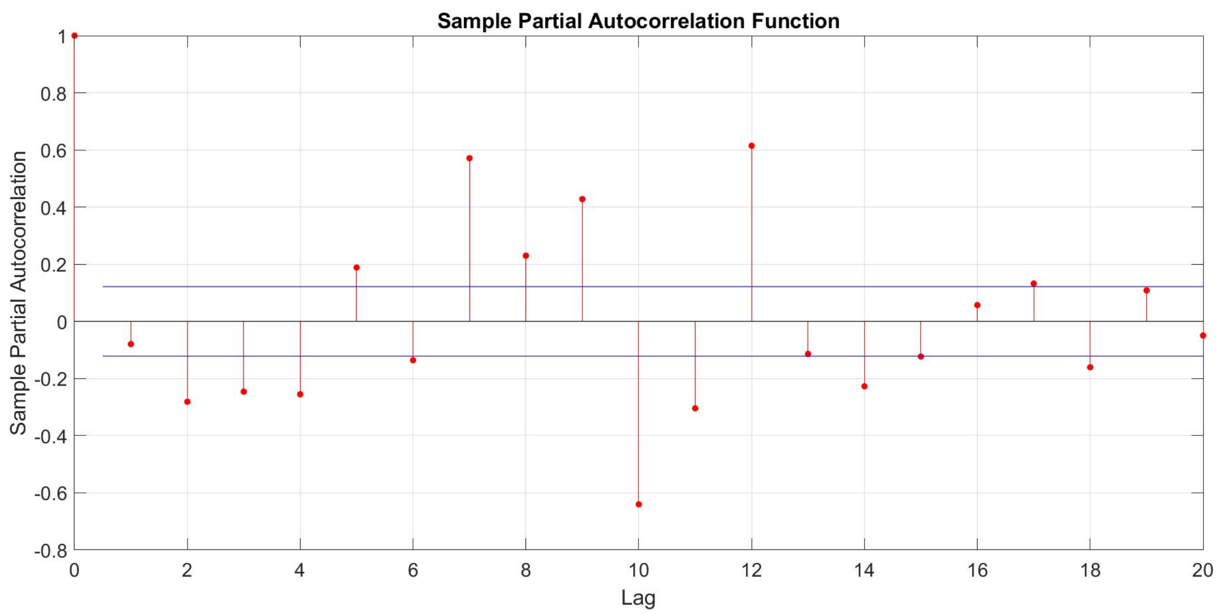
$$\begin{aligned} \hat{y}(t) &= a_1 y(t-1) + \cdots + a_{k-1} y(t-k+1) \\ \epsilon(t) &= y(t) - \hat{y}(t) \end{aligned}$$

- predicting $y(t-k)$ linearly based on $y(t-k+1), \cdots, y(t-1)$ and computing the simple errors:

$$\begin{aligned} \hat{y}(t-k) &= a_1 y(t-k+1) + \cdots + a_{k-1} y(t-1) \\ \epsilon(t-k) &= y(t-k) - \hat{y}(t-k) \end{aligned}$$

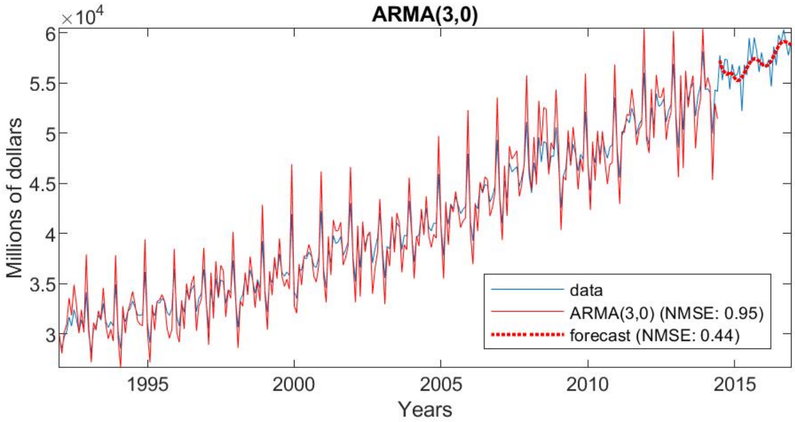
- computing the correlation coefficient $(\alpha(k))$ between $\epsilon(t)$ and $\epsilon(t-k)$. It captures the correlation between $y(t)$ and $y(t-k)$ not explained by $y(t-k+1), \cdots, y(t-1)$.

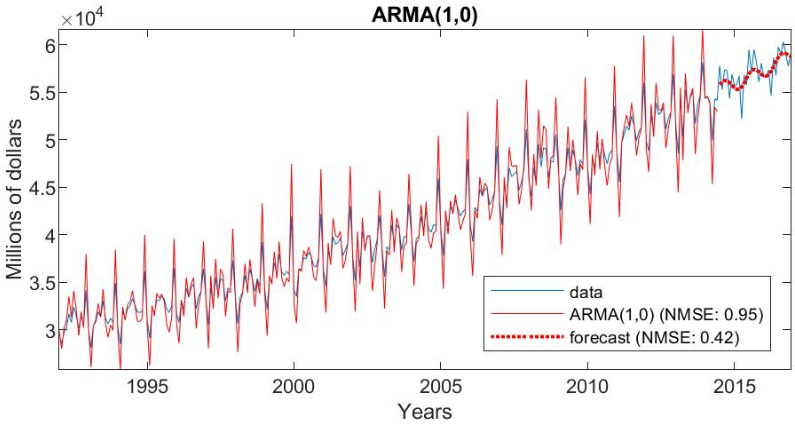
```
>> parcorr(dtds_train_data)
```

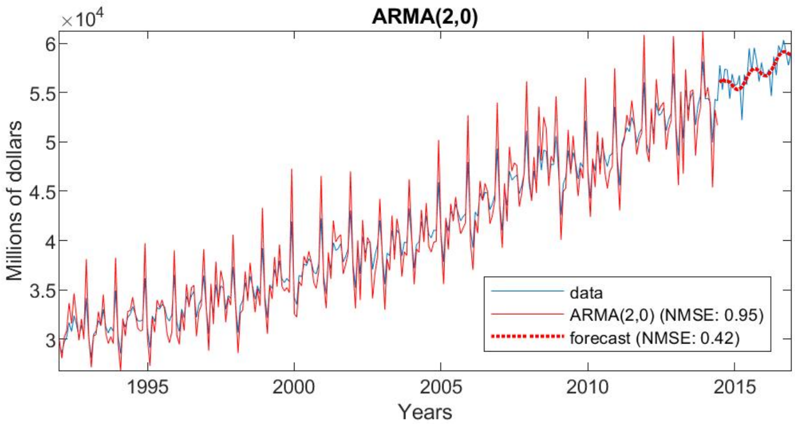


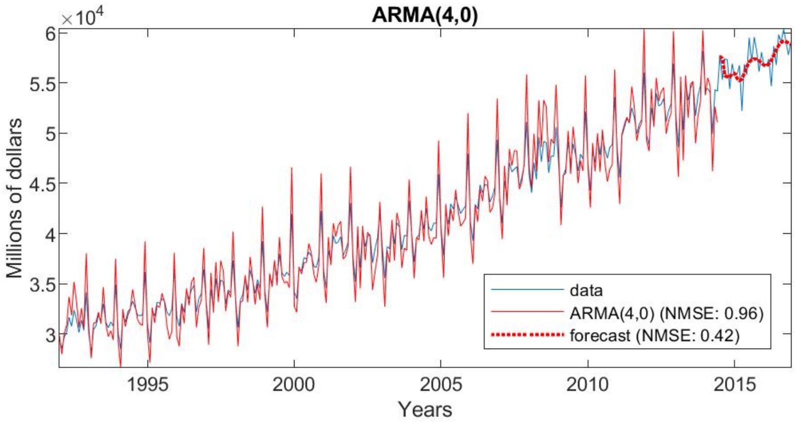
Given unclear insights from the PACF plot, AR models with order ranging from 1 to 12 are fitted; as for the MA model, the normalized mean squared error fitness value increment w.r.t. to trend and seasonality is very low: the best model, with a value of 44% is, AR(3).

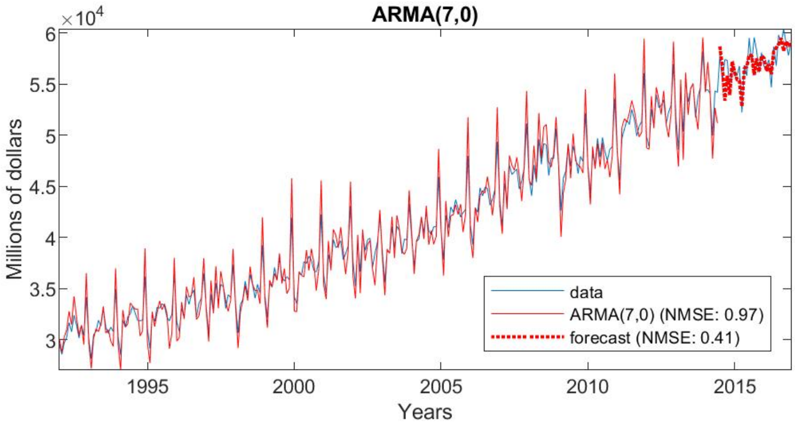
```
>> FindBestARMA(1:12, 0, all_data, components);
```

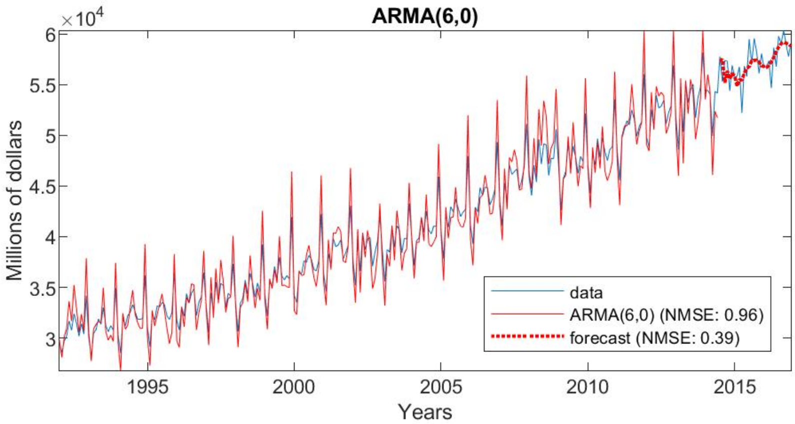












2.3 Autoregressive moving average model

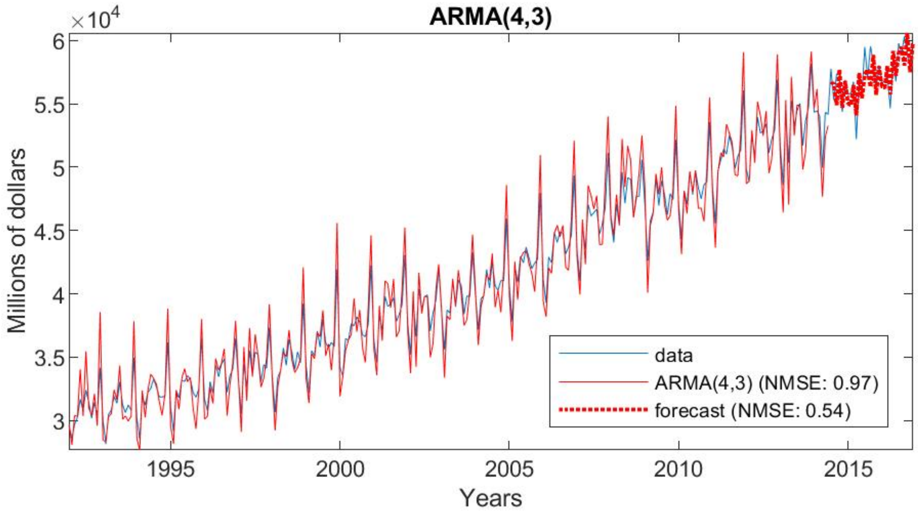
An ARMA(p, q) combines the two abovementioned models in a single one:

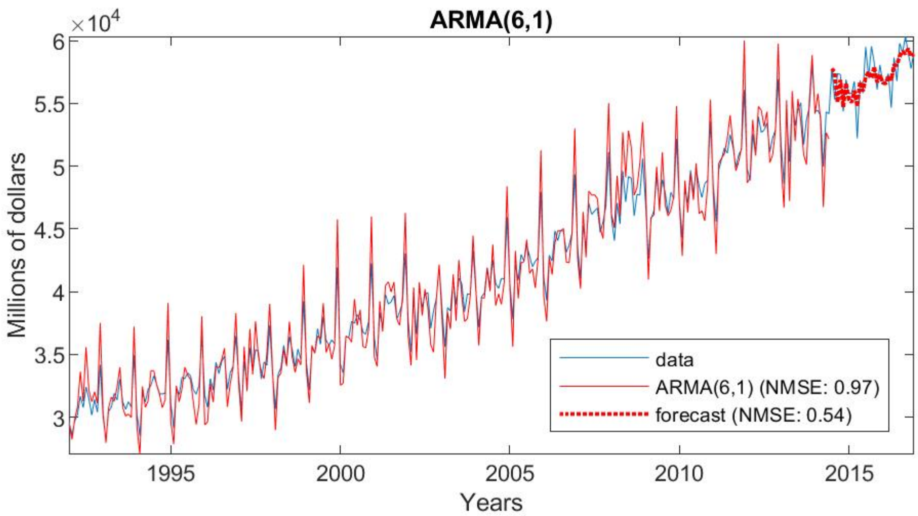
$$\hat{y}(t) = a_1 y(t-1) + \cdots + a_q y(t-p) +$$

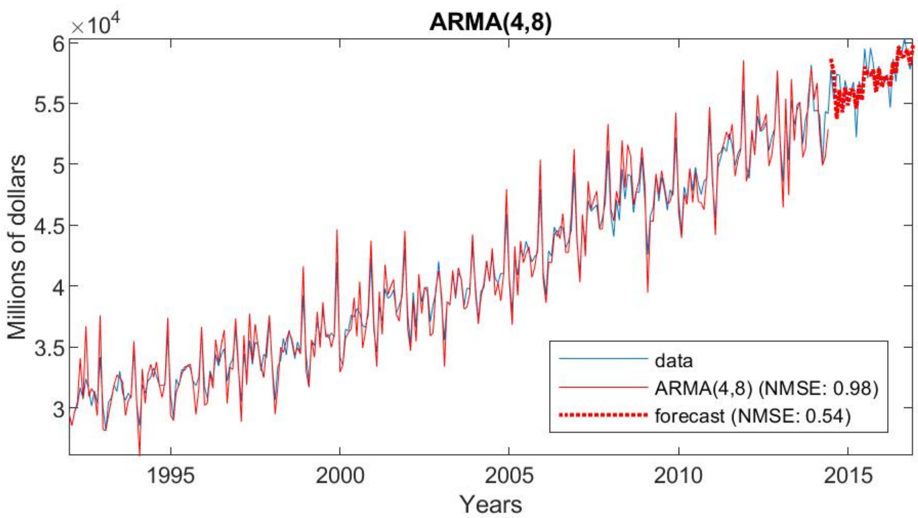
$$
\begin{aligned}
&+\xi(t) + c_1\xi(t-1) + \cdots + c_q\xi(t-q) \\
A(z)\hat{y}(t) &= C(z)\xi(t)
\end{aligned}
$$

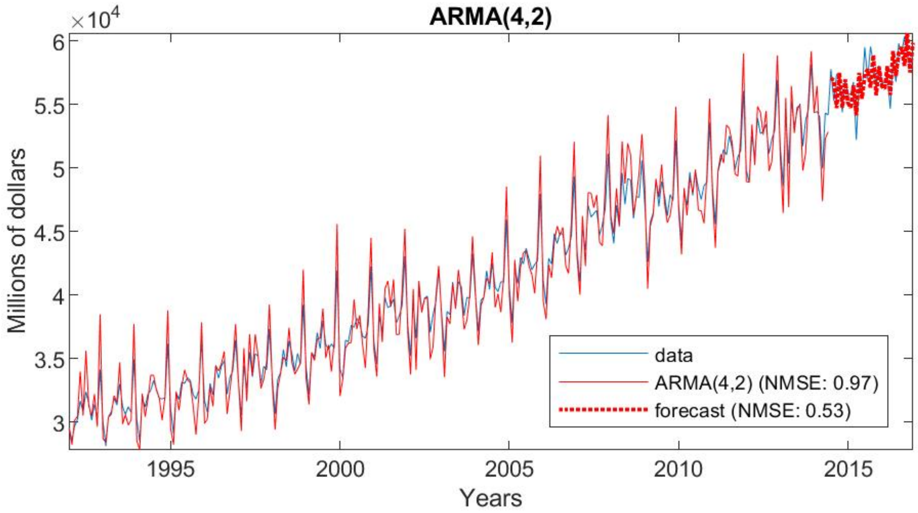
With the following experiments, with orders varying from 1 to 12 for both AR and MA parts, a greater increase in performance is met: ARMA(4,3) and ARMA(6,1) reach 54% of NMSE.

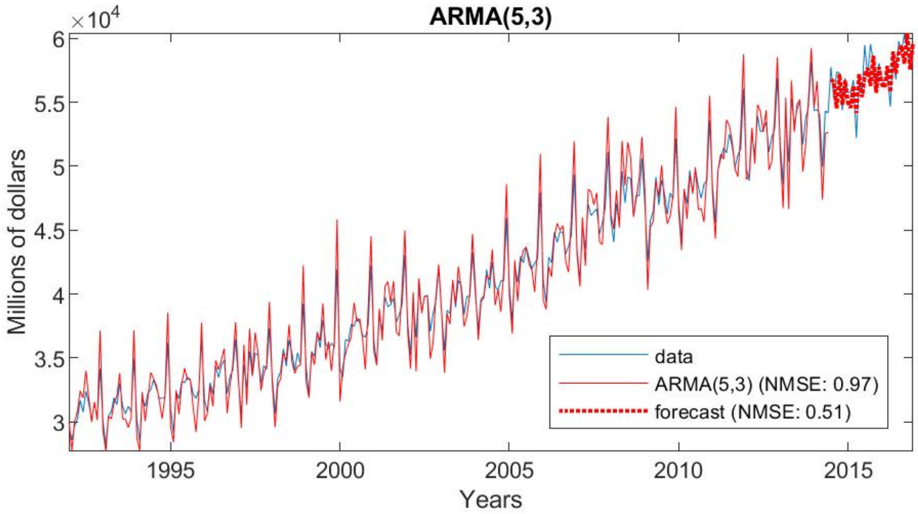
```
>> FindBestARMA(1:12, 1:12, all_data, components);
Warning: ARMA(2,8) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(2,12) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(5,11) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(5,12) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(6,11) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(7,5) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(7,7) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(7,8) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(7,9) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(8,9) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(8,10) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(8,11) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(9,6) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(10,4) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(10,5) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(11,5) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(11,11) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(12,11) unstable, discharged
> In FindBestARMA (line 35)
Warning: ARMA(12,12) unstable, discharged
> In FindBestARMA (line 35) )
```

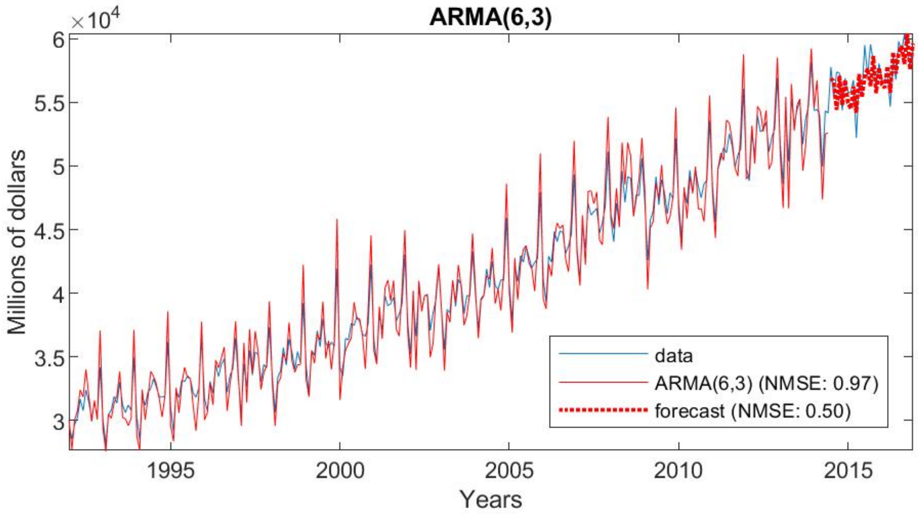

ARMA(4,3)
×10^4
Millions of dollars
Years
data
ARMA(4,3) (NMSE: 0.97)
forecast (NMSE: 0.54)


ARMA(6,1)
×10^4
Millions of dollars
Years
data
ARMA(6,1) (NMSE: 0.97)
forecast (NMSE: 0.54)


ARMA(4,8)
×10^4
Millions of dollars
Years
data
ARMA(4,8) (NMSE: 0.98)
forecast (NMSE: 0.54)


ARMA(4,2)
×10^4
Millions of dollars
Years
data
ARMA(4,2) (NMSE: 0.97)
forecast (NMSE: 0.53)


ARMA(5,3)
×10^4
Millions of dollars
Years
data
ARMA(5,3) (NMSE: 0.97)
forecast (NMSE: 0.51)


ARMA(6,3)
×10^4
Millions of dollars
Years
data
ARMA(6,3) (NMSE: 0.97)
forecast (NMSE: 0.50)

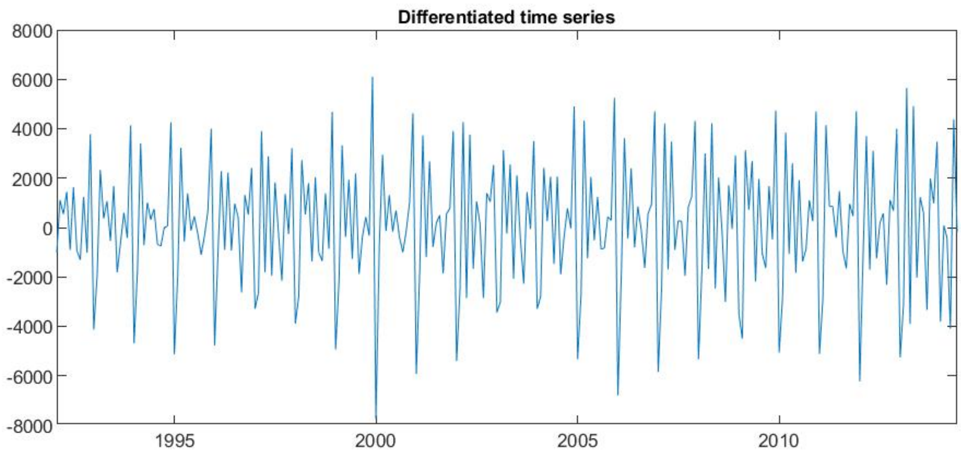
2.4 Seasonal autoregressive integrated moving average model

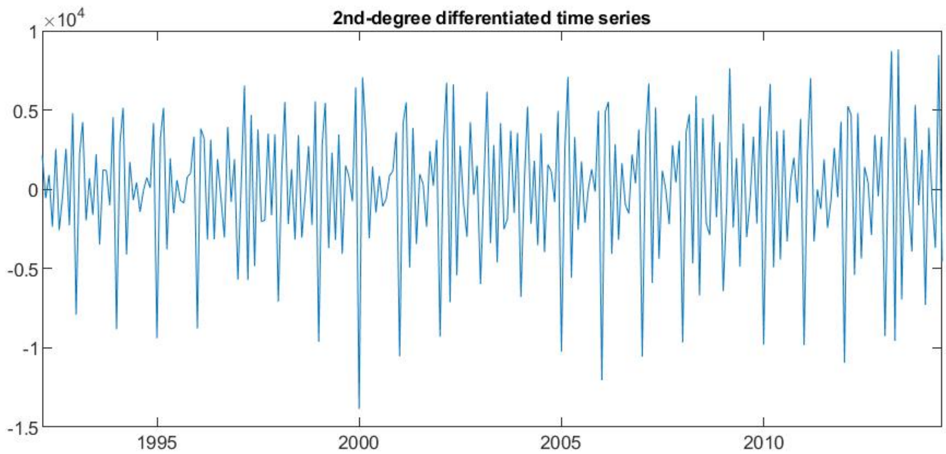
SARIMA models represents non-stationary signals directly, without the need of prior detrending and deseasing operations. For the secular part, this is done by differencing the series d times (operation which implies the removal of a stochastic trend); e.g. for $d = 1, 2$ differencing respectively means:

$$y^{\text{new}}(t) = \Delta_1 y(t) = y(t) - y(t-1) = (1 - z^{-1})y(t)$$
$$y^{\text{new}}(t) = \Delta_2 y(t) = (y(t) - y(t-1)) - (y(t-1) - y(t-2)) = y(t) - 2y(t-1) + y(t-2)$$
$$= (1 - z^{-1})^2 y(t)$$

Applying first and second degree differentiation to `train_data` (and losing respectively 1 and 2 data points), we get:

```
>> subplot(2,1,1)
>> plot(train_time(2:end), diff(train_data));
>> title('Differentiated time series')
>> subplot(2,1,2)
>> plot(train_time(3:end),  diff(train_data, 2));
>> title('2nd-degree differentiated time series')
```





The full ARIMA model is the following:

$$A(z)(1 - z^{-1})^d \hat{y}(t) = C(z)\xi(t)$$

Similarly, it's possible to differenciate seasonally (with period S), to get the SARIMA$(p, d, q) \times (P, D, Q)_S$ model, in which we distinguish seasonal and nonseasonal AR and MA orders:

$$A_p(z)\mathscr{A}_P(z^S)(1 - z^{-1})^d(1 - z^{-S})^D \hat{y}(t) = C_q(z)\mathscr{C}_Q(z^S)\xi(t)$$

where:

$$\mathscr{A}_P(z^S) = 1 - \tilde{a}_1 z^{-S} + \cdots + \tilde{a}_P z^{-S^P}$$
$$\mathscr{C}_Q(z^S) = 1 - \tilde{c}_1 z^{-S} + \cdots + \tilde{c}_Q z^{-S^Q}$$

E.g. SARIMA$(1, 1, 1) \times (2, 1, 1)_{12}$ is:

$$(1 - a_1 z^{-1})(1 - \tilde{a}_1 z^{-12} - \tilde{a}_1 z^{-24})(1 - z^{-1})(1 - z^{-12})\hat{y}(t) = (1 - c_1 z^{-1})(1 - \tilde{c}_1 z^{-12})\xi(t)$$

The following tests will be done:

- ARIMA with ar order (p) ranging from 0 to 12, ma order (q) ranging from 0 to 12 (escluding the 0,0 case) and order of integration $d = 1, 2, 3$;
- SARIMA with $p = 0, \cdots, 12$, $q = 0, \cdots, 12$ (escluding the 0,0 case), $d = 1, 2, 3$, periodicity S with values 4, 6 and 12, seasonal ar order P ranging from 0 to 2, seasonal D differentiation equal to 1, seasonal ma order Q ranging from 0 to 2[4].

```
function [train_pred, test_pred, ...
    train_nmse, test_nmse] = FitAndForecastSARIMA(p, d, q, P, D, Q, train_data, test_data)
% ======== fit model =====================================================
model = estimate(arima( ...
   'ARLags', p, 'D', d,'ARLags', q, 'SARLags', P, 'Seasonality', D, 'SMALags', Q), ...
   train_data, 'Display', 'off');
residuals = infer(model, train_data);
train_pred = train_data + residuals;
% ======== compute train error ===========================================
train_nmse = 1- min([1 ...
                   power( ...
                     norm( train_data - train_pred ) / ...
                     norm( train_data - mean(train_data)) ...
                         ,2) ...
                ]);
% ======== forecast ======================================================
[test_pred, ~] = forecast(model, numel(test_data), train_data);
% ======== compute test error ============================================
test_nmse = 1- min([1 ...
                   power( ...
                     norm( test_data - test_pred) / ...
                     norm( test_data - mean(test_data)) ...
                         ,2) ...
                ]);
```

[4] In Matlab's Econometric toolbox notation, $(1, 1, 1)_{12}$ is written as "ARLags"=12, "Seasonality"=12, "MALags"=12; $(2, 1, 2)_{12}$ is written as "ARLags"=[12 24], "Seasonality"=12, "MALags"=[12 24], and so on.

```
function [best_train_pred, best_test_pred, best_train_nmse, ...
    best_test_nmse, best_p, best_d, best_q, best_P, ...
                     best_Seasonality, best_Q] = FindBestSARIMA(p_sequence, d_sequence, q_sequence, ...
                                                P_sequence, Seasonality_sequence, Q_sequence, all_data)
% ========================================================================
train_data = all_data{1};
train_time = all_data{2};
test_data = all_data{3};
test_time = all_data{4};
% ======= initialization ({:,4} position is the best test_nmse) =========
best_models = cell(6,10);
best_models(1:6,4) = num2cell(repmat(-1000, 6,1));
% ======= fit ============================================================
for d = d_sequence
    for p = p_sequence
        for q = q_sequence
          if ~(p==0 && q==0)
           %% ======= ARIMA =============================================
           %% ======= adjusting ar,ma order for arima function ===========
           if p==0
               p_real=[];
           else
               p_real = 1:p;
           end
           if q==0
               q_real=[];
           else
               q_real = 1:q;
           end
           %% ======= fit ================================================
           try
                [train_pred, test_pred, ...
                    train_nmse, test_nmse] = FitAndForecastSARIMA( ...
                                             p_real, d, q_real, [], 0, [], train_data, test_data);
                if test_nmse>best_models{1,4}
                    best_models = [ {train_pred, test_pred, train_nmse, test_nmse, p, d, q, 0, 0, 0}; ...
                        best_models(1:5, 1:10) ];
                elseif test_nmse>best_models{2,4}
                    best_models = [ best_models(1, 1:10); {train_pred, test_pred, train_nmse, test_nmse, ...
                                                    p, d, q, 0, 0, 0}; ...
                                    best_models(2:5, 1:10) ];
                elseif test_nmse>best_models{3,4}
                    best_models = [ best_models(1:2, 1:10); {train_pred, test_pred, train_nmse, test_nmse, ...
                                                    p, d, q, 0, 0, 0}; ...
                                    best_models(3:5, 1:10) ];
                elseif test_nmse>best_models{4,4}
                   best_models = [ best_models(1:3, 1:10); {train_pred, test_pred, train_nmse, test_nmse, ...
                                                        p, d, q, 0, 0, 0}; ...
                                    best_models(4:5, 1:10) ];
                elseif test_nmse>best_models{5,4}
                   best_models = [ best_models(1:4, 1:10); {train_pred, test_pred, train_nmse, test_nmse, ...
                                                        p, d, q, 0, 0, 0}; ...
                                   best_models(5, 1:10) ];
                elseif test_nmse>best_models{6,4}
                   best_models = [ best_models(1:5, 1:10); ...
                        {train_pred, test_pred, train_nmse, test_nmse, p, d, q, 'no', 'no', 'no'} ];
                end
            catch
                warning('ARIMA(%d,%d,%d) unstable, discharged',p, d, q);
            end
```

(... continue ...)

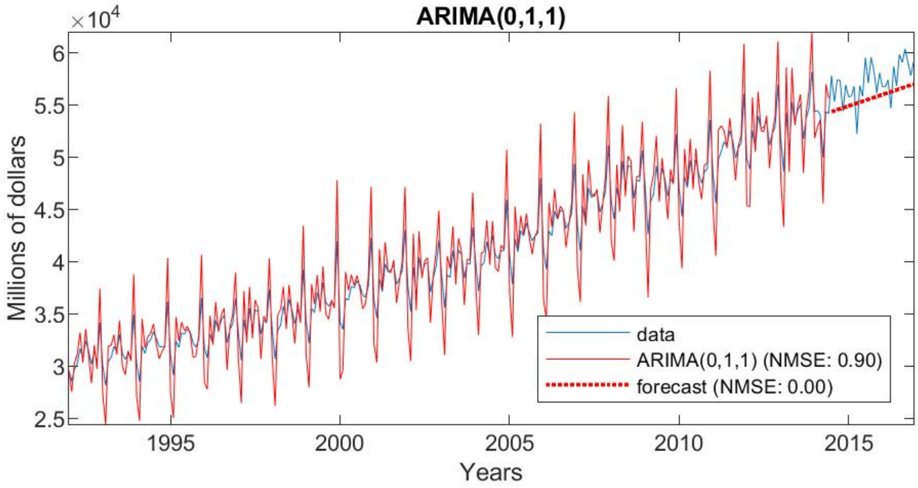
```
            %% ======== SARIMA ==========================================
            for Seasonality = Seasonality_sequence
                for P = P_sequence
                    %% ======== adjusting ma order for arima function =====
                    P_real = [];
                    if P>0
                        for number=1:P
                            P_real = [P_real,  number*Seasonality];
                        end
                    end
                    for Q = Q_sequence
                        %% ======== adjusting ar order for arima function =
                        Q_real = [];
                        if Q>0
                          for number=1:P
                            Q_real = [Q_real,  number*Seasonality];
                          end
                        end
                        %% ======== fit ===================================
                        try
                            [train_pred, test_pred, ...
                                train_nmse, test_nmse] = FitAndForecastSARIMA( ...
                                                     p_real, d, q_real, P_real, Seasonality, ...
                                                     Q_real, train_data, test_data);
                            if test_nmse>best_models{1,4}
                                best_models = [ {train_pred, test_pred, train_nmse, test_nmse, ...
                                                        p, d, q, P, Seasonality, Q}; ...
                                                best_models(1:5, 1:10) ];
                            elseif test_nmse>best_models{2,4}
                                best_models = [ best_models(1, 1:10); {train_pred, test_pred, ...
                                                        train_nmse, test_nmse, p, d, q, P, Seasonality, Q}; ...
                                                best_models(2:5, 1:10) ];
                            elseif test_nmse>best_models{3,4}
                                best_models = [ best_models(1:2, 1:10); {train_pred, test_pred, ...
                                                        train_nmse, test_nmse, p, d, q, P, Seasonality, Q}; ...
                                                best_models(3:5, 1:10) ];
                            elseif test_nmse>best_models{4,4}
                                best_models = [ best_models(1:3, 1:10); {train_pred, test_pred, ...
                                                           train_nmse, test_nmse, p, d, q, P, Seasonality, Q}; ...
                                               best_models(4:5, 1:10) ];
                            elseif test_nmse>best_models{5,4}
                               best_models = [ best_models(1:4, 1:10); {train_pred, test_pred, ...
                                                           train_nmse, test_nmse, p, d, q, P, Seasonality, Q}; ...
                                               best_models(5, 1:10) ];
                            elseif test_nmse>best_models{6,4}
                               best_models = [ best_models(1:5, 1:10); ...
                                    {train_pred, test_pred, train_nmse, test_nmse, p, d, q, P, ...
                                        Seasonality, Q} ];
                            end
                        catch
                            warning('SARIMA(%d,%d,%d)x(%d,1,%d)_{%d} unstable, discharged',p,d,q,P,Q,Seasonality);
                        end
                    end
                end
            end
            fprintf('... (%d,%d,%d) done.\n',p,d,q)
          end
        end
    end
end
```

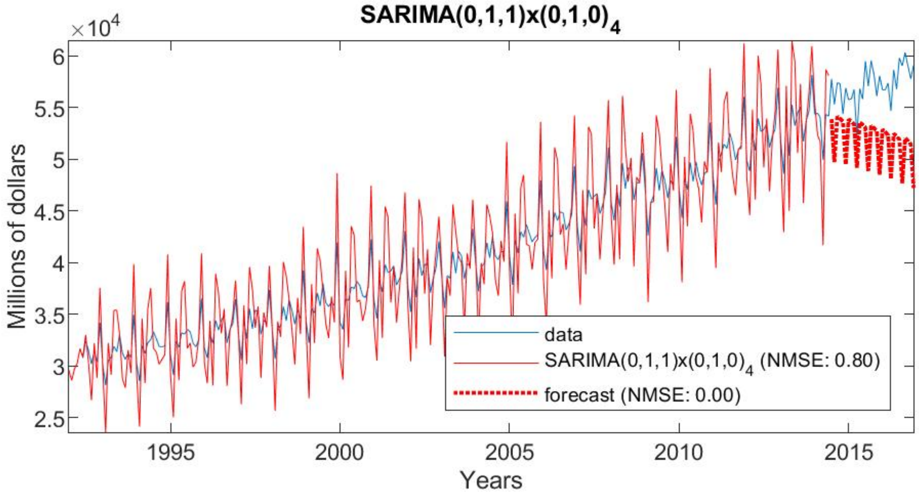
(... continue ...)

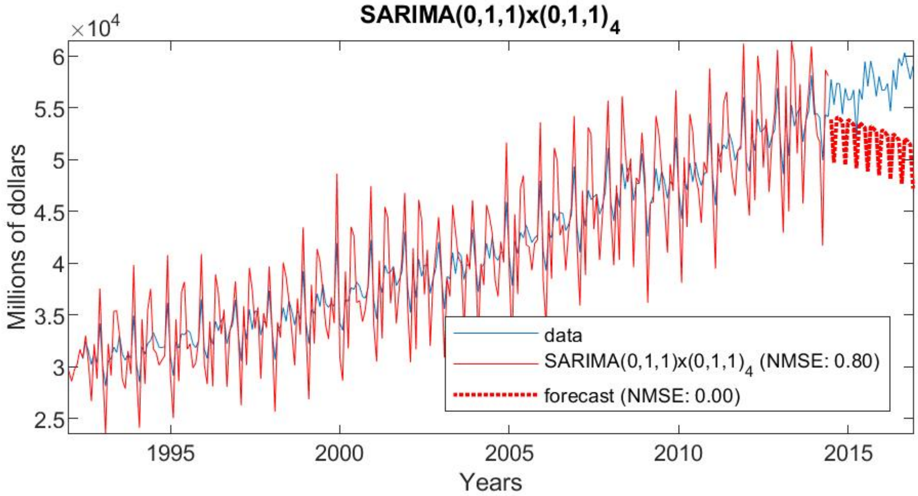
```
% ======== plot best models (2x (3x1) plots)==============================
for model_index=1:6
    train_pred = best_models{model_index,1};
    test_pred = best_models{model_index, 2};
    train_nmse = best_models{model_index, 3};
    test_nmse = best_models{model_index, 4};
    p = best_models{model_index, 5};
    d = best_models{model_index, 6};
    q = best_models{model_index, 7};
    P = best_models{model_index, 8};
    Seasonality = best_models{model_index, 9};
    Q = best_models{model_index, 10};
    if model_index < 4
        if model_index==1
            % ========== save for output the first model only =============
            best_train_pred = train_pred;
            best_test_pred = test_pred;
            best_train_nmse = train_nmse;
            best_test_nmse = test_nmse;
            best_p = p;
            best_d = d;
            best_q = q;
            best_P = p;
            best_Seasonality = Seasonality;
            best_Q = Q;
        end
        subplot(3,1,model_index)
    else
        if model_index == 4
            figure()
        end
        subplot(3,1,model_index-3)
    end
    plot([train_time; test_time], [train_data; test_data])
    hold on
    plot(train_time, train_pred, "r")
    plot(test_time, test_pred, "r:", 'LineWidth',1.8)
    hold off
    if Seasonality==0
        legend("data", ...
        sprintf('ARIMA(%d,%d,%d) (NMSE: %.2f)', p, d, q, train_nmse), ...
        sprintf("forecast (NMSE: %.2f)", test_nmse), ...
        'Location', "southeast")
        title(sprintf('ARIMA(%d,%d,%d)', p, d, q));
    else
        legend("data", ...
        sprintf('SARIMA(%d,%d,%d)x(%d,1,%d)_{%d} (NMSE: %.2f)', p, d, q, P, Q, Seasonality, train_nmse), ...
        sprintf("forecast (NMSE: %.2f)", test_nmse), ...
        'Location', "southeast")
        title(sprintf('SARIMA(%d,%d,%d)x(%d,1,%d)_{%d}', p, d, q, P, Q, Seasonality));
    end
    xlabel("Years");
    ylabel("Millions of dollars");
end
```

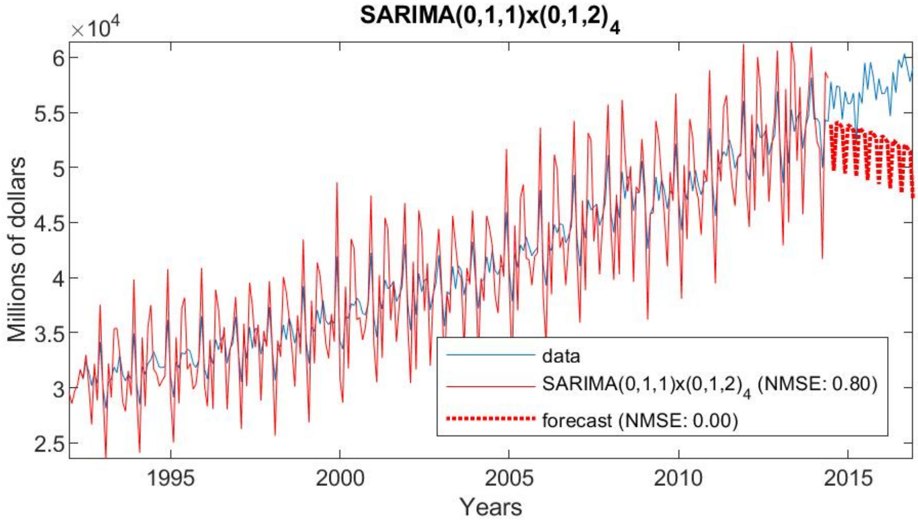
The integrated approach isn't able to catch the hidden relationship for prediction.

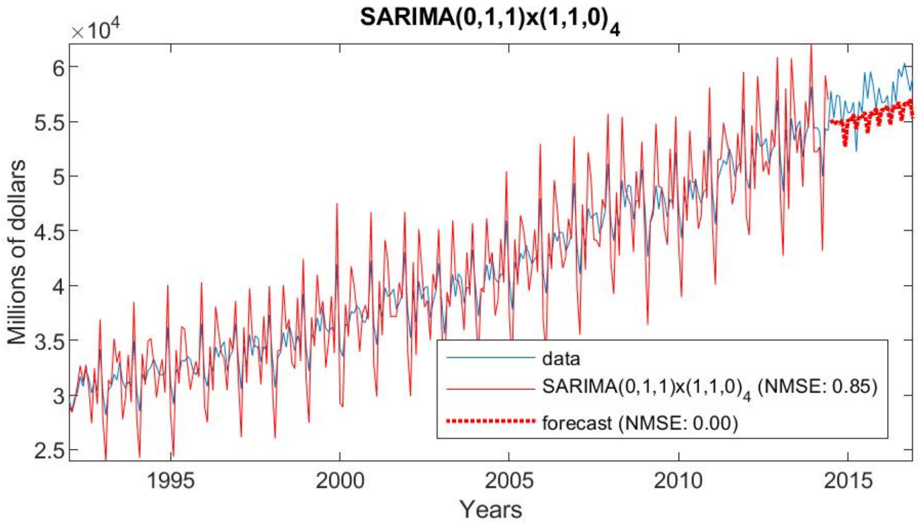
```
>> FindBestSARIMA(0:12, 1:3, 0:12, 0:2, [4 6 12], 0:2, all_data);
```

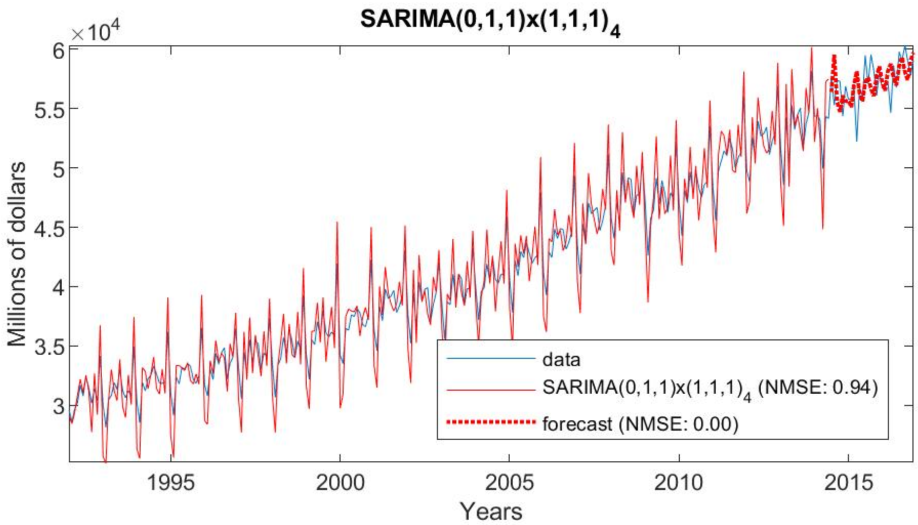

ARIMA(0,1,1)
×10^4
Millions of dollars
Years
data
ARIMA(0,1,1) (NMSE: 0.90)
forecast (NMSE: 0.00)


SARIMA(0,1,1)x(0,1,0)$_4$
×10^4
Millions of dollars
Years
data
SARIMA(0,1,1)x(0,1,0)$_4$ (NMSE: 0.80)
forecast (NMSE: 0.00)


SARIMA(0,1,1)x(0,1,1)$_4$
×10^4
Millions of dollars
Years
data
SARIMA(0,1,1)x(0,1,1)$_4$ (NMSE: 0.80)
forecast (NMSE: 0.00)


SARIMA(0,1,1)x(0,1,2)4
×10^4
Millions of dollars
Years
data
SARIMA(0,1,1)x(0,1,2)4 (NMSE: 0.80)
forecast (NMSE: 0.00)

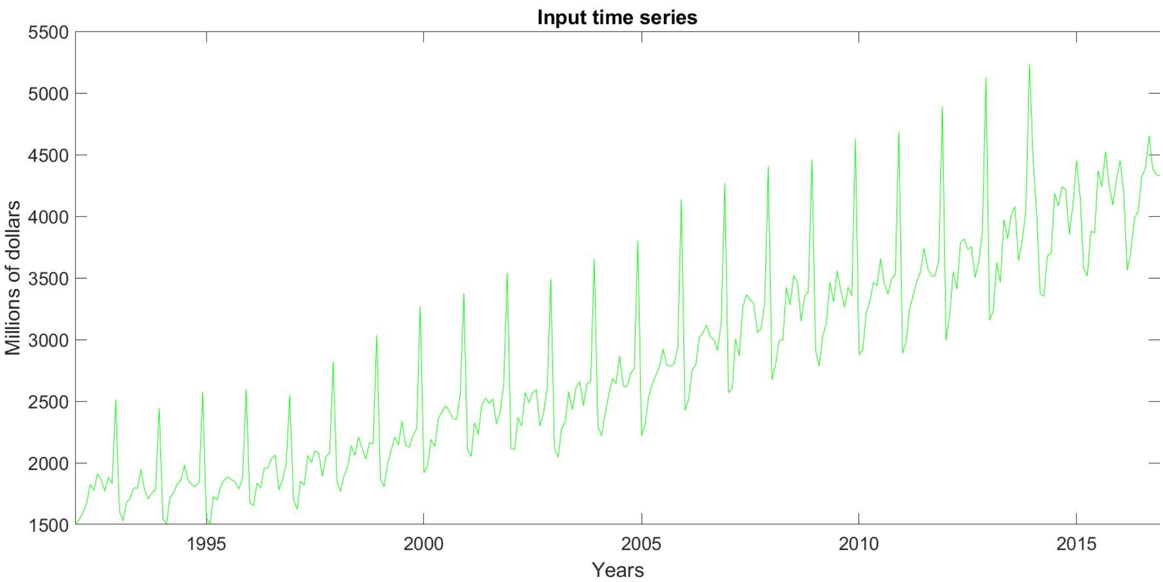

SARIMA(0,1,1)x(1,1,0)4
×10^4
Millions of dollars
Years
data
SARIMA(0,1,1)x(1,1,0)4 (NMSE: 0.85)
forecast (NMSE: 0.00)


SARIMA(0,1,1)x(1,1,1)4
×10^4
Millions of dollars
Years
data
SARIMA(0,1,1)x(1,1,1)4 (NMSE: 0.94)
forecast (NMSE: 0.00)

2.5 Autoregressive moving average model with an exogenous input

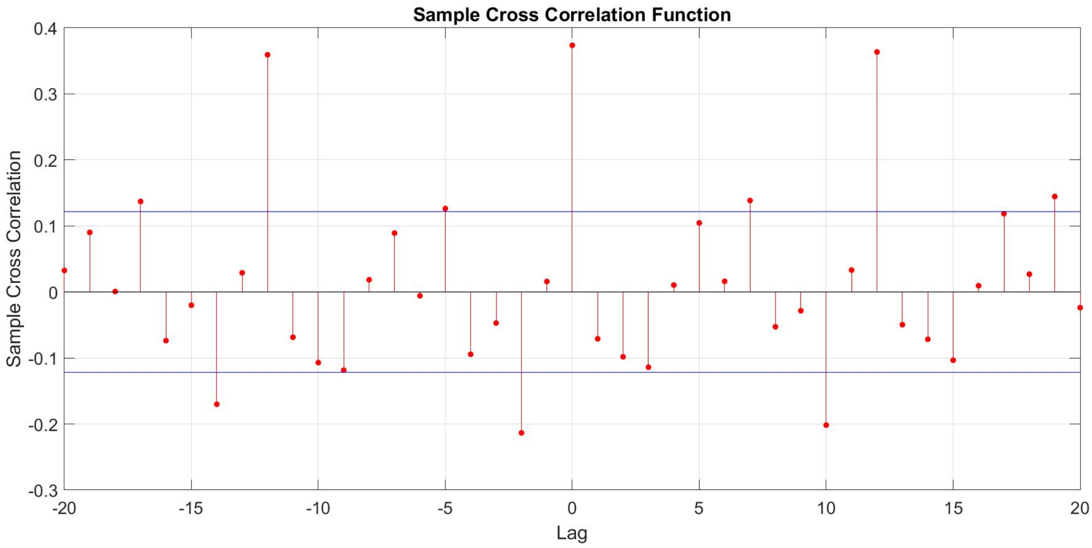
Let's consider another time series from the US Census Bureau, describing beer, wine and liquor monthly sales (300 samples in million of dollars, from 1992 to 2016, same as `Food`):

```
>> disp(head(Beer));
      Time         Data
   ___________    ____
   01-Jan-1992    1509
   01-Feb-1992    1541
   01-Mar-1992    1597
   01-Apr-1992    1675
   01-May-1992    1822
   01-Jun-1992    1775
   01-Jul-1992    1912
   01-Aug-1992    1862
>> plot(Beer.Time, Beer.Data);
>> title("Input time series")
>> xlabel("Years")
>> ylabel("Millions of dollars")
>> disp(numel(Beer.Data))
   300
```



One could try to exploit the linear correlation with `Beer` until $t-1$, to better predict $\hat{y}(t)$.

```
>> train_input = Beer.Data(1:270);
>> test_input = Beer.Data(271:end);
>> input = {train_input, test_input};
>> crosscorr(dtds_train_data,train_input)
```

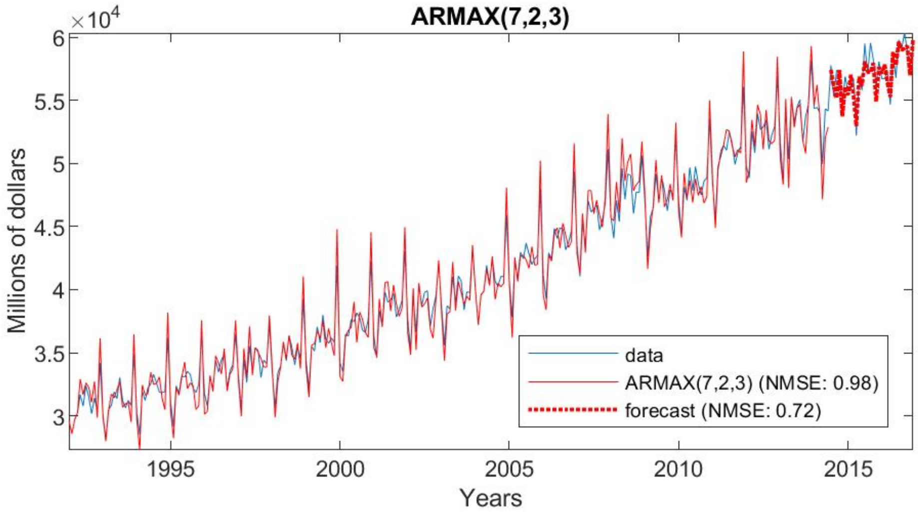


The ARMAX(p, g, q) model is defined by the following difference equation:

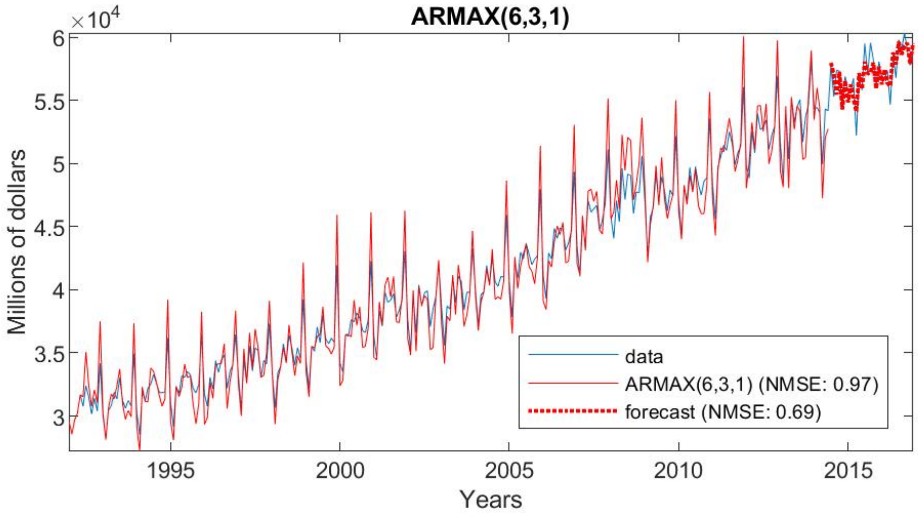
$$\begin{aligned}\hat{y}(t) = a_1 y(t-1) + \cdots + a_q y(t-p) + \\ + b_1 u(t-1) + \cdots + b_g u(t-g) \\ + \xi(t) + c_1 \xi(t-1) + \cdots + c_q \xi(t-q) \\ A(z)\hat{y}(t) = B(z)u(t-1) + C(z)\xi(t)\end{aligned}$$

```
function [train_pred, test_pred, ...
    train_nmse, test_nmse] = FitAndForecastARMAX(ar_order, x_order, ma_order, ...
                                                train_data, test_data, input, components)
% ===================================================================
train_comp = components{1};
test_comp = components{2};
train_input = input{1};
test_input = input{2};
dtds_train_data = train_data - train_comp;
train = iddata(dtds_train_data, train_input, 'TimeUnit','months');
% ======== fit model ================================================
model = armax(train, [ar_order [x_order] ma_order 1]);
train_pred = dtds_train_data + resid(train,model).y + train_comp;
% ======== compute train error ======================================
train_nmse = 1- min([1 ...
                   power( ...
                     norm( train_data - train_pred ) / ...
                     norm( train_data - mean(train_data)) ...
                         ,2) ...
                ]);
% ======== forecast =================================================
test_pred = forecast(model, train, numel(test_data), test_input);
test_pred = test_pred.y + test_comp;
% ======== compute test error =======================================
test_nmse = 1- min([1 ...
                   power( ...
                     norm( test_data - test_pred) / ...
                     norm( test_data - mean(test_data)) ...
                         ,2) ...
                ]);
```

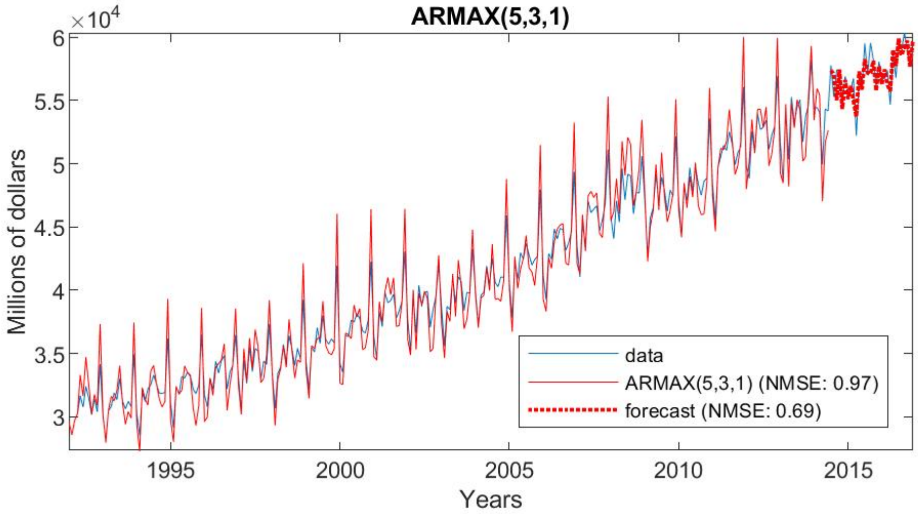
```
>> FindBestARMAX(0:12,1:12,0:12, all_data, input, components)
```



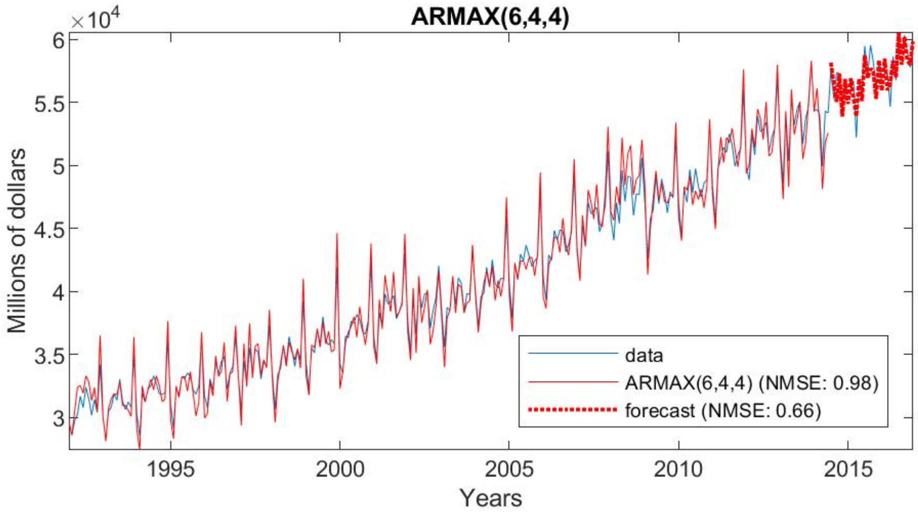
ARMAX(7,2,3)
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
ARMAX(7,2,3) (NMSE: 0.98)
forecast (NMSE: 0.72)

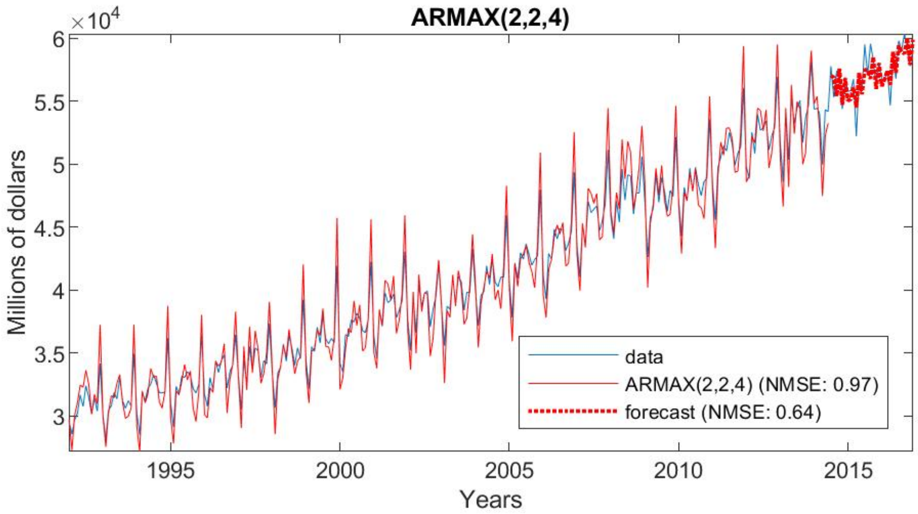


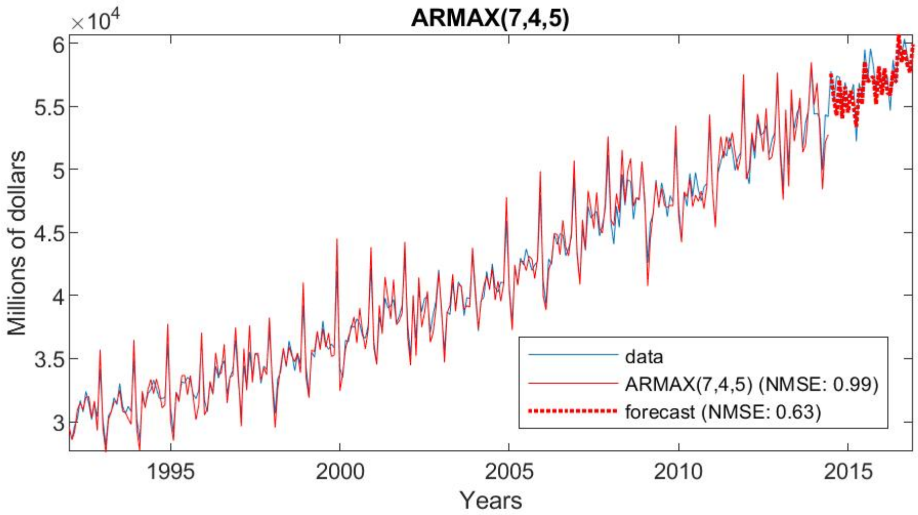
ARMAX(6,3,1)
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
ARMAX(6,3,1) (NMSE: 0.97)
forecast (NMSE: 0.69)



ARMAX(5,3,1)
×10^4
6
5.5
5
4.5
4
3.5
3
Millions of dollars
1995
2000
2005
2010
2015
Years
data
ARMAX(5,3,1) (NMSE: 0.97)
forecast (NMSE: 0.69)


ARMAX(6,4,4)
×10^4
Millions of dollars
6
5.5
5
4.5
4
3.5
3
1995
2000
2005
2010
2015
Years
data
ARMAX(6,4,4) (NMSE: 0.98)
forecast (NMSE: 0.66)


ARMAX(2,2,4)
×10^4
Millions of dollars
6
5.5
5
4.5
4
3.5
3
1995
2000
2005
2010
2015
Years
data
ARMAX(2,2,4) (NMSE: 0.97)
forecast (NMSE: 0.64)


ARMAX(7,4,5)
×10^4
Millions of dollars
6
5.5
5
4.5
4
3.5
3
1995
2000
2005
2010
2015
Years
data
ARMAX(7,4,5) (NMSE: 0.99)
forecast (NMSE: 0.63)

Between all orders combinations in $p = 0, \cdots, 12, g = 1, \cdots, 12,\ q = 0, \cdots, 12$ (except for $q = 0, p = 0$)[5], ARMAX(7,2,3) obtains the best test NMSE value (72%), overtaking by a long shot the ARMA performance.

3 Conclusion

The best model, combining trend, seasonality and the following ARMAX for the irregular component:

```
>> train = iddata(dtds_train_data, train_input, 'TimeUnit','months');
>> model = armax(train, [7 [2] 3 1]);
>> model
model =
Discrete-time ARMAX model: A(z)y(t) = B(z)u(t) + C(z)e(t)
  A(z) = 1 + 0.286 z^-1 + 0.5765 z^-2 + 0.5766 z^-3
          + 0.3397 z^-4 - 0.003062 z^-5 + 0.3332 z^-6
                                        - 0.4586 z^-7
  B(z) = 0.8756 z^-1 - 0.9045 z^-2
  C(z) = 1 + 0.07747 z^-1 + 0.4971 z^-2 + 0.7111 z^-3
Sample time: 1 months
Parameterization:
   Polynomial orders:   na=7   nb=2   nc=3
   nk=1
   Number of free coefficients: 12
   Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.
Status:
Estimated using ARMAX on time domain data "train".
Fit to estimation data: 41.2% (prediction focus)
FPE: 1.158e+06, MSE: 1.059e+06
```

has equation:

$$
\begin{aligned}
\hat{y}(t) = {} & \underbrace{30170.22 + 50.7t + 0.15t^2}_{\text{trend}} \\
& \underbrace{- 4.71 - 338.86\cos\left(\frac{1}{36}\pi t\right) - 88.44\sin\left(\frac{1}{36}\pi t\right) - 123.31\cos\left(\frac{1}{18}\pi t\right) + 38.37\sin\left(\frac{1}{18}\pi t\right)}_{\text{seasonality(1)}} \\
& \underbrace{+ 23.91\cos\left(\frac{1}{12}\pi t\right) + 3\sin\left(\frac{1}{12}\pi t\right) + 8.07\cos\left(\frac{1}{9}\pi t\right) - 21.47\sin\left(\frac{1}{9}\pi t\right)}_{\text{seasonality(2)}} \\
& \underbrace{+ 6.65\cos\left(\frac{5}{36}\pi t\right) + 55.48\sin\left(\frac{5}{36}\pi t\right) - 134.05\cos\left(\frac{1}{6}\pi t\right) - 652.18\sin\left(\frac{1}{6}\pi t\right)}_{\text{seasonality(3)}} \\
& \underbrace{- 59.58\cos\left(\frac{1}{4}\pi t\right) - 84.89\sin\left(\frac{1}{4}\pi t\right)}_{\text{seasonality(4)}} \\
& \underbrace{- 0.29y(t-1) - 0.58y(t-2) - 0.58y(t-3) - 0.34y(t-4) + 0.003y(t-5)}_{\text{autoregression(1)}} \\
& \underbrace{- 0.33y(t-6) + 0.46y(t-7)}_{\text{autoregression(2)}}
\end{aligned}
$$

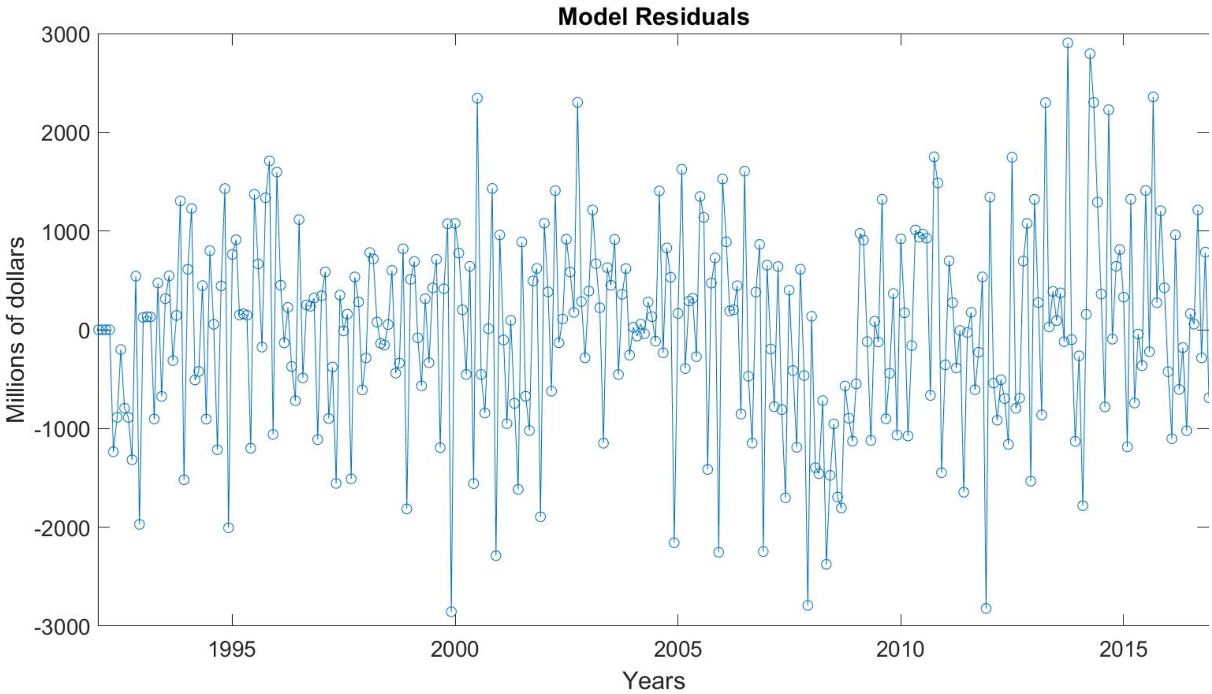
[5] `FindBestARMAX` can be found in the appendix.

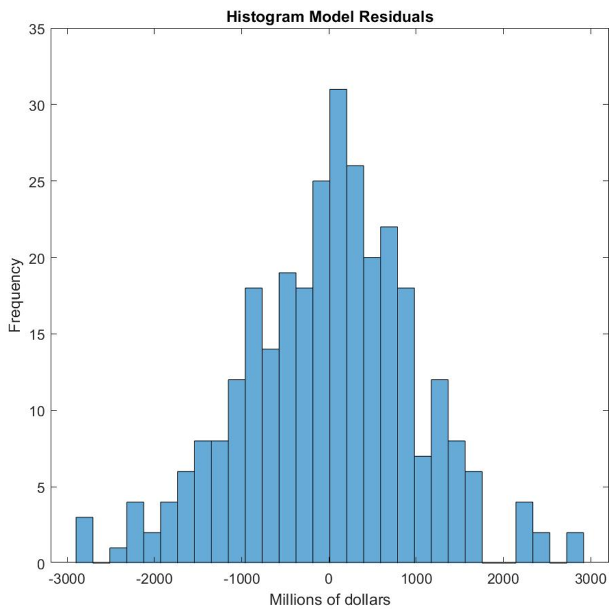
$$\underbrace{+\ \xi(t) + 0.08\xi(t-1) + 0.49\xi(t-2) + 0.71\xi(t-3)}_{\text{moving average}}$$

$$\underbrace{+\ 0.8756u(t-1) - 0.9045u(t-2)}_{\text{regression}}$$

The model residuals show that some regularities still persist, which implies that there is room for modeling improvements.

```
>> train_pred = dtds_train_data + resid(train,model).y + trendAndSeas;
>> test_pred = forecast(model,train, numel(test_data),test_input).y + forecast_trendAndSeas;
>> model_residuals = Food.Data - [train_pred; test_pred];
>> plot(Food.Time, model_residuals, 'o-')
>> title("Model Residuals")
>> xlabel("Years")
>> ylabel("Millions of dollars")
```





4 Appendix

```
function [best_train_pred, best_test_pred, ...
                    best_train_nmse, best_test_nmse, ...
      best_ar_order, best_ma_order] = FindBestARMA(ar_order_sequence, ...
                                                   ma_order_sequence, all_data, components)
% ===========================================================================
train_data = all_data{1};
train_time = all_data{2};
test_data = all_data{3};
test_time = all_data{4};
% ======== initialization ({:,4} position is the best test_nmse) =========
best_models = cell(6,6);
best_models(1:6,4) = num2cell(repmat(-1000, 6,1));
% ======== fit ==============================================================
for ar_order = ar_order_sequence
    for ma_order = ma_order_sequence
        try
            [train_pred, test_pred, train_nmse, test_nmse] = FitAndForecastARMA(ar_order, ma_order, ...
                                                           train_data, test_data, components);
            if test_nmse>best_models{1,4}
                best_models = [ {train_pred, test_pred, train_nmse, test_nmse, ar_order, ma_order}; ...
                    best_models(1:5, 1:6) ];
            elseif test_nmse>best_models{2,4}
                best_models = [ best_models(1, 1:6); {train_pred, test_pred, train_nmse, ...
                                      test_nmse,ar_order,ma_order}; ...
                                best_models(2:5, 1:6) ];
            elseif test_nmse>best_models{3,4}
                best_models = [ best_models(1:2, 1:6); {train_pred, test_pred, train_nmse, ...
                                        test_nmse,ar_order,ma_order}; ...
                                best_models(3:5, 1:6) ];
            elseif test_nmse>best_models{4,4}
               best_models = [ best_models(1:3, 1:6); {train_pred, test_pred, train_nmse, ...
                                           test_nmse,ar_order,ma_order}; ...
                               best_models(4:5, 1:6) ];
            elseif test_nmse>best_models{5,4}
               best_models = [ best_models(1:4, 1:6); {train_pred, test_pred, train_nmse, ...
                                           test_nmse,ar_order,ma_order}; ...
                               best_models(5, 1:6) ];
            elseif test_nmse>best_models{6,4}
               best_models = [ best_models(1:5, 1:6); ...
                    {train_pred, test_pred, train_nmse, test_nmse, ar_order, ma_order} ];
            end
        catch
            warning('ARMA(%d,%d) unstable, discharged',ar_order, ma_order);
        end
    end
end
```

(... continue ...)

```
% ======== plot best models (2x (3x1) plots)==============================
for model_index=1:6
    train_pred = best_models{model_index,1};
    test_pred = best_models{model_index, 2};
    train_nmse = best_models{model_index, 3};
    test_nmse = best_models{model_index, 4};
    ar_order = best_models{model_index, 5};
    ma_order = best_models{model_index, 6};
    if model_index < 4
        if model_index==1
            % ========== save for output the first model only =============
            best_train_pred = train_pred;
            best_test_pred = test_pred;
            best_train_nmse = train_nmse;
            best_test_nmse = test_nmse;
            best_ar_order = ar_order;
            best_ma_order = ma_order;
        end
        subplot(3,1,model_index)
    else
        if model_index == 4
            figure()
        end
        subplot(3,1,model_index-3)
    end
    plot([train_time; test_time], [train_data; test_data])
    hold on
    plot(train_time, train_pred, "r")
    plot(test_time, test_pred, "r:", 'LineWidth',1.8)
    hold off
    legend("data", ...
    sprintf('ARMA(%d,%d) (NMSE: %.2f)', ar_order, ma_order, train_nmse), ...
    sprintf("forecast (NMSE: %.2f)", test_nmse), ...
    'Location', "southeast")
    title(sprintf('ARMA(%d,%d)', ar_order, ma_order));
    xlabel("Years");
    ylabel("Millions of dollars");
end
```

```
function [best_train_pred, best_test_pred, ...
    best_train_nmse, best_test_nmse, ...
    best_ar_order, best_x_order, best_ma_order] = FindBestARMAX(ar_order_sequence, x_order_sequence, ...
                                                  ma_order_sequence, all_data, input, components)
% ======================================================================
train_data = all_data{1};
train_time = all_data{2};
test_data = all_data{3};
test_time = all_data{4};
% ======== initialization ({:,4} position is the best test_nmse) =========
best_models = cell(6,7);
best_models(1:6,4) = num2cell(repmat(-1000, 6,1));
% ======== fit ===========================================================
for x_order = x_order_sequence
    for ar_order = ar_order_sequence
        for ma_order = ma_order_sequence
           if ~(ma_order==0 && ar_order==0)
            try
                [train_pred, test_pred, train_nmse, test_nmse] = FitAndForecastARMAX(ar_order, ...
                                          x_order, ma_order, train_data, test_data, input,components);
                if test_nmse>best_models{1,4}
                    best_models = [ {train_pred, test_pred, train_nmse, test_nmse, ar_order, x_order, ma_order }; ...
                        best_models(1:5, 1:7) ];
                elseif test_nmse>best_models{2,4}
                    best_models = [ best_models(1, 1:7); {train_pred, test_pred, train_nmse, test_nmse, ar_order, ...
      x_order, ma_order}; ...
                        best_models(2:5, 1:7) ];
                elseif test_nmse>best_models{3,4}
                    best_models = [ best_models(1:2, 1:7); {train_pred, test_pred, train_nmse, test_nmse, ar_order, ...
      x_order, ma_order}; ...
                        best_models(3:5, 1:7) ];
                elseif test_nmse>best_models{4,4}
                   best_models = [ best_models(1:3, 1:7); {train_pred, test_pred, train_nmse, test_nmse, ar_order, ...
        x_order, ma_order}; ...
                        best_models(4:5, 1:7) ];
                elseif test_nmse>best_models{5,4}
                   best_models = [ best_models(1:4, 1:7); {train_pred, test_pred, train_nmse, test_nmse, ar_order, ...
        x_order, ma_order}; ...
                        best_models(5, 1:7) ];
                elseif test_nmse>best_models{6,4}
                   best_models = [ best_models(1:5, 1:7); ...
                        {train_pred, test_pred, train_nmse, test_nmse, ar_order, x_order, ma_order} ];
                end
            catch
                warning('ARMAX(%d,%d,%d) unstable, discharged',ar_order, x_order, ma_order);
            end
           end
        end
    end
end
```

(... continue ...)

```
% ======== plot best models (2x (3x1) plots)==============================
for model_index=1:6
    train_pred = best_models{model_index,1};
    test_pred = best_models{model_index, 2};
    train_nmse = best_models{model_index, 3};
    test_nmse = best_models{model_index, 4};
    ar_order = best_models{model_index, 5};
    x_order = best_models{model_index, 6};
    ma_order = best_models{model_index, 7};
    if model_index < 4
        if model_index==1
            % ========== save for output the first model only =============
            best_train_pred = train_pred;
            best_test_pred = test_pred;
            best_train_nmse = train_nmse;
            best_test_nmse = test_nmse;
            best_ar_order = ar_order;
            best_x_order = x_order;
            best_ma_order = ma_order;
        end
        subplot(3,1,model_index)
    else
        if model_index == 4
            figure()
        end
        subplot(3,1,model_index-3)
    end
    plot([train_time; test_time], [train_data; test_data])
    hold on
    plot(train_time, train_pred, "r")
    plot(test_time, test_pred, "r:", 'LineWidth',1.8)
    hold off
    legend("data", ...
    sprintf('ARMAX(%d,%d,%d) (NMSE: %.2f)', ar_order, x_order, ma_order, train_nmse), ...
    sprintf("forecast (NMSE: %.2f)", test_nmse), ...
    'Location', "southeast")
    title(sprintf('ARMAX(%d,%d,%d)', ar_order, x_order, ma_order));
    xlabel("Years");
    ylabel("Millions of dollars");
end
```