ARIMAX models

Introduction

Autoregressive integrated moving average (ARIMAX) models extend ARIMA models through the inclusion of exogenous variables X. We write an $ARIMAX(p, d, q)$ model for some time series data y_t and exogenous data X_t, where p is the number of autoregressive lags, d is the degree of differencing and q is the number of moving average lags as:

$$\Delta^D y_t = \sum_{i=1}^{p} \phi_i \Delta^D y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \sum_{m=1}^{M} \beta_m Xm, t + \epsilon_t$$

$$\epsilon_t \sim N\left(0, \sigma^2\right)$$

Example

We will combine ARIMA dynamics with intervention analysis for monthly UK driver death data. There are two interventions we are interested in: the 1974 oil crisis and the introduction of the seatbelt law in 1983. We will model the effects of these events as structural breaks.

```
import numpy as np
import pandas as pd
import pyflux as pf
from datetime import datetime
import matplotlib.pyplot as plt
%matplotlib inline

data = pd.read_csv("https://vincentarelbundock.github.io/Rdatasets/csv/MASS/drivers.csv")
data.index = data['time'];
data.loc[(data['time']>=1983.05), 'seat_belt'] = 1;
data.loc[(data['time']<1983.05), 'seat_belt'] = 0;
data.loc[(data['time']>=1974.00), 'oil_crisis'] = 1;
data.loc[(data['time']<1974.00), 'oil_crisis'] = 0;
plt.figure(figsize=(15,5));
plt.plot(data.index,data['drivers']);
plt.ylabel('Driver Deaths');
plt.title('Deaths of Car Drivers in Great Britain 1969-84');
plt.plot();
```

The structural breaks can be included via patsy notation. Below we estimate a point mass estimate z^{MLE} of the latent variables:

```
model = pf.ARIMAX(data=data, formula='drivers~1+seat_belt+oil_crisis',
                  ar=1, ma=1, family=pf.Normal())
x = model.fit("MLE")
x.summary()

ARIMAX(1,0,1)
======================================== ============================================
Dependent Variable: drivers              Method: MLE
Start Date: 1969.08333333                Log Likelihood: -1278.7616
End Date: 1984.91666667                  AIC: 2569.5232
Number of observations: 191              BIC: 2589.0368
================================================================================================
Latent Variable          Estimate   Std Error  z         P>|z|    95% C.I.
======================== ========== ========== ========= ======== ========================
AR(1)                    0.5002     0.0933     5.3607    0.0      (0.3173 | 0.6831)
MA(1)                    0.1713     0.0991     1.7275    0.0841   (-0.023 | 0.3656)
Beta 1                   946.585    176.9439   5.3496    0.0      (599.7749 | 1293.3952)
Beta seat_belt           -57.8924   57.8211    -1.0012   0.3167   (-171.2217 | 55.437)
Beta oil_crisis          -151.673   44.119     -3.4378   0.0006   (-238.1462 | -65.1998)
Sigma                    195.6042
================================================================================================
```

We can plot the in-sample fit using `plot_fit()`:

```
model.plot_fit(figsize=(15,10))
```

To forecast forward we need exogenous variables for future dates. Since the interventions carry forward, we can just use a slice of the existing dataframe and use func:*plot_predict*:

```
model.plot_predict(h=10, oos_data=data.iloc[-12:], past_values=100, figsize=(15,5))
```

Class Description

class **ARIMAX**(*data, formula, ar, ma, integ, target, family*)

Autoregressive Integrated Moving Average Exogenous Variable Models (ARIMAX).

Parameter	Type	Description
data	pd.DataFrame or np.ndarray	Contains the univariate time series
formula	string	Patsy notation specifying the regression
ar	int	The number of autoregressive lags
ma	int	The number of moving average lags

Parameter	Type	Description
integ	int	How many times to difference the data (default: 0)
target	string or int	Which column of DataFrame/array to use.
family	pf.Family instance	The distribution for the time series, e.g `pf.Normal()`

Attributes

`latent_variables`

A pf.LatentVariables() object containing information on the model latent variables, prior settings. any fitted values, starting values, and other latent variable information. When a model is fitted, this is where the latent variables are updated/stored. Please see the documentation on Latent Variables for information on attributes within this object, as well as methods for accessing the latent variable information.

Methods

`adjust_prior`(*index*, *prior*)

Adjusts the priors for the model latent variables. The latent variables and their indices can be viewed by printing the `latent_variables` attribute attached to the model instance.

Parameter	Type	Description
index	int	Index of the latent variable to change
prior	pf.Family instance	Prior distribution, e.g. `pf.Normal()`

Returns: void - changes the model `latent_variables` attribute

`fit`(*method*, ***kwargs*)

Estimates latent variables for the model. User chooses an inference option and the method returns a results object, as well as updating the model's `latent_variables` attribute.

Parameter	Type	Description
method	str	Inference option: e.g. 'M-H' or 'MLE'

See Bayesian Inference and Classical Inference sections of the documentation for the full list of inference options. Optional parameters can be entered that are relevant to the particular mode of inference chosen.

Returns: pf.Results instance with information for the estimated latent variables

`plot_fit`(***kwargs*)

Plots the fit of the model against the data. Optional arguments include *figsize*, the dimensions of the figure to plot.

Returns : void - shows a matplotlib plot

`plot_ppc`(*T*, *nsims*)

Plots a histogram for a posterior predictive check with a discrepancy measure of the user's choosing. This method only works if you have fitted using Bayesian inference.

Parameter	Type	Description
T	function	Discrepancy, e.g. `np.mean` or `np.max`
nsims	int	How many simulations for the PPC

Returns: void - shows a matplotlib plot

`plot_predict`(*h*, *oos_data*, *past_values*, *intervals*, ***kwargs*)

Plots predictions of the model, along with intervals.

Parameter	Type	Description
h	int	How many steps to forecast ahead
oos_data	pd.DataFrame	Exogenous variables in a frame for h steps
past_values	int	How many past datapoints to plot
intervals	boolean	Whether to plot intervals or not

To be clear, the *oos_data* argument should be a DataFrame in the same format as the initial dataframe used to initialize the model instance. The reason is that to predict future values, you need to specify assumptions about exogenous variables for the future. For example, if you predict *h* steps ahead, the method will take the h first rows from *oos_data* and take the values for the exogenous variables that you asked for in the patsy formula.

Optional arguments include *figsize* - the dimensions of the figure to plot. Please note that if you use Maximum Likelihood or Variational Inference, the intervals shown will not reflect latent variable uncertainty. Only Metropolis-Hastings will give you fully Bayesian prediction intervals. Bayesian intervals with variational inference are not shown because of the limitation of mean-field inference in not accounting for posterior correlations.

Returns : void - shows a matplotlib plot

`plot_predict_is`(*h*, *fit_once*, *fit_method*, ***kwargs*)

Plots in-sample rolling predictions for the model. This means that the user pretends a last subsection of data is out-of-sample, and forecasts after each period and assesses how well they did. The user can choose whether to fit parameters once at the beginning or every time step.

Parameter	Type	Description
h	int	How many previous timesteps to use
fit_once	boolean	Whether to fit once, or every timestep
fit_method	str	Which inference option, e.g. 'MLE'

Optional arguments include *figsize* - the dimensions of the figure to plot. **h** is an int of how many previous steps to simulate performance on.

Returns : void - shows a matplotlib plot

`plot_sample`(*nsims*, *plot_data=True*)

Plots samples from the posterior predictive density of the model. This method only works if you fitted the model using Bayesian inference.

Parameter	Type	Description
nsims	int	How many samples to draw
plot_data	boolean	Whether to plot the real data as well

Returns : void - shows a matplotlib plot

`plot_z`(*indices*, *figsize*)

Returns a plot of the latent variables and their associated uncertainty.

Parameter	Type	Description
indices	int or list	Which latent variable indices to plot
figsize	tuple	Size of the matplotlib figure

Returns : void - shows a matplotlib plot

`ppc`(*T*, *nsims*)

Returns a p-value for a posterior predictive check. This method only works if you have fitted using Bayesian inference.

Parameter	Type	Description
T	function	Discrepancy, e.g. `np.mean` or `np.max`
nsims	int	How many simulations for the PPC

Returns: int - the p-value for the discrepancy test

`predict`(*h*, *oos_data*, *intervals=False*)

Returns a DataFrame of model predictions.

Parameter	Type	Description
h	int	How many steps to forecast ahead
oos_data	pd.DataFrame	Exogenous variables in a frame for h steps
intervals	boolean	Whether to return prediction intervals

To be clear, the *oos_data* argument should be a DataFrame in the same format as the initial dataframe used to initialize the model instance. The reason is that to predict future values, you need to specify assumptions about exogenous variables for the future. For example, if you predict *h* steps ahead, the method will take the 5 first rows from *oos_data* and take the values for the exogenous variables that you specified as exogenous variables in the patsy formula.

Please note that if you use Maximum Likelihood or Variational Inference, the intervals shown will not reflect latent variable uncertainty. Only Metropolis-Hastings will give you fully Bayesian prediction intervals. Bayesian intervals with variational inference are not shown because of the limitation of mean-field inference in not accounting for posterior correlations.

Returns : pd.DataFrame - the model predictions

`predict_is`(*h*, *fit_once*, *fit_method*)

Returns DataFrame of in-sample rolling predictions for the model.

Parameter	Type	Description
h	int	How many previous timesteps to use
fit_once	boolean	Whether to fit once, or every timestep
fit_method	str	Which inference option, e.g. 'MLE'

Returns : pd.DataFrame - the model predictions

sample(*nsims*)

Returns np.ndarray of draws of the data from the posterior predictive density. This method only works if you have fitted the model using Bayesian inference.

Parameter	Type	Description
nsims	int	How many posterior draws to take

Returns : np.ndarray - samples from the posterior predictive density.

References

Box, G; Jenkins, G. (1970). Time Series Analysis: Forecasting and Control. San Francisco: Holden-Day.

Hamilton, J.D. (1994). Time Series Analysis. Taylor & Francis US.