

# Time Series Regression and Forecasting

---

Juergen Meinecke

Time Series Terminology, Autocorrelation  
Lags, First Differences, Growth Rates

Notation is now slightly different

Instead of an  $i$ -subscript, variables will have a  $t$ -subscript  
(this is not a substantive change, just convention for time series)

The variable  $Y_t$  is the value of  $Y$  (for example real GDP) in period  $t$   
(for example year)

Data set:  $\{Y_1, \dots, Y_T\}$  are  $T$  observations on the time series  $Y$

We consider only consecutive, evenly-spaced observations  
(for example, monthly, 1960 to 1999, no missing months)

Missing and unevenly spaced data do not pose a principal problem  
and only introduce technical complications which we are happy to  
ignore at this stage

### Definition

The **first lag** of time series  $Y_t$  is  $Y_{t-1}$ .

The  **$j$ -th lag** of time series  $Y_t$  is  $Y_{t-j}$ .

### Definition

The **first difference** of time series  $Y_t$  is  $\Delta Y_t := Y_t - Y_{t-1}$ .

### Definition

The **first difference of the logarithm** of time series  $Y_t$  is  
 $\Delta \ln(Y_t) := \ln(Y_t) - \ln(Y_{t-1})$ .

With these definitions it is easy to determine the percentage change of a time series  $Y_t$  between the periods  $t - 1$  and  $t$ :  
it is approximately  $100 \cdot \Delta \ln(Y_t)$

Example: Quarterly CPI data for the US

I'm starting out with a time series on the price level in the US

Price level here is measured by the consumer price index (CPI)

The specific time series I'm using is labelled **CPIAUCSL**

It is the *Consumer Price Index for All Urban Consumers* provided by the Federal Reserve Bank of St. Louis (FRED)

Let's look at two recent measurements

- CPI in the fourth quarter of 2023 (2023:Q4) = 308.10
- CPI in the first quarter of 2024 (2024:Q1) = 310.99

Given this price level data, how do we back out inflation?

We study two approaches: exact and approximate

- CPI in the fourth quarter of 2023 (2023:Q4) = 308.10
- CPI in the first quarter of 2024 (2024:Q1) = 310.99
- Inflation via *exact* percentage change in CPI,  
2023:Q4 to 2024:Q1

$$100 \cdot \left( \frac{310.99 - 308.10}{308.10} \right) = 0.94\%$$

- Inflation via logarithmic *approximation* instead:

$$100 \cdot (\ln(310.99) - \ln(308.10)) = 0.93\%$$

The two approaches give slightly different results

It is common to extrapolate up the quarter-to-quarter change to an annual rate

Quarter-to-quarter change *at an annual rate*

- Annualized inflation via *exact* percentage change in CPI

$$4 \cdot 100 \cdot \left( \frac{310.99 - 308.10}{308.10} \right) = 3.75\%$$

- Annualized inflation via logarithmic *approximation* instead:

$$4 \cdot 100 \cdot (\ln(310.99) - \ln(308.10)) = 3.73\%$$

Answers the question: if the current quarter inflation continued throughout the year, what would annual inflation be?

It's a simple extrapolation really

# Time Series Regression and Forecasting

---

Juergen Meinecke



Time Series Terminology, Autocorrelation

Autocorrelation

The correlation of a time series with its own lagged values is called autocorrelation or serial correlation

### Definition

The  $j$ -th **autocovariance** of a time series  $Y_t$  is the covariance between  $Y_t$  and its  $j$ -th lag,  $Y_{t-j}$ :  $\text{Cov}(Y_t, Y_{t-j})$ .

The  $j$ -th **autocorrelation** of a time series  $Y_t$  is the correlation between  $Y_t$  and its  $j$ -th lag,  $Y_{t-j}$ :

$$\rho(j) := \frac{\text{Cov}(Y_t, Y_{t-j})}{\sqrt{\text{Var}(Y_t)\text{Var}(Y_{t-j})}}.$$

The sample autocorrelation is the *estimated* autocorrelation

## Definition

The  $j$ -th **sample autocorrelation** of a time series  $Y_t$  is the correlation between  $Y_t$  and its  $j$ -th lag,  $Y_{t-j}$ :

$$\hat{\rho}(j) := \frac{\widehat{\text{Cov}}(Y_t, Y_{t-j})}{\widehat{\text{Var}}(Y_t)},$$

$$\text{with } \widehat{\text{Cov}}(Y_t, Y_{t-j}) := \frac{1}{T} \sum_{t=j+1}^T (Y_t - \bar{Y}_{j+1,T})(Y_{t-j} - \bar{Y}_{1,T-j})$$

$$\bar{Y}_{p,q} := \frac{1}{T-j} \sum_{t=p}^q Y_t$$

$$\widehat{\text{Var}}(Y_t) := \frac{1}{T} \sum_{t=1}^T (Y_t - \bar{Y})^2$$

Two little comments:

Although we only compare  $T - j$  pairs of the time series, the division is by  $T$  (this is conventional in time series analysis)

When computing the sample autocorrelation, we have implicitly assumed that

- variances are constant over time
- covariances are constant over time  
(only dependent on the lag length  $j$ )

This is justified by *stationarity* (which we will define next week)

# Python example: quarterly CPI data for the US

Using the time series **CPIAUCSL** on quarterly CPI in the US,  
I create the quarter-to-quarter inflation at an annualized rate

## Python Code

```
> import pandas as pd
> import statsmodels.formula.api as smf
> import numpy as np
> # reading data from spreadsheet (downloaded from FRED):
> df = pd.read_csv('CPIAUCSL.csv')

> # creating quarterly index
> df['date'] = pd.to_datetime(df['DATE'], format='%Y-%m-%d')
> df.index = pd.DatetimeIndex(df.date, name='quarter').to_period('Q')

> # copy of CPI series with easy-to-access name:
> df['cpi'] = df.CPIAUCSL

> # taking logarithm of original series:
> df['logcpi'] = np.log(df.cpi)

> # creating annualised inflation via differences in logs:
> # (this is the 'first derivative' of 'cpi')
> df['infl'] = 400 * df.logcpi.diff()

> # creating quarter-on-quarter differences in inflation:
> # (this is the 'second derivative' of 'cpi')
> df['dinfl'] = df.infl.diff()

> df = df.drop(['DATE', 'CPIAUCSL'], axis=1)
```

## Let's take a look at the time series

### Python Code

```
> # looking at data: top two years
> print(df.head(8))
```

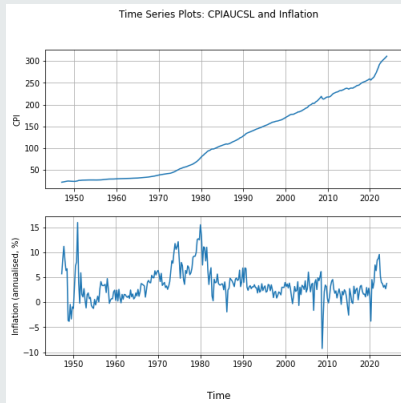
	date	cpi	logcpi	infl	dinfl
quarter					
1947Q1	1947-01-01	21.700000	3.077312	NaN	NaN
1947Q2	1947-04-01	22.010000	3.091497	5.673854	NaN
1947Q3	1947-07-01	22.490000	3.113071	8.629548	2.955694
1947Q4	1947-10-01	23.126667	3.140986	11.166235	2.536687
1948Q1	1948-01-01	23.616667	3.161953	8.386530	-2.779705
1948Q2	1948-04-01	23.993333	3.177776	6.329335	-2.057195
1948Q3	1948-07-01	24.396667	3.194447	6.668199	0.338864
1948Q4	1948-10-01	24.173333	3.185250	-3.678565	-10.346764

```
> # looking at data: bottom two years
> print(df.tail(8))
```

	date	cpi	logcpi	infl	dinfl
quarter					
2022Q2	2022-04-01	291.706333	5.675748	9.547648	0.822408
2022Q3	2022-07-01	295.509000	5.688699	5.180682	-4.366966
2022Q4	2022-10-01	298.441000	5.698572	3.949186	-1.231496
2023Q1	2023-01-01	301.203000	5.707784	3.684879	-0.264307
2023Q2	2023-04-01	303.466667	5.715272	2.994927	-0.689952
2023Q3	2023-07-01	306.034333	5.723697	3.370209	0.375281
2023Q4	2023-10-01	308.099000	5.730421	2.689546	-0.680663
2024Q1	2024-01-01	310.989667	5.739760	3.735410	1.045865

# Python Code

```
> fig, axs = plt.subplots(2, 1, figsize=(8,7))
> axs[0].plot(df.date, df.cpi)
> axs[0].set_ylabel('CPI')
> axs[1].plot(df.date, df.infl)
> axs[1].set_ylabel('Inflation (annualised, %)')
> fig.supxlabel('Time')
> fig.suptitle('Time Series Plots: CPIAUCSL and Inflation')
> plt.show()
```



Then I look at sample autocorrelations

## Python Code

```
> from matplotlib import pyplot as plt
> from statsmodels.graphics.tsaplots import plot_acf

> # creating a 'stacked' plot of 3 rows
> fig, axs = plt.subplots(3, 1, figsize = (8, 14))

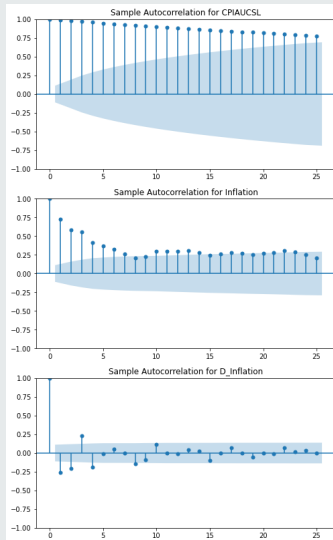
> # stacking them
> plot_acf(df.cpi, ax=axs[0], title = 'Sample Autocorrelation for CPIAUCSL')
> plot_acf(df.infl, missing='drop', ax=axs[1], title = 'Sample Autocorrelation for Inflation')
> plot_acf(df.dinfl, missing='drop', ax=axs[2], title = 'Sample Autocorrelation for D_Inflation')
> plt.show()
```

which creates the following plot ...



# Increasing degree of 'differentiation' reduces autocorrelation

## Python Code Output



These sample autocorrelations show

- the original time series **CPIAUCSL** (price level as measured by cpi) is very highly serially or auto-correlated
- **infl** (the first derivative of **CPIAUCSL**) is still highly serially correlated
- **dinfl** (the first derivative of **infl** and second derivative of **CPIAUCSL**) is not serially correlated anymore

Please bear this in mind, as it will have important ramifications when we want to run auto-regressions using price level or inflation data

Detecting serial correlation by visual inspection is tricky:

both series are highly auto-correlated, yet only obvious for CPI

# Time Series Regression and Forecasting

---

Juergen Meinecke

Autoregressive Models and Forecasting

The First Order Autoregressive (AR(1)) Model

A natural starting point for a forecasting model is to use past values of  $Y$  (that is,  $Y_{t-1}, Y_{t-2}, \dots$ ) to forecast  $Y_t$

An autoregression is a regression model in which  $Y_t$  is regressed against its own lagged values

The number of lags used as regressors is called the *order* of the autoregression

In a first order autoregression,  $Y_t$  is regressed against  $Y_{t-1}$

In a  $p$ -th order autoregression,  $Y_t$  is regressed against  $Y_{t-1}, Y_{t-2}, \dots, Y_{t-p}$

The population AR(1) model is

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + u_t$$

The coefficient  $\beta_1$  does NOT have a causal interpretation

If  $\beta_1 = 0$  then  $Y_{t-1}$  is not useful for forecasting  $Y_t$

The AR(1) model is estimated by OLS regression of  $Y_t$  on  $Y_{t-1}$

Testing  $\beta_1 = 0$  versus  $\beta_1 \neq 0$  provides a test of the hypothesis that  $Y_{t-1}$  is not useful for forecasting  $Y_t$

## Python Code

```
> # creating lagged inflation
> # (will be used as explanatory variable in AR(1) estimation)
> df['l1infl'] = df.infl.shift(1)
```

```
> # looking at data: top two years
> print(df[['cpi', 'infl', 'l1infl']].head(8))
```

	cpi	infl	l1infl
quarter			
1947Q1	21.700000	NaN	NaN
1947Q2	22.010000	5.673854	NaN
1947Q3	22.490000	8.629548	5.673854
1947Q4	23.126667	11.166235	8.629548
1948Q1	23.616667	8.386530	11.166235
1948Q2	23.993333	6.329335	8.386530
1948Q3	24.396667	6.668199	6.329335
1948Q4	24.173333	-3.678565	6.668199

```
> # looking at data: bottom two years
> print(df[['cpi', 'infl', 'l1infl']].tail(8))
```

	cpi	infl	l1infl
quarter			
2022Q2	291.706333	9.547648	8.725240
2022Q3	295.509000	5.180682	9.547648
2022Q4	298.441000	3.949186	5.180682
2023Q1	301.203000	3.684879	3.949186
2023Q2	303.466667	2.994927	3.684879
2023Q3	306.034333	3.370209	2.994927
2023Q4	308.099000	2.689546	3.370209
2024Q1	310.989667	3.735410	2.689546

Here I'm running an AR(1) estimation for `infl`

## Python Code (output edited)

```
> # first order autoregression:
> ar1 = smf.ols('infl ~ l1infl', data=df, missing='drop').fit(use_t=False)
> print(ar1.summary())
```

OLS Regression Results

```
=====
Dep. Variable:          infl    R-squared:                0.523
Model:                  OLS     Adj. R-squared:           0.521
Method:                 Least Squares   F-statistic:              334.3
No. Observations:      307
Df Residuals:          305
Df Model:               1
Covariance Type:       nonrobust
=====
```

	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.9526	0.186	5.127	0.000	0.588	1.317
l1infl	0.7226	0.040	18.285	0.000	0.645	0.800

```
=====
```

Notice: We don't need to use heteroskedasticity-robust standard errors because we are not really interested in statistical inference, instead we want to use the coefficient estimates to produce forecasts



# Forecasting

Our main objective when estimating autoregressions is to produce *forecasts*

We are not interested in causal effects

As a consequence, we are not usually interested in the coefficient estimates of AR models

We only use the coefficient estimates to create a forecast for the dependent variable

External validity is paramount: the model estimated using historical data must hold into the (near) future

But what do I mean by *forecast*?

## Notation

- For an AR(1) model:

$$Y_{T+1|T} = \beta_0 + \beta_1 Y_T$$

$$\hat{Y}_{T+1|T} = \hat{\beta}_0 + \hat{\beta}_1 Y_T$$

- $Y_{T+1|T}$ : forecast of  $Y_{T+1}$  based on  $Y_T, Y_{T-1}, \dots$  using the population coefficients (typically unknown)
- $\hat{Y}_{T+1|T}$ : forecast of  $Y_{T+1}$  based on  $Y_T, Y_{T-1}, \dots$  using the estimated coefficients
- Forecast errors are defined by  $Y_{T+1} - \hat{Y}_{T+1|T}$

Do not confuse predicted values with forecasts

- *Predicted values* are “in-sample”
- *Forecasts* are “out-of-sample”  
(looking into the future)

Let me explain the difference between predicted values and forecasts

Earlier we estimated the following AR(1) model for inflation:

$$\widehat{\text{infl}}_t = 0.9526 + 0.7226 \cdot \text{infl}_{t-1}$$

We used data from 1947:Q1–2024:Q1 for the estimation

This means:

- $\widehat{\text{infl}}_{2024:Q1}$  is a predicted value
- $\widehat{\text{infl}}_{2024:Q2|2024:Q1}$  is a forecast

Let's calculate both

These are simple common sense calculations

Calculation for the predicted value

In the data we observe  $\text{infl}_{2023:Q4} = 2.6895$

Resulting in the predicted value

$$\widehat{\text{infl}}_{2024:Q1} = 0.9526 + 0.7226 \cdot 2.6895 = 2.896$$

In my data set I do observe  $\text{infl}_{2024:Q1} = 3.7354$

therefore  $\text{infl}_{2024:Q1} - \widehat{\text{infl}}_{2024:Q1}$  is the *residual* for Q1 2024

Calculation for the forecast

In the data we observe  $\text{infl}_{2024:Q1} = 3.7354$

Resulting in the forecast values

$$\widehat{\text{infl}}_{2024:Q2|2024:Q1} = 0.9526 + 0.7226 \cdot 3.7354 = 3.6518$$

I could wait until July when  $\text{infl}_{2024:Q2}$  is released and calculate the *forecast error*  $\text{infl}_{2024:Q2} - \widehat{\text{infl}}_{2024:Q2|2024:Q1}$

Easy to produce predicted values and forecasts in **Python**

Just use the post-regression **predict** function

It will produce a predicted value when in-sample

It will produce a forecast value when out-of-sample

## Python Code

```
> # Prediction for 2024:Q1, and forecast for 2024:Q2  
> newdata = {'l1infl' : [df.infl[-2], df.infl[-1]]}  
> ar1.predict(newdata)
```

```
0    2.896081  
1    3.651812
```

# Time Series Regression and Forecasting

---

Juergen Meinecke

Autoregressive Models and Forecasting

The  $p$ -th Order Autoregressive (AR( $p$ )) Model



The population AR(p) model is

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \dots + \beta_p Y_{t-p} + u_t$$

The coefficients do NOT have a causal interpretation

To test hypothesis that  $Y_{t-2}, \dots, Y_{t-p}$  do not add value over and above  $Y_{t-1}$ , use an  $F$ -test

We will look at choosing  $p$  using a suitable information criterion

# Here I'm preparing an AR(4) estimation for `infl`

## Python Code

```
> # creating more lags for inflation
> df['l2infl'] = df.infl.shift(2)
> df['l3infl'] = df.infl.shift(3)
> df['l4infl'] = df.infl.shift(4)

># looking at data: top two years
> print(df[['cpi', 'infl', 'l1infl', 'l2infl', 'l3infl', 'l4infl']].head(8))
```

	cpi	infl	l1infl	l2infl	l3infl	l4infl
quarter						
1947Q1	21.700000	NaN	NaN	NaN	NaN	NaN
1947Q2	22.010000	5.673854	NaN	NaN	NaN	NaN
1947Q3	22.490000	8.629548	5.673854	NaN	NaN	NaN
1947Q4	23.126667	11.166235	8.629548	5.673854	NaN	NaN
1948Q1	23.616667	8.386530	11.166235	8.629548	5.673854	NaN
1948Q2	23.993333	6.329335	8.386530	11.166235	8.629548	5.673854
1948Q3	24.396667	6.668199	6.329335	8.386530	11.166235	8.629548
1948Q4	24.173333	-3.678565	6.668199	6.329335	8.386530	11.166235

```
> # looking at data: bottom two years
> print(df[['cpi', 'infl', 'l1infl', 'l2infl', 'l3infl', 'l4infl']].tail(8))
```

	cpi	infl	l1infl	l2infl	l3infl	l4infl
quarter						
2022Q2	291.706333	9.547648	8.725240	8.398633	6.304446	7.443233
2022Q3	295.509000	5.180682	9.547648	8.725240	8.398633	6.304446
2022Q4	298.441000	3.949186	5.180682	9.547648	8.725240	8.398633
2023Q1	301.203000	3.684879	3.949186	5.180682	9.547648	8.725240
2023Q2	303.466667	2.994927	3.684879	3.949186	5.180682	9.547648
2023Q3	306.034333	3.370209	2.994927	3.684879	3.949186	5.180682
2023Q4	308.099000	2.689546	3.370209	2.994927	3.684879	3.949186
2024Q1	310.989667	3.735410	2.689546	3.370209	2.994927	3.684879

Here I'm running an AR(4) estimation for `infl`

## Python Code (output edited)

```
> # fourth order autoregression:  
> ar4 = smf.ols('infl ~ l1infl + l2infl + l3infl + l4infl',  
               data=df, missing='drop').fit(use_t=False)  
> print(ar4.summary())
```

OLS Regression Results

```
=====
```

Dep. Variable:	infl	R-squared:	0.560
Model:	OLS	Adj. R-squared:	0.554
Method:	Least Squares	F-statistic:	95.02
No. Observations:	304		
Df Residuals:	299		
Df Model:	4		
Covariance Type:	nonrobust		

```
=====
```

	coef	std err	z	P> z	[0.025	0.975]
Intercept	0.7567	0.194	3.902	0.000	0.377	1.137
l1infl	0.6183	0.057	10.845	0.000	0.507	0.730
l2infl	0.0082	0.065	0.128	0.898	-0.118	0.135
l3infl	0.3125	0.065	4.844	0.000	0.186	0.439
l4infl	-0.1672	0.056	-2.966	0.003	-0.278	-0.057

```
=====
```

Again producing prediction and forecast

We estimated the following AR(4) model for inflation:

$$\widehat{\text{infl}}_t = 0.7567 + 0.6183 \cdot \text{infl}_{t-1} + 0.0082 \cdot \text{infl}_{t-2} + \\ 0.3125 \cdot \text{infl}_{t-3} - 0.1672 \cdot \text{infl}_{t-4}$$

In the data we observe

## Python Code

```
> df.infl.tail(5)
quarter
2023Q1    3.684879
2023Q2    2.994927
2023Q3    3.370209
2023Q4    2.689546
2024Q1    3.735410
```

$$\widehat{\text{infl}}_{2024:Q1} = 0.7567 + 0.6183 \cdot (2.6895) + 0.0082 \cdot (3.3702) \\ + 0.3125 \cdot (2.9949) - 0.1672 \cdot (3.6849) = 2.7673$$

$$\widehat{\text{infl}}_{2024:Q2|2024:Q1} = 0.7567 + 0.6183 \cdot (3.7354) + 0.0082 \cdot (2.6895) \\ + 0.3125 \cdot (3.3702) - 0.1672 \cdot (2.9949) = 3.6409$$

# Forecasting

Still easy to produce predicted values and forecasts in Python

Again use the post-regression `predict` function

It will produce a predicted value when in-sample

It will produce a forecast value when out-of-sample

## Python Code

```
> # Prediction for 2024:Q1, and forecast for 2024:Q2
> newdata = {'l1infl' : [df.infl[-2], df.infl[-1]],
             'l2infl' : [df.infl[-3], df.infl[-2]],
             'l3infl' : [df.infl[-4], df.infl[-3]],
             'l4infl' : [df.infl[-5], df.infl[-4]]}
> ar4.predict(newdata)

0    2.767258
1    3.640964
```