Expressing Steady State Assumptions in Time Series Forecasting: AR and VAR Models

Grant Holtes

January 28, 2024

Summary

This note introduces the application of steady state adjustments in time series modeling, focusing on univariate autoregression (AR) and vector autoregression (VAR) models. It outlines a three-step process involving model estimation, solving for constant terms based on given coefficients and steady state assumptions, and subsequently replacing these terms in the model. The note concludes with an application to a macroeconomic forecasting example, showcasing a $VAR(4, 1)$ model for U.S. economic variables and visualizing the impact of steady state assumptions. The provided Python code in the appendix details the implementation of the adjustments in the steady state.

1 Introduction

When applying time series modelling, it may be required to adjust the steady state of the model to reflect known properties of the system or to apply assumptions on how the steady state will change. For example, consider a model that predicts oil prices and GDP figures for various nations. To test the impact of the gradual increase in oil price from the current state to a assumed value, the practitioner needs to ensure that the price of oil will converge to a given value, while preserving the interactions in the model.

As explored in the following sections, this can be achieved by replacing the constant term in the autoregression model. The basic methodology used is:

1. Estimate the model
2. Given the coefficients on the lags of the variable(s) and the required steady state, solve for the required constant term(s).
3. Replace the constant term(s) in the model with the new estimated ones.

This is shown for the univariate autoregression, $AR(m)$, and full vector autoregression, $VAR(n, m)$, cases.

A more statistically rigorous method to achieve a similar expression of priors in a time series models is to use a Bayesian VAR model (BVAR), as described by Wozniak, 2021 and as applied by Louzis, 2015. The application of BVAR models is beyond the scope of this note.

2 Univariate autoregression

The most basic case is a $AR(1)$ model, where only one variable is predicted and one lag is used, as defined in Equation 1, where $\epsilon_t \sim N(0, \sigma^2)$.

$$x_t = \beta_0 + \beta_1 x_{t-1} + \epsilon_t \tag{1}$$

In the steady state, $x_t = x_{t-1} = x^*$,

$$x^* = \beta_0 + \beta_1 x^* \tag{2}$$

$$\beta_0 = (1 - \beta_1) x^* \tag{3}$$

Given a steady state, x^*, Equation 3 can be used to determine the required constant term. This can be expanded for the case of m lags, as in Equation 6

$$x_t = \beta_0 + \sum_{k=1}^{m} \beta_k x_{t-k} + \epsilon_t \tag{4}$$

$$x^* = \beta_0 + \sum_{k=1}^{m} \beta_k x^* \tag{5}$$

$$\beta_0 = (1 - \sum_{k=1}^{m} \beta_k) x^* \tag{6}$$

3 Vector autoregression

The method used to derive the constant term for the $AR(m)$ case can also be applied to the case of $VAR(n, m)$ models. The simpler $VAR(n, 1)$ case is explored first, where n variables are predicted using one lag of all variables, as in Equation 7.

$$\mathbf{x}_t = \beta_0 + \beta_1 \mathbf{x}_{t-1} + \epsilon_t \tag{7}$$

Where:

- $\mathbf{x}_t$ is a $(n, 1)$ column vector of variables at time t.
- β_0 is a $(n, 1)$ column vector of constants
- ϵ_t is a $(n, 1)$ column vector of random variables, $\sim N(0, \sigma_n^2)$
- β_1 is a (n, n) matrix of coefficients that describe the relationships between the variables.

Again we can define the steady state as the case where $x_t = x_{t-1} = x^*$.

$$\mathbf{x}^* = \beta_0 + \beta_1 \mathbf{x}^* \tag{8}$$

$$\beta_0 = (\mathbf{I} - \beta_1) \mathbf{x}^* \tag{9}$$

Where $\mathbf{I}$ is the (n, n) identity matrix.

This can be extended for the m lag case, as in Equation 11.

$$\mathbf{x}_t = \beta_0 + \sum_{k=1}^{m} \beta_k \mathbf{x}_{t-k} + \epsilon_t \tag{10}$$

$$\beta_0 = (\mathbf{I} - \sum_{k=1}^{m} \beta_k)\mathbf{x}^* \tag{11}$$

4 Application to macroeconomic forecasts

This approach is shown in the simple example of a $VAR(n, 1)$ model on some common macroeconomic variables: real GDP growth, unemployment, inflation, and interest rate. The model graphed in Figure 1 is the 100 month forecast of these variables from July 2017.

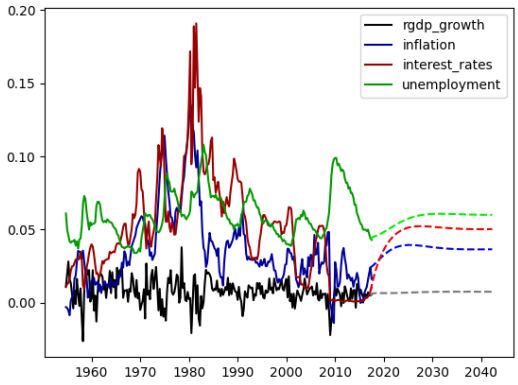

Figure 1: VAR(4,1) model of core macroeconomic variables for the US economy. Data from FRED, n.d.

As is shown in Figure 1, the series revert to their sample means in the steady state. Consider the case that instead that the following steady state assumptions were made and the constant term vector was adjusted as in Equation 9:

Real GDP growth: 1%

Inflation: 2%

Interest rate: 3.5%

Unemployment rate: 5%

This results in the forecast in Figure 2, which shows that the series now converge to these new steady states, but exhibit the same short run interactions and mean reversion properties as the original unmodified model.

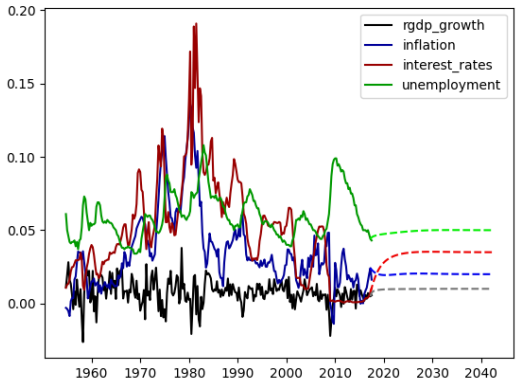

Figure 2: VAR(4,1) model of core macroeconomic variables for the US economy, with steady state assumptions. Data from FRED, n.d.

Code for the adjustment is given in the appendix.

References

FRED. (n.d.). *Economic data* [Accessed via https://fred.stlouisfed.org/].

Louzis, D. P. (2015). Steady-state priors and bayesian variable selection in var forecasting. *Bank of Greece - Working Paper*.

Wozniak, T. (2021). *Bayesian vector autoregressions* [Accessed via https://fbe.unimelb.edu.au/].

5 Code

Python code used is given below.

5.1 VAR model

```
def VAR_1(df):
    Xt_1 = df.shift(1)[1:] # Lag of 1
    Xt_0 = df.iloc[1:] # Lag of 0
    # explantaory vars + const
    X = np.concatenate([np.ones((Xt_1.shape[0],1)), Xt_1], axis=1)
    Y = np.array(Xt_0)
    B, _, _, _ = np.linalg.lstsq(X,Y)
    beta_0 = B[0,:]
    beta_1 = B[1:,:]
    return beta_0, beta_1
```

5.2 Steady state adjustment

```
def adjust_beta_0(beta_0, beta_1, steady_state_target):
    # Find implied steady states at t=inf, where Xt = Xt-1
    # X = X @ beta_1 + beta_0
    # X @ I - X @ beta_1 = beta_0
    # X @ (I - beta_1) = beta_0
    # X = beta_0 @ (I - beta_1)'
    I = np.identity(beta_1.shape[0])
    implied_steady_states = beta_0 @ np.linalg.inv(I - beta_1)
    # If the steady state is not provided, use existing steady state
    for i, m in enumerate(steady_state_target):
        if m == None:
            steady_state_target[i] = implied_steady_states[i]

    # XT = XT @ beta_1 + beta_0T
    # beta_0T = XT - XT @ beta_1
    # beta_0T = XT @ I - XT @ beta_1
    # beta_0T = XT @ (I - beta_1)
    n = beta_1.shape[0]
    beta_0_a = steady_state_target @ (np.identity(n) - beta_1)
    return beta_0_a
```