Documentation for `KalmanFilter.py`: Implementation of MLA, VAR(1), and Random Walk State-Space Models

Based on Buccheri, Corsi & Peluso (2021)

October 14, 2025

Contents

1 Overview

This package provides implementations of several linear-Gaussian state-space models using Kalman filtering, smoothing, and maximum-likelihood or EM estimation:

- The **Multi-Asset Lagged Adjustment (MLA)** model — Buccheri, Corsi & Peluso (2021);
- The standard **Vector Autoregressive (VAR(1))** model;
- The **Random Walk (RW)** model with drift.

All models share common Kalman routines but have their own specific parameterizations and estimation functions.

—

2 Core MLA Functions

(Existing MLA documentation preserved — see previous section for equations and inputs/outputs.)

- `LeadLagKF`, `FastLeadLagKF`, `LeadLagSmoothing`, `LeadLagMLfit`, `LeadLagllem`

—

3 VAR(1) Model Functions

3.1 Model Definition

The VAR(1) model is defined as:

$$\begin{aligned} \boldsymbol{\alpha}_t &= A\boldsymbol{\alpha}_{t-1} + \boldsymbol{\eta}_t, \qquad \boldsymbol{\eta}_t \sim \mathcal{N}(0, Q), \\ \boldsymbol{y}_t &= Z\boldsymbol{\alpha}_t + \boldsymbol{\varepsilon}_t, \qquad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(0, H), \end{aligned}$$

where A is the transition matrix, Z the observation matrix, and Q, H the process and observation noise covariances.

—

3.2 `VAR1_kalman_filter(yt, A, Z, H, Q)`

Purpose: Runs a Kalman filter for the VAR(1) model.

Inputs:

- `yt` ($T \times n$ array): observed multivariate time series.
- `A` ($n \times n$ array): state transition matrix.
- `Z` ($n \times n$ array): observation matrix (usually identity).
- `H` ($n \times n$ array): observation noise covariance matrix.

- `Q` ($n \times n$ array): process noise covariance matrix.

Outputs:

- `att`: filtered state estimates $\hat{\boldsymbol{\alpha}}_{t|t}$.
- `Ptt`: filtered covariance matrices $P_{t|t}$.
- `at`: predicted state means $\hat{\boldsymbol{\alpha}}_{t|t-1}$.
- `Pt`: predicted covariances $P_{t|t-1}$.
- `vt`: innovations $\boldsymbol{y}_t - Z\hat{\boldsymbol{\alpha}}_{t|t-1}$.
- `Ft`: innovation covariance matrices.
- `Kt`: Kalman gain matrices.
- `loglike`: total log-likelihood value.

—

3.3 `VAR1Smoothing(yt, A, Z, att, Ptt, Pt, vt, Ft, Kt)`

Purpose: Applies backward Kalman smoothing for the VAR(1) model.
Inputs:

- Filter outputs from `VAR1_kalman_filter`.

Outputs:

- `x_smooth`: smoothed state means $\hat{\boldsymbol{\alpha}}_{t|T}$.
- `V_smooth`: smoothed state covariances $V_{t|T}$.
- `Vt_smooth`: smoothed lag-one covariances.

—

3.4 `VAR1_ml_fit(params, yt)`

Purpose: Estimates A, H, Q by maximizing the Kalman filter log-likelihood.
Inputs:

- `params` (1D array): initial vectorized parameters.
- `yt` ($T \times n$ array): observed time series.

Outputs:

- `A_mlfit`: estimated transition matrix.
- `H_mlfit`: estimated observation covariance matrix.
- `Q_mlfit`: estimated process covariance matrix.

—

3.5 `VAR1_em_fit(A0, H0, Q0, yt, maxiter=1000, tol=10^{-6})`

Purpose: Estimates A, H, Q via the EM algorithm.

Inputs:

- `A0`, `H0`, `Q0`: initial guesses for system matrices.
- `yt`: observed series.
- `maxiter` (int): maximum EM iterations.
- `tol` (float): convergence tolerance on log-likelihood.

Outputs:

- Updated estimates (A, H, Q).
- Smoothed states and log-likelihood trajectory.

—

3.6 Example (VAR(1) Simulation and Estimation)

```
from KalmanFilter import VAR1_kalman_filter, VAR1Smoothing, VAR1_ml_fit

A = np.eye(2)
Z = np.eye(2)
H = np.array([[0.2**2, 0], [0, 0.21**2]])
Q = np.array([[0.3**2, 0.4*0.3*0.31],
              [0.4*0.3*0.31, 0.31**2]])

yt = np.random.multivariate_normal([0,0], H, 1000)
att, Ptt, at, Pt, vt, Ft, Kt, loglike = VAR1_kalman_filter(yt, A, Z, H,
    Q)
x_smooth, V_smooth, Vt_smooth = VAR1Smoothing(yt, A, Z, att, Ptt, Pt,
    vt, Ft, Kt)
A_hat, H_hat, Q_hat = VAR1_ml_fit(np.ones(12), yt)
```

—

4 Random Walk (RW) Model Functions

4.1 Model Definition

The Random Walk (RW) model is a special case of the VAR(1) model with $A = I_n$ and optional drift $\boldsymbol{\mu}$:

$$\begin{aligned}\boldsymbol{\alpha}_t &= \boldsymbol{\mu} + \boldsymbol{\alpha}_{t-1} + \boldsymbol{\eta}_t, \quad \boldsymbol{\eta}_t \sim \mathcal{N}(0, Q),\\ \boldsymbol{y}_t &= \boldsymbol{\alpha}_t + \boldsymbol{\varepsilon}_t, \quad \boldsymbol{\varepsilon}_t \sim \mathcal{N}(0, H).\end{aligned}$$

—

4.2 `RW_kalman_filter(yt, drift, H, Q)`

Purpose: Runs a Kalman filter for a multivariate random-walk process with drift.

Inputs:

- `yt` ($T \times n$ array): observed series.
- `drift` (n-vector): drift vector $\boldsymbol{\mu}$.
- `H` ($n \times n$ array): observation noise covariance.
- `Q` ($n \times n$ array): process noise covariance.

Outputs:

- `att`: filtered states $\hat{\boldsymbol{\alpha}}_{t|t}$.
- `Ptt`: filtered covariances.
- `at`, `Pt`: predictions.
- `vt`, `Ft`, `Kt`: innovations, innovation covariances, and gains.
- `loglike`: log-likelihood value.

—

4.3 `RW_ml_fit(params, yt)`

Purpose: Performs maximum-likelihood estimation of drift, H, and Q parameters for the RW model.

Inputs:

- `params` (1D array): initial guess vector combining

$$(\text{drift}, \sigma_H, \sigma_Q, \rho_Q)$$

- `yt`: observed series.

Outputs:

- `drift_mlfit`: estimated drift vector $\hat{\boldsymbol{\mu}}$.
- `H_mlfit`: estimated observation covariance.
- `Q_mlfit`: estimated process covariance.

—

4.4 Example (RW Simulation and Estimation)

```
from KalmanFilter import RW_kalman_filter, RW_ml_fit, matrix_generator
import numpy as np

n_assets, n_steps = 4, 600
Ht = np.diag(np.random.uniform(1e-4, 1e-3, n_assets))
s_eta = np.random.uniform(1e-3, .2, n_assets)
rho_eta = np.random.uniform(-.5, .5, int(n_assets*(n_assets-1)/2))
Qt = matrix_generator(np.hstack((s_eta, rho_eta)), n_assets)
drift = np.random.uniform(-1e-2, 2.5e-2, n_assets)

# Simulate random walk
yt = np.zeros((n_steps, n_assets))
alpha = np.zeros((n_steps+1, n_assets))
for t in range(1, n_steps+1):
    alpha[t] = drift + alpha[t-1] + np.random.multivariate_normal(np.
        zeros(n_assets), Qt)
    yt[t-1] = alpha[t-1] + np.random.multivariate_normal(np.zeros(
        n_assets), Ht)

# Filter and ML fit
att, Ptt, at, Pt, vt, Ft, Kt, loglike = RW_kalman_filter(yt, drift, Ht,
    Qt)
params = np.hstack((drift, np.diag(Ht), s_eta, rho_eta))
drift_ml, H_ml, Q_ml = RW_ml_fit(params, yt)
```

—

5 References

- Buccheri, G., Corsi, F., & Peluso, S. (2021). *High-Frequency Lead-Lag Effects and Cross-Asset Linkages: A Multi-Asset Lagged Adjustment Model. Journal of Business & Economic Statistics*, 39(3), 605–621.
- Durbin, J. & Koopman, S. J. (2012). *Time Series Analysis by State Space Methods.* Oxford University Press.
- Shumway, R. H. & Stoffer, D. S. (2015). *Time Series Analysis and Its Applications.*