Stock Market Prediction Using Kalman Filter

Shawn Hagler & Jaehyun Lee

Introduction to Kalman Filters

Kalman Filters are a recursive algorithm for estimating the state of a dynamic system from noisy measurements.

- Developed by Rudolf E. Kálmán in 1960
- Widely used in various fields, such as aerospace, robotics, and finance

Advantages of Kalman Filters:

- Handle noisy data well by combining prior knowledge with current observations
- Adapt over time, adjusting to changes in the underlying system
- Incorporate both past and current observations to update state estimates

In the context of stock price prediction, Kalman Filters can be used to model the hidden underlying factors that influence stock prices and update the predicted prices based on new observations.

Derivation from Markov Models to Kalman Filters

Markov Models:

 - Assumes that the future state depends only on the current state and not on the previous states

 - Can be used to model simple financial systems, but may not capture the complexity of stock price movements

Transition to Kalman Filters:

 - Kalman Filters are an extension of Markov models for systems with continuous states and noisy observations

 - Derive from the Bayesian framework for state estimation, providing a more sophisticated approach to modeling dynamic systems

 - In the context of stock price prediction, Kalman Filters model the hidden underlying factors that influence stock prices, while accounting for the noise in the observations

By transitioning from Markov models to Kalman Filters, we can better capture the dynamics of stock prices and improve the accuracy of our predictions.

Markov Process: $P(x_t \mid x_{t-1}, x_{t-2}, \cdots, x_0) = P(x_t \mid x_{t-1})$

Hidden Markov Model: $P(x_t, y_t \mid x_{t-1}) = P(x_t \mid x_{t-1}) P(y_t \mid x_t)$

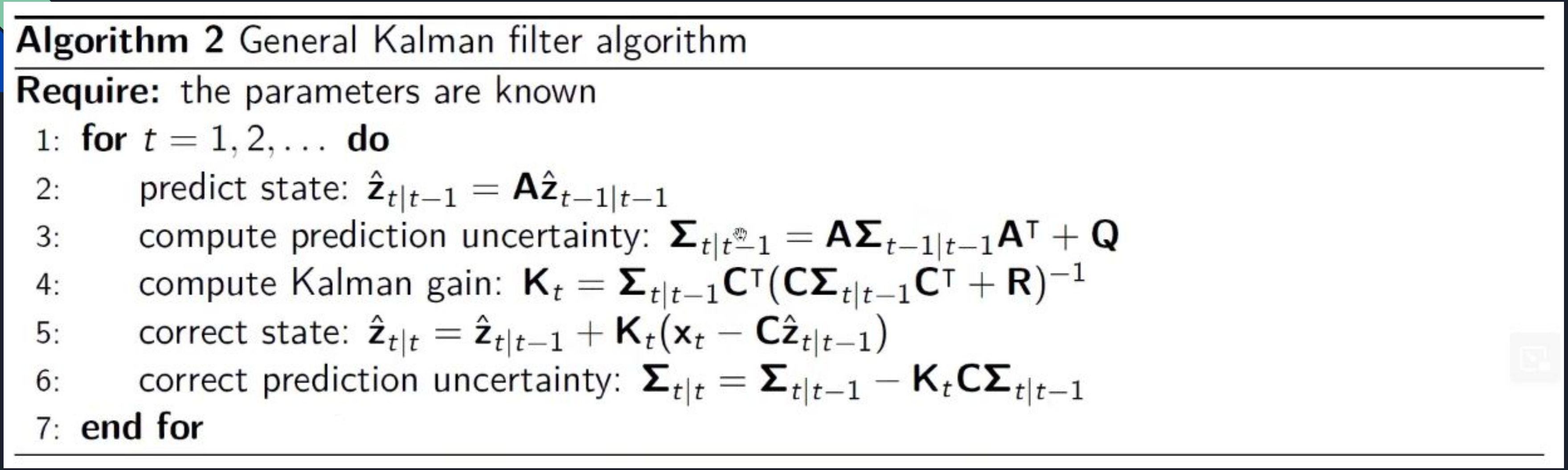
Kalman Filter:

- Prediction Step => $\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1}$ $P_{t|t-1} = AP_{t-1|t-1}A^T + Q$
- Update Step =>

$$K_t = P_{t|t-1}C^T(CP_{t|t-1}C^T + R)^{-1}$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(y_t - C\hat{x}_{t|t-1})$$

$$P_{t|t} = (I - K_tC)P_{t|t-1}$$



Algorithm 2 General Kalman filter algorithm

```
Require: the parameters are known
1: for $t = 1, 2, \dots$ do
2:     predict state: $\hat{\mathbf{z}}_{t|t-1} = \mathbf{A}\hat{\mathbf{z}}_{t-1|t-1}$
3:     compute prediction uncertainty: $\mathbf{\Sigma}_{t|t-1} = \mathbf{A}\mathbf{\Sigma}_{t-1|t-1}\mathbf{A}^\top + \mathbf{Q}$
4:     compute Kalman gain: $\mathbf{K}_t = \mathbf{\Sigma}_{t|t-1}\mathbf{C}^\top(\mathbf{C}\mathbf{\Sigma}_{t|t-1}\mathbf{C}^\top + \mathbf{R})^{-1}$
5:     correct state: $\hat{\mathbf{z}}_{t|t} = \hat{\mathbf{z}}_{t|t-1} + \mathbf{K}_t(\mathbf{x}_t - \mathbf{C}\hat{\mathbf{z}}_{t|t-1})$
6:     correct prediction uncertainty: $\mathbf{\Sigma}_{t|t} = \mathbf{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{C}\mathbf{\Sigma}_{t|t-1}$
7: end for
```

Advantages and Disadvantages

Advantages:

- Allows for noise in initial estimate, state transitions, and observations
- Historical data isn’t required to be stored
- Computationally efficient

Disadvantages:

- Is for linear-Gaussian state space models (in practice most state transitions would be non-linear or noise is non-gaussian)

Problem Definition

Goal: Predict future stock prices using Kalman Filters

- Improve decision-making for trading strategies
- Enhance portfolio management and risk assessment

Dataset: Historical stock price data

- Daily closing prices
- Example: Verizon Communications Inc. (VZ) stock prices over one year

Approach:

- Apply Kalman Filters to model the hidden state (log-transformed prices)
- Optimize Kalman Filter parameters using the Expectation-Maximization (EM) algorithm
- Predict future stock prices based on the estimated hidden state

Introduction to Trading

Trading is buying and selling financial instruments to profit from market fluctuations

Stock price prediction is crucial for informed investment decisions and strategy development

Common prediction methods:

- Technical analysis: Historical price data analysis
- Fundamental analysis: Company financial health evaluation
- Quantitative analysis: Mathematical models
- Machine learning: Data-driven algorithms

Observation and Hidden State

Observation: The actual stock prices, which are the data we can observe and collect.

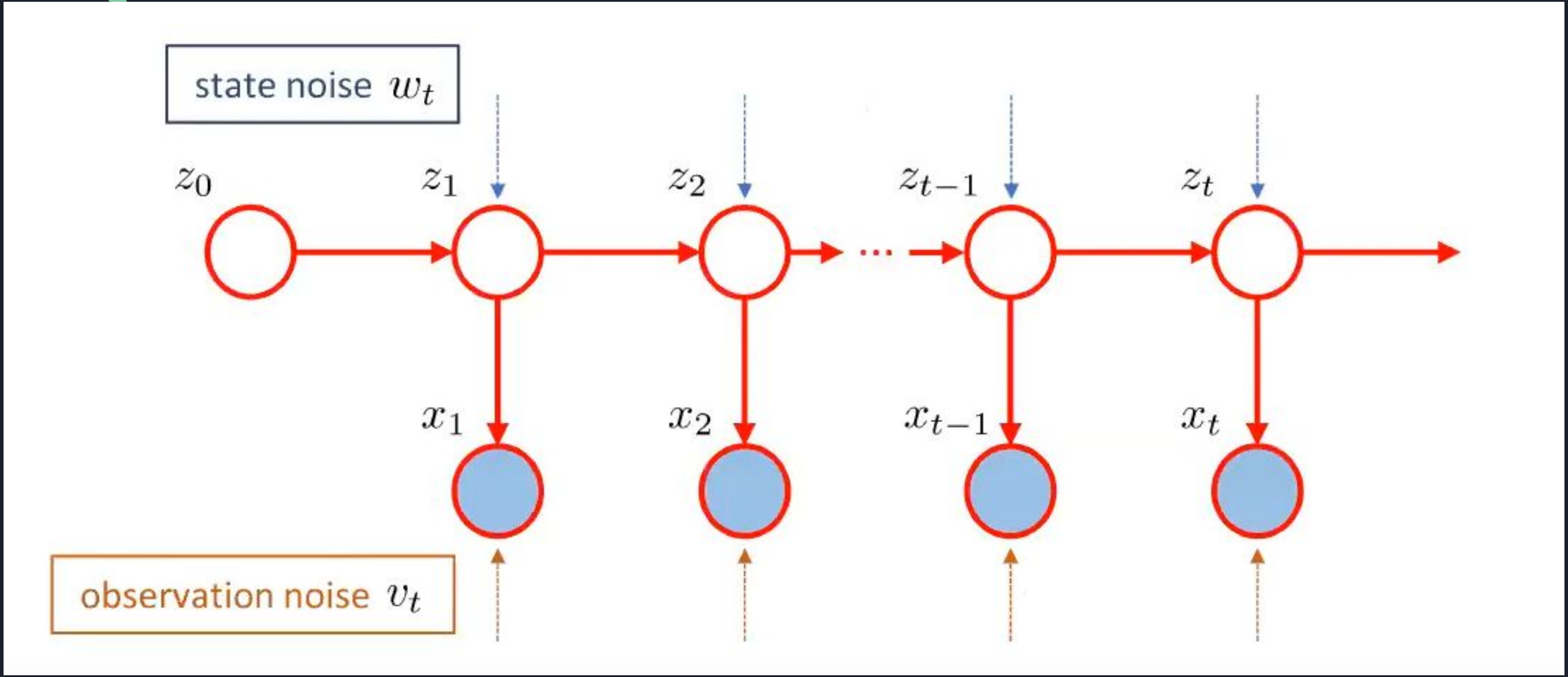
Hidden state: The underlying log-transformed stock prices, which are the true values we are trying to estimate.

- Represents the underlying process driving the stock prices, and it evolves linearly over time.

The Kalman Filter is designed to estimate the hidden state based on the observations while accounting for the inherent noise in the data.

Estimating the hidden state can be useful for tracking the underlying trend, predicting future stock prices, and identifying anomalies in the stock price data.

State Space Model



Transition Matrix Output

The transition matrix is a matrix that describes the evolution of the hidden state from time t-1 to time t.

Role in the Kalman Filter:

- The transition matrix is a key component in the prediction step of the Kalman Filter

- It helps to estimate the hidden state at the next time step based on the current hidden state and the system's dynamics

- Predicted state mean: $\hat{\mu}_{t|t-1} = A \cdot \mu_{t-1|t-1}$

- Predicted state covariance: $\hat{P}_{t|t-1} = A \cdot P_{t-1|t-1} \cdot A^T + Q$

Since the transition matrix is 1, the predicted state mean at time t is equal to the updated state mean at time t-1

The simplicity of the transition matrix allows for a straightforward and efficient estimation of the hidden state's future values

Expectation-Maximization (EM) Algorithm

Purpose: Optimize the parameters of the Kalman Filter to improve prediction accuracy

Two-step iterative process:

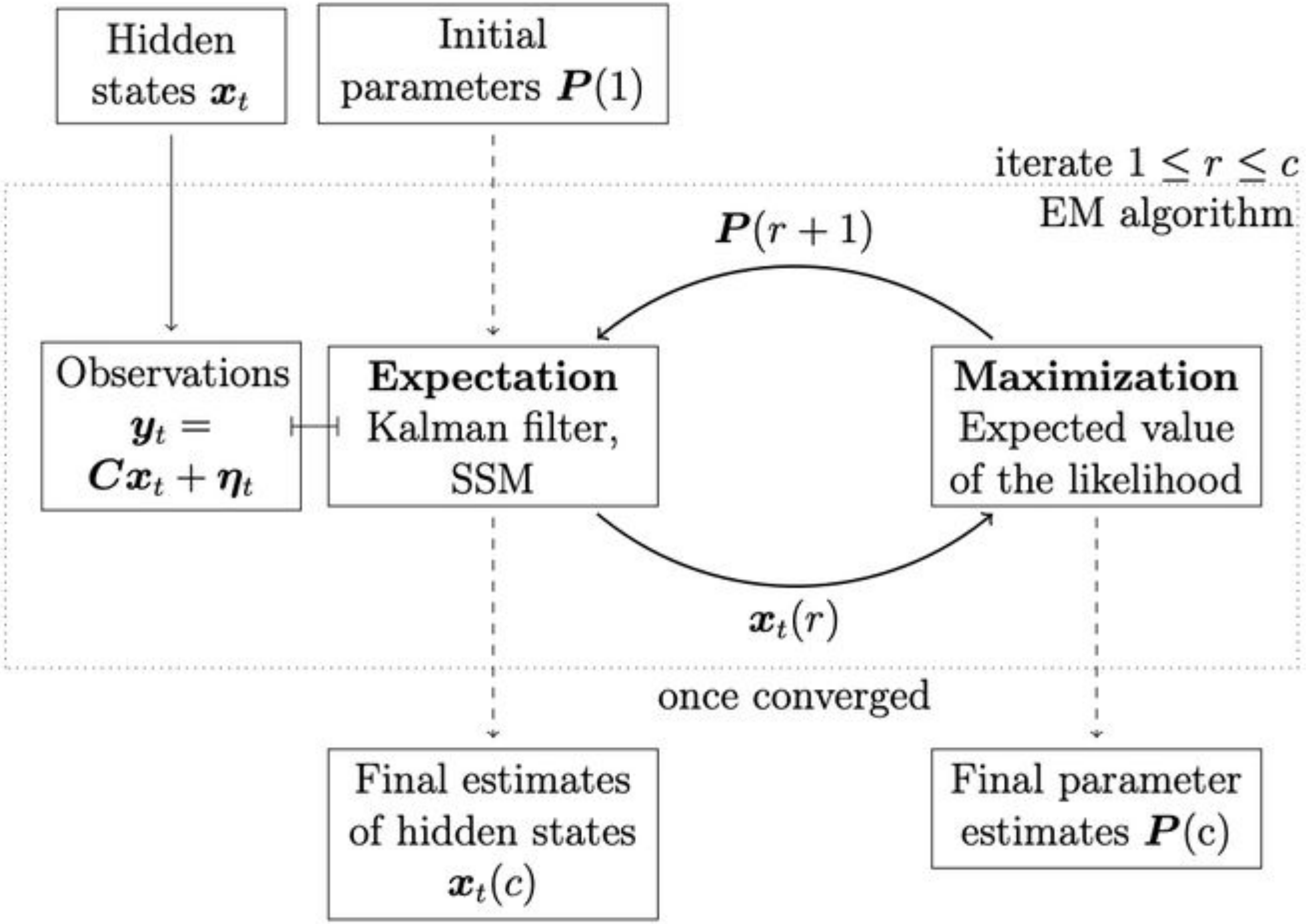
- Expectation step (E-step): Estimate the hidden state using the current model parameters
- Maximization step (M-step): Update the model parameters to maximize the likelihood of the observed data given the current hidden state estimates

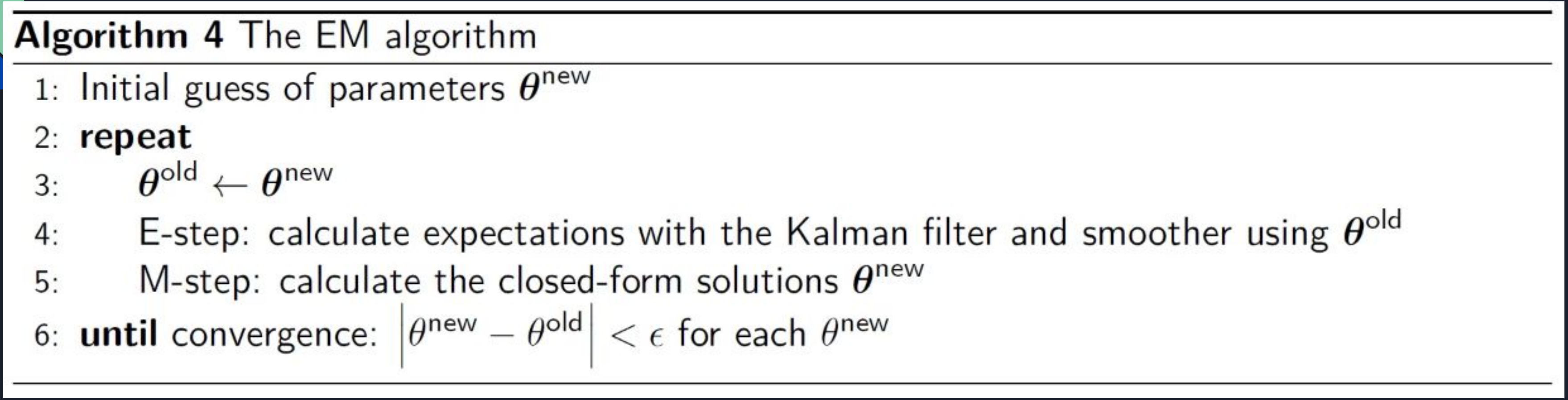
Advantages:

- Handles missing or incomplete data well
- Converges to a local optimum, providing improved predictions

Implementation:

- Iterate through the E-step and M-step for a fixed number of iterations or until convergence
- Use the optimized parameters obtained from the EM algorithm for final stock price prediction


Hidden states $\boldsymbol{x}_t$
Initial parameters $\boldsymbol{P}(1)$
iterate $1 \leq r \leq c$
EM algorithm
$\boldsymbol{P}(r+1)$
Observations $\boldsymbol{y}_t = \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{\eta}_t$
Expectation Kalman filter, SSM
Maximization Expected value of the likelihood
$\boldsymbol{x}_t(r)$
once converged
Final estimates of hidden states $\boldsymbol{x}_t(c)$
Final parameter estimates $\boldsymbol{P}(c)$



Algorithm 4 The EM algorithm

```
1: Initial guess of parameters $\boldsymbol{\theta}^{\text{new}}$
2: repeat
3:     $\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$
4:     E-step: calculate expectations with the Kalman filter and smoother using $\boldsymbol{\theta}^{\text{old}}$
5:     M-step: calculate the closed-form solutions $\boldsymbol{\theta}^{\text{new}}$
6: until convergence: $\left|\theta^{\text{new}} - \theta^{\text{old}}\right| < \epsilon$ for each $\theta^{\text{new}}$
```

Implementation

1) Log-transform prices

- Convert stock prices to log prices: $log\_prices_t = \log(prices_t)$

- Purpose: Stabilize variance and make the time series more stationary

2) Initialize Kalman Filter

3) Optimize parameters using the EM algorithm

- Apply the Expectation-Maximization (EM) algorithm to find the optimal parameters for the Kalman Filter

- Iterate the process for a fixed number of times (e.g., 10 iterations)

4) Predict future prices

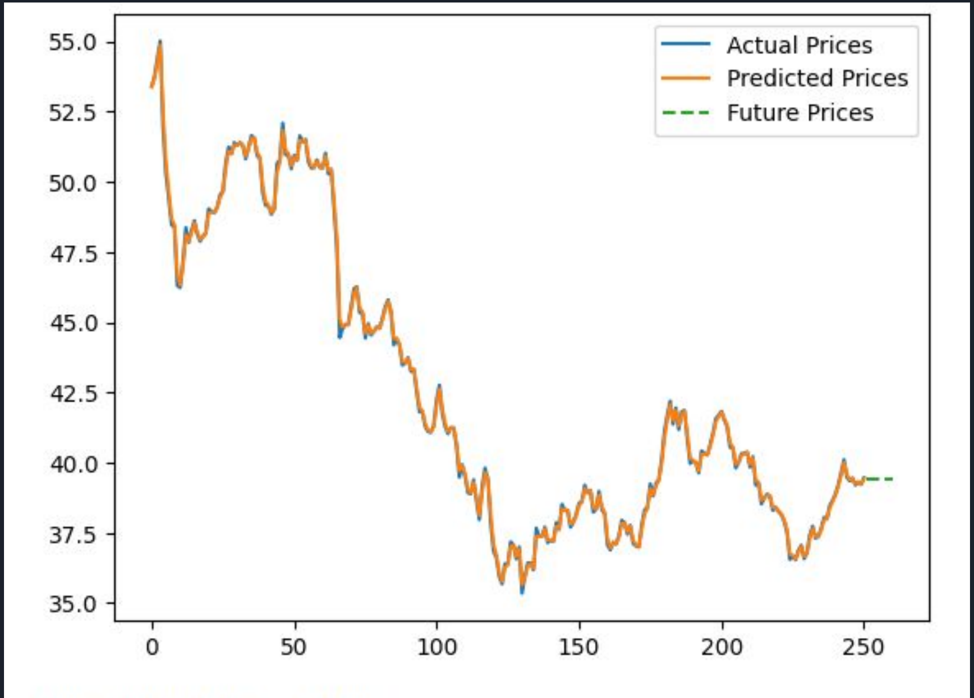
$$predicted\_prices_t = \exp(state\_means_t)$$

- Use the Kalman Filter to estimate the hidden state for each time step

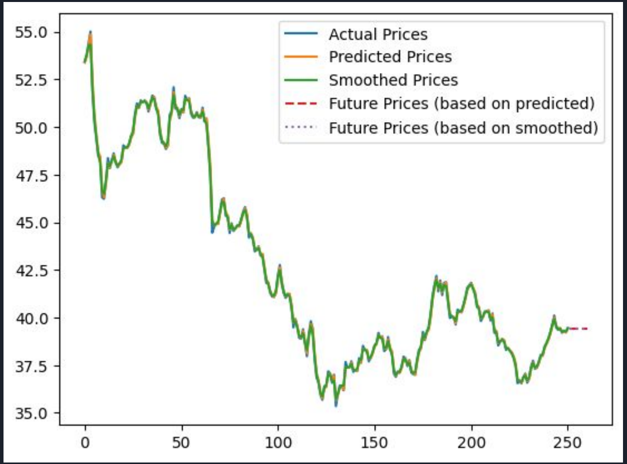
- Convert the hidden state (log prices) back to actual prices:

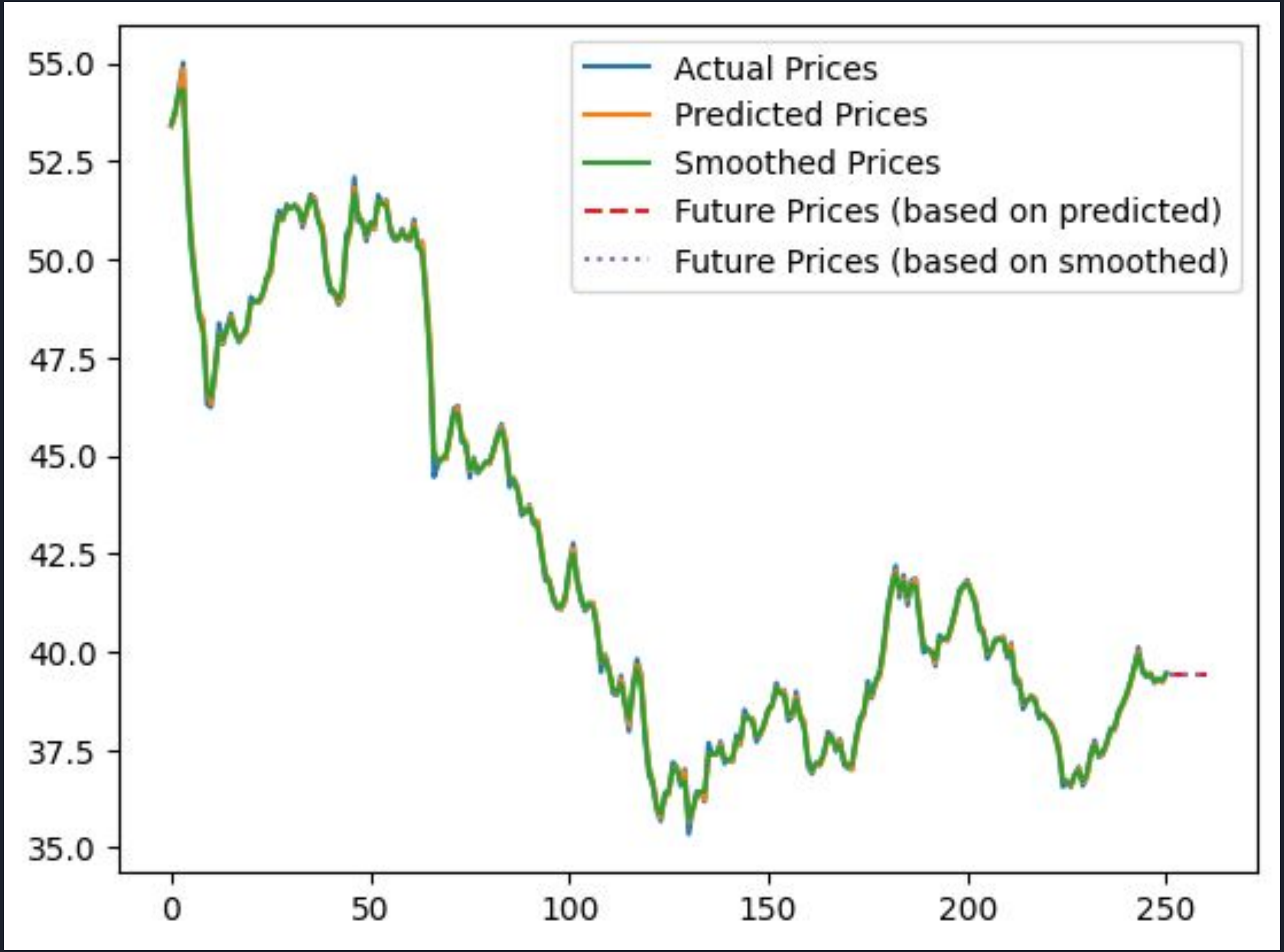
- Predict future prices using the optimized Kalman Filter parameters

Results

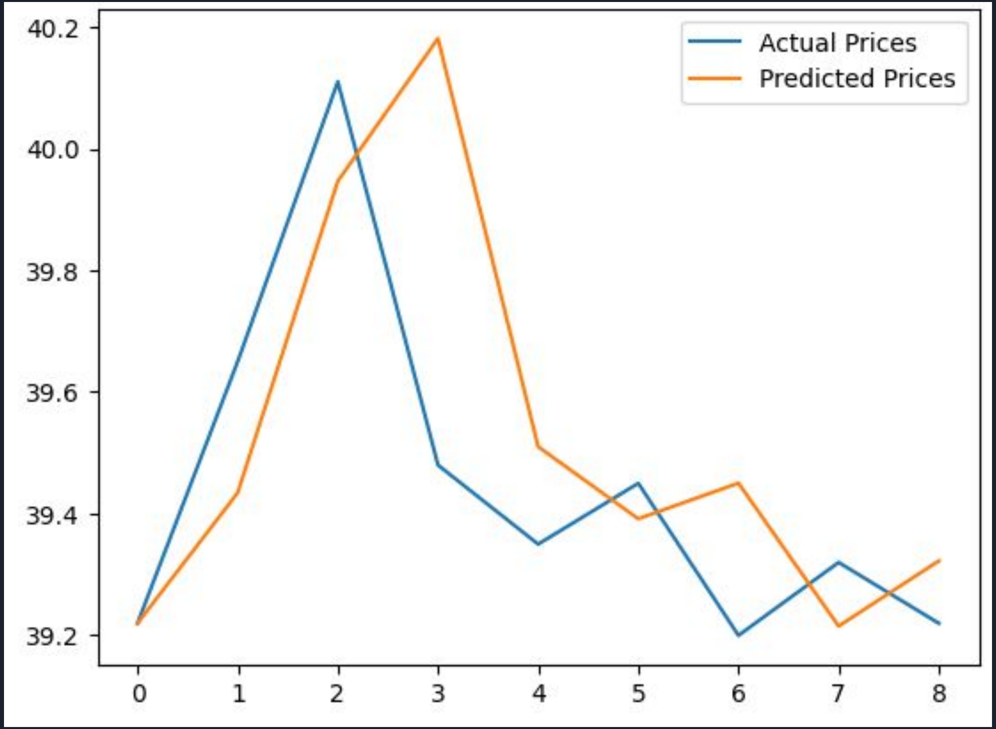


```
Mean Square Error: 0.0160
```

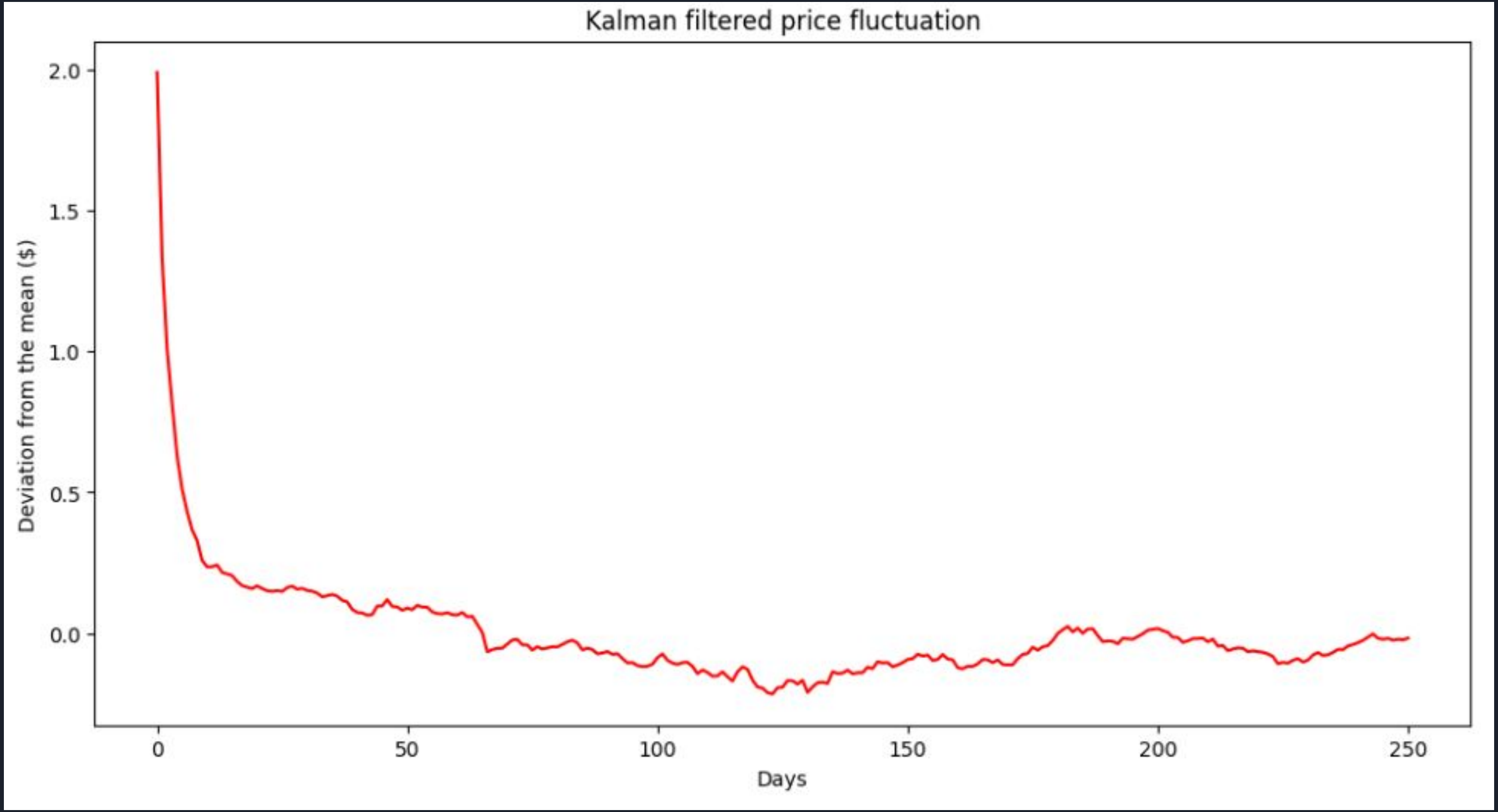



Actual Prices
Predicted Prices
Smoothed Prices
Future Prices (based on predicted)
Future Prices (based on smoothed)
55.0
52.5
50.0
47.5
45.0
42.5
40.0
37.5
35.0
0
50
100
150
200
250

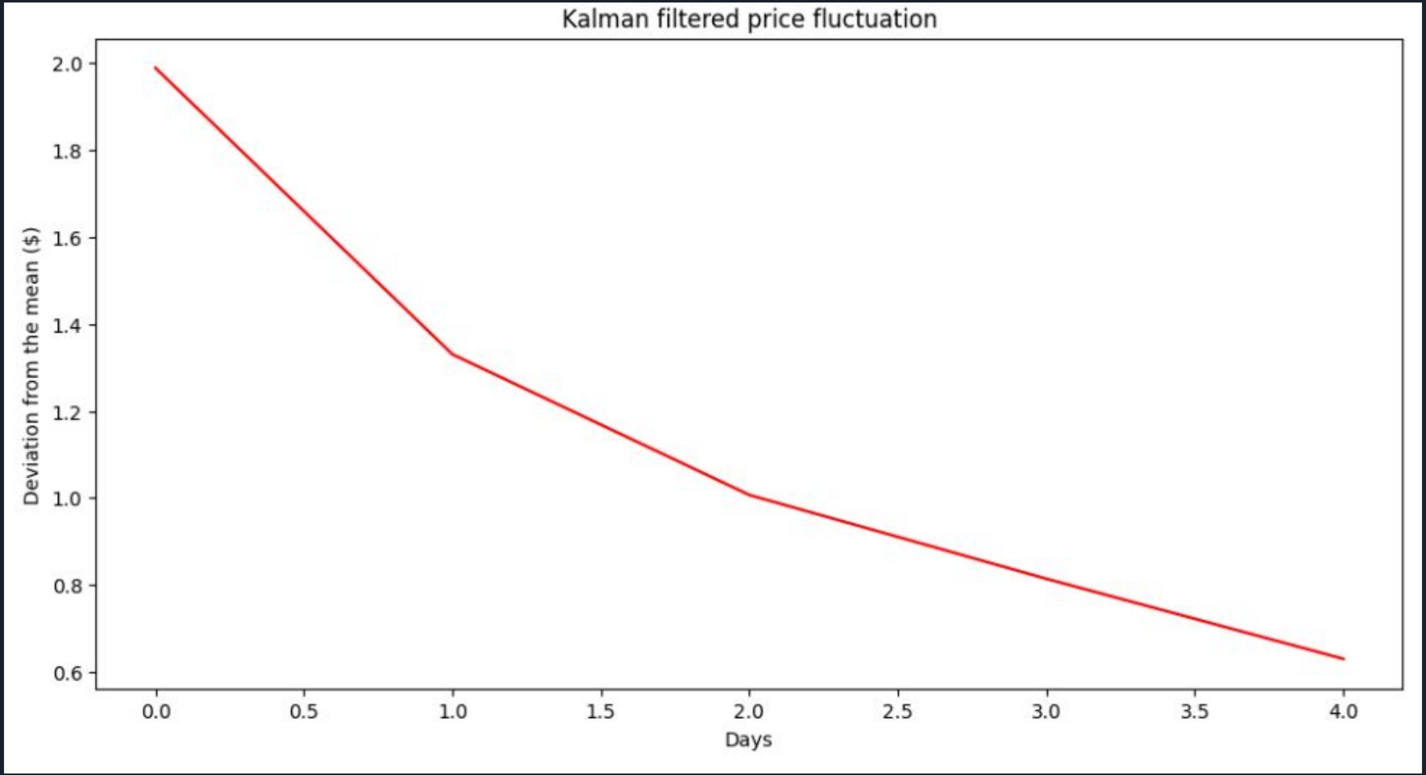
Results (cont.)

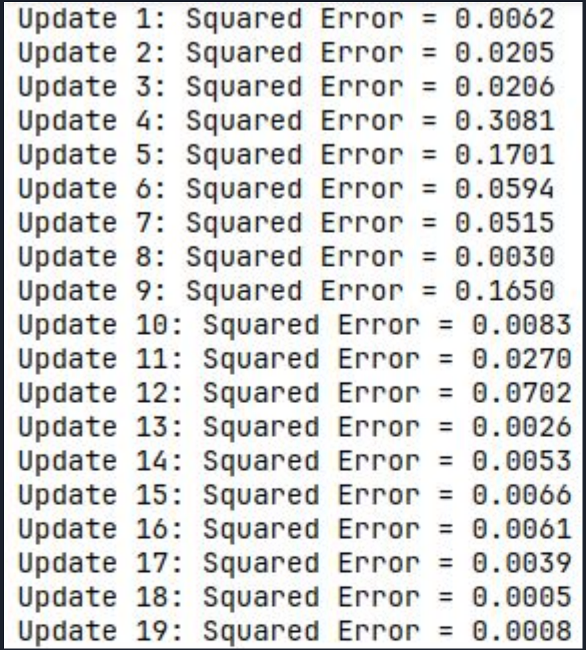


Results (cont.)

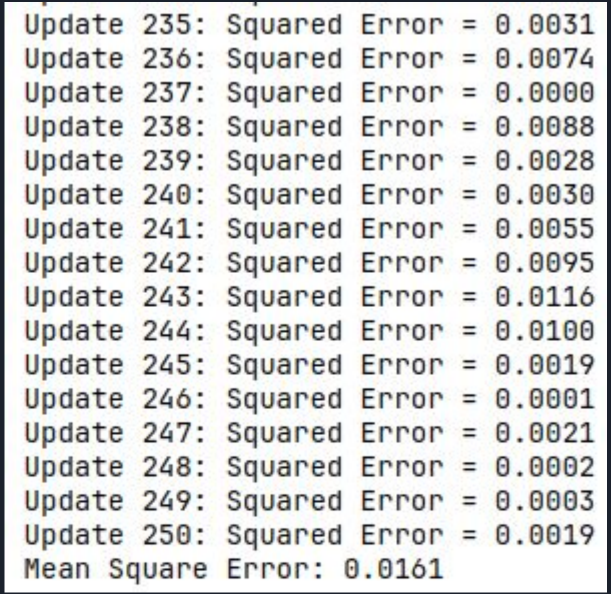


Results (cont.)

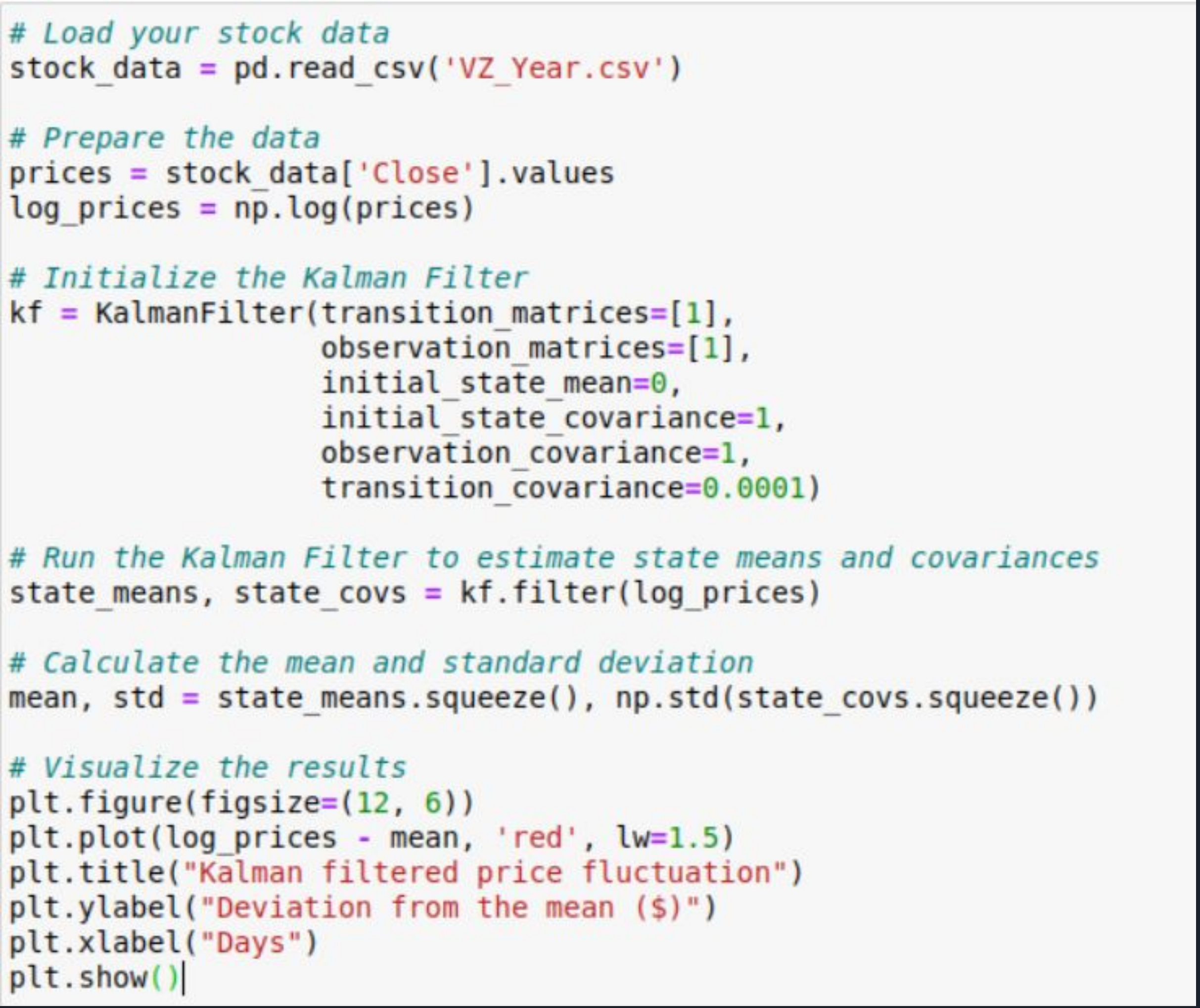




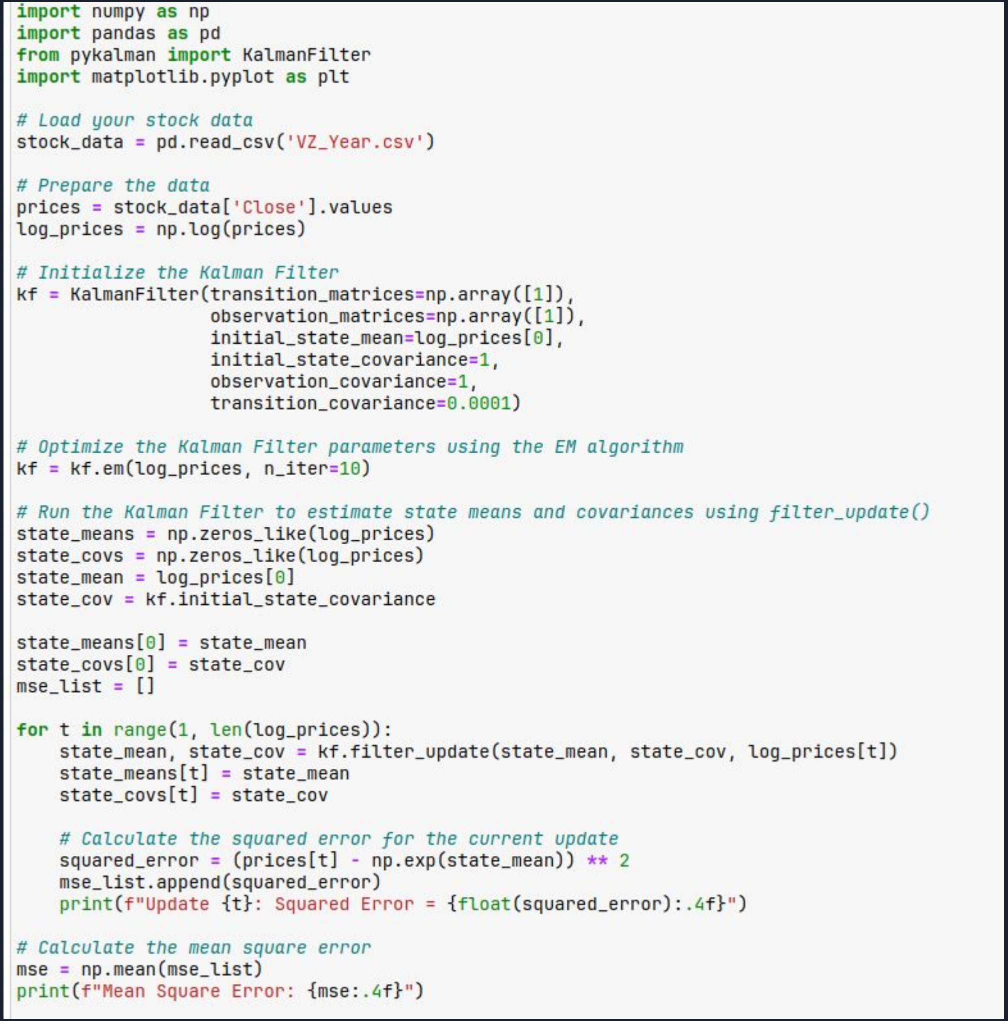
```
Update 1: Squared Error = 0.0062
Update 2: Squared Error = 0.0205
Update 3: Squared Error = 0.0206
Update 4: Squared Error = 0.3081
Update 5: Squared Error = 0.1701
Update 6: Squared Error = 0.0594
Update 7: Squared Error = 0.0515
Update 8: Squared Error = 0.0030
Update 9: Squared Error = 0.1650
Update 10: Squared Error = 0.0083
Update 11: Squared Error = 0.0270
Update 12: Squared Error = 0.0702
Update 13: Squared Error = 0.0026
Update 14: Squared Error = 0.0053
Update 15: Squared Error = 0.0066
Update 16: Squared Error = 0.0061
Update 17: Squared Error = 0.0039
Update 18: Squared Error = 0.0005
Update 19: Squared Error = 0.0008
```



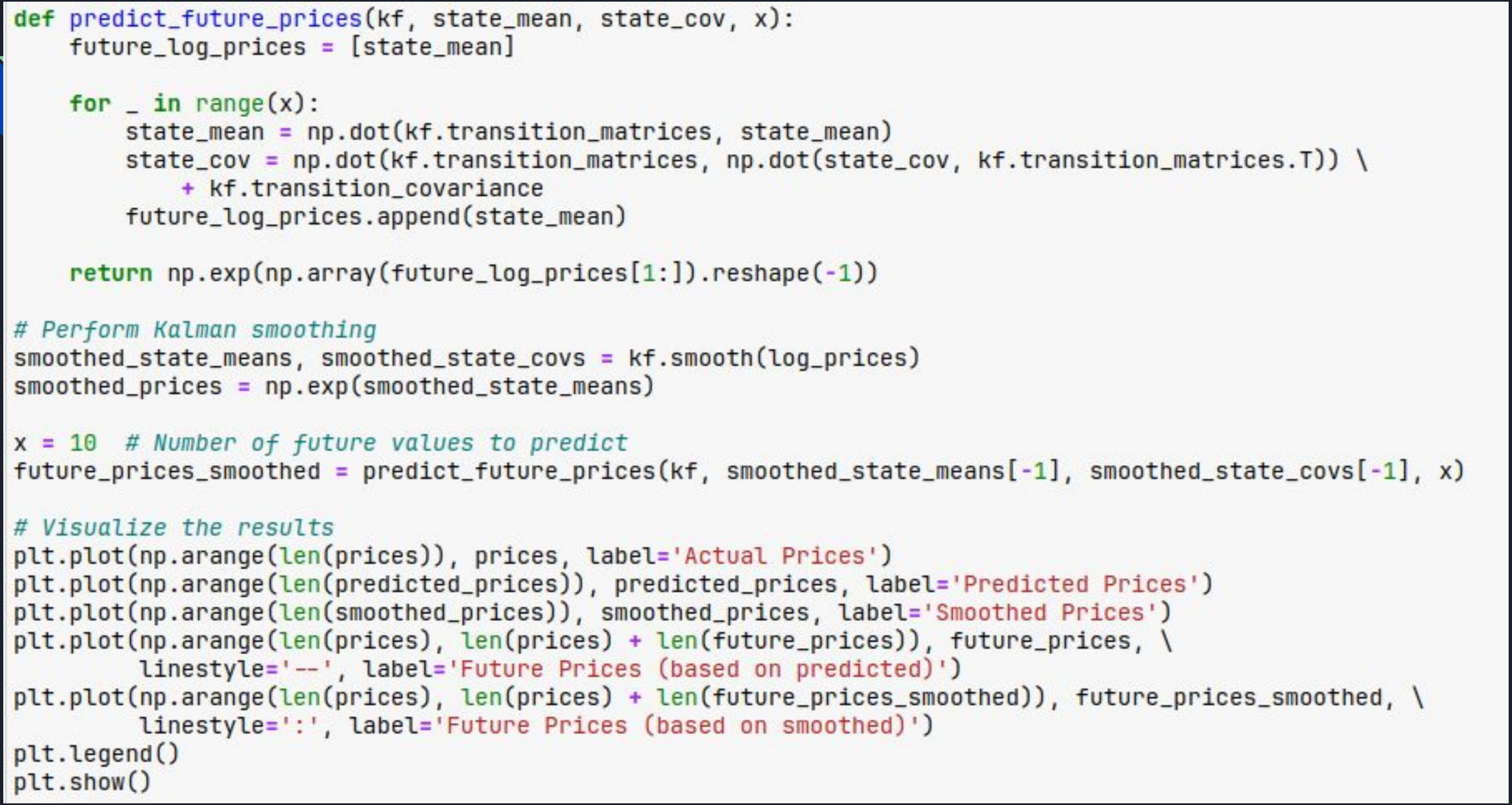
```
Update 235: Squared Error = 0.0031
Update 236: Squared Error = 0.0074
Update 237: Squared Error = 0.0000
Update 238: Squared Error = 0.0088
Update 239: Squared Error = 0.0028
Update 240: Squared Error = 0.0030
Update 241: Squared Error = 0.0055
Update 242: Squared Error = 0.0095
Update 243: Squared Error = 0.0116
Update 244: Squared Error = 0.0100
Update 245: Squared Error = 0.0019
Update 246: Squared Error = 0.0001
Update 247: Squared Error = 0.0021
Update 248: Squared Error = 0.0002
Update 249: Squared Error = 0.0003
Update 250: Squared Error = 0.0019
Mean Square Error: 0.0161
```



```python
# Load your stock data
stock_data = pd.read_csv('VZ_Year.csv')

# Prepare the data
prices = stock_data['Close'].values
log_prices = np.log(prices)

# Initialize the Kalman Filter
kf = KalmanFilter(transition_matrices=[1],
                  observation_matrices=[1],
                  initial_state_mean=0,
                  initial_state_covariance=1,
                  observation_covariance=1,
                  transition_covariance=0.0001)

# Run the Kalman Filter to estimate state means and covariances
state_means, state_covs = kf.filter(log_prices)

# Calculate the mean and standard deviation
mean, std = state_means.squeeze(), np.std(state_covs.squeeze())

# Visualize the results
plt.figure(figsize=(12, 6))
plt.plot(log_prices - mean, 'red', lw=1.5)
plt.title("Kalman filtered price fluctuation")
plt.ylabel("Deviation from the mean ($)")
plt.xlabel("Days")
plt.show()
```



```python
import numpy as np
import pandas as pd
from pykalman import KalmanFilter
import matplotlib.pyplot as plt

# Load your stock data
stock_data = pd.read_csv('VZ_Year.csv')

# Prepare the data
prices = stock_data['Close'].values
log_prices = np.log(prices)

# Initialize the Kalman Filter
kf = KalmanFilter(transition_matrices=np.array([1]),
                  observation_matrices=np.array([1]),
                  initial_state_mean=log_prices[0],
                  initial_state_covariance=1,
                  observation_covariance=1,
                  transition_covariance=0.0001)

# Optimize the Kalman Filter parameters using the EM algorithm
kf = kf.em(log_prices, n_iter=10)

# Run the Kalman Filter to estimate state means and covariances using filter_update()
state_means = np.zeros_like(log_prices)
state_covs = np.zeros_like(log_prices)
state_mean = log_prices[0]
state_cov = kf.initial_state_covariance

state_means[0] = state_mean
state_covs[0] = state_cov
mse_list = []

for t in range(1, len(log_prices)):
    state_mean, state_cov = kf.filter_update(state_mean, state_cov, log_prices[t])
    state_means[t] = state_mean
    state_covs[t] = state_cov

    # Calculate the squared error for the current update
    squared_error = (prices[t] - np.exp(state_mean)) ** 2
    mse_list.append(squared_error)
    print(f"Update {t}: Squared Error = {float(squared_error):.4f}")

# Calculate the mean square error
mse = np.mean(mse_list)
print(f"Mean Square Error: {mse:.4f}")
```



```python
def predict_future_prices(kf, state_mean, state_cov, x):
    future_log_prices = [state_mean]

    for _ in range(x):
        state_mean = np.dot(kf.transition_matrices, state_mean)
        state_cov = np.dot(kf.transition_matrices, np.dot(state_cov, kf.transition_matrices.T)) \
            + kf.transition_covariance
        future_log_prices.append(state_mean)

    return np.exp(np.array(future_log_prices[1:]).reshape(-1))

# Perform Kalman smoothing
smoothed_state_means, smoothed_state_covs = kf.smooth(log_prices)
smoothed_prices = np.exp(smoothed_state_means)

x = 10  # Number of future values to predict
future_prices_smoothed = predict_future_prices(kf, smoothed_state_means[-1], smoothed_state_covs[-1], x)

# Visualize the results
plt.plot(np.arange(len(prices)), prices, label='Actual Prices')
plt.plot(np.arange(len(predicted_prices)), predicted_prices, label='Predicted Prices')
plt.plot(np.arange(len(smoothed_prices)), smoothed_prices, label='Smoothed Prices')
plt.plot(np.arange(len(prices), len(prices) + len(future_prices)), future_prices, \
         linestyle='--', label='Future Prices (based on predicted)')
plt.plot(np.arange(len(prices), len(prices) + len(future_prices_smoothed)), future_prices_smoothed, \
         linestyle=':', label='Future Prices (based on smoothed)')
plt.legend()
plt.show()
```

Conclusion

Potential Improvements and Extensions:

- Incorporate additional features, such as technical indicators, to enhance the model's predictive power

- Test different optimization algorithms for refining the Kalman Filter parameters

- Apply the method to other financial data, such as exchange rates or market indices, to evaluate its versatility and effectiveness across various assets

The use of Kalman Filters for stock price prediction holds promise, and with further refinement and exploration, it can potentially contribute to more accurate and effective trading strategies.