

Kalman Filter for Financial Time Series

Technical Implementation Guide

Precog Quant Task 2026

Technical Documentation

February 9, 2026

Contents

1 Introduction

The Kalman Filter is a recursive algorithm for estimating the state of a dynamic system from noisy observations. In quantitative finance, it serves multiple purposes:

- **Signal Extraction:** Separating “true” price trends from market noise
- **Uncertainty Quantification:** Providing confidence bounds on estimates
- **Feature Generation:** Creating predictive features for machine learning models
- **Dynamic Hedge Ratios:** Estimating time-varying relationships (pairs trading)

1.1 Why Kalman Filter for Trading?

Traditional moving averages suffer from:

- Fixed lag (EMA) or equal weighting (SMA)
- No formal uncertainty quantification
- No adaptive response to volatility regimes

The Kalman Filter addresses these by:

- Optimally weighting new information based on noise estimates
- Providing state covariance as uncertainty measure
- Adapting its “gain” based on observation reliability

2 Mathematical Foundation

2.1 State-Space Representation

Definition 1 (Linear Gaussian State-Space Model). A discrete-time linear Gaussian state-space model consists of:

$$\mathbf{x}_t = \mathbf{F}_t \mathbf{x}_{t-1} + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \quad (1)$$

$$\mathbf{y}_t = \mathbf{H}_t \mathbf{x}_t + \mathbf{v}_t, \quad \mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \quad (2)$$

where:

- $\mathbf{x}_t \in \mathbb{R}^n$ is the hidden state vector
- $\mathbf{y}_t \in \mathbb{R}^m$ is the observation vector
- $\mathbf{F}_t \in \mathbb{R}^{n \times n}$ is the state transition matrix
- $\mathbf{H}_t \in \mathbb{R}^{m \times n}$ is the observation matrix
- $\mathbf{Q}_t \in \mathbb{R}^{n \times n}$ is the process noise covariance
- $\mathbf{R}_t \in \mathbb{R}^{m \times m}$ is the observation noise covariance

2.2 Local Level Model (Random Walk + Noise)

For financial applications, we use the simplest meaningful model:

$$x_t = x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, Q) \quad (\text{State: "true" price level}) \quad (3)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, R) \quad (\text{Observation: noisy price}) \quad (4)$$

This is a scalar version with:

- $F = 1$ (state persists)
- $H = 1$ (direct observation)
- $Q > 0$ (process noise variance)
- $R > 0$ (observation noise variance)

Remark 1. The ratio Q/R (signal-to-noise ratio) determines filter behavior:

- Large Q/R : State changes rapidly \Rightarrow Filter trusts observations more
- Small Q/R : State is stable \Rightarrow Filter smooths more aggressively

3 Kalman Filter Algorithm

3.1 Notation

- $\hat{x}_{t|s}$: Estimate of x_t given observations up to time s
- $P_{t|s}$: Variance (uncertainty) of $\hat{x}_{t|s}$

3.2 Recursion Steps

Algorithm 1 Kalman Filter (Forward Pass)

Require: Initial state $\hat{x}_{0|0}$, initial covariance $P_{0|0}$, observations $\{y_1, \dots, y_T\}$

Ensure: Filtered states $\{\hat{x}_{t|t}\}$, covariances $\{P_{t|t}\}$

```

1: for  $t = 1$  to  $T$  do
2:   Predict:
3:    $\hat{x}_{t|t-1} = F \cdot \hat{x}_{t-1|t-1}$  ▷ State prediction
4:    $P_{t|t-1} = F \cdot P_{t-1|t-1} \cdot F^T + Q$  ▷ Covariance prediction
5:   Update:
6:    $\nu_t = y_t - H \cdot \hat{x}_{t|t-1}$  ▷ Innovation (prediction error)
7:    $S_t = H \cdot P_{t|t-1} \cdot H^T + R$  ▷ Innovation covariance
8:    $K_t = P_{t|t-1} \cdot H^T \cdot S_t^{-1}$  ▷ Kalman gain
9:    $\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \cdot \nu_t$  ▷ State update
10:   $P_{t|t} = (I - K_t \cdot H) \cdot P_{t|t-1}$  ▷ Covariance update
11: end for
```

3.3 Scalar Version (Local Level Model)

For our scalar model, the recursion simplifies to:

$$\text{Predict: } \hat{x}_{t|t-1} = \hat{x}_{t-1|t-1} \quad (5)$$

$$P_{t|t-1} = P_{t-1|t-1} + Q \quad (6)$$

$$\text{Update: } \nu_t = y_t - \hat{x}_{t|t-1} \quad (7)$$

$$S_t = P_{t|t-1} + R \quad (8)$$

$$K_t = \frac{P_{t|t-1}}{P_{t|t-1} + R} \quad (9)$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \cdot \nu_t \quad (10)$$

$$P_{t|t} = (1 - K_t) \cdot P_{t|t-1} \quad (11)$$

Remark 2 (Kalman Gain Interpretation). The Kalman gain $K_t \in (0, 1)$ determines how much to trust the new observation:

- $K_t \rightarrow 1$: Prediction uncertain ($P_{t|t-1}$ large) \Rightarrow Trust observation
- $K_t \rightarrow 0$: Observation noisy (R large) \Rightarrow Trust prediction

4 Rauch-Tung-Striebel (RTS) Smoother

The RTS smoother improves estimates by using **all** observations (past and future). This creates “oracle” estimates for training purposes.

Algorithm 2 RTS Smoother (Backward Pass)

Require: Filtered states $\{\hat{x}_{t|t}\}$, covariances $\{P_{t|t}\}$, predicted $\{P_{t|t-1}\}$

Ensure: Smoothed states $\{\hat{x}_{t|T}\}$, covariances $\{P_{t|T}\}$

- 1: Initialize: $\hat{x}_{T|T}, P_{T|T}$ (from forward pass)
 - 2: **for** $t = T - 1$ down to 1 **do**
 - 3: $J_t = P_{t|t} \cdot F^T \cdot P_{t+1|t}^{-1}$ \triangleright Smoother gain
 - 4: $\hat{x}_{t|T} = \hat{x}_{t|t} + J_t \cdot (\hat{x}_{t+1|T} - \hat{x}_{t+1|t})$ \triangleright Smoothed state
 - 5: $P_{t|T} = P_{t|t} + J_t \cdot (P_{t+1|T} - P_{t+1|t}) \cdot J_t^T$ \triangleright Smoothed covariance
 - 6: **end for**
-

Remark 3 (Critical Warning). **RTS Smoother uses future data!** At time t , the smoothed estimate $\hat{x}_{t|T}$ depends on y_{t+1}, \dots, y_T .

Usage Rules:

- ✓ In-sample: Use RTS to create training labels
- ✗ Out-of-sample: **NEVER** use RTS (look-ahead bias!)

5 Parameter Estimation via EM Algorithm

The noise parameters Q and R are unknown and must be estimated from data. We use the Expectation-Maximization (EM) algorithm.

5.1 EM Algorithm for State-Space Models

Algorithm 3 EM Algorithm for Kalman Filter Parameters

Require: Observations $\{y_1, \dots, y_T\}$, initial guesses $Q^{(0)}, R^{(0)}$

Ensure: Estimated parameters \hat{Q}, \hat{R}

```

1: Initialize  $Q^{(0)}, R^{(0)}$ 
2: for iteration  $k = 1$  to  $K$  do
3:   E-step: Run Kalman filter + RTS smoother with  $Q^{(k-1)}, R^{(k-1)}$ 
4:   Obtain smoothed states  $\{\hat{x}_{t|T}\}$  and covariances  $\{P_{t|T}\}$ 
5:   M-step: Update parameters
6:    $Q^{(k)} = \frac{1}{T-1} \sum_{t=2}^T [P_{t|T} + P_{t-1|T} - 2P_{t,t-1|T} + (\hat{x}_{t|T} - \hat{x}_{t-1|T})^2]$ 
7:    $R^{(k)} = \frac{1}{T} \sum_{t=1}^T [(y_t - \hat{x}_{t|T})^2 + P_{t|T}]$ 
8:   if  $|\mathcal{L}^{(k)} - \mathcal{L}^{(k-1)}| < \epsilon$  then
9:     break ▷ Converged
10:  end if
11: end for

```

5.2 Log-Likelihood Function

The log-likelihood for monitoring convergence:

$$\mathcal{L} = -\frac{T}{2} \log(2\pi) - \frac{1}{2} \sum_{t=1}^T \left[\log(S_t) + \frac{\nu_t^2}{S_t} \right] \quad (12)$$

5.3 Initialization Strategy

Good initialization is critical for EM convergence:

$$Q^{(0)} = \text{Var}(\Delta y) = \text{Var}(y_t - y_{t-1}) \quad (13)$$

$$R^{(0)} = \text{Var}(y) \quad (14)$$

6 Implementation

6.1 Python Implementation

```

1 import numpy as np
2
3 class KalmanTrendFilter:
4     """Local Level Kalman Filter with EM parameter estimation."""
5
6     def __init__(self, q_init=0.01, r_init=1.0):
7         self.Q = q_init # Process noise
8         self.R = r_init # Observation noise
9
10    def forward_filter(self, y):
11        """Forward Kalman filter pass."""
12        T = len(y)
13
14        # Storage
15        x_filt = np.zeros(T) # Filtered states
16        P_filt = np.zeros(T) # Filtered covariances
17        x_pred = np.zeros(T) # Predicted states

```

```

18     P_pred = np.zeros(T) # Predicted covariances
19     K = np.zeros(T)      # Kalman gains
20     nu = np.zeros(T)     # Innovations
21
22     # Initialize
23     x_filt[0] = y[0]
24     P_filt[0] = self.R
25
26     for t in range(1, T):
27         # Predict
28         x_pred[t] = x_filt[t-1]
29         P_pred[t] = P_filt[t-1] + self.Q
30
31         # Innovation
32         nu[t] = y[t] - x_pred[t]
33         S = P_pred[t] + self.R
34
35         # Kalman gain
36         K[t] = P_pred[t] / S
37
38         # Update
39         x_filt[t] = x_pred[t] + K[t] * nu[t]
40         P_filt[t] = (1 - K[t]) * P_pred[t]
41
42     return {
43         'x_filt': x_filt,
44         'P_filt': P_filt,
45         'x_pred': x_pred,
46         'P_pred': P_pred,
47         'K': K,
48         'nu': nu
49     }
50
51     def rts_smoother(self, y, filter_results):
52         """RTS backward smoother (IN-SAMPLE ONLY!)."""
53         T = len(y)
54         x_filt = filter_results['x_filt']
55         P_filt = filter_results['P_filt']
56         P_pred = filter_results['P_pred']
57
58         # Storage
59         x_smooth = np.zeros(T)
60         P_smooth = np.zeros(T)
61
62         # Initialize from last filtered
63         x_smooth[-1] = x_filt[-1]
64         P_smooth[-1] = P_filt[-1]
65
66         # Backward pass
67         for t in range(T-2, -1, -1):
68             J = P_filt[t] / P_pred[t+1] if P_pred[t+1] > 0 else 0
69             x_smooth[t] = x_filt[t] + J * (x_smooth[t+1] - x_filt[t])
70             P_smooth[t] = P_filt[t] + J**2 * (P_smooth[t+1] - P_pred[t
+1])
71
72         return {'x_smooth': x_smooth, 'P_smooth': P_smooth}
73
74     def fit_em(self, y, max_iter=50, tol=1e-6):

```

```

75     """Estimate Q and R via EM algorithm."""
76     T = len(y)
77     log_liks = []
78
79     for iteration in range(max_iter):
80         # E-step: Filter + Smooth
81         filt = self.forward_filter(y)
82         smooth = self.rts_smoother(y, filt)
83
84         x_s = smooth['x_smooth']
85         P_s = smooth['P_smooth']
86
87         # Log-likelihood
88         S = filt['P_pred'][1:] + self.R
89         nu = filt['nu'][1:]
90         log_lik = -0.5 * np.sum(np.log(S) + nu**2 / S)
91         log_liks.append(log_lik)
92
93         # M-step: Update Q and R
94         # Q: variance of state changes
95         state_diffs = np.diff(x_s)
96         self.Q = np.mean(state_diffs**2 + P_s[1:] + P_s[:-1])
97
98         # R: observation noise variance
99         resids = y - x_s
100        self.R = np.mean(resids**2 + P_s)
101
102        # Convergence check
103        if iteration > 0 and abs(log_liks[-1] - log_liks[-2]) < tol
104    :
105        break
106
107    return {'Q': self.Q, 'R': self.R, 'log_liks': log_liks}

```

Listing 1: Kalman Filter Class

7 Feature Generation for ML Models

The Kalman Filter provides several valuable features for machine learning:

7.1 Kalman-Derived Features

Feature	Formula	Interpretation
kf_innovation	$\nu_t = y_t - \hat{x}_{t t-1}$	Prediction error; surprises
kf_innovation_abs	$ \nu_t $	Magnitude of surprise
kf_uncertainty	$P_{t t}$	State estimation uncertainty
kf_gain	K_t	Filter responsiveness
kf_state_gap	$y_t - \hat{x}_{t t}$	Price vs filtered state
kf_likelihood_ratio	ν_t^2 / S_t	Normalized surprise

Table 1: Kalman-Derived ML Features

7.2 Feature Generation Code

```

1 def generate_kalman_features(kf, prices):
2     """Generate ML features from Kalman Filter."""
3     # Fit parameters on training data
4     kf.fit_em(prices)
5
6     # Forward filter (valid for OOS)
7     results = kf.forward_filter(prices)
8
9     features = {
10         'kf_innovation': results['nu'],
11         'kf_innovation_abs': np.abs(results['nu']),
12         'kf_uncertainty': results['P_filt'],
13         'kf_gain': results['K'],
14         'kf_state_gap': prices - results['x_filt'],
15         'kf_likelihood_ratio': results['nu']**2 / (results['P_pred'] +
16         kf.R),
17     }
18     return pd.DataFrame(features)

```

Listing 2: Feature Generation

8 Critical Implementation Rules

8.1 In-Sample vs Out-of-Sample Usage

Component	IS	OOS	Reasoning
Parameter Estimation (EM)	✓	×	Freeze params from IS
Forward Filter	✓	✓	No future information
RTS Smoother	✓	×	Uses future data (bias!)
Oracle Labels (training)	✓	×	Smoothed targets for IS only
ML Features	✓	✓	From forward filter

Table 2: Kalman Component Usage Rules

8.2 Validation Checks

1. **Innovation Whiteness:** Innovations should be uncorrelated

$$\text{Corr}(\nu_t, \nu_{t-k}) \approx 0 \quad \text{for } k > 0 \quad (15)$$

2. **Innovation Normality:** Innovations should be approximately Gaussian

$$\nu_t / \sqrt{S_t} \sim \mathcal{N}(0, 1) \quad (16)$$

3. **Parameter Stability:** Q and R should not drift dramatically

9 Application in Our Pipeline

9.1 Training Phase (In-Sample: 2016-2023)

1. Fit EM on IS data to estimate \hat{Q} and \hat{R}
2. Run RTS smoother to create oracle training labels

3. Generate Kalman features from forward filter
4. Train ML model on Kalman features + other features

9.2 Production Phase (Out-of-Sample: 2024-2026)

1. Use frozen parameters \hat{Q} and \hat{R} from IS
2. Run forward filter only (no smoothing!)
3. Generate Kalman features
4. Apply trained ML model for predictions

9.3 Key Insight

“The Kalman Filter does not generate alpha by prediction alone; it structures uncertainty. The value is in the features, not the raw filtered states.”

10 Conclusion

The Kalman Filter provides:

- **Principled signal extraction** from noisy price data
- **Uncertainty quantification** via state covariance
- **Valuable ML features** (innovation, gain, uncertainty)
- **Oracle labels** for training (via RTS smoother, IS only)

Key implementation rules:

- Estimate parameters on IS data only
- Use forward filter for production/OOS
- Never use RTS smoother on OOS data
- Validate innovation whiteness and normality