Notes on Using a Kalman Filter for Futures Prices

Introduction

This project is a small C++ implementation of a one-dimensional Kalman Filter applied to futures price data. I wrote it mainly to understand how traders use simple Bayesian tools to smooth noisy tick data. The code is intentionally minimal so that the math and logic can be seen directly, without the usual layers of abstraction.

Why a Kalman Filter?

Futures markets such as ES, NQ, and CL move very quickly and produce a lot of "noise." Many ticks are not meaningful price changes; they are just bid/ask updates, microstructure effects, or random fluctuations. A trader watching the tape sees the price jump up and down constantly, even when the real underlying value has barely changed.

The Kalman Filter gives a clean way to separate:

$$\text{observed price} = \text{true price} + \text{noise}.$$

What makes the filter useful is that it updates its estimate every time a new tick comes in, but it does not overreact. It chooses mathematically how much to trust the new tick versus how much to trust the previous estimate. This balance is controlled by the Kalman Gain, which comes directly from Bayesian statistics applied to Gaussian uncertainty.

How the Model Is Set Up

The idea is simple: the true price changes slowly, while the market prints noisy observations around it. So the model assumes:

$$x_k = x_{k-1} + w_k, \qquad w_k \sim N(0, Q),$$

$$z_k = x_k + v_k, \qquad v_k \sim N(0, R).$$

Here:

- Q describes how much the true price can drift between ticks.
- R describes how noisy the market ticks are.

The filter uses these two variance assumptions to compute the Kalman Gain:

$$K = \frac{P_{\text{prior}}}{P_{\text{prior}} + R}.$$

This is the entire reason the filter behaves the way it does. If the noise R is large, the filter ignores the tick. If the uncertainty P is large, the filter follows the tick closely.

Why the Code Looks the Way It Does

The C++ implementation follows the standard one-dimensional Kalman update:

$$x_{\text{new}} = x_{\text{prior}} + K(z - x_{\text{prior}}),$$

$$P_{\text{new}} = (1 - K)P_{\text{prior}}.$$

I did not add any additional smoothing, windows, or market-specific tricks, because the goal was to show the pure statistical mechanism. The code simply takes each incoming tick, predicts a small drift, and then updates the estimate according to the formulas above.

In short:

- The code is small because the Kalman Filter itself is small.
- The formulas come directly from Bayesian updating of two Gaussians.
- The behavior makes sense for futures because the noise is large and constant, while the underlying value changes gradually.

Summary

This project demonstrates how a classic filtering technique can clean up noisy futures data. The Kalman Filter provides a mathematically simple way to track a hidden price that moves slowly relative to its noisy observations. The C++ code mirrors the math directly so the logic is transparent and easy to follow.