To help me understand and perhaps implement Kalman filtering, this cheat sheet condenses and complements the explanation of the Kalman filter in Bishop PRML (pdf) section 13.3. I'm having trouble with math typesetting on the web, so here's the markdown and pdf of this post.

--

The Kalman filter

The Kalman filter is a statistical model often used for tracking objects through space. It uses a sequence of hidden variables (z_n below) that follows a discrete random walk. These are related to the observed data (x_n below) via a regression-like model: linear plus Normal noise. Here is the full specification, with mnemonics based on the latin equivalents of the Greek letters: $G = \Gamma, S = \Sigma$.

- $z_1 = \mu_0 + u$ with $u \sim N(0, P_0)$ (mnemonic: P_0 rior)
- $z_n = Az_{n-1} + w_n$ with $w_n \sim N(0, \Gamma)$ (mnemonic: $w$here it Actually Γoes)
- $x_n = Cz_n + v_n$ with $v_n \sim N(0, \Sigma)$ (mnemonic: $v$ision is what you Can Σee)

Bishop covers two common goals:

1. Predict z_n given X.
2. Infer the parameters $A, C, \mu_0, \Sigma, \Gamma, P_0$.

Building blocks

For these tasks, it is useful to define some building blocks and simplify them. Bishop does this through unpleasant matrix identities, results on multivariate Gaussians, and re-use of conditional independence properties that he develops earlier in the chapter for another model that shares the same structure. I'm just going to rattle them off, adding explanation only where I couldn't understand Bishop's.

These building blocks are computed recursively with one forward pass and one backward pass. These will be useful throughout the post. The forward recursion provides details to predict z_n given $x_1 \ldots x_n$. The backward recursion allows predictions of z_n including all of X. An EM alorithm for inference can also be built on these components.

Forward recursion:

- $P_{n-1} \equiv \mathrm{Cov}(z_n | x_1, \ldots, x_{n-1}) = AV_{n-1}A + \Gamma$
- $R_{n-1} \equiv \mathrm{Cov}(x_n | x_1, \ldots, x_{n-1}) = CP_{n-1}C^T + \Sigma$
- $K_n \equiv P_{n-1}C(CP_{n-1}C^T + \Sigma)^{-1} = P_{n-1}CR_{n-1}^{-1}$
- $V_n \equiv \mathrm{Cov}(z_n | x_1, \ldots, x_n) = (I - K_nC)P_{n-1}$
- $\mu_n \equiv \mathbb{E}(z_n | x_1, \ldots, x_n) = A\mu_{n-1} + K_n(x_n - CA\mu_{n-1})$

Backward recursion:

- $J_n \equiv V_nAP_n^{-1} = V_nA(AV_nA + \Gamma)^{-1}$
- $\hat{V}_n \equiv \mathrm{Cov}(z_n | X) = V_n - J_n(\hat{V}_{n+1} - P_n)J_n^T$
- $\hat{\mu}_n \equiv \mathbb{E}(z_n | X) = \mu_n + J_n(\hat{\mu}_{n+1} - A\mu_n)$

Notes and explanations to supplement Bishop:

- You can start the forward recursion with P_0 and μ_0. For the backward recursion, you can set $\hat{V}_N = V_N$ and $\hat{\mu}_N = \mu_N$, where N is the length of the observed sequence. If you check the definitions, you'll see why these things should match. Also, *N.B.*, I learned the hard way not to use n and N as separate variables in my code.
- For the backward recursion, be careful with the treacherous mix of hatted and non-hatted variables; $V, \hat{V}, \mu, \hat{\mu}$. I think you always use the hatted version unless it hasn't been computed yet. You often use the hatted version from the previous iteration with the non-hatted version from the current iteration.
- Bishop doesn't use R -- I added it. Bishop also doesn't explain that P has a straightforward meaning; he just gives the formula.
- K_n and J_n are mysterious to me, but they are very similar to the "hat matrices" that solve regularized and weighted least squares problems. Check out the wikipedia articles on ridge regression and WLS and tell me if you can figure out more details.
- Covariance matrices of the form $AVA^T + \Gamma$ or $CPC^T + \Sigma$ can arise from doing a linear transformation and adding noise. In those cases, it's $Az + w$ or $Cz + v$.

Learning the parameters

The log likelihood for the whole model is this (plus a constant). I multiply by -2 just because it's cleaner to look at; otherwise there's an ugly ubiquitous $-\frac{1}{2}$. Multiplying by -2 doesn't change the expectation or the location of the optimum. We just have to minimize it instead of maximizing it later on.

$$\begin{aligned} -2LogL \equiv & \operatorname{logdet}(P_0) + (z_1 - \mu_0)^T P_0^{-1}(z_1 - \mu_0) \\ & + \sum_{n=2}^{N} \operatorname{logdet}(\Gamma) + (z_n - Az_{n-1})^T \Gamma^{-1}(z_n - Az_{n-1}) \\ & + \sum_{n=1}^{N} \operatorname{logdet}(\Sigma) + (x_n - Cz_n)^T \Sigma^{-1}(x_n - Cz_n) \end{aligned}$$

For EM, we need $\mathrm{argmax}_\theta \mathrm{E}_{\mathrm{z}|\mathrm{X},\theta^{\mathrm{old}}}[\mathrm{LogL}(\mathrm{x},\mathrm{z},\theta)]$. The LogL depends on z via a linear function of a few sufficient stats, so the following building blocks are useful and sufficient for the update. This partly depends on unrolling quadratic terms like $\mathbb{E}[z^T M z] = \mathrm{Tr}(M\mathbb{E}[zz^T])$.

- $E_n \equiv \mathbb{E}(z_n|X) = \hat{\mu}_n$
- $E_{n,n} \equiv \mathbb{E}(z_n z_n^T|X) = \hat{V}_n + \hat{\mu}_n\hat{\mu}_n^T$
- $E_{n-1,n} \equiv E_{n-1,n}^T \equiv \mathbb{E}(z_{n-1} z_n^T|X) = J_{n-1}\hat{V}_n + \hat{\mu}_{n-1}\hat{\mu}_n^T$

The actual update can be found separately for each line of the LogL as it appears above. Unless noted, the RHS uses only the "old" parameters. Here is the result.

- $\mu_0 \leftarrow \hat{E}_n = \mu_1$
- $P_0 \leftarrow E_{n,n} - E_n E_n^T = \hat{V}_1$
- $A \leftarrow (\sum_{n=2}^N E_{n,n})^{-1}(\sum_{n=2}^N E_{n,n-1})$
- $\Gamma \leftarrow \frac{1}{N-1}\sum_{n=2}^N E_{n,n} - AE_{n-1,n} - E_{n,n-1}A^T + AE_{n-1,n-1}A^T$ (Note: use the new A in this update.)
- $C \leftarrow \sum_{n=1}^N x_n E_n (\sum_{n=1}^N E_{n,n})^{-1}$
- $\Sigma \leftarrow \frac{1}{N}\sum_{n=1}^N x_n x_n^T - x_n E_n^T C^T - CE_n x_n^T + CE_{n,n}C^T$ (Note: use the new C in this update.)

In the Σ update, Bishop uses C where I use C^T and vice versa. I suspect he's wrong because if C is rectangular, Cz works but $C^T z$ has the wrong sizes. Σ has the same size as x on each axis.

The updates for A and C resemble the classic regression estimate $(X^T X)^{-1} X^T Y$, but they are transposed because in a typical regression looking like $Y \approx X\beta$, you would estimate β, but here we estimate X. The variance estimates are also basically the same as linear regression.

Testing

If I were to code this up, Bishop or I will have made typos (probably already did), and I would need to test the code and catch them. How?

- Matrix size checks. Most languages will check this at run-time when you try to multiply matrices that don't conform. Maybe Rust could even check that at compile time.
- Visual checks. It is comforting to plot and examine some sample paths -- true, observed, and filtered -- for a simple system in 2d or 1d.
- Simulation. It's pretty easy to run this model generatively, then check whether the recovered parameters and sample paths are right. You just have to calibrate your expectations when doing this. For z, you won't get it exactly, ever. For μ_0 or P_0, I suspect it is impossible to get a consistent estimate from a single sequence, but running many sequences would work. The other parameters probably do converge with one long sequence as $N \to \infty$.
- Simple mode. For each parameter, I would want an option where it is constrained to a simple default option such as I for matrices and 0 for means. It would be easier to isolate problems if I could run simulations etc with all but one parameter fixed.
- Invariants. With EM, the observed-data likelihood is supposed to increase monotonically.
- Existing software. There is Python code out there for Kalman filtering. It may not do everything (e.g. EM), but it could help in checking the building blocks.

Testing aside, another implementation consideration is that you'll have to invert a bunch of matrices. One of the biggest things Misha Kilmer taught me was not to do this blindly. Since all the inverses are of symmetric positive definite matrices, Cholesky decomposition and forward+backward substitution would be a nice default option.