**In short, I think the problem is not the single observation but is the outlier.**

First, the convexity is not a problem for me. The natural parameters for COM-Poisson is . Since,

, it’s log-concave in terms of natural parameters. Loosely speaking, since for keeps the sign for hessian, so the convexity is not broken. We can get the hints from the CMP regression: if the log-convexity is not hold for and , then all previous researches about CMP regression are kind of problematic (they just do different kinds of MLE)… This gives a more detailed discussion <https://arxiv.org/pdf/1610.08244.pdf>.

I think the problem you see in the MLE simulation for single observation comes from . To give an intuition, let’s consider a rough approximation:

If we replace them with sample variance and expectation (assume MLE is close to MME), since and   is finite, we need shoot to infinity (under this bad approximation). But , that force to be super large. This is what you see in the simulation example, the MLE is super large in this case…

**However, in the adaptive filtering we have the prior prediction, which is equivalent to adding prior samples. So, we are actually dealing with prior samples + single observation when doing MLE.**

**OK, let me try to explain the “jump” in the filtering now.** When we take a look at the adaptive filtering update (i.e. n=1 in the addtfill\_compois\_v2.docx)

If the observation is much smaller than prior expectation, i.e. , then will be very small (roughly) or even to a negative value. This will cause 2 issues: 1) when but very small, will be large, i.e. we assign too much weight on observation. This leads to the “jump”; 2) when , the covariance matrix is even negative-definite! That’s invalid no matter there’s a “jump” or not. On the contrary, when , the is very small, so the outlier observation will not contribute a lot.

To kill the annoying “outlier”, one natural way is to think about the Fisher scoring, i.e. replace the observed information by expected (Fisher) information:

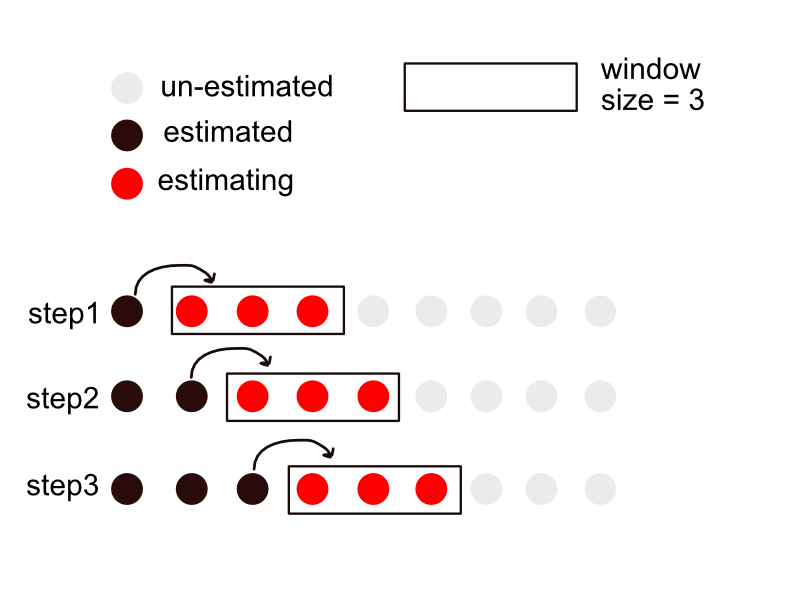
OK, now we will always have a positive-definite posterior variance, and we never need to worry about the “outliers”, by sacrificing some information in the observation. There’s another potential problem, if we initialize badly, the Fisher scoring version may not “adjust” itself quickly, and this might lead to singular .

To compare the raw adaptive filtering and the Fisher scoring, I use the **two case 3 as the examples**. Here, T = 10 and Q is set as [1e-3, 1e-3]. These two are initialized by the corresponding smoothing version.

|  |  |
| --- | --- |
| “raw” adaptive filtering | Fisher scoring |
|  |  |
|  |  |

OK, “Fisher scoring” resolves the “jump”/ robustness problem well, but it is not sensitive enough for observation: it just responds too slow.

The way to keep robustness and improve the accuracy meanwhile is to increase the sample size, i.e. let the observation but not the prior lead the estimation (Moreover, this will be more robust to outliers). Here, I choose to let the algorithm “see further”. The basic idea is illustrated in the following figure:



Each dot represents the we want to estimate. Basically, in each iteration, I use observation in the window to facilitate the estimation by assuming state vectors in the window are the same. After estimation, I only let the first position in the window keep the estimates.

Similarly, we can change the position of the window (the above one is the “forward” case):



This will be discussed in the section “update”. The remaining parts of this section are all “forward” window.

OK, let’s see if it works. Still use case 3 as the example, set the window size k = 1 (original), 5, 10 and 100 (crazy).

|  |  |  |
| --- | --- | --- |
|  | “raw” adaptive filtering | Fisher scoring |
| K=1 |  |  |
| K=5 |  |  |
| K=10 |  |  |
| K=100 |  |  |

OK, it works, although not elegant… We can see that: 1) the algorithm is not very sensitive to the window size, but when the size is too large the assumption “same state vectors in the window” no longer holds; 2) “Fisher scoring” doesn’t sacrifice a lot of information when we use the window. (This is as expected, since as by WLLN).

**To ensure robustness and improve accuracy, maybe we can use the window + fisher scoring? (ALL results in this file are fisher version to ensure the robustness)**

When using the window, the window size becomes another tuning parameter. However, since the algorithm is not sensitive to it (because the fine partition), we can just search it in a coarse grid. Still use case 3 as the example, here I just use window + fisher scoring.

I first search across k = [1, linspace(5, 30, 6)] by maximizing the prediction likelihood, with Q = [1e-4, 1e-4]. (Maybe this time we should use filtering/ smoothing likelihood? Will investigate it in more detail in section “update”). Plot the prediction log-likelihood against the window size:



Then turn on the Q-tuning under k=10, and do both filtering and smoothing:



Although everything looks fine, here’s a big problem: **the Q-tuning is not correct, it just use upper bound diag([1e-3 1e-3])**. Basically, when I use the forward window, the Q will be over-estimated; but when I use the backward and centering window, the Q will be under-estimated. See details in the following section.

# Update

To investigate the performance of Q-tuning, I use the example in “demo\_QTune.m”. Now I set the forward and backward window size be 10, and centering window size be 11 (5 forward + 5 backward). The code can be found in “demo\_QTune\_window.m” In the following plots, the 1st column show heatmaps, the second column show fitting under true Q and tuned Q. The true Q is diag([1e-4 1e-5]).

|  |  |  |
| --- | --- | --- |
|  | Heatmap: prediction llhd | fitting |
| Forward |  |  |
| Backward |  |  |
| Centering |  | Similar to “backward” one. Didn’t run. |

So, before finding a way to unbiasedly estimate Q, I temporarily tune Q without window (window size = 1) at first, and then tune window size under the optimized Q. This is done in the “demo\_QTune\_new.m” and plots (only forward case) can be found in “plots-single-fisher” folder.

When selecting window size under the optimized Q, I plot different log-likelihoods (prediction, filtering and smoothing) against window size:

|  |  |  |
| --- | --- | --- |
| forward | backward | centering |
|  |  |  |

It seems forward case is OK for prediction/ filtering/ smoothing. But the things are more wired for backward & centering…

Fitting:

|  |  |  |
| --- | --- | --- |
| No window | Backward, K=15 | Center, k = 31 |
|  |  |  |

**In this case, no window case seems best, since the Q is perfectly estimated.** However, in the real case, perfect Q estimation is **nearly impossible**, since Q tend to change step by step. Maybe we can do prediction-MLE locally? I previously tried to implement this (<https://www.sciencedirect.com/science/article/pii/S0263224116300732>), but failed… adding window seems another way to rescue the wrongly assumed constant Q.

when window is turned on, these three versions perform similarly. In some other cases, forward version is a little bit (just a little bit) better than the other 2 versions. In the following parts, I only use the forward version and optimize based on filtering/prediction log-likelihood.

**OK, let’s go back to previous example (“case 3”) and do it in the right order: tune Q without window first and then select the window size under the tuned Q.** The code can be found in “demo\_newCases\_fisher\_window\_v2.m”. The optimized Q is diag([1e-3, 5.43e-5])



Use filtering log-likelihood, k = 10. The fitting results:

|  |  |
| --- | --- |
| **Current correct version:**  **Q = diag([1e-3, 5.43e-5]), k = 10** | Previous wrong version:  Q = diag([1e-3 1e-3]), k = 10 |
|  |  |

Well, pretty similar. But there’s another problem: it must guarantee that the Q-estimation is correct. If Q is incorrectly estimated, we may need a super large window size to rescue this… Here’s an example (case 2). The optimized Q is diag([2.5e-6 2.4e-7]), and the log-likelihoods against window size:



|  |  |
| --- | --- |
| No Q tune, no window. Q = diag([1e-3 1e-3]) |  |
| No Q tune: Q = diag([1e-3 1e-3]), k = 15 |  |
| No Q tune: Q = diag([1e-3 1e-3]), k = 100 |  |
| Q tune: Q = diag([2.5e-6 2.4e-7]), k = 400 |  |

So, to use the window correctly:

1. it’s best if we can estimate Q for window version correctly.
2. If not, first try to do Q-tune without window🡪optimize window size.
3. If the optimized window size is super large, maybe we can consider just use large Q, say diag([1e-3 1e-3]) and optimize under the pre-set Q.

TODO:

1. Allow Q change along the time and estimate it locally?
2. See if I can estimate Q correctly in the window version?