# Naming issue

I’m still not very comfortable about calling them as “smoother”. Basically, no matter “smoother/ smoothing” refers to the problem or the method, it’s all about inference. However, we are comparing different models, but not different methods for one model.

Just follow the “model 🡪 inference problem 🡪 inference method”. Under one model, we can ask either (1) filtering problem or (2) smoothing problem . But no matter what inference question we ask or what inference method we use, the model is unchanged. Based on the same rationale, I think we should never name our model with “MAP”, because it’s about inference. Also in Eden et al.’s paper, they never call their model “adaptive filter”.

OK, let’s move on to statistics traditions. These traditions are mostly followed by 2 famous Bayesian time-series textbooks from Mike West et al.(1. <https://www2.stat.duke.edu/~mw/West&HarrisonBook/> & 2. <https://www2.stat.duke.edu/~mw/Prado&Ferreira&WestBook2021/>).

In statistics, “state-space model” always refers to linear evolution by default. The more general version you mentioned is called “general state-space model”. (Just like when we say LM, we will never treat it as a GLM)

Text

Description automatically generated

Another name for the model (in statistics) is dynamic (generalized) linear model (DLM/ DGLM). But I think it’s rarely used in computational neuroscience. (This is why I previously borrow the name “LDS”, but it seems cause some confusions)

BTW, The “state-space model” is used in Liam Paninski et al’s paper about doing MAP directly (<https://link.springer.com/article/10.1007/s10827-009-0179-x>). Cite this.

Well, although I don’t like “smoother”, I’m less important than our readers. If in neuroscience, “smoother” can routinely refer to the model, then use it.

In the following writing, to save words, I just use prefix “d-” and “s-” to refer “dynamic” and “static”.

# dCMP with constant

I simply borrow the optimized Q from dCMP with single . The optimized Q are similar in these 3 cases:

|  |  |  |
| --- | --- | --- |
|  | Intercept for | remainders for |
| dCMP-(5,3) | 0.2928\*1e-4 | 0.1208\*1e-4 |
| dCMP-(5,1) | 0.2934\*1e-4 | 0.1115\*1e-4 |
| dPoi-(5) | 0.2776\*1e-4 | 0.1719\*1e-4 |

Maybe use Q from dPoi when doing the formal fitting? The coordinate descent code can be found in [v1\_comparison\_v3.m](https://github.com/weigcdsb/COM_POISSON/blob/main/demo/v1/v1_comparison_v3.m) (line283-331), and  . The llhd-per-spike for neuron 13 are (half training, half testing):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | dCMP-(5,3) | dCMP-(5,1) | dCMP-(5)-constant nu | dPoi-(5) | sCMP-(5,3) | sPoi-(5) |
| Training llhd/spike | -0.9101 | -0.9108 | -0.9132 | -0.9307 | -0.9266 | -0.9507 |
| Held-out llhd/spike | -0.9099 | -0.9099 | -0.9126 | -0.9338 | -0.9189 | -0.9452 |

Nothing surprise. Conceptually, constant nu dCMP should lie between sCMP-(5,1) and dCMP-(5,1). But we can see it even lies between sCMP-(5,3) and dCMP-(5,1). The training & testing llhd/spk for sCMP-(5,1) are -0.9270 and -0.9193.

Since 1) the results are not surprising, 2) optimizing Q exactly (not treating it as d-Poisson) is not trivial and 3) coordinate descent is somewhat slow, maybe we can just show it in one case but not run for the whole 74 neurons?

# Null & full models

To calculate the “bit/spike”, I need to calculate null model. Here, the null model is null model for CMP. So, the parameters are calculated by MLE and (code: line 483-489). However, this would be tricky for full model: as you found previously, the MLE for single observation might be un-identifiable (because we have no information about variance).

The Poisson model can be even worse than the null CMP, because the is forced to be 1. (Just like LM, the null is  , but if we force it to be some other constants, surely things we be worse).

The null llhd/spike for training and testing are: -0.9505 & -0.9437. Based on this, the “bits/spike” are:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | dCMP-(5,3) | dCMP-(5,1) | dCMP-(5)-constant nu | dPoi-(5) | sCMP-(5,3) | sPoi-(5) |
| Training llhd/spike | 0.0582 | 0.0573 | 0.0539 | 0.0286 | 0.0345 | -0.0003 |
| Held-out llhd/spike | 0.0488 | 0.0488 | 0.0449 | 0.0143 | 0.0357 | -0.0022 |

For reference, training & testing “bits/spike” for sCMP-(5,1) are 0.0338 & 0.0353.

# All 74 neurons

Since “constant nu” is developing, I didn’t run it here. May add sCMP-(5,1) later. Instead of using boxplot/beeswarm, I just use the line chart, because these values should be aligned. (The code: [comparison\_all.m](https://github.com/weigcdsb/COM_POISSON/blob/main/demo/v1/comparison_all.m)).

The training & testing “bits/spike” for all 74 neurons (red lines show the **medians**):



It seems there are 3 neuron with bad d-Poisson fitting. This is because the Poisson smoother fails (ill-conditioned matrix), which gives bad initials and NR is not converged after 1000 iterations. (**This shows another drawback for Poisson smoother: it’s not robust for highly variable data**)

Let’s get rid of these 3 (neuron = 9, 60 and 67) and plot the remaining 71:



It seems some d-Poissons are worse than s-Poisson. Maybe this is also caused by failure of Poisson smoother. OK, let’s further get rid of these (neuron = 9, 38, 43, 50, 59, 60, 65, 67, 69) and plot the remaining 65 neurons.

