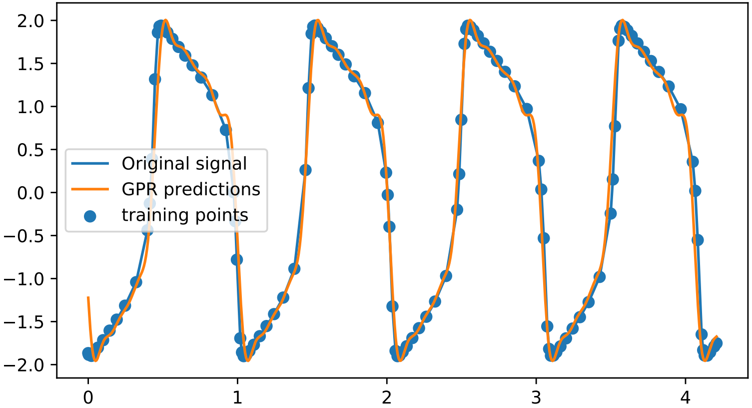
SOTA GPRs for neural data

Mark Blyth

Goals

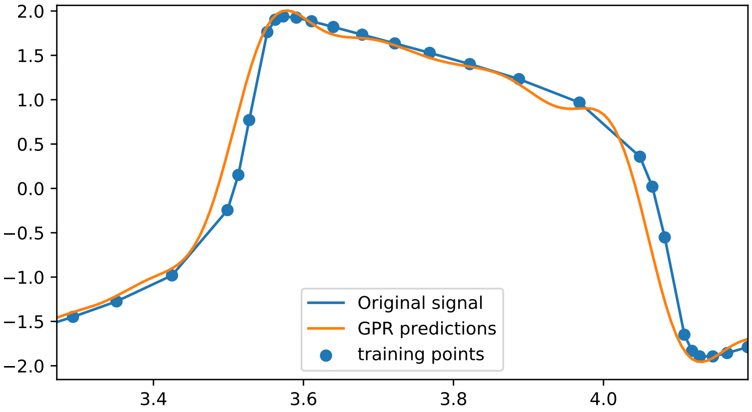
- Redraft the continuation paper
  - Week 1
- Implement and test some GP schemes
  - Week 2

My SEKernel



Reasonable fit, but fixed lengthscales means it struggles at timescale changes.
Good baseline. Bad for real neurons.

My SEKernel



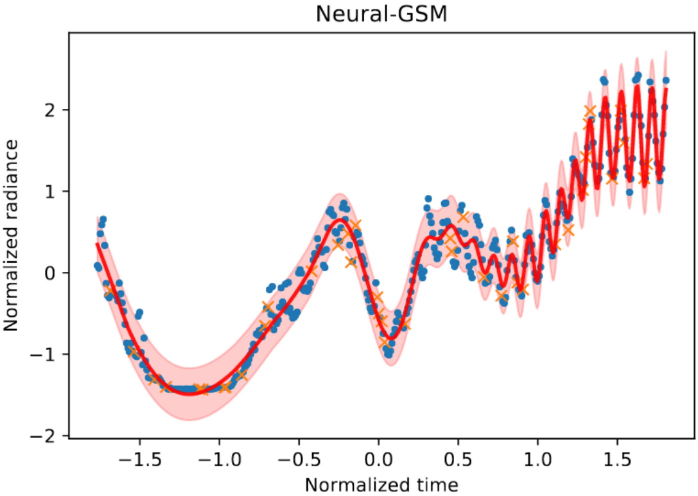
Reasonable fit, but fixed lengthscales means it struggles at timescale changes.
Good baseline. Bad for real neurons.

Generalised nonstationary spectral kernels

- Method identified in the literature review as being applicable to neuron data
- Fits into a sparse GP framework – good for experiments
- Models nonstationarity – varying length scales, function variance
  - Lengthscales quantify local similarity (think: wiggliness)
  - Multiple timescale dynamics means wiggliness changes across the signal
- Open source code available!

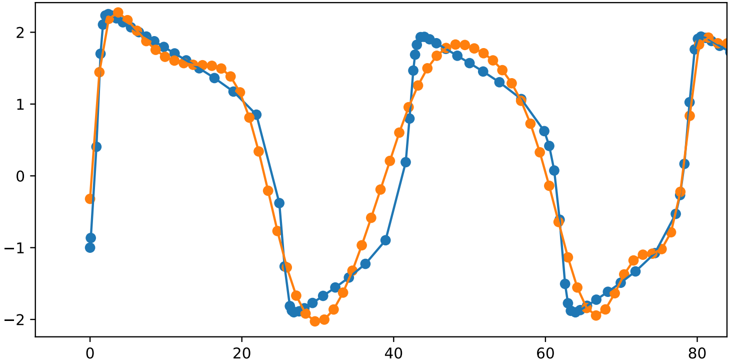
`https://github.com/sremes/nssm-gp`

Published results



Provides state-of-the-art performance on test data

My results



It does look to have varying lengthscales, but it doesn’t work well!

Possible issues

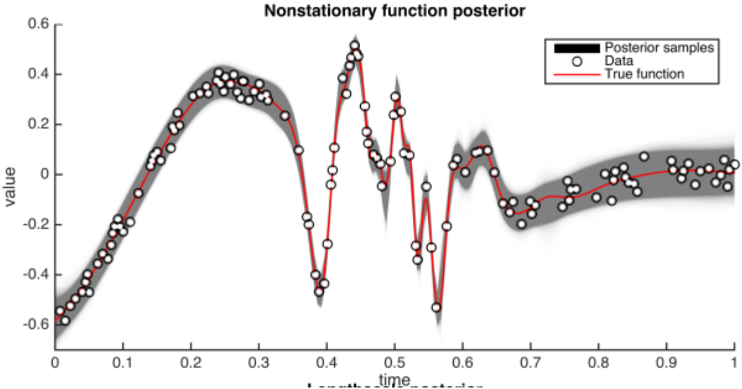
- I don't trust the code
  - Provided code relies on outdated, incompatible tensorflow, GPFlow versions; wouldn't run
  - I rewrote so it would run, but don't know tensorflow, GPFlow, so bad fit could be a code issue
- Not enough data?
- Bad training?
  - I don't know anything about the tensorflow optimizers
- Generally bad method?
  - This was tested using algos from a preprint
  - A near-identical algo was published in a conference, might work better?

Other approaches

Try other nonstationary kernels, or...

- Could use a good stationary kernel and hope its good enough
- Real neurons have very short, fast spikes. Could use one kernel for the spikes, and another for the rest
- GPFlow implements a switching kernel
  - Develop some sort of algo to detect where to switch kernels
  - Fit a switching kernel, based around these changepoints
- Could use hidden Markov chains for a piecewise model

Other nonstationary kernels



Source: Heinonen, Markus, et al. "Non-stationary gaussian process regression with hamiltonian monte carlo." Artificial Intelligence and Statistics. 2016.

Similar idea to the method I already tried, but hopefully with more usable code.

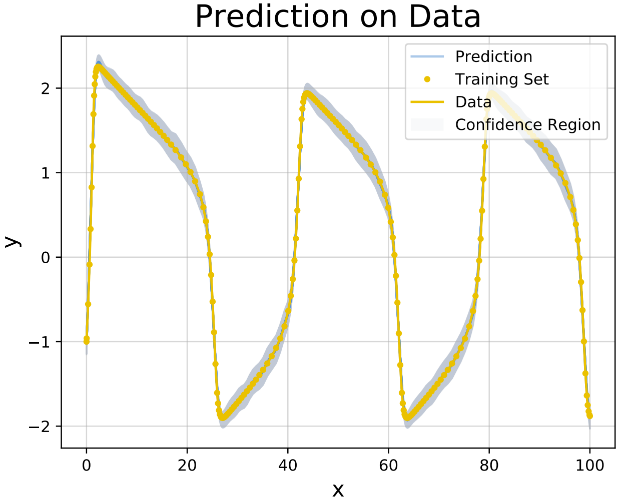
Hidden Markov chain model

- Assume there's two dynamics, $f_q(t)$ for quiescence, $f_s(t)$ for spiking
- Each dynamics are modelled as a random process
- Hidden (latent) variable θ dictates whether the neuron is spiking or quiescent
- θ follows a random process to initiate the transition from quiescence to spiking
- Model:

$$f(t,\theta) = \begin{cases} f_q(t) & \text{if } \theta = 0 \\ f_s(t) & \text{if } \theta = 1 \end{cases} \tag{1}$$

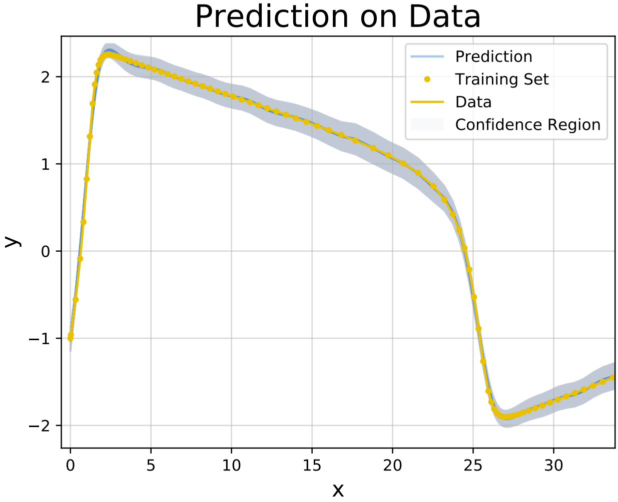
Might be easy, might be hard

Good stationary kernel



Uses the function-space distribution over kernels method; code adapted from
`https://github.com/wjmaddox/spectralgp`

Good stationary kernel



Uses the function-space distribution over kernels method; code adapted from `https://github.com/wjmaddox/spectralgp`

Good stationary kernel

Caveat:

- Fitzhugh-Nagumo is slower changing than real neuron data
  - Real data would likely be strongly nonstationary, in which case this wouldn't work
- Good stationary kernels would be useful for a switching kernel, or a hidden Markov chain model
  - Have a good stationary kernel for the active phase
  - Have a good stationary kernel for the quiescent phase
  - Switch between them at the appropriate points

Good stationary kernel

Note:

- The stationary method shown here is a spectral mixture kernel – SMK
- The unsuccessful method was a generalised (nonstationary) spectral mixture kernel – GSMK
- A GSMK can model any SMK
  - Anything an SMK can model, a GSMK can model equally well
  - Reverse is not true
- Unsuccessful results are likely down to practice (coding issues), rather than theory (invalid kernel choice)

Sidenote on GPFlow

- Based on tensorflow
  - Very fast, very powerful
- Lots of the SOTA work uses GPFlow or GPyTorch
- Might be worth learning how to use it
  - Can implement and test more advanced kernels that way

Next steps

- GPs are tricky on fast-changing data; I still think they'd be useful / worth the time and effort:
  - Clean data source
  - Could allow CBC to be interfaced with existing continuation methods. . .
  - . . . or could be used to make a novel, application-specific / discretization-free continuation method
- More GPR testing
  - Try other kernels (GPFlow periodic, Heinonen Hamiltonian Monte-Carlo, switching, . . .)
  - Try to get GSM kernel to work?
  - Switching kernels?
  - Learn about Tensorflow and GPFlow?